<table>
<thead>
<tr>
<th>Title</th>
<th>Compiling Finite Linear CSP into SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Tamura, Naoyuki / Taga, Akiko / Kitagawa, Satoshi / Banbara, Mutsunori</td>
</tr>
<tr>
<td>Citation</td>
<td>Lecture Notes in Computer Science, 4204 - Principles and Practice of Constraint Programming - CP 2006:590-603</td>
</tr>
<tr>
<td>Issue date</td>
<td>2006-09</td>
</tr>
<tr>
<td>Resource Type</td>
<td>Journal Article / 学術雑誌論文</td>
</tr>
<tr>
<td>Resource Version</td>
<td>author</td>
</tr>
<tr>
<td>DOI</td>
<td>10.1007/11889205_42</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://www.lib.kobe-u.ac.jp/handle_kernel/90000146">http://www.lib.kobe-u.ac.jp/handle_kernel/90000146</a></td>
</tr>
</tbody>
</table>

Create Date: 2018-10-11
Compiling Finite Linear CSP into SAT

Naoyuki Tamura\textsuperscript{1}, Akiko Taga\textsuperscript{2}, Satoshi Kitagawa\textsuperscript{2}, and Mutsunori Banbara\textsuperscript{1}

\begin{flushleft}
\textsuperscript{1} Information Science and Technology Center, Kobe University, JAPAN
tamura@kobe-u.ac.jp
\textsuperscript{2} Graduate School of Science and Technology, Kobe University, JAPAN
\end{flushleft}

Abstract. In this paper, we propose a 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 is basically the same with the one used to encode Job-Shop Scheduling Problems by Crawford and Baker. Comparison $x \leq a$ 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 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.

1 Introduction

Recent advances in SAT solver technologies [1–5] have enabled solving a problem by encoding it to a SAT problem, and then to use the efficient SAT solver to find a solution, such as for model checking, planning, and scheduling [6–12].

In this paper, we propose a method to encode Constraint Satisfaction Problems (CSP) and Constraint Optimization Problems (COP) with integer linear constraints into Boolean Satisfiability Testing Problems (SAT) of CNF (product-of-sums) formulas.

As Hoos discussed in [8], basically two encoding methods are known: “sparse encoding” and “compact encoding”. Sparse encoding [13] 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 $a$. Compact encoding [14, 7] assigns a Boolean variable for each bit of each integer variable.

Encoding method used in this paper is different from these. The method is basically the same with the one used to encode Job-Shop Scheduling Problems by Crawford and Baker in [9] and studied by Soh, Inoue, and Nabeshima in [10–12]. It encodes a comparison $x \leq a$ 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 \leq a), x \leq a + 1\} for each integer $a$, represent the order relation for an integer variable $x$. Clauses,
for example \(\{x \leq a, \neg(y \leq a)\}\) for each integer \(a\), can be used to represent the constraint among integer variables, i.e. \(x \leq y\).

The original encoding method in [9–12] is only for Job-Shop Scheduling Problems. In this paper, we extend the method so that it can be applied for any finite linear CSPs and COPs.

To evaluate the effectiveness of this approach, we applied the method to Open-Shop Scheduling Problems (OSS). All 192 instances in three OSS benchmark sets [15–17] are examined, and our program found and proved the optimal results for all instances including three previously undecided problems [18–20].

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.

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

We also prepare two countably infinite sets of integer variables \(V\) and Boolean variables \(B\). 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.

Literals over \(V \subset V\) and \(B \subset B\), denoted by \(L(V, B)\), consist of Boolean variables \(\{p | p \in B\}\), negations of Boolean variables \(\{\neg p | p \in B\}\), and comparisons \(\{e \leq c | e \in E(V), c \in \mathbb{Z}\}\). Please note that we restrict comparison literals to only appear positively and in the form of \(\sum a_i x_i \leq c\) without loss of generality. For example, \(\neg(a_1 x_1 + a_2 x_2 \leq c)\) can be represented with \(\neg a_1 x_1 - a_2 x_2 \leq -c - 1\), and \(x \neq y\) (that is, \((x < y) \vee (x > y)\)) can be represented with \((x - y \leq -1) \vee (-x + y \leq -1)\).

Clauses over \(V \subset V\) and \(B \subset B\), 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 \((x \leq 0)\) does not mean \(\forall x.(x \leq 0)\).

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

1. \(V\) is a finite subset of integer variables \(V\),
2. \(\ell\) is a mapping from \(V\) to \(\mathbb{Z}\) representing the lower bound of the integer variable,
3. \(u\) is a mapping from \(V\) to \(\mathbb{Z}\) representing the upper bound of the integer variable,
4. \(B\) is a finite subset of Boolean variables \(B\), 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 \( \ell \) 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 \((V, \ell, u, B, S)\) is a pair \((\alpha, \beta)\) where \( \alpha \) is a mapping from \( V \) to \( \mathbb{Z} \) and \( \beta \) is a mapping from \( B \) to \( \{\top, \bot\} \).

Definition 2 (Satisfiability). Let \((V, \ell, u, B, S)\) be a CSP. A clause \( C \in C(V, B) \) is satisfiable by an assignment \((\alpha, \beta)\) if the assignment makes the clause \( C \) be true and \( \ell(x) \leq \alpha(x) \leq u(x) \) 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 \( S \) 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 \( x \leq c \) where \( x \) is an integer variable and \( c \) is an integer satisfying \( \ell(x) - 1 \leq c \leq u(x) \). In fact, it is possible to restrict the range of \( c \) to \( \ell(x) \leq c \leq u(x) - 1 \) since \( x \leq \ell(x) - 1 \) is always false and \( x \leq u(x) \) is always true. However, we use the wider range to simplify the discussion.

Let us consider a comparison of \( x + y \leq 7 \) when \( \ell(x) = \ell(y) = 0 \) and \( u(x) = u(y) = 6 \). As shown in Figure 1, the comparison can be equivalently expressed as \((x \leq 1 \lor y \leq 5) \land (x \leq 2 \lor y \leq 4) \land (x \leq 3 \lor y \leq 3) \land (x \leq 4 \lor y \leq 2) \land (x \leq 5 \lor y \leq 1)\) in which 10 black dotted points are contained as satisfiable assignments since \( 0 \leq x, y \leq 6 \). Please note that conditions \((x \leq 1 \lor y \leq 5)\) and \((x \leq 5 \lor y \leq 1)\), which are equivalent to \( y \leq 5 \) and \( x \leq 5 \) 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 \((V, \ell, u, B, S)\) be a CSP, then for any assignment \((\alpha, \beta)\) of the CSP, for any linear expressions \(e, f \in E(V)\), and for any integer \(c \geq \ell(e) + \ell(f)\), the following holds.

\[
(\alpha, \beta) \models e + f \leq c
\]

\[
\iff (\alpha, \beta) \models \bigwedge_{a+b=c-1} (e \leq a \lor f \leq b)
\]

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

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

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

\((\iff)\) From the hypotheses, \(\alpha(e) \leq a \lor \alpha(f) \leq b\) is true for any \(a\) and \(b\) satisfying \(a + b = c - 1\), \(\ell(e) - 1 \leq a \leq u(e)\), and \(\ell(f) - 1 \leq b \leq u(f)\). From the definition of satisfiability, we also have \(\ell(e) \leq \alpha(e) \leq u(e)\) and \(\ell(f) \leq \alpha(f) \leq u(f)\). 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 \(\alpha(e) \geq c - \ell(f) + 1\), we choose \(a = c - \ell(f)\) and \(b = \ell(f) - 1\). It is easy to check the conditions \(\ell(e) - 1 \leq a \leq u(e)\) and \(\ell(f) - 1 \leq b \leq u(f)\) are satisfied, and \(\alpha(e) \leq a \lor \alpha(f) \leq b\) 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 \( \ell(e) - 1 \leq a \leq u(e) \) and \( \ell(f) - 1 \leq b \leq u(f) \) are satisfied, and \( \alpha(e) \leq a \lor \alpha(f) \leq b \) becomes false for such \( a \) and \( b \), which contradicts the hypotheses. \( \square \)

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

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

\[
(a, \beta) \models \sum_{i=1}^{n} a_i x_i \leq c \\
\iff (a, \beta) \models \bigwedge_{i=1}^{n} a_i x_i \leq b_i \]

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

\[
(a x \leq b) \equiv \begin{cases} 
 x \leq \left\lfloor \frac{b}{a} \right\rfloor & (a > 0) \\
 \neg \left( x \leq \left\lfloor \frac{b}{a} \right\rfloor - 1 \right) & (a < 0) 
\end{cases}
\]

**Proof.** The satisfiability of \( \sum_{i=1}^{n} a_i x_i \leq c \) is equivalent to the satisfiability of \( \bigwedge \bigvee (a_i x_i \leq b_i) \) from Lemma 1, and the satisfiability of each \( a_i x_i \leq b_i \) is equivalent to the satisfiability of \((a x \leq b)\).

Therefore, any comparison literal \( \sum_{i=1}^{n} a_i x_i \leq c \) 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 \((x \leq -1 \lor y \leq -1 \lor \neg z \leq 1) \land (x \leq -1 \lor y \leq 0 \lor \neg (z \leq 2)) \land (x \leq -1 \lor y \leq 1 \lor \neg (z \leq 3)) \land (x \leq 0 \lor y \leq -1 \lor \neg (z \leq 2)) \land (x \leq 0 \lor y \leq 0 \lor \neg (z \leq 3)) \land (x \leq 1 \lor y \leq -1 \lor \neg (z \leq 3)) \).

### 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 \( x \leq c \ (x \in V, \ell(x) - 1 \leq c \leq u(x)) \) by replacing it with a newly introduced Boolean variable \( p(x, c) \) which is chosen from \( B \). We denote a set of these new Boolean variables as follows.

\[
B' = \{p(x, c) \mid x \in V, \ell(x) - 1 \leq c \leq u(x)\}
\]
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.

$$A(x) = \{ \{ \neg p(x, \ell(x) - 1) \}, \{ p(x, u(x)) \} \}$$

$$\cup \{ \{ \neg p(x, c - 1), p(x, c) \mid \ell(x) \leq c \leq u(x) \} \}$$

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 much affect the performance of the solver.

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

$$(V, \ell, u, B, S) \text{ is satisfiable} \iff (\emptyset, \emptyset, \emptyset, B \cup B', S^* \cup A) \text{ is satisfiable}$$

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

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

Then an assignment $(\alpha, \beta^*)$ satisfies $S^* \cup A$.

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

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

It is straightforward to check the assignment $(\alpha, \beta)$ satisfies $S$. $\Box$

### 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 $\{ x - y \leq -1, -x + y \leq -1 \}$ which means $x \neq y$. Each of $x - y \leq -1$ and $-x + y \leq -1$ 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 $\{ C_{i1}, C_{i2}, \ldots, C_{in} \}$ 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 $B$, 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 $1 \leq i \leq n$ and $1 \leq j \leq n$.

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

**Lemma 2.** Let $(V, \ell, u, B, S)$ 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.

$\{L_1, L_2, \ldots, L_n\}$ is satisfiable

$\iff \{\{p_1, p_2, \ldots, p_n\} \{\neg p_1, L_1\}, \{\neg p_2, L_2\}, \ldots, \{\neg p_n, L_n\}\}$ is satisfiable

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

($\iff$) From the hypotheses, there is an assignment $(\alpha, \beta)$ 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 $\{x - y \leq -1, -x + y \leq -1\}$ when $\ell(x) = \ell(y) = 0$ and $u(x) = u(y) = 2$. $x - y \leq -1$ and $-x + y \leq -1$ are converted into

$S_1 = (p(x, -1) \lor \neg p(y, 0)) \land (p(x, 0) \lor \neg p(y, 1)) \land (p(x, 1) \lor \neg p(y, 2))$ and $S_2 = \neg p(x, 2) \lor p(y, 1)) \land \neg p(x, 1) \lor p(y, 0)) \land (\neg p(x, 0) \lor p(y, -1))$

respectively. Expanding $S_1 \lor S_2$ generates 9 clauses. However, by introducing new Boolean variables $p$ and $q$, we obtain the following seven clauses.

$$\{p, q\} \quad \{\neg p, p(x, -1), \neg p(y, 0)\} \quad \{\neg p, p(x, 0), \neg p(y, 1)\} \quad \{\neg p, p(x, 1), \neg p(y, 2)\} \quad \{\neg q, \neg p(x, 2), p(y, 1)\} \quad \{\neg q, \neg p(x, 1), p(y, 0)\} \quad \{\neg q, \neg p(x, 0), p(y, -1)\}$$

### 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 $\sum_{i=1}^{m} a_i x_i \leq c$ 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 at most three. For example, $x_1 + x_2 + x_3 + x_4 \leq c$ can be replaced with $x + x_3 + x_4 \leq c$ by introducing a new integer variable $x$ and new constraints $x - x_1 - x_2 \leq 0$ and $-x + x_1 + x_2 \leq 0$, that is, $x = x_1 + x_2$.

Therefore, each comparison $\sum_{i=1}^{m} a_i x_i \leq c$ can be encoded by at most $O(md^2)$ clauses\(^3\) even when $m \geq 4$, and the number of literals in each clause is

\(^3\) We corrected the number of clauses which was $O(d^2) + O(md)$ in the final version of CP2006 paper.
at most four (three for integer variables and one for the case handling described in the previous subsection).

4 Encoding finite linear COP to SAT

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

(1) \((V, \ell, u, B, S)\) 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 \((V, \ell, u, B, S, v)\) can be obtained by repeatedly solving CSPs.

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

Of course, instead of linear search, binary search method is useful to find the optimal value efficiently as used in previous works [10–12]. It is also possible to encode COP to SAT once at first, and repeatedly modify only the clause \(\{v \leq c\}\) for a given \(c\). This procedure substantially reduces the time spent for encoding.

5 Solving OSS

In order to show the applicability of our method, we applied it to OSS (Open-Shop Scheduling) problems. There are three well-known sets of OSS benchmark problems by Guéret and Prins [15] (80 instances denoted by gp*), Taillard [16] (60 instances denoted by tai_*), and Brucker et al. [17] (52 instances denoted by j*), which are also used in [18–20]. 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 [20].

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

In the figure, integer variables \(m\) represents the makespan and each \(s_{ij}\) represents the start time of each operation \(O_{ij}\). Clauses \(s_{ij} + p_{ij} \leq m\) represent deadline constraint such that operations should be completed before \(m\). Clauses \(s_{ij} + p_{ij} \leq s_{kl}, s_{kl} + p_{kl} \leq s_{ij}\) represent resource capacity constraint such that the operation \(O_{ij}\) and \(O_{kl}\) should not be overlapped each other.
\[(p_{ij}) = \begin{pmatrix} 661 & 6 & 333 \\ 168 & 489 & 343 \\ 171 & 505 & 324 \end{pmatrix} \]

Fig. 2. OSS benchmark instance gp03-01

\[
\begin{align*}
\{s_{00} + 661 \leq m\} & \quad \{s_{01} + 6 \leq m\} & \quad \{s_{02} + 333 \leq m\} \\
\{s_{10} + 168 \leq m\} & \quad \{s_{11} + 489 \leq m\} & \quad \{s_{12} + 343 \leq m\} \\
\{s_{20} + 171 \leq m\} & \quad \{s_{21} + 505 \leq m\} & \quad \{s_{22} + 324 \leq m\} \\
\{s_{00} + 661 \leq s_{01}, s_{01} + 6 \leq s_{00}\} & \quad \{s_{00} + 661 \leq s_{02}, s_{02} + 333 \leq s_{00}\} \\
\{s_{01} + 6 \leq s_{02}, s_{02} + 333 \leq s_{01}\} & \quad \{s_{10} + 168 \leq s_{11}, s_{11} + 489 \leq s_{10}\} \\
\{s_{10} + 168 \leq s_{12}, s_{12} + 343 \leq s_{10}\} & \quad \{s_{11} + 489 \leq s_{12}, s_{12} + 343 \leq s_{11}\} \\
\{s_{20} + 171 \leq s_{21}, s_{21} + 505 \leq s_{20}\} & \quad \{s_{20} + 171 \leq s_{22}, s_{22} + 324 \leq s_{20}\} \\
\{s_{21} + 505 \leq s_{22}, s_{22} + 324 \leq s_{21}\} & \quad \{s_{00} + 661 \leq s_{20}, s_{20} + 171 \leq s_{00}\} \\
\{s_{00} + 661 \leq s_{20}, s_{20} + 171 \leq s_{00}\} & \quad \{s_{01} + 6 \leq s_{21}, s_{21} + 505 \leq s_{01}\} \\
\{s_{11} + 489 \leq s_{21}, s_{21} + 505 \leq s_{11}\} & \quad \{s_{02} + 333 \leq s_{22}, s_{22} + 324 \leq s_{02}\} \\
\{s_{02} + 333 \leq s_{22}, s_{22} + 324 \leq s_{02}\} & \quad \{s_{12} + 343 \leq s_{22}, s_{22} + 324 \leq s_{12}\} \\
\end{align*}
\]

Fig. 3. CSP representation of gp03-01

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

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

\[
u = \sum_{0 \leq k < n} \max_{(i-j) \mod n = k} p_{ij}
\]

The value \(\nu\) is used for the upper bound of \(s_{ij}\)’s and \(m\), and the value \(\ell\) is used for the lower bound of \(m\) (the lower bound 0 is used for \(s_{ij}\)’s). For example, \(\ell = 1000\) and \(\nu = 1509\) 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\(^4\). We used MiniSat [5] as the backend complete SAT solver because it is known to be very efficient (MiniSat is a winner of all industrial categories of the SAT 2005 competition).

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

\(^4\) The program will be available at http://bach.istc.kobe-u.ac.jp/csp2sat/.
\[
(s_{ij}) = \begin{pmatrix}
247 & 296 & 110 & 618 & 537 & 31 & 500 & 127 \\
815 & 50 & 328 & 274 & 311 & 672 & 550 & 6 \\
1 & 583 & 120 & 339 & 876 & 842 & 675 & 58 \\
293 & 669 & 5 & 72 & 250 & 502 & 403 & 994 \\
286 & 517 & 870 & 594 & 612 & 347 & 0 & 297 \\
404 & 252 & 73 & 28 & 83 & 25 & 300 & 734 \\
707 & 997 & 560 & 12 & 48 & 87 & 842 & 340 \\
53 & 6 & 703 & 285 & 342 & 872 & 526 & 547 \\
\end{pmatrix}
\]

Fig. 4. Optimal Scheduling of \texttt{j8-per10-2} found by CSP2SAT

Figures 7, 8, and 9 provides the results. 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 second to 1163 seconds 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 (\texttt{j8-per10-2}) within 3 hours.

The known upper bound of \texttt{j8-per10-2} was 1009. CSP2SAT improved the result to 1002 and proved there are no solutions for 1001. Figure 4 shows the start times \(s_{ij}\) of the optimal scheduling found by the program.

Figure 5 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 used for gp* benchmarks, × is used for tai* benchmarks, and ◊ 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.

For the remaining two open problems \texttt{j7-per0-0} and \texttt{j8-per0-1}, we solved and proved their optimal values by using 10 Mac mini machines (PowerPC G4 1.42GHz 1GB memory) running in parallel on Xgrid system [21] 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 within 13 hours.

Figure 6 summarizes the newly obtained results. All three remaining open problems in [18–20] are now closed.

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 the encoding, we applied the method to 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.

Acknowledgments

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

References

### Table 1: Results for benchmark instances provided by Guéret and Prins

<table>
<thead>
<tr>
<th>Instance</th>
<th>Variables</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp03-01</td>
<td>1168</td>
<td>14155</td>
</tr>
<tr>
<td>gp03-02</td>
<td>1170</td>
<td>13945</td>
</tr>
<tr>
<td>gp03-03</td>
<td>1166</td>
<td>13955</td>
</tr>
<tr>
<td>gp03-04</td>
<td>1170</td>
<td>13855</td>
</tr>
<tr>
<td>gp03-05</td>
<td>1169</td>
<td>13915</td>
</tr>
<tr>
<td>gp03-06</td>
<td>1165</td>
<td>13925</td>
</tr>
<tr>
<td>gp03-07</td>
<td>1167</td>
<td>13955</td>
</tr>
<tr>
<td>gp03-08</td>
<td>1162</td>
<td>13945</td>
</tr>
<tr>
<td>gp03-09</td>
<td>1167</td>
<td>13995</td>
</tr>
<tr>
<td>gp03-10</td>
<td>1170</td>
<td>13855</td>
</tr>
<tr>
<td>gp04-01</td>
<td>1281</td>
<td>28097</td>
</tr>
<tr>
<td>gp04-02</td>
<td>1270</td>
<td>33928</td>
</tr>
<tr>
<td>gp04-03</td>
<td>1288</td>
<td>28182</td>
</tr>
<tr>
<td>gp04-04</td>
<td>1261</td>
<td>32925</td>
</tr>
<tr>
<td>gp04-05</td>
<td>1289</td>
<td>27927</td>
</tr>
<tr>
<td>gp04-06</td>
<td>1269</td>
<td>27383</td>
</tr>
<tr>
<td>gp04-07</td>
<td>1267</td>
<td>25955</td>
</tr>
<tr>
<td>gp04-08</td>
<td>1259</td>
<td>26516</td>
</tr>
<tr>
<td>gp04-09</td>
<td>1280</td>
<td>26737</td>
</tr>
<tr>
<td>gp04-10</td>
<td>1283</td>
<td>27736</td>
</tr>
<tr>
<td>gp05-01</td>
<td>1245</td>
<td>72727</td>
</tr>
<tr>
<td>gp05-02</td>
<td>1247</td>
<td>65993</td>
</tr>
<tr>
<td>gp05-03</td>
<td>1265</td>
<td>75457</td>
</tr>
<tr>
<td>gp05-04</td>
<td>1258</td>
<td>50497</td>
</tr>
<tr>
<td>gp05-05</td>
<td>1280</td>
<td>68151</td>
</tr>
<tr>
<td>gp05-06</td>
<td>1269</td>
<td>74131</td>
</tr>
<tr>
<td>gp05-07</td>
<td>1269</td>
<td>68801</td>
</tr>
<tr>
<td>gp05-08</td>
<td>1287</td>
<td>55489</td>
</tr>
<tr>
<td>gp05-09</td>
<td>1286</td>
<td>70587</td>
</tr>
<tr>
<td>gp05-10</td>
<td>1265</td>
<td>75457</td>
</tr>
<tr>
<td>gp06-01</td>
<td>1264</td>
<td>96410</td>
</tr>
<tr>
<td>gp06-02</td>
<td>1265</td>
<td>90659</td>
</tr>
<tr>
<td>gp06-03</td>
<td>1255</td>
<td>11155</td>
</tr>
<tr>
<td>gp06-04</td>
<td>1275</td>
<td>104957</td>
</tr>
<tr>
<td>gp06-05</td>
<td>1299</td>
<td>107806</td>
</tr>
<tr>
<td>gp06-06</td>
<td>1284</td>
<td>106400</td>
</tr>
<tr>
<td>gp06-07</td>
<td>1290</td>
<td>113726</td>
</tr>
<tr>
<td>gp06-08</td>
<td>1265</td>
<td>113726</td>
</tr>
<tr>
<td>gp06-09</td>
<td>1243</td>
<td>118943</td>
</tr>
<tr>
<td>gp06-10</td>
<td>1243</td>
<td>90559</td>
</tr>
</tbody>
</table>

Fig. 7. Results for benchmark instances provided by Guéret and Prins

(1996) 521–532


Instance | Optim. CPU | SAT | Variables Clause | Instance | Optim. CPU | SAT | Variables Clause
--- | --- | --- | --- | --- | --- | --- | ---
\(tai_4\) | 184 | 2 | 5944 | 31\,006 | \(tai_{10}\) | 637 | 98 | 94184 | 169888
\(tai_4\) | 236 | 1 | 4643 | 27426 | \(tai_{10}\) | 588 | 95 | 95343 | 1716326
\(tai_4\) | 271 | 2 | 5460 | 329925 | \(tai_{10}\) | 598 | 92 | 92303 | 1651992
\(tai_4\) | 250 | 2 | 5358 | 32341 | \(tai_{10}\) | 577 | 92 | 91314 | 1639647
\(tai_4\) | 295 | 2 | 6081 | 36418 | \(tai_{10}\) | 640 | 96 | 93978 | 1677177
\(tai_4\) | 189 | 2 | 4721 | 29194 | \(tai_{10}\) | 538 | 95 | 91151 | 1642608
\(tai_4\) | 201 | 2 | 4743 | 29188 | \(tai_{10}\) | 616 | 103 | 92285 | 1648788
\(tai_4\) | 217 | 2 | 5629 | 35110 | \(tai_{10}\) | 595 | 95 | 91094 | 1631685
\(tai_4\) | 261 | 2 | 5328 | 31517 | \(tai_{10}\) | 595 | 97 | 94528 | 1697235
\(tai_4\) | 217 | 2 | 5611 | 35444 | \(tai_{10}\) | 596 | 95 | 93135 | 1674220
\(tai_5\) | 300 | 6 | 11526 | 94098 | \(tai_{1x}\) | 937 | 523 | 309784 | 8563684
\(tai_5\) | 262 | 5 | 10110 | 82314 | \(tai_{1x}\) | 918 | 567 | 325397 | 9026993
\(tai_5\) | 323 | 6 | 11318 | 90297 | \(tai_{1x}\) | 871 | 543 | 315726 | 8767426
\(tai_5\) | 310 | 5 | 11047 | 88190 | \(tai_{1x}\) | 934 | 560 | 326511 | 9067128
\(tai_5\) | 326 | 6 | 10356 | 80906 | \(tai_{1x}\) | 946 | 541 | 321309 | 8940331
\(tai_5\) | 326 | 6 | 10356 | 80906 | \(tai_{1x}\) | 933 | 560 | 326512 | 9067124
\(tai_5\) | 310 | 6 | 10951 | 87906 | \(tai_{1x}\) | 891 | 566 | 322034 | 8943618
\(tai_5\) | 300 | 6 | 11009 | 88852 | \(tai_{1x}\) | 893 | 546 | 319320 | 8866998
\(tai_5\) | 355 | 6 | 11940 | 94884 | \(tai_{1x}\) | 899 | 568 | 324660 | 8998985
\(tai_5\) | 326 | 7 | 11344 | 90508 | \(tai_{1x}\) | 902 | 586 | 329605 | 9051491
\(tai_7\) | 453 | 21 | 30952 | 370295 | \(tai_{20}\) | 1155 | 3105 | 775142 | 29178719
\(tai_7\) | 443 | 24 | 31244 | 372853 | \(tai_{20}\) | 1241 | 3559 | 777061 | 29153596
\(tai_7\) | 468 | 30 | 31669 | 374258 | \(tai_{20}\) | 1257 | 2990 | 770228 | 28898909
\(tai_7\) | 463 | 20 | 31224 | 370305 | \(tai_{20}\) | 1248 | 3442 | 779059 | 29238508
\(tai_7\) | 416 | 22 | 30171 | 360661 | \(tai_{20}\) | 1256 | 3603 | 785066 | 29485803
\(tai_7\) | 451 | 45 | 30986 | 367026 | \(tai_{20}\) | 1204 | 2741 | 773849 | 29073596
\(tai_7\) | 442 | 33 | 32415 | 385956 | \(tai_{20}\) | 1294 | 2912 | 779814 | 2925385
\(tai_7\) | 424 | 20 | 30863 | 370287 | \(tai_{20}\) | 1169 | 2990 | 773336 | 29262619
\(tai_7\) | 458 | 21 | 31929 | 380761 | \(tai_{20}\) | 1289 | 3204 | 785835 | 29493666
\(tai_7\) | 398 | 20 | 29693 | 359194 | \(tai_{20}\) | 1241 | 3208 | 779045 | 28977588

Fig. 8. Results for benchmark instances provided by Taillard

<table>
<thead>
<tr>
<th>Instance</th>
<th>Optim. CPU</th>
<th>SAT Variance</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>j3-per0-1</td>
<td>1127</td>
<td>10880</td>
<td>42705</td>
</tr>
<tr>
<td>j3-per0-2</td>
<td>1084</td>
<td>20335</td>
<td>95123</td>
</tr>
<tr>
<td>j3-per10-0</td>
<td>1111</td>
<td>12675</td>
<td>53453</td>
</tr>
<tr>
<td>j3-per10-1</td>
<td>1069</td>
<td>15355</td>
<td>68062</td>
</tr>
<tr>
<td>j3-per10-2</td>
<td>1053</td>
<td>15355</td>
<td>70546</td>
</tr>
<tr>
<td>j4-per0-0</td>
<td>1055</td>
<td>22062</td>
<td>133215</td>
</tr>
<tr>
<td>j4-per0-1</td>
<td>1180</td>
<td>32160</td>
<td>209841</td>
</tr>
<tr>
<td>j4-per0-2</td>
<td>1071</td>
<td>26057</td>
<td>163530</td>
</tr>
<tr>
<td>j4-per10-0</td>
<td>1041</td>
<td>29457</td>
<td>190740</td>
</tr>
<tr>
<td>j4-per10-1</td>
<td>1019</td>
<td>22538</td>
<td>137589</td>
</tr>
<tr>
<td>j4-per10-2</td>
<td>1000</td>
<td>26057</td>
<td>164892</td>
</tr>
<tr>
<td>j4-per20-0</td>
<td>1000</td>
<td>28726</td>
<td>186429</td>
</tr>
<tr>
<td>j4-per20-1</td>
<td>1004</td>
<td>26074</td>
<td>165849</td>
</tr>
<tr>
<td>j4-per20-2</td>
<td>1012</td>
<td>26822</td>
<td>171587</td>
</tr>
<tr>
<td>j5-per0-0</td>
<td>1042</td>
<td>40825</td>
<td>335726</td>
</tr>
<tr>
<td>j5-per0-1</td>
<td>1054</td>
<td>58687</td>
<td>508163</td>
</tr>
<tr>
<td>j5-per0-2</td>
<td>1063</td>
<td>44127</td>
<td>367603</td>
</tr>
<tr>
<td>j5-per10-0</td>
<td>1005</td>
<td>67661</td>
<td>705462</td>
</tr>
<tr>
<td>j5-per10-1</td>
<td>1021</td>
<td>76467</td>
<td>808206</td>
</tr>
<tr>
<td>j5-per10-2</td>
<td>1012</td>
<td>77799</td>
<td>823964</td>
</tr>
<tr>
<td>j5-per20-0</td>
<td>1000</td>
<td>69400</td>
<td>727773</td>
</tr>
<tr>
<td>j5-per20-1</td>
<td>1000</td>
<td>69400</td>
<td>727773</td>
</tr>
<tr>
<td>j5-per20-2</td>
<td>1000</td>
<td>69400</td>
<td>727773</td>
</tr>
<tr>
<td>j6-per0-0</td>
<td>1056</td>
<td>85887</td>
<td>1053419</td>
</tr>
<tr>
<td>j6-per0-1</td>
<td>1045</td>
<td>85887</td>
<td>1053419</td>
</tr>
<tr>
<td>j6-per0-2</td>
<td>1063</td>
<td>85887</td>
<td>1053419</td>
</tr>
<tr>
<td>j6-per10-0</td>
<td>1005</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j6-per10-1</td>
<td>1021</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j6-per10-2</td>
<td>1012</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j6-per20-0</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j6-per20-1</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j6-per20-2</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per0-0</td>
<td>1055</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per0-1</td>
<td>1055</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per0-2</td>
<td>1055</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per10-0</td>
<td>1005</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per10-1</td>
<td>1005</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per10-2</td>
<td>1005</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per20-0</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per20-1</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j7-per20-2</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per0-1</td>
<td>1055</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per0-2</td>
<td>1055</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per10-0</td>
<td>1013</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per10-1</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per10-2</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per20-0</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per20-1</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
<tr>
<td>j8-per20-2</td>
<td>1000</td>
<td>109837</td>
<td>1380492</td>
</tr>
</tbody>
</table>

Fig. 9. Results for benchmark instances provided by Brucker et al.