Constraints (2009) 14:254–272
DOI 10.1007/s10601-008-9061-0

Compiling finite linear CSP into SAT

**Naoyuki Tamura · Akiko Taga ·
Satoshi Kitagawa · Mutsunori Banbara**

Published online: 24 December 2008

Abstract In this paper, we propose a new method to encode Constraint Satisfaction Problems (CSP) and Constraint Optimization Problems (COP) with integer linear constraints into Boolean Satisfiability Testing Problems (SAT). The encoding method (named order encoding) is basically the same as the one used to encode Job-Shop Scheduling Problems by Crawford and Baker. Comparison $x \le a$ is encoded by a different Boolean variable for each integer variable x and integer value a. To evaluate the effectiveness of this approach, we applied the method to the Open-Shop Scheduling Problems (OSS). All 192 instances in three OSS benchmark sets are examined, and our program found and proved the optimal results for all instances including three previously undecided problems.

Keywords Constraint satisfaction problems · SAT encoding · Open-shop scheduling problems

1 Introduction

Recent advances in SAT solver technologies [6, 16–18, 20] have enabled solving a problem by encoding it as a SAT problem, and then using an efficient SAT solver to find a solution, such as for model checking, planning, and scheduling [4, 7, 10, 11, 14, 19, 21].

In this paper, we propose a new method to encode Constraint Satisfaction Problems (CSP) and Constraint Optimization Problems (COP) with integer linear

N. Tamura (✉) · M. Banbara
Information Science and Technology Center, Kobe University, Kobe, Japan
e-mail: tamura@kobe-u.ac.jp

A. Taga · S. Kitagawa
Graduate School of Science and Technology, Kobe University, Kobe, Japan

constraints into Boolean Satisfiability Testing Problems (SAT) of CNF (product-of-sums) formulas [23].

As Hoos discussed in [10], basically two encoding methods are known: “sparse encoding” and “compact encoding”. Sparse encoding [5] encodes each assignment of a value to an integer variable by a different Boolean variable, that is, Boolean variable representing $x = a$ is used for each integer variable x and integer value a. The direct encoding [24] and the support encoding [8] are based on the sparse encoding. The compact encoding [7, 12] assigns a Boolean variable for each bit of each integer variable.

The encoding method used in this paper (named order encoding) is different from these. The method is basically the same as the one used to encode Job-Shop Scheduling Problems by Crawford and Baker in [4] and studied by Soh, Inoue, and Nabeshima in [11, 19, 21]. It encodes a comparison $x \le a$ by a different Boolean variable for each integer variable x and integer value a.

The benefit of this encoding is the natural representation of the order relation on integers. Axiom clauses with two literals, such as $\{\neg(x \le a), x \le a+1\}$ for each integer a, represent the order relation for an integer variable x. Clauses, for example $\{x \le a, \neg(y \le a)\}$ for each integer a, can be used to represent the constraint among integer variables, i.e. $x \le y$.

The original encoding method used in [4, 11, 19, 21] is only for Job-Shop Scheduling Problems. In this paper, we extend the method so that it can be applied to any finite linear CSPs and COPs.

To evaluate the effectiveness of this approach, we applied the method to the Graph Coloring Problems and the Open-Shop Scheduling Problems (OSS).

The proposed method gives the better performance compared with the direct encoding [24] and the support encoding [8] for the Graph Coloring Problems.

As for OSS problems, all 192 instances in three OSS benchmark sets proposed in [3, 9, 22] are examined, and our program found and proved the optimal results for all instances including three previously undecided problems [2, 13, 15].

2 Finite linear CSP and SAT

In this section, we define finite linear *Constraint Satisfaction Problems* (CSP) and *Boolean Satisfiability Testing Problems* (SAT) of CNF formulas.

$\mathbf{Z}$ is used to denote a set of integers and $\mathbf{B}$ is used to denote a set of Boolean constants ($\top$ and $\bot$ are the only elements of $\mathbf{B}$ representing “true” and “false” respectively).

We also prepare two countably infinite sets of *integer variables* $\mathcal{V}$ and *Boolean variables* $\mathcal{B}$. Although only a finite number of variables are used in a specific CSP or SAT, countably infinite variables are prepared to introduce new variables during the translation. Symbols $x, y, z, x_1, y_1, z_1, \ldots,$ are used to denote integer variables, and symbols $p, q, r, p_1, q_1, r_1, \ldots,$ are used to denote Boolean variables.

Linear expressions over $V \subset \mathcal{V}$, denoted by $E(V)$, are algebraic expressions in the form of $\sum a_i x_i$ where a_i's are non-zero integers and x_i's are integer variables (elements of V). We also add the restriction that x_i's are mutually distinct.

Literals over $V \subset \mathcal{V}$ and $B \subset \mathcal{B}$, denoted by $L(V, B)$, consist of Boolean variables $\{p \mid p \in B\}$, negations of Boolean variables $\{\neg p \mid p \in B\}$, and *comparisons* $\{e \le c \mid e \in E(V), c \in \mathbf{Z}\}$. Please note that we restrict comparison literals to only appear positively and in the form of $\sum a_i x_i \le c$ without loss of generality. For example, $\neg(a_1x_1 + a_2x_2 \le c)$ can be represented with $-a_1x_1 - a_2x_2 \le -c - 1$, and $x \ne y$ (that is, $(x < y) \vee (x > y)$) can be represented with $(x - y \le -1) \vee (-x + y \le -1)$. Encoding of other expressions, such as max, abs, etc., will be explained in Section 3.5.

Clauses over $V \subset \mathcal{V}$ and $B \subset \mathcal{B}$, denoted by $C(V, B)$, are defined as usual where literals are chosen from $L(V, B)$, that is, a clause represents a disjunction of element literals. Integer variables occurring in a clause are treated as free variables, that is, a clause $\{x \le 0\}$ does not mean $\forall x.(x \le 0)$.

Definition 1 (Finite linear CSP) A (finite linear) CSP (Constraint Satisfaction Problem) is defined as a tuple (V, ℓ, u, B, S) where

(1) V is a finite subset of *integer variables* $\mathcal{V}$,
(2) ℓ is a mapping from V to $\mathbf{Z}$ representing the *lower bound* of the integer variable,
(3) u is a mapping from V to $\mathbf{Z}$ representing the *upper bound* of the integer variable,
(4) B is a finite subset of *Boolean variables* $\mathcal{B}$, and
(5) S is a finite set of clauses (that is, a finite subset of $C(V, B)$) representing the constraint to be satisfied.

In the rest of this paper, we simply call finite linear CSP as CSP.

We extend the functions ℓ and u for any linear expressions $e \in E(V)$, e.g. $\ell(2x - 3y) = -9$ and $u(2x - 3y) = 6$ when $\ell(x) = \ell(y) = 0$ and $u(x) = u(y) = 3$.

An *assignment* of a CSP (V, ℓ, u, B, S) is a pair (α, β) where α is a mapping from V to $\mathbf{Z}$ and β is a mapping from B to $\{\top, \bot\}$.

Definition 2 (Satisfiability) Let (V, ℓ, u, B, S) be a CSP. A clause $C \in C(V, B)$ is *satisfiable* by an assignment (α, β) if the assignment makes the clause C be true and $\ell(x) \le \alpha(x) \le u(x)$ for all $x \in V$. We denote this satisfiability relation as follows.

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

A clause C is satisfiable if C is satisfiable by some assignment.

A set of clauses is satisfiable when all clauses in the set are satisfiable by the same assignment. A logical formula is satisfiable when its clausal form is satisfiable. The CSP is satisfiable if the set of clauses S is satisfiable.

Finally, we define SAT as a special form of CSP.

Definition 3 (SAT) A SAT (Boolean Satisfiability Testing Problem) is a CSP without integer variables, that is, $(\emptyset, \emptyset, \emptyset, B, S)$.

3 Encoding finite linear CSP to SAT

3.1 Converting comparisons to primitive comparisons

In this section, we will explain a method to transform a comparison into primitive comparisons.

A *primitive comparison* is a comparison in the form of $x \le c$ where x is an integer variable and c is an integer satisfying $\ell(x) - 1 \le c \le u(x)$. In fact, it is possible to restrict the range of c to $\ell(x) \le c \le u(x) - 1$ since $x \le \ell(x) - 1$ is always false and $x \le u(x)$ is always true. However, we use the wider range to simplify the discussion.

Let us consider a comparison of $x + y \le 7$ when $\ell(x) = \ell(y) = 2$ and $u(x) = u(y) = 6$. As shown in Fig. 1, the comparison can be equivalently expressed as $(x \le 1 \vee y \le 5) \wedge (x \le 2 \vee y \le 4) \wedge (x \le 3 \vee y \le 3) \wedge (x \le 4 \vee y \le 2) \wedge (x \le 5 \vee y \le 1)$ in which 10 black dotted points are contained as satisfiable assignments since $0 \le x$, $y \le 6$. Please note that conditions $(x \le 1 \vee y \le 5)$ and $(x \le 5 \vee y \le 1)$, which are equivalent to $y \le 5$ and $x \le 5$ respectively, are necessary to exclude cases of $x = 2$, $y = 6$ and $x = 6$, $y = 2$.

Now, we will show the following lemma before describing the conversion to primitive comparisons in general.

Lemma 1 *Let (V, ℓ, u, B, S) be a CSP, then for any assignment (α, β) of the CSP, for any linear expressions e, $f \in E(V)$, and for any integer $c \ge \ell(e) + \ell(f)$, the following holds.*

$$
\begin{aligned}
& (\alpha, \beta) \models e + f \le c \\
\Longleftrightarrow \quad & (\alpha, \beta) \models \bigwedge_{a+b=c-1} (e \le a \ \vee \ f \le b)
\end{aligned}
$$

Parameters a and b range over $\mathbf{Z}$ satisfying $a + b = c - 1$, $\ell(e) - 1 \le a \le u(e)$, and $\ell(f) - 1 \le b \le u(f)$. The conjunction represents $\top$ if there are no such a and b.

Proof ($\Longrightarrow$) From the hypotheses and the definition of satisfiability, we get $\alpha(e) + \alpha(f) \le c$, $\ell(e) \le \alpha(e) \le u(e)$, and $\ell(f) \le \alpha(f) \le u(f)$. Let a and b be any integers satisfying $a + b = c - 1$, $\ell(e) - 1 \le a \le u(e)$, and $\ell(f) - 1 \le b \le u(f)$. If there are no such a and b, the conclusion holds.

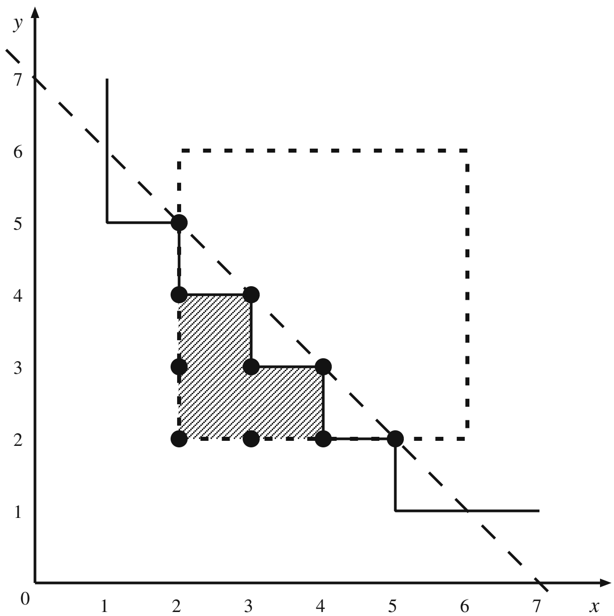

Fig. 1 Converting $x + y \le 7$ to primitive comparisons

If $\alpha(e) \le a$, $e \le a$ in the conclusion is satisfied. Otherwise, $f \le b$ in the conclusion is satisfied since $\alpha(f) \le c - \alpha(e) \le c - a - 1 = (a + b + 1) - a - 1 = b$. Therefore, $e \le a \vee f \le b$ is satisfied for any a and b.

($\Longleftarrow$) From the hypotheses, $\alpha(e) \le a \vee \alpha(f) \le b$ is true for any a and b satisfying $a + b = c - 1$, $\ell(e) - 1 \le a \le u(e)$, and $\ell(f) - 1 \le b \le u(f)$. From the definition of satisfiability, we also have $\ell(e) \le \alpha(e) \le u(e)$ and $\ell(f) \le \alpha(f) \le u(f)$. Now, we show the conclusion through a proof by contradiction. Assume that $\alpha(e) + \alpha(f) > c$ which is the negation of the conclusion.

When $\alpha(e) \ge c - \ell(f) + 1$, we choose $a = c - \ell(f)$ and $b = \ell(f) - 1$. It is easy to check the conditions $\ell(e) - 1 \le a \le u(e)$ and $\ell(f) - 1 \le b \le u(f)$ are satisfied, and $\alpha(e) \le a \vee \alpha(f) \le b$ becomes false for such a and b, which contradicts the hypotheses.

When $\alpha(e) < c - \ell(f) + 1$, we choose $a = \alpha(e) - 1$ and $b = c - \alpha(e)$. It is easy to check the conditions $\ell(e) - 1 \le a \le u(e)$ and $\ell(f) - 1 \le b \le u(f)$ are satisfied, and $\alpha(e) \le a \vee \alpha(f) \le b$ becomes false for such a and b, which contradicts the hypotheses. □

The following proposition shows a general method to convert a (linear) comparison into primitive comparisons.

Proposition 1 *Let (V, ℓ, u, B, S) be a CSP, then for any assignment (α, β) of the CSP, for any linear expression $\sum_{i=1}^n a_i x_i \in E(V)$, and for any integer $c \ge \ell(\sum_{i=1}^n a_i x_i)$ the following holds.*

$$\begin{aligned} & (\alpha, \beta) \models \sum_{i=1}^{n} a_i x_i \le c \\ \Longleftrightarrow \quad & (\alpha, \beta) \models \bigwedge_{\sum_{i=1}^n b_i = c-n+1} \ \bigvee_i (a_i x_i \le b_i)^{\#} \end{aligned}$$

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

$$(a\,x \le b)^{\#} \ \equiv \ \begin{cases} x \le \left\lfloor \dfrac{b}{a} \right\rfloor & (a > 0) \\ \neg \left(x \le \left\lceil \dfrac{b}{a} \right\rceil - 1 \right) & (a < 0) \end{cases}$$

Proof The satisfiability of $\sum a_i x_i \le c$ is equivalent to the satisfiability of $\bigwedge \bigvee (a_i x_i \le b_i)$ from Lemma 1, and the satisfiability of each $a_i x_i \le b_i$ is equivalent to the satisfiability of $(a_i x_i \le b_i)^{\#}$. □

Therefore, any comparison literal $\sum a_i x_i \le c$ in a CSP can be converted to a CNF (product-of-sums) formula of primitive comparisons (or Boolean constants) without changing its satisfiability. Please note that the comparison literal should occur positively in the CSP to perform this conversion.

Example 1 When $\ell(x) = \ell(y) = \ell(z) = 0$ and $u(x) = u(y) = u(z) = 3$, comparison $x + y < z - 1$ is converted into $(x \le -1 \vee y \le -1 \vee \neg(z \le 1)) \wedge (x \le -1 \vee y \le 0 \vee \neg(z \le 2)) \wedge (x \le -1 \vee y \le 1 \vee \neg(z \le 3)) \wedge (x \le 0 \vee y \le -1 \vee \neg(z \le 2)) \wedge (x \le 0 \vee y \le 0 \vee \neg(z \le 3)) \wedge (x \le 1 \vee y \le -1 \vee \neg(z \le 3))$.

3.2 Encoding to SAT

As shown in the previous subsection, any (finite linear) CSP can be converted into a CSP with only primitive comparisons.

Now, we eliminate each primitive comparison $x \le c$ ($x \in V$, $\ell(x) - 1 \le c \le u(x)$) by replacing it with a newly introduced Boolean variable $p(x, c)$ which is chosen from $\mathcal{B}$. We denote a set of these new Boolean variables as follows.

$$B' = \{p(x, c) \mid x \in V,\ \ell(x) - 1 \le c \le u(x)\}$$

We also need to introduce the following axiom clauses $A(x)$ for each integer variable x in order to represent the bound and the order relation.

$$\begin{aligned} A(x) = \{&\{\neg p(x, \ell(x) - 1)\}, \{p(x, u(x))\}\} \\ &\cup \{\{\neg p(x, c - 1), p(x, c)\} \mid \ell(x) \le c \le u(x)\} \end{aligned}$$

As previously described, clauses of $\{\neg p(x, \ell(x) - 1)\}$ and $\{p(x, u(x))\}$ are redundant. However, these will be removed in the early stage of SAT solving and will not affect much the performance of the solver.

Proposition 2 *Let (V, ℓ, u, B, S) be a CSP with only primitive comparisons, let S^* be a clausal form formula obtained from S by replacing each primitive comparison $x \le c$ with $p(x, c)$, and let $A = \bigcup_{x \in V} A(x)$. Then, the following holds.*

$$\begin{aligned} &(V, \ell, u, B, S) \textit{ is satisfiable} \\ \Longleftrightarrow\quad &(\emptyset, \emptyset, \emptyset, B \cup B', S^* \cup A) \textit{ is satisfiable} \end{aligned}$$

Proof ($\Longrightarrow$) Since (V, ℓ, u, B, S) is satisfiable, there is an assignment (α, β) which makes S be true and $\ell(x) \le \alpha(x) \le u(x)$ for all $x \in V$. We extend the mapping β to β^* as follows.

$$\beta^*(p) = \begin{cases} \beta(p) & (p \in B) \\ \alpha(x) \le c & (p = p(x, c) \in B') \end{cases}$$

Then an assignment (α, β^*) satisfies $S^* \cup A$.

($\Longleftarrow$) From the hypotheses, there is an assignment $(\emptyset, \beta)$ which makes $S^* \cup A$ be true. We define a mapping α as follows.

$$\alpha(x) = \min\,\{c \mid \ell(x) \le c \le u(x),\ \ p(x, c)\}$$

It is straightforward to check the assignment (α, β) satisfies S. □

3.3 Keeping clausal form

When encoding a clause of CSP to SAT, the encoded formula is no more a clausal form in general.

Consider a case of encoding a clause $\{x - y \le -1, -x + y \le -1\}$ which means $x \ne y$. Each of $x - y \le -1$ and $-x + y \le -1$ is encoded into a CNF formula of primitive comparisons. Therefore, when we expand the conjunctions to get a clausal form, the number of obtained clauses is the multiplication of two numbers of primitive comparisons.

As it is well known, introduction of new Boolean variables is useful to reduce the size. Suppose $\{c_1, c_2, \ldots, c_n\}$ is a clause of original CSP where c_i's are comparison literals, and $\{C_{i1}, C_{i2}, \ldots, C_{in_i}\}$ is an encoded CNF formula (in clausal form) of c_i for each i.

We introduce new Boolean variables $p_1, p_2, \ldots, p_n$ chosen from $\mathcal{B}$, and replace the original clause with $\{p_1, p_2, \ldots, p_n\}$. We also introduce new clauses $\{\neg p_i\} \cup C_{ij}$ for each $1 \le i \le n$ and $1 \le j \le n_i$.

This conversion does not affect the satisfiability which can be shown from the following lemma.

Lemma 2 *Let (V, ℓ, u, B, S) be a CSP, $\{L_1, L_2, \ldots, L_n\}$ be a clause of the CSP, and $p_1, p_2, \ldots, p_n$ be new Boolean variables. Then, the following holds.*

$$\begin{aligned} & \{L_1, L_2, \ldots, L_n\} \text{ is satisfiable} \\ \iff\ & \{\{p_1, p_2, \ldots, p_n\} \{\neg p_1, L_1\}, \{\neg p_2, L_2\}, \ldots, \{\neg p_n, L_n\}\} \text{ is satisfiable} \end{aligned}$$

Proof ($\Longrightarrow$) From the hypotheses, there is an assignment (α, β) which satisfies some L_i. We extend the mapping β so that $\beta(p_i) = \top$ and $\beta(p_j) = \bot$ $(j \neq i)$. Then, the assignment satisfies converted clauses.

($\Longleftarrow$) From the hypotheses, there is an assignment (α, β) which satisfies some p_i. The assignment also satisfies $\{\neg p_i, L_i\}$, and therefore L_i. Hence the conclusion holds. □

Example 2 Consider an example of encoding a clause $\{x - y \le -1, -x + y \le -1\}$ when $\ell(x) = \ell(y) = 0$ and $u(x) = u(y) = 2$. $x - y \le -1$ and $-x + y \le -1$ are converted into $S_1 = (p(x, -1) \vee \neg p(y, 0)) \wedge (p(x, 0) \vee \neg p(y, 1)) \wedge (p(x, 1) \vee \neg p(y, 2))$ and $S_2 = (\neg p(x, 2) \vee p(y, 1)) \wedge (\neg p(x, 1) \vee p(y, 0)) \wedge (\neg p(x, 0) \vee p(y, -1))$ respectively. Expanding $S_1 \vee S_2$ generates 9 clauses. However, by introducing new Boolean variables p and q, we obtain the following seven clauses.

$$\begin{array}{ccc} & \{p, q\} & \\ \{\neg p, p(x, -1), \neg p(y, 0)\} & \{\neg p, p(x, 0), \neg p(y, 1)\} & \{\neg p, p(x, 1), \neg p(y, 2)\} \\ \{\neg q, \neg p(x, 2), p(y, 1)\} & \{\neg q, \neg p(x, 1), p(y, 0)\} & \{\neg q, \neg p(x, 0), p(y, -1)\} \end{array}$$

3.4 Size of the encoded SAT problem

Usually the size of the encoded SAT problem becomes large.

Suppose the number of integer variables is n, and the size of integer variable domains is d, that is, $d = u(x) - \ell(x) + 1$ for all $x \in V$. Then the size of newly introduced Boolean variables B' is $O(n\,d)$, the size of axiom clauses A is also $O(n\,d)$, and the number of literals in each axiom clause is at most two.

Each comparison $\sum_{i=1}^{m} a_i x_i \le c$ will be encoded into $O(d^{m-1})$ clauses in general by Proposition 1.

However, it is possible to reduce the number of integer variables in each comparison to at most three. For example, $x_1 + x_2 + x_3 + x_4 \le c$ can be replaced with $x + x_3 + x_4 \le c$ by introducing a new integer variable x and new constraints $x - x_1 - x_2 \le 0$ and $-x + x_1 + x_2 \le 0$, that is, $x = x_1 + x_2$.

Therefore, each comparison $\sum_{i=1}^{m} a_i x_i \le c$ can be encoded by at most $O(m\,d^2)$ clauses even when $m \ge 4$, and the number of literals in each clause is at most four (three for integer variables and one for the case handling described in the previous subsection).

3.5 Encoding expressions other than $\sum a_i x_i \le b$

Comparisons and expressions other than $\sum a_i x_i \le b$ can also be encoded to SAT by using the conversion described in Fig. 2.

Comparisons and expressions (the first column) can be replaced with the replacement (the second column) with some extra condition (the third column) where E div c and E mod c are integer quotient and remainder of E divided by an integer constant c.

3.6 Comparison with direct and support encodings

This section compares the proposed encoding with other encodings, especially the direct encoding [24] and the support encoding [8], through the following CSP example.

$$x \in \{2, 3, 4, 5, 6\}$$
$$y \in \{2, 3, 4, 5, 6\}$$
$$x + y \le 7$$

Direct encoding: The *direct encoding* [24] uses Boolean variables p_{xi} meaning $x = i$ for each CSP variable x and each integer constant i ($\ell(x) \le i \le u(x)$). Therefore, 10 Boolean variables are used for the above CSP example.

$$\begin{array}{ccccc} p_{x2} & p_{x3} & p_{x4} & p_{x5} & p_{x6} \\ p_{y2} & p_{y3} & p_{y4} & p_{y5} & p_{y6} \end{array}$$

The following at-least-one and at-most-one clauses are used as axioms for each CSP variable x.

$$p_{x\ell(x)} \vee \cdots \vee p_{xu(x)}$$
$$\neg p_{xi} \vee \neg p_{xj} \qquad (\ell(x) \le i < j \le u(x))$$

Expression	Replacement	Extra condition
$E < F$	$E + 1 \le F$	
$E = F$	$(E \le F) \wedge (E \ge F)$	
$E \ne F$	$(E < F) \vee (E > F)$	
$\max(E, F)$	x	$(x \ge E) \wedge (x \ge F) \wedge ((x \le E) \vee (x \le F))$
$\min(E, F)$	x	$(x \le E) \wedge (x \le F) \wedge ((x \ge E) \vee (x \ge F))$
$\mathrm{abs}(E)$	$\max(E, -E)$	
E div c	q	$(E = cq + r) \wedge (0 \le r) \wedge (r < c)$
E mod c	r	$(E = cq + r) \wedge (0 \le r) \wedge (r < c)$

Fig. 2 Encoding expressions other than $\sum a_i x_i \le b$

Therefore, the following 22 clauses are required for the example.

$$p_{x2} \vee p_{x3} \vee p_{x4} \vee p_{x5} \vee p_{x6}$$
$$\neg p_{x2} \vee \neg p_{x3} \qquad \neg p_{x2} \vee \neg p_{x4} \qquad \neg p_{x2} \vee \neg p_{x5}$$
$$\neg p_{x2} \vee \neg p_{x6} \qquad \neg p_{x3} \vee \neg p_{x4} \qquad \neg p_{x3} \vee \neg p_{x5}$$
$$\neg p_{x3} \vee \neg p_{x6} \qquad \neg p_{x4} \vee \neg p_{x5} \qquad \neg p_{x4} \vee \neg p_{x6} \qquad \neg p_{x5} \vee \neg p_{x6}$$

(similar clauses for y)

Constraints are encoded into clauses representing conflict points. When $x_1 = i_1$, $\ldots$, $x_n = i_n$ violates the constraint, the following clause is added.

$$\neg p_{x_1 i_1} \vee \cdots \vee \neg p_{x_n i_n}$$

Therefore, 15 clauses are used to encode $x + y \le 7$.

$$\neg p_{x2} \vee \neg p_{y6} \quad \neg p_{x3} \vee \neg p_{y5} \quad \neg p_{x3} \vee \neg p_{y6} \quad \neg p_{x4} \vee \neg p_{y4} \quad \neg p_{x4} \vee \neg p_{y5}$$
$$\neg p_{x4} \vee \neg p_{y6} \quad \neg p_{x5} \vee \neg p_{y3} \quad \neg p_{x5} \vee \neg p_{y4} \quad \neg p_{x5} \vee \neg p_{y5} \quad \neg p_{x5} \vee \neg p_{y6}$$
$$\neg p_{x6} \vee \neg p_{y2} \quad \neg p_{x6} \vee \neg p_{y3} \quad \neg p_{x6} \vee \neg p_{y4} \quad \neg p_{x6} \vee \neg p_{y5} \quad \neg p_{x6} \vee \neg p_{y6}$$

Support encoding: The *support encoding* [8] uses the same Boolean variables and axiom clauses with the direct encoding.

Constraints are encoded into clauses representing supports. When $y = j_1, \ldots, y = j_n$ is the support of $x = i$, the following clause is added.

$$\neg p_{xi} \vee p_{yj_1} \vee \cdots \vee p_{yj_n}.$$

Therefore, the following 8 clauses are used for $x + y \le 7$.

$$\neg p_{x2} \vee p_{y2} \vee p_{y3} \vee p_{y4} \vee p_{y5}$$
$$\neg p_{x3} \vee p_{y2} \vee p_{y3} \vee p_{y4}$$
$$\neg p_{x4} \vee p_{y2} \vee p_{y3}$$
$$\neg p_{x5} \vee p_{y2}$$
$$\neg p_{y2} \vee p_{x2} \vee p_{x3} \vee p_{x4} \vee p_{x5}$$
$$\neg p_{y3} \vee p_{x2} \vee p_{x3} \vee p_{x4}$$
$$\neg p_{y4} \vee p_{x2} \vee p_{x3}$$
$$\neg p_{y5} \vee p_{x2}$$

Order encoding: The *order encoding* uses Boolean variables p_{xi} meaning $x \le i$ for each CSP variable x and each integer constant i ($\ell(x) - 1 \le i \le u(x)$). Therefore, the following 12 Boolean variables are used to encode the example.

$$p_{x1} \quad p_{x2} \quad p_{x3} \quad p_{x4} \quad p_{x5} \quad p_{x6}$$
$$p_{y1} \quad p_{y2} \quad p_{y3} \quad p_{y4} \quad p_{y5} \quad p_{y6}$$

The following bound and order clauses are used as axioms for each CSP variable x.

$$\neg p_{x\,\ell(x)-1}$$

$$p_{xu(x)}$$

$$\neg p_{x\,i-1} \vee p_{xi} \qquad (\ell(x) \le i \le u(x))$$

Therefore, the following 14 clauses are required.

$$\neg p_{x1} \qquad\qquad p_{x6}$$
$$\neg p_{x1} \vee p_{x2} \quad \neg p_{x2} \vee p_{x3} \quad \neg p_{x3} \vee p_{x4} \quad \neg p_{x4} \vee p_{x5} \quad \neg p_{x5} \vee p_{x6}$$
(similar clauses for y)

Constraints are encoded into clauses representing conflict regions instead of conflict points. When all points $(x_1, \ldots, x_n)$ in the region $i_1 < x_1 \le j_1, \ \ldots, \ i_n < x_n \le j_n$ violate the constraint, the following clause is added.

$$p_{x_1 i_1} \vee \neg p_{x_1 j_1} \vee \cdots \vee p_{x_n i_n} \vee \neg p_{x_n j_n}$$

Therefore, the following 5 clauses are used to encode $x + y \le 7$.

$$p_{x1} \vee p_{y5} \qquad p_{x2} \vee p_{y4} \qquad p_{x3} \vee p_{y3} \qquad p_{x4} \vee p_{y2} \qquad p_{x5} \vee p_{y1}$$

4 Encoding finite linear COP to SAT

Definition 4 (Finite linear COP) A (finite linear) COP (Constraint Optimization Problem) is defined as a tuple (V, ℓ, u, B, S, v) where

(1) (V, ℓ, u, B, S) is a finite linear CSP, and
(2) $v \in V$ is an integer variable representing the objective variable to be minimized (without loss of generality we assume COPs as minimization problems).

The optimal value of COP (V, ℓ, u, B, S, v) can be obtained by repeatedly solving CSPs.

$$\min \{c \mid \ell(v) \le c \le u(v), \ \text{CSP } (V, \ell, u, B, S \cup \{\{v \le c\}\}) \text{ is satisfiable}\}$$

Of course, instead of linear search, binary search method is useful to find the optimal value efficiently as used in previous works [11, 19, 21].

It is also possible to encode COP to SAT once at first, and repeatedly modify only the clause $\{v \le c\}$ for a given c. This procedure substantially reduces the time spent for encoding.

5 Experimental results

In order to show the applicability of our method, we applied it to the Graph Coloring Problems and the Open-Shop Scheduling (OSS) Problems.

5.1 Graph coloring problems

Graph Coloring Problem (GCP) is a problem to find the minimum number of colors (the chromatic number) for a given undirected graph such that no two adjacent vertices have the same color.

Fig. 3 Comparison of MiniSat (CPU seconds) for 44 GCP instances (a)

Instance	Colors		Order Encoding	Direct Encoding	Support Encoding
DSJC125.1	4	U	0.02	0.01	0.01
	5	S	0.28	0.01	0.06
DSJR500.1	11	U	223.86	-	-
	12	S	0.24	0.13	0.95
le450_5a	4	U	0.16	0.04	0.08
	5	S	1.63	0.20	1.05
le450_5c	4	U	0.24	0.02	0.26
	5	S	0.47	0.09	0.21
le450_5d	4	U	0.09	0.06	0.26
	5	S	0.32	0.08	0.47
anna	10	U	2.16	301.25	1238.57
	11	S	0.02	0.00	0.04
david	10	U	2.11	174.40	1359.68
	11	S	0.02	0.01	0.04
homer	12	U	106.73	-	-
	13	S	0.08	-	-
huck	10	U	1.18	139.06	1189.80
	11	S	0.02	0.00	0.01
jean	9	U	0.22	15.68	36.83
	10	S	0.00	0.01	0.01
games120	8	U	0.10	3.31	7.75
	9	S	0.04	0.01	0.03
miles250	7	U	0.04	0.11	0.53
	8	S	0.01	0.01	0.04
queen5_5	4	U	0.00	0.00	0.00
	5	S	0.00	0.00	0.00
queen6_6	6	U	4.86	1.37	6.42
	7	S	0.01	0.01	0.10
queen7_7	6	U	0.02	0.01	0.04
	7	S	0.01	0.00	0.01
queen8_12	11	U	26.27	-	-
	12	S	0.03	0.03	0.28
myciel3	3	U	0.00	0.00	0.00
	4	S	0.00	0.00	0.00
myciel4	4	U	0.04	0.04	0.05
	5	S	0.00	0.00	0.00
myciel5	5	U	57.30	273.22	570.50
	6	S	0.00	0.00	0.00
mug88_1	3	U	0.01	0.00	0.01
	4	S	0.00	0.00	0.00
mug88_25	3	U	0.01	0.01	0.01
	4	S	0.00	0.00	0.00
mug100_1	3	U	0.01	0.00	0.01
	4	S	0.00	0.00	0.00

All of 119 benchmark instances listed in the web site of the Computational Symposium on Graph Coloring[1] are used to compare the order encoding with the direct encoding [24] and the support encoding [8].

The encoding programs are written in Perl and MiniSat [6] is used as the SAT solver. These programs find the minimum chromatic number by changing the number of colors with binary search method.

The order encoding program decided the minimum chromatic number of 44 instances within the time limit of 1800 s executed on Intel Xeon 2.8 GHz 4 GB memory machine, while the direct and support encoding programs solved 39 and 38 instances respectively which are included in the 44 instances.

[1] http://mat.gsia.cmu.edu/COLOR04/

Fig. 4 Comparison of MiniSat (CPU seconds) for 44 GCP instances (b)

Instance	Colors		Order Encoding	Direct Encoding	Support Encoding
mug100_25	3	U	0.01	0.01	0.01
	4	S	0.00	0.00	0.00
abb313GPIA	8	U	649.23	-	-
	9	S	19.06	-	-
ash331GPIA	3	U	0.08	0.01	0.02
	4	S	0.03	0.01	0.07
ash608GPIA	3	U	0.20	0.01	0.11
	4	S	0.09	0.06	0.16
ash958GPIA	3	U	0.16	0.03	0.17
	4	S	0.44	0.08	0.24
will199GPIA	6	U	0.18	0.16	0.69
	7	S	0.18	0.10	0.27
1-Insertions_4	4	U	131.34	403.12	881.53
	5	S	0.00	0.00	0.00
2-Insertions_3	3	U	0.04	0.02	0.03
	4	S	0.00	0.00	0.00
3-Insertions_3	3	U	0.24	0.11	0.22
	4	S	0.00	0.00	0.01
4-Insertions_3	3	U	1.80	1.62	2.42
	4	S	0.00	0.00	0.00
1-FullIns_3	3	U	0.00	0.00	0.00
	4	S	0.00	0.00	0.00
1-FullIns_4	4	U	0.03	0.01	0.03
	5	S	0.02	0.00	0.02
1-FullIns_5	5	U	10.46	15.34	60.34
	6	S	0.04	0.01	0.12
2-FullIns_3	4	U	0.00	0.00	0.00
	5	S	0.00	0.00	0.01
2-FullIns_4	5	U	0.14	0.10	0.39
	6	S	0.03	0.01	0.03
2-FullIns_5	6	U	217.55	999.09	-
	7	S	0.62	0.04	-
3-FullIns_3	5	U	0.02	0.01	0.02
	6	S	0.02	0.00	0.02
3-FullIns_4	6	U	1.48	3.87	14.96
	7	S	0.16	0.05	0.13
4-FullIns_3	6	U	0.06	0.05	0.12
	7	S	0.00	0.00	0.02
4-FullIns_4	7	U	17.47	192.58	314.54
	8	S	0.19	0.04	0.31
5-FullIns_3	7	U	0.23	0.68	4.46
	8	S	0.01	0.01	0.02
5-FullIns_4	8	U	277.85	-	-
	9	S	0.69	0.22	0.33

Figures 3 and 4 list the solved instances for each encoding method along with the CPU seconds of MiniSat solver for the SAT problem with the optimum chromatic number c (followed by a mark of "S") and the SAT problem with $c-1$ colors (followed by a mark of "U"). For example, the instance DSJC125.1 can be colored with five colors but not with four colors.

The result shows the order encoding is also effective for CSPs with not-equals constraints.

Fig. 5 OSS benchmark instance `gp03-01`

$$(p_{ij}) = \begin{pmatrix} 661 & 6 & 333 \\ 168 & 489 & 343 \\ 171 & 505 & 324 \end{pmatrix}$$

$$\{s_{00} + 661 \le m\} \quad \{s_{01} + 6 \le m\} \quad \{s_{02} + 333 \le m\}$$
$$\{s_{10} + 168 \le m\} \quad \{s_{11} + 489 \le m\} \quad \{s_{12} + 343 \le m\}$$
$$\{s_{20} + 171 \le m\} \quad \{s_{21} + 505 \le m\} \quad \{s_{22} + 324 \le m\}$$
$$\{s_{00} + 661 \le s_{01},\ s_{01} + 6 \le s_{00}\} \quad \{s_{00} + 661 \le s_{02},\ s_{02} + 333 \le s_{00}\}$$
$$\{s_{01} + 6 \le s_{02},\ s_{02} + 333 \le s_{01}\} \quad \{s_{10} + 168 \le s_{11},\ s_{11} + 489 \le s_{10}\}$$
$$\{s_{10} + 168 \le s_{12},\ s_{12} + 343 \le s_{10}\} \quad \{s_{11} + 489 \le s_{12},\ s_{12} + 343 \le s_{11}\}$$
$$\{s_{20} + 171 \le s_{21},\ s_{21} + 505 \le s_{20}\} \quad \{s_{20} + 171 \le s_{22},\ s_{22} + 324 \le s_{20}\}$$
$$\{s_{21} + 505 \le s_{22},\ s_{22} + 324 \le s_{21}\} \quad \{s_{00} + 661 \le s_{10},\ s_{10} + 168 \le s_{00}\}$$
$$\{s_{00} + 661 \le s_{20},\ s_{20} + 171 \le s_{00}\} \quad \{s_{10} + 168 \le s_{20},\ s_{20} + 171 \le s_{10}\}$$
$$\{s_{01} + 6 \le s_{11},\ s_{11} + 489 \le s_{01}\} \quad \{s_{01} + 6 \le s_{21},\ s_{21} + 505 \le s_{01}\}$$
$$\{s_{11} + 489 \le s_{21},\ s_{21} + 505 \le s_{11}\} \quad \{s_{02} + 333 \le s_{12},\ s_{12} + 343 \le s_{02}\}$$
$$\{s_{02} + 333 \le s_{22},\ s_{22} + 324 \le s_{02}\} \quad \{s_{12} + 343 \le s_{22},\ s_{22} + 324 \le s_{12}\}$$

Fig. 6 CSP representation of `gp03-01`

5.2 Open-shop scheduling problems

There are three well-known sets of OSS (Open-Shop Scheduling) benchmark problems by Guéret and Prins [9] (80 instances denoted by `gp*`), Taillard [22] (60 instances denoted by `tai_*`), and Brucker et al. [3] (52 instances denoted by `j*`), which are also used in [2, 13, 15].

Some problems in these benchmark sets are very hard to solve. Actually, three instances (`j7-per0-0`, `j8-per0-1`, and `j8-per10-2`) are still open, and 37 instances are closed recently in 2005 by complete MCS-based search solver of ILOG [15].

Representing OSS problem as CSP is straightforward. Figure 5 defines a benchmark instance `gp03-01` of 3 jobs and 3 machines. Each element p_{ij} represents the process time of the operation O_{ij} ($0 \le i, j \le 2$). The instance `gp03-01` can be represented as a CSP of 27 clauses as shown in Fig. 6.

In the figure, integer variables m represents the makespan and each s_{ij} represents the start time of each operation O_{ij}. Clauses $\{s_{ij} + p_{ij} \le m\}$ represent deadline constraint such that operations should be completed before m. Clauses $\{s_{ij} + p_{ij} \le s_{kl}, s_{kl} + p_{kl} \le s_{ij}\}$ represent resource capacity constraint such that the operation O_{ij} and O_{kl} should not be overlapped each other.

Fig. 7 Number of solved instances within a specified CPU time

CPU time		Number of solved instances
0 min. ..	1 min.	96
1min. ..	10 min.	77
10 min. ..	1 hour	14
1 hour ..	3 hours	3
Unsolved		2
Total		192

Fig. 8 Optimal scheduling of j8-per10-2 found by CSP2SAT

$$(s_{ij}) = \begin{pmatrix} 247 & 296 & 110 & 618 & 537 & 31 & 500 & 127 \\ 815 & 50 & 328 & 274 & 311 & 672 & 550 & 6 \\ 1 & 583 & 120 & 339 & 876 & 842 & 675 & 58 \\ 293 & 669 & 5 & 72 & 250 & 502 & 403 & 994 \\ 286 & 517 & 870 & 594 & 612 & 347 & 0 & 297 \\ 404 & 252 & 73 & 28 & 83 & 25 & 300 & 734 \\ 707 & 997 & 560 & 12 & 48 & 87 & 842 & 340 \\ 53 & 6 & 703 & 285 & 342 & 872 & 526 & 547 \end{pmatrix}$$

Before encoding the CSP to SAT, we also need to determine the lower and upper bound of integer variables. We used the following values ℓ and u (where n is the number of jobs and machines).

$$\ell = \max\left(\max_{0 \le i < n} \sum_{0 \le j < n} p_{ij},\ \max_{0 \le j < n} \sum_{0 \le i < n} p_{ij} \right)$$

$$u = \sum_{0 \le k < n} \max_{(i-j) \bmod n = k} p_{ij}$$

The value u is used for the upper bound of s_{ij}'s and m, and the value ℓ is used for the lower bound of m (the lower bound 0 is used for s_{ij}'s). For example, $\ell = 1000$ and $u = 1509$ for the instance gp03-01.

We developed a program called CSP2SAT which encodes a CSP representation (of a given OSS problem) into SAT and repeatedly invokes a complete SAT solver to find the optimal solution by binary search.[2] We used MiniSat [6] as the backend complete SAT solver because it is known to be very efficient.

We run CSP2SAT for all 192 instances of the three benchmark sets on Intel Xeon 2.8 GHz 4 GB memory machine with the time limit of 3 h (10800 s).

Figures 12, 13, and 14 provides the results for each benchmark instance. The column named “Optim.” describes the optimal value found by the program, and “CPU” describes the total CPU time in seconds including encoding process. The column named “SAT” describes the numbers of Boolean variables and clauses in the encoded SAT problem. Although time spent for encoding is not shown separately in the figures, it ranges from 1 s to 1163 s and fits linearly with the number of clauses in the encoded SAT program.

CSP2SAT found the optimal solutions for 189 known problems and one unknown problem (j8-per10-2) within 3 h. Figure 7 shows the overall performance. 96 instances (50%) are solved within 1 min, and 173 instances (90%) are solved within 10 min.

The known upper bound of j8-per10-2 was 1009. CSP2SAT improved the result to 1002 and proved there are no solutions for 1001. Figure 8 shows the start times s_{ij} of the optimal scheduling found by the program.

Figure 9 provides the log scale plot of the number of clauses in the encoded SAT problem (x-axis) and the total CPU time (y-axis) for 190 problems. The mark + is

[2]The program will be available at http://bach.istc.kobe-u.ac.jp/csp2sat/.

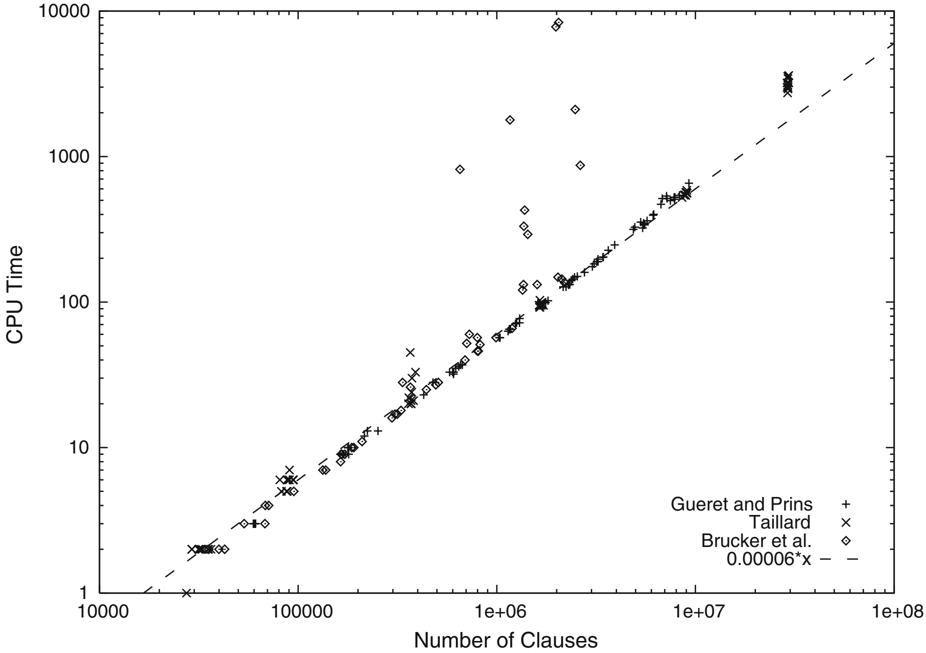

Fig. 9 Log scale plot of the number of clauses and the CPU time (seconds)

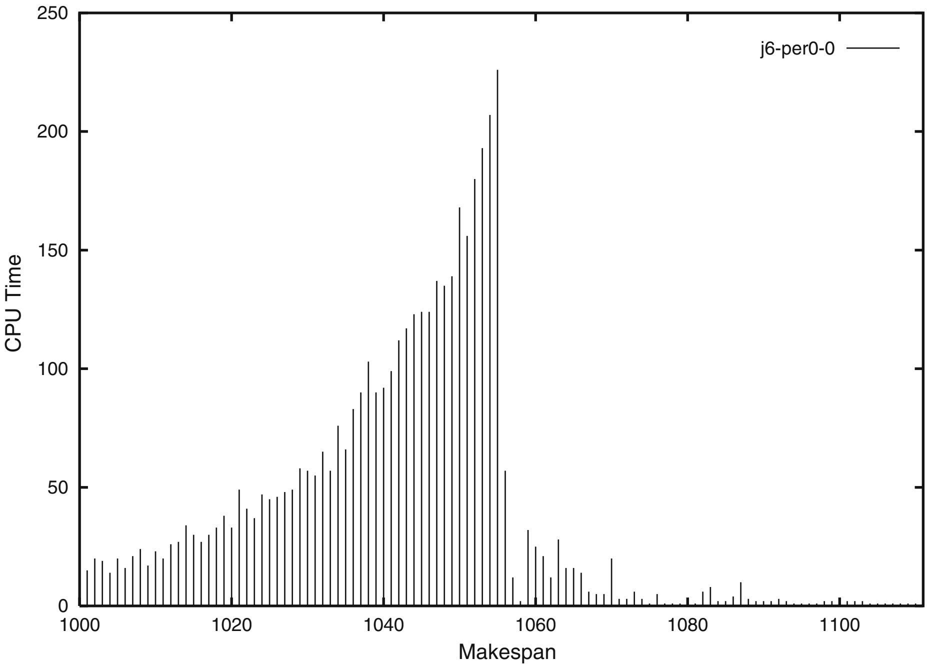

Fig. 10 CPU seconds of checking satisfiability for various makespan values

Fig. 11 New results found and proved to be optimal

Instance	Makespan	Previously known bounds	
		Lower bound	Upper bound
`j7-per0-0`	1048	1039	1048
`j8-per0-1`	1039	1018	1039
`j8-per10-2`	**1002**	1000	1009

used for `gp*` benchmarks, $\times$ is used for `tai*` benchmarks, and $\diamond$ is used for `j*` benchmarks. Dotted line is a plot of $y = 0.00006x$.

Except some instances of `j*` benchmarks, it seems the total CPU time linearly fits with the number of clauses. This shows that the encoding used in this paper is natural and does not uselessly increase the complexity for SAT solver.

Instance	Optim.	CPU	SAT	
			Variables	Clauses
gp03-01	1168	3	14155	61133
gp03-02	1170	3	13945	59978
gp03-03	1168	3	13945	59978
gp03-04	1166	3	13995	60253
gp03-05	1170	3	13855	59483
gp03-06	1169	3	13915	59813
gp03-07	1165	3	13925	59868
gp03-08	1167	3	13955	60033
gp03-09	1162	3	14075	60693
gp03-10	1165	3	13945	59978
gp04-01	1281	10	28097	179010
gp04-02	1270	13	33928	223257
gp04-03	1288	9	28182	179655
gp04-04	1261	12	32925	215646
gp04-05	1289	10	27927	177720
gp04-06	1269	9	27383	173592
gp04-07	1267	9	25955	162756
gp04-08	1259	9	26516	167013
gp04-09	1280	9	26737	168690
gp04-10	1263	13	37736	252153
gp05-01	1245	36	72727	643703
gp05-02	1247	33	65993	578694
gp05-03	1265	37	75457	670058
gp05-04	1258	23	50497	429098
gp05-05	1280	33	68151	599527
gp05-06	1269	37	74131	657257
gp05-07	1269	32	68801	605802
gp05-08	1287	28	55489	477290
gp05-09	1262	35	70387	621113
gp05-10	1254	33	69009	607810
gp06-01	1264	57	96410	1038543
gp06-02	1285	65	106659	1158484
gp06-03	1255	72	115317	1259806
gp06-04	1275	63	104957	1138566
gp06-05	1299	65	107806	1171907
gp06-06	1284	65	106400	1155453
gp06-07	1290	77	119091	1303972
gp06-08	1265	71	113726	1241187
gp06-09	1243	72	118943	1302240
gp06-10	1254	57	95559	1028584

Instance	Optim.	CPU	SAT	
			Variables	Clauses
gp07-01	1159	99	137537	1761090
gp07-02	1185	148	188537	2461830
gp07-03	1237	132	179037	2331300
gp07-04	1167	131	176437	2295576
gp07-05	1157	141	182137	2373894
gp07-06	1193	127	166587	2160237
gp07-07	1185	102	141187	1811241
gp07-08	1180	144	184787	2410305
gp07-09	1220	150	194437	2542896
gp07-10	1270	127	171837	2232372
gp08-01	1130	160	186315	2762188
gp08-02	1135	190	216215	3233688
gp08-03	1110	197	215955	3229588
gp08-04	1153	227	242020	3640613
gp08-05	1218	247	259830	3921463
gp08-06	1115	175	203085	3026638
gp08-07	1126	204	229215	3438688
gp08-08	1148	183	207245	3092238
gp08-09	1114	189	213225	3186538
gp08-10	1161	203	227980	3419213
gp09-01	1129	323	317881	5423978
gp09-02	1110	327	291477	4954180
gp09-03	1115	395	357077	6121380
gp09-04	1130	340	322063	5498387
gp09-05	1180	362	333871	5708483
gp09-06	1093	401	359455	6163691
gp09-07	1090	339	325507	5559665
gp09-08	1105	349	321325	5485256
gp09-09	1123	316	286803	4871017
gp09-10	1110	355	310993	5301422
gp10-01	1093	470	353491	6705492
gp10-02	1097	526	412677	7878078
gp10-03	1081	535	376317	7157718
gp10-04	1077	515	378438	7199739
gp10-05	1071	515	358743	6809544
gp10-06	1071	508	410960	7844061
gp10-07	1079	523	408839	7802040
gp10-08	1093	498	392578	7479879
gp10-09	1112	541	434897	8318298
gp10-10	1092	656	483276	9276777

Fig. 12 Results for benchmark instances provided by Guéret and Prins

Instance	Optim.	CPU	SAT	
			Variables	Clauses
tai_4x4_1	193	2	5043	31706
tai_4x4_2	236	1	4643	27426
tai_4x4_3	271	2	5460	32925
tai_4x4_4	250	2	5358	32341
tai_4x4_5	295	2	6081	36418
tai_4x4_6	189	2	4721	29194
tai_4x4_7	201	2	4743	29188
tai_4x4_8	217	2	5629	35110
tai_4x4_9	261	2	5328	31517
tai_4x4_10	217	2	5611	35444
tai_5x5_1	300	6	11526	94098
tai_5x5_2	262	5	10110	82314
tai_5x5_3	323	6	11318	90297
tai_5x5_4	310	5	11047	88190
tai_5x5_5	326	6	10356	80906
tai_5x5_6	312	5	10942	87344
tai_5x5_7	303	6	10951	87906
tai_5x5_8	300	6	11009	88852
tai_5x5_9	353	6	11940	94884
tai_5x5_10	326	7	11344	90508
tai_7x7_1	435	21	30952	370295
tai_7x7_2	443	24	31244	372853
tai_7x7_3	468	30	31669	374258
tai_7x7_4	463	20	31224	370305
tai_7x7_5	416	22	30171	360661
tai_7x7_6	451	45	30986	367026
tai_7x7_7	422	33	32415	389596
tai_7x7_8	424	20	30863	370287
tai_7x7_9	458	21	31929	380761
tai_7x7_10	398	20	29939	359194
tai_10x10_1	637	98	94183	1678890
tai_10x10_2	588	95	95343	1716326
tai_10x10_3	598	92	92303	1651992
tai_10x10_4	577	92	91314	1639647
tai_10x10_5	640	96	93978	1677177
tai_10x10_6	538	95	91151	1642608
tai_10x10_7	616	103	92285	1648788
tai_10x10_8	595	95	91094	1631685
tai_10x10_9	595	97	94528	1697235
tai_10x10_10	596	95	93315	1674220
tai_15x15_1	937	523	309784	8563684
tai_15x15_2	918	567	325397	9026993
tai_15x15_3	871	543	315726	8767426
tai_15x15_4	934	560	326511	9067128
tai_15x15_5	946	541	323109	8940331
tai_15x15_6	933	560	326512	9067214
tai_15x15_7	891	566	322034	8943618
tai_15x15_8	893	546	319320	8866998
tai_15x15_9	899	568	324060	8998985
tai_15x15_10	902	586	325865	9053491
tai_20x20_1	1155	3105	775142	29178719
tai_20x20_2	1241	3559	777061	29153596
tai_20x20_3	1257	2990	770228	28898989
tai_20x20_4	1248	3442	779059	29238508
tai_20x20_5	1256	3603	785066	29485803
tai_20x20_6	1204	2741	773489	29073596
tai_20x20_7	1294	2912	779414	29225385
tai_20x20_8	1169	2990	778336	29262619
tai_20x20_9	1289	3204	785835	29493666
tai_20x20_10	1241	3208	770645	28917758

Fig. 13 Results for benchmark instances provided by Taillard

Instance	Optim.	CPU	SAT	
			Variables	Clauses
j3-per0-1	1127	2	10805	42708
j3-per0-2	1084	5	20335	95123
j3-per10-0	1131	3	12675	53453
j3-per10-1	1069	3	15335	68062
j3-per10-2	1053	4	15355	68341
j3-per20-0	1026	2	10015	39923
j3-per20-1	1000	2	9245	35496
j3-per20-2	1000	4	15755	71137
j4-per0-0	1055	7	22062	133215
j4-per0-1	1180	11	32160	209841
j4-per0-2	1071	8	26057	163530
j4-per10-0	1041	10	29457	190740
j4-per10-1	1019	7	22538	137589
j4-per10-2	1000	9	26057	164892
j4-per20-0	1000	10	28726	186429
j4-per20-1	1004	9	26074	165849
j4-per20-2	1009	9	26822	171525
j5-per0-0	1042	28	40825	335726
j5-per0-1	1054	28	58687	508163
j5-per0-2	1063	26	44127	367603
j5-per10-0	1004	18	39967	329523
j5-per10-1	1002	17	37653	307928
j5-per10-2	1006	16	36509	296700
j5-per20-0	1000	17	38329	315830
j5-per20-1	1000	27	56607	492707
j5-per20-2	1012	25	51485	442196
j6-per0-0	1056	817	63443	652740
j6-per0-1	1045	57	92340	990913
j6-per0-2	1063	57	75801	797362
j6-per10-0	1005	52	67661	705462
j6-per10-1	1021	46	76467	808206
j6-per10-2	1012	51	77799	823964
j6-per20-0	1000	60	69400	727773
j6-per20-1	1000	46	75431	798740
j6-per20-2	1000	40	66181	692002
j7-per0-0	–	–	85887	1051419
j7-per0-1	1055	428	109837	1380492
j7-per0-2	1056	292	113537	1431330
j7-per10-0	1013	332	108687	1368170
j7-per10-1	1000	121	107087	1347411
j7-per10-2	1011	1786	93887	1165467
j7-per20-0	1000	66	95487	1193523
j7-per20-1	1005	132	125087	1595847
j7-per20-2	1003	132	107987	1361349
j8-per0-1	–	–	145495	2118473
j8-per0-2	1052	870	177995	2630988
j8-per10-0	1017	2107	168310	2481679
j8-per10-1	1000	8346	140620	2047787
j8-per10-2	1002	7789	136655	1984646
j8-per20-0	1000	148	139255	2030756
j8-per20-1	1000	136	149265	2191364
j8-per20-2	1000	144	145300	2125157

Fig. 14 Results for benchmark instances provided by Brucker et al.

We found the most of the CPU time was spent for showing satisfiability and unsatisfiability of the boundary makespan values. Figure 10 plots the CPU seconds spent for checking satisfiability by changing makespan values of the `j6-per0-0` instance. The time of 226 s is required to prove the makespan value 1055 is unsatisfiable, and the time of 57 s is required to prove 1056 is satisfiable for `j6-per0-0` which spent 817 s in total.

For the remaining two open problems `j7-per0-0` and `j8-per0-1`, we solved and proved their optimal values by using 10 Mac mini machines (PowerPC G4 1.42 GHz 1 GB memory) running in parallel on Xgrid system [1] and by dividing the problem into 120 subproblems where each subproblem is obtained by specifying the order of six operations.

Optimal solutions were found and proved for both of the two remaining instances. The makespan value 1048 is proved to be optimal for the instance `j7-per0-0` within 6 h, and the makespan value 1039 is proved to be optimal for the instance `j8-per0-1` within 13 h.

Figure 11 summarizes the newly obtained results. All three remaining open problems in [2, 13, 15] are now closed (Figs. 12, 13, and 14).

6 Conclusion

In this paper, we proposed a method to encode Constraint Satisfaction Problems (CSP) and Constraint Optimization Problems (COP) with integer linear constraints into Boolean Satisfiability Testing Problems (SAT).

To evaluate the effectiveness of this approach, we applied the method to the Graph Coloring Problems and the Open-Shop Scheduling Problems (OSS). The proposed method gives the better performance compared with the direct encoding and the support encoding for the Graph Coloring Problems. As for OSS problems, all 192 instances in three OSS benchmark sets are examined, and our program found and proved the optimal results for all instances including three previously undecided problems.

Acknowledgements We would like to give thanks to Katsumi Inoue, Hidetomo Nabeshima, Takehide Soh, and Shuji Ohnishi for their helpful suggestions.

References

1. Apple Computer Inc. (2004). Xgrid guide. Cupertino: Apple Computer Inc.
2. Blum, C. (2005). Beam-ACO—hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & OR, 32*, 1565–1591.
3. Brucker, P., Hurink, J., Jurisch, B., & Wöstmann, B. (1997). A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics, 76*, 43–59.
4. Crawford, J. M., & Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th national conference on artificial intelligence (AAAI-94)* (pp. 1092–1097).

5. de Kleer, J. (1989). A comparison of ATMS and CSP techniques. In *Proceedings of the 11th international joint conference on artificial intelligence (IJCAI 89)* (pp. 290–296).
6. Eén, N., & Sörensson, N. (2003). An extensible SAT-solver. In *Proceedings of the 6th international conference on theory and applications of satisfiability testing (SAT 2003)* (pp. 502–518).
7. Ernst, M. D., Millstein, T. D., & Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *Proceedings of the 15th international joint conference on artificial intelligence (IJCAI 97)* (pp. 1169–1177).
8. Gent, I. P. (2002). Arc consistency in SAT. In *Proceedings of the 15th european conference on artificial intelligence (ECAI 2002)* (pp. 121–125).
9. Guéret, C., & Prins, C. (1999). A new lower bound for the open-shop problem. *Annals of Operations Research, 92*, 165–183.
10. Hoos, H. H. (1999). SAT-encodings, search space structure, and local search performance. In *Proceedings of the 16th international joint conference on artificial intelligence (IJCAI 99)* (pp. 296–303).
11. Inoue, K., Soh, T., Ueda, S., Sasaura, Y., Banbara, M., & Tamura, N. (2006). A competitive and cooperative approach to propositional satisfiability. *Discrete Applied Mathematics, 154*, 2291–2306.
12. Iwama, K., & Miyazaki, S. (1994). SAT-variable complexity of hard combinatorial problems. In *Proceedings of the IFIP 13th world computer congress* (pp. 253–258).
13. Jussien, N., & Lhomme, O. (2002). Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence, 139*, 21–45.
14. Kautz, H. A., McAllester, D. A., & Selman, B. (1996). Encoding plans in propositional logic. In *Proceedings of the 5th international conference on principles of knowledge representation and reasoning (KR'96)* (pp. 374–384).
15. Laborie, P. (2005). Complete MCS-based search: Application to resource constrained project scheduling. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)* (pp. 181–186).
16. Li, C. M., & Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the fifteenth international joint conference on artificial intelligence (IJCAI 97)* (pp. 366–371).
17. Marques-Silva, J. P., & Sakallah, K. A. (1999). GRAPS: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers, 48*, 506–521.
18. Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th design automation conference (DAC 2001)* (pp. 530–535).
19. Nabeshima, H., Soh, T., Inoue, K., & Iwanuma, K. (2006). Lemma reusing for SAT based planning and scheduling. In *Proceedings of the international conference on automated planning and scheduling 2006 (ICAPS'06)* (pp. 103–112).
20. Selman, B., Kautz, H. A., & Cohen, B. (1996). Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 26*, 521–532.
21. Soh, T., Inoue, K., Banbara, M., & Tamura, N. (2005). Experimental results for solving job-shop scheduling problems with multiple SAT solvers. In *Proceedings of the 1st international workshop on distributed and speculative constraint processing (DSCP'05)*.
22. Taillard, E. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.
23. Tamura, N., Taga, A., Kitagawa, S., & Banbara, M. (2006). Compiling finite linear CSP into SAT. In *Proceedings of the 12th international conference on principles and practice of constraint programming (CP 2006)* (pp. 590–603).
24. Walsh, T. (2000). SAT v CSP. In *Proceedings of the 6th international conference on principles and practice of constraint programming (CP 2000)* (pp. 441–456).