A SAT-based Method for Solving the Two-dimensional Strip Packing Problem

Takehide Soh[1], Katsumi Inoue[12], Naoyuki Tamura[3], Mutsunori Banbara[3], and Hidetomo Nabeshima[4]

[1] Department of Informatics, Graduate University for Advanced Studies, soh@nii.ac.jp

[2] National Institute of Informatics, ki@nii.ac.jp

[3] Kobe University, {tamura, banbara}@kobe-u.ac.jp

[4] University of Yamanashi, nabesima@yamanashi.ac.jp

Abstract

We propose a satisfiability testing (SAT) based exact approach for solving the two-dimensional strip packing problem (2SPP). In this problem, we are given a set of rectangles and one large rectangle called a strip. The goal of the problem is to pack all rectangles without overlap, into the strip by minimizing the overall height of the packing. We show the method solves a 2SPP by translating it to SAT problems through a SAT encoding called order encoding. Generated SAT problems tend to be large, then we show techniques to reduce the search space by utilizing symmetries and relations of rectangles. To solve a 2SPP, that is, to compute the minimum height of a 2SPP, we need to repeatedly solve similar SAT problems. We then reuse learned clauses, assumptions, and models, which are obtained from previously solved SAT problems, to efficiently compute the minimum height. We attempt to solve 38 instances from the literature and obtain the total of 29 optimal solutions, including the solutions of two open problems.

1 Introduction

Packing problems have many practical applications such as truck loading, LSI layouts and assignments of newspaper articles. There has been a great deal of research on these problems, for example, knapsack problems and bin packing problems. In this paper, we consider a *satisfiability testing* (SAT) based exact approach for solving the *two-dimensional strip packing problem.* (2SPP) [2]. This problem is NP-hard in the strong sense because the one-dimensional bin packing problem which is strongly NP-hard can easily be transformed into a 2SPP [8].

The input of the 2SPP is a set $R = \{r_1, \ldots, r_n\}$ of n rectangles. Each rectangle has a width w_i and a height h_i. We are also given a large rectangle, called a *strip*, of width W. The goal is to pack all rectangles without overlap into the strip by minimizing the height H. Although rectangles are allowed

Proceedings of the 15th International RCRA workshop (RCRA 2008):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Udine, Italy, 12–13 December 2008

to be rotated by 90 degrees in the general case of the 2SPP, we assume that rectangles cannot be rotated according to convention of previous research [1, 4, 13, 19]. Furthermore, we assume that only integer values are allowed for $w_i, h_i, W, and\ H$.

The 2SPP has been well studied in the last decade. There are two types of methods to solve the 2SPP: the *exact method* and the *incomplete method.* The exact method can get the optimal solution of the problem. Martello *et al.* solve relatively small 38 instances and obtained 27 optimal solutions [13]. The incomplete method cannot prove the optimality of the solution, *i.e.*, the obtained minimum height H. The method can only confirm the solution as optimum provided that the solution corresponds to the lower bound of the problem. The method can pack over thousand rectangles with keeping quality of solutions [1, 4, 15, 19]. Although both types of methods have been well studied, it is difficult to reach the optimal height even so small problems that have up to 200 rectangles as inputs.

For the problem, we propose a SAT-based method for solving the 2SPP. Recent advance of SAT technologies has been tremendous. Many *SAT solvers* have been developed and solve SAT problems. Most state-of-the-art solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [5] and use a *clause learning* technique [12]. With this technique, SAT solvers generate learned clauses when a conflict is reached and avoid encountering the same conflict in the later search. These techniques make SAT solvers applicable to huge problems which have over millions of clauses. The main advantage of using SAT solvers is that it make possible to use several SAT techniques.

In order to solve the 2SPP with a SAT solver, we represent the problem as a *constraint satisfaction problem* (CSP) and solve the CSP as a SAT problem through a SAT encoding called *order encoding* [16]. The feature of the encoding is that a CSP comparison $x \leq a$ is encoded into a Boolean variable and it enable a compact encoding for the 2SPP. However, generated SAT problems have to grow with the number of input rectangles. Then we propose techniques to reduce the search space. These techniques utilize symmetries and relations of rectangles. To reach the optimal solution, we have to repeatedly solve similar sub-problems. For solving these problems efficiently, we reuse learned clauses generated from previously solved problem. We also reuse assumptions and models which are obtained from previously solved sub-problems. In computational experiments, we evaluate techniques to reduce the search space and attempted to solve 38 instances from the literature [13].

The remainder of the paper is organized as follows. Section 2 provides a necessarily brief perspective on the 2SPP and related concepts. Section 3 describes how to encode the 2SPP into SAT problems. Section 4 explains how to solve the optimal height of a 2SPP and several techniques to solve problems efficiently. Section 5 shows computational results. Section 6 discusses related works. Finally, Section 7 concludes the paper.

2 Preliminaries

In this section, we give preliminaries to take a SAT-based approach to the 2SPP. In the following, $\mathbb{N}$ denotes the set of natural numbers and $\mathbb{Z}$ denotes the set of integers.

2.1 2SPP and 2OPP

We consider a SAT-based approach to the *two-dimensional strip packing problem* (2SPP). Although we want to obtain the optimal height of a 2SPP, SAT solvers can only determine the satisfiability of a given problem. We thus approach to the 2SPP by solving a sequence of *two-dimensional orthogonal packing problems* (2OPPs), which are decision problems of the 2SPP with a fixed height of the strip. We give more details of this method in Section 4.1. Here, we define the 2SPP and the 2OPP as follows [2].

Two-dimensional strip packing problem (2SPP)

Input. A set $R = \{r_1, \ldots, r_n\}$ of n rectangles. Each rectangle $r_i \in R$ has a width w_i and a height h_i ($w_i, h_i \in \mathbb{N}$). A *Strip* of width $W \in \mathbb{N}$.

Constraints. Each rectangle cannot overlap with the others and the edges of the strip and parallel to the horizontal and the vertical axis.

Question. What is the minimum height such that the set of rectangles can be packed in the given strip?

Two-dimensional orthogonal packing problem (2OPP)

Input. A set $R = \{r_1, \ldots, r_n\}$ of n rectangles. Each rectangle $r_i \in R$ has width w_i and height h_i ($w_i, h_i \in \mathbb{N}$). A *Strip* of width W and height H ($W, H \in \mathbb{N}$).

Constraints. Each rectangle cannot overlap with the others and the edges of the strip and parallel to the horizontal and the vertical axis.

Question. Can the set of rectangles be packed in the given strip?

2.2 CSP formulation of 2OPP

We now represent the 2OPP as a *constraint satisfaction problem* (CSP). A CSP is a triple $\langle V, D, C\rangle$, where V is a finite subset of integer variables, D is a function which maps every variable in V to a subset of $\mathbb{Z}$. We use $D(x)$ as the subset of $\mathbb{Z}$ mapped from $x \in V$ and call the set $D(x)$ the domain of x. C is a finite set of constraints over a subset of variables in V.

The CSP formulation of the 2OPP is as follows. Let x_i and y_i be integer variables such that the pair (x_i, y_i) of variables represents the position of lower left coordinates of the rectangle r_i in the strip. The domains of x_i and y_i are as follows.

$$\begin{array}{c} D(x_i) = \{a \in \mathbb{N} \mid 0 \le a \le W - w_i\} \\ D(y_i) = \{a \in \mathbb{N} \mid 0 \le a \le H - h_i\} \end{array} \tag{1}$$

These domains represent possible values of the coordinates of the rectangle r_i and guarantee that r_i must not overlap with edges of the strip. For each pair of rectangles r_i and r_j $(1 \le i < j \le n)$, we associate the *non-overlapping constraints.*

$$(x_i + w_i \le x_j) \lor (x_j + w_j \le x_i) \lor (y_i + h_i \le y_j) \lor (y_j + h_j \le y_i) \tag{2}$$

2.3 Order Encoding

There have been several studies on translation methods which encode a CSP into a SAT problem, *e.g.*, direct encoding, log encoding [18], support encoding [10] and log support encoding [9]. Among them, *order encoding* [16] aims to make a more natural explanation of the order relation of integers. In order encoding, there are two encoding steps. Let x be an integer variable, and c be an integer value. At the first step, a constraint with comparison is translated into *primitive comparisons* which are in the form of $x \le c$. At the next step, a primitive comparison is encoded into a Boolean variable $p_{x,c}$. Due to space limitation, we illustrate the encoding method with a simple constraint $x_1 + 1 \le x_2$ $(x_1, x_2 \in \{0, 1, 2, 3\})$. This constraint is encoded into the set of primitive comparisons as follows:

$$\neg(x_2 \le 0), \quad (x_1 \le 0) \lor \neg(x_2 \le 1), \quad (x_1 \le 1) \lor \neg(x_2 \le 2), \quad (x_1 \le 2)$$

Then, these constraints are translated into the following formula of a SAT problem:

$$\neg px_{2,0}, \quad px_{1,0} \lor \neg px_{2,1}, \quad px_{1,1} \lor \neg px_{2,2}, \quad px_{1,2}$$

where $px_{i,c}$ denotes $x_i \le c$ for a simple expression. In order encoding, the following axiom clauses are also added:

$$\neg px_{1,0} \lor px_{1,1}, \quad \neg px_{1,1} \lor px_{1,2}, \quad \neg px_{2,0} \lor px_{2,1}, \quad \neg px_{2,1} \lor px_{2,2}$$

3 From 2OPP into SAT Problems

In this section, we explain how to translate a 2OPP into a SAT problem with order encoding. Let $r_i, r_j \in R$ $(i \ne j)$ be two rectangles in a 2OPP. Let e and f be any integer. Then, the SAT encoding of a 2OPP uses four

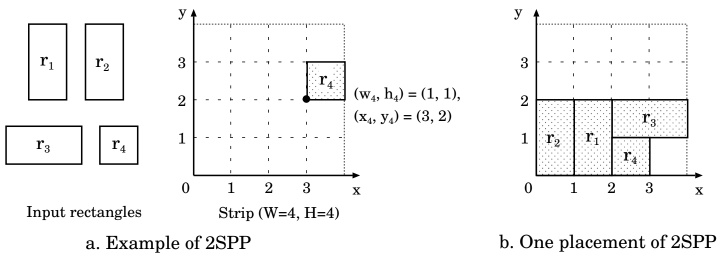


Figure 1: Example of a 2SPP

kinds of atoms, $lr_{i,j}$, $ud_{i,j}$, $px_{i,e}$, and $py_{i,f}$. $lr_{i,j}$ is *true* if r_i are placed at the left to the r_j. $ud_{i,j}$ is *true* if r_i are placed at the downward to the r_j. $px_{i,e}$ is *true* if r_i are placed at less than or equal to e. $py_{i,f}$ is *true* if r_i are placed at less than or equal to f. Then, inputs and constraints of a 2OPP can be encoded into a SAT problem as follows.

For each rectangle r_i, and integer e and f such that $0 \leq e < W - w_i$ and $0 \leq f < H - h_i$, we have the 2-literal axiom clauses due to order encoding,

$$\begin{gathered} \neg px_{i,e} \vee px_{i,e+1} \\ \neg py_{i,f} \vee py_{i,f+1} \end{gathered} \tag{3}$$

For each rectangles r_i, r_j $(i < j)$, we have the following 4-literal clauses as the non-overlapping constraints (2):

$$lr_{i,j} \vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i} \tag{4}$$

For each rectangles r_i, r_j $(i < j)$, and integer e and f such that $0 \leq e < W - w_i$ and $0 \leq f < H - h_j$, we also have the following 3-literal clauses as the non-overlapping constraints (2):

$$\begin{gathered} \neg lr_{i,j} \vee px_{i,e} \vee \neg px_{j,e+w_i} \\ \neg lr_{j,i} \vee px_{j,e} \vee \neg px_{i,e+w_j} \\ \neg ud_{i,j} \vee py_{i,f} \vee \neg py_{j,f+h_i} \\ \neg ud_{j,i} \vee py_{j,f} \vee \neg py_{i,f+h_j} \end{gathered} \tag{5}$$

Example. Consider the simple example of 2OPP shown in Figure 1a. We are given four rectangles $(w_1, h_1) = (1, 2), (w_2, h_2) = (1, 2), (w_3, h_3) = (2, 1), (w_4, h_4) = (1, 1)$ and a strip $(W, H) = (4, 4)$. We obtain the SAT-encoded 2OPP shown in Figure 2. This SAT problem is satisfiable and the figure of packed rectangles corresponding to a model is shown in Fig. 1b. In this case, Boolean variables of the SAT problem are assigned as follows.

$$px_{1,0} = F,\ px_{1,1} = T,\ px_{2,0} = T,\ px_{3,1} = F,\ px_{3,2} = T,\ px_{4,1} = F,\ px_{4,2} = T$$
$$py_{1,0} = T,\ py_{2,0} = T,\ py_{3,0} = F,\ py_{3,1} = T,\ py_{4,0} = T$$

Variables

$px_{1,0},\dots,px_{1,3}\quad py_{1,0},\dots,py_{1,3}\quad px_{2,0},\dots,px_{2,2}\quad py_{2,0},\dots,py_{2,3}$

$px_{3,0},\dots,px_{3,3}\quad py_{3,0},\dots,py_{3,2}\quad px_{4,0},\dots,px_{4,3}\quad py_{4,0},py_{4,1}$

Order Constraint (3)

$\neg px_{1,0} \vee px_{1,1}, \neg px_{1,1} \vee px_{1,2}, \neg px_{1,2} \vee px_{1,3}$

$\vdots$

$\neg py_{4,0} \vee py_{4,1}, \neg py_{4,1} \vee py_{4,2}, \neg py_{4,2} \vee py_{4,3}$

Non-overlapping Constraint (4), (5)

$lr_{1,2} \vee lr_{2,1} \vee ud_{1,2} \vee ud_{2,1}$

$\vdots$

$lr_{3,4} \vee lr_{4,3} \vee ud_{3,4} \vee ud_{4,3}$

$\neg lr_{1,2} \vee \neg px_{2,0}\quad \neg lr_{1,2} \vee px_{1,0} \vee \neg px_{2,1}\quad \neg lr_{1,2} \vee px_{1,1} \vee \neg px_{2,2}\quad \neg lr_{1,2} \vee px_{1,2}$

$\vdots$

$\neg ud_{3,4} \vee \neg py_{3,0}\quad \neg ud_{3,4} \vee py_{4,0} \vee \neg py_{3,1}\quad \neg ud_{3,4} \vee py_{4,1} \vee \neg py_{3,2}\quad \neg ud_{3,4} \vee py_{4,2}$

Figure 2: Example of SAT-encoded 2OPP

These assignments are converted into the following assignments of the 2OPP:

$$x_1 = 1,\quad x_2 = 0,\quad x_3 = 2,\quad x_4 = 2$$
$$y_1 = 0,\quad y_2 = 0,\quad y_3 = 1,\quad y_4 = 0$$

4 Solving 2SPP with a SAT Solver

In Section 3, we explained how to translate a 2OPP into a SAT problem. In this section, we show how to compute the optimal height of a 2SPP with a SAT solver by repeatedly solving SAT-encoded 2OPP problems.

4.1 Searching Optimum Height of 2SPP

Let ub and lb be *upper* and *lower bounds* of a solution of a 2SPP, respectively. In practice, the lower bound is given by exact methods and the upper bound is given by either exact or incomplete methods. Let o be an integer value such that $lb \le o \le ub-1$. We introduce a new Boolean variable ph_o which is *true* if all rectangles are packed at the downward to the height o. Then, to solve a 2SPP, for each rectangle r_i, and a height o such that $lb \le o \le ub-1$, we have the 2-literal clauses:

$$\neg ph_o \vee py_{i,o-h_i} \tag{6}$$

Furthermore, for each o ($lb \le o \le ub-1$), we have the 2-literal clauses due to order encoding:

$$\neg ph_o \vee ph_{o+1} \tag{7}$$

Let Ψ be the set of clauses consisting of all clauses obtained from (3), (4), (5), (6) and (7). Then, we can decide the satisfiability of a 2OPP with the height H by solving the set of clauses[1]:

$$\Psi \cup \{ph_H\} \tag{8}$$

Note that Ψ is common in all 2OPPs associated with a 2SPP.

The optimal height of a 2SPP can be obtained by repeatedly solving SAT-encoded 2OPPs. The *bisection method*, as used in previous studies [11, 14], is useful for efficiently finding the optimal height. First, we set the lower and upper bounds. Then we solve the height which is at the half of the region. If we obtain the satisfiability of the 2OPP, then upper bound updated by the height, or if we obtain the unsatisfiability of the 2OPP, then lower bound updated by the height. We execute these operations repeatedly until the optimal height, which is the boundary between the satisfiable and unsatisfiable problems, is obtained. For example, let us consider the 2SPP which has the optimal height 140. To compute the optimal height, suppose that the lower bound is 48 and that the upper bound is 393. Then the height would change as: 48 (`UNSAT`), 393 (`SAT`), 221 (`SAT`), 134 (`UNSAT`), 178 (`SAT`), 157 (`SAT`),.... The solution area becomes more and more constrained until the optimal height is obtained.

4.2 Reducing Techniques

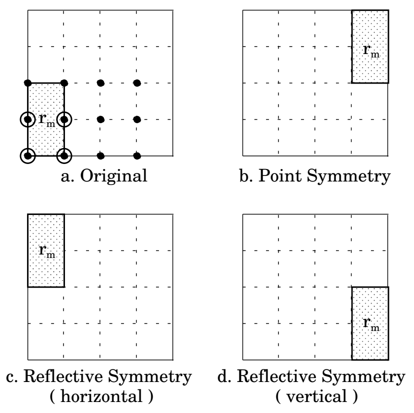
Now, we propose techniques which enhance the SAT-based approach. Supposing that $W = H$, the size of clauses of a SAT-encoded 2OPP are $O(n^2)$, where n is the number of rectangles. Generated SAT problems have to grow with the number of input rectangles. We thus propose four techniques to reduce the search space. We give evaluation of these techniques in Section 5.

Domain reduction. To prune the search space, we reduce the domain of the maximum rectangle defined by w_m and h_m with symmetry (see Figure 3). There are three cases wherein maximum means maximum width, maximum height, or maximum area. We choose the maximum width and obtain a new domain for the maximum rectangle in the horizontal direction:

$$D(x_m) = \{a \in \mathbb{N} \mid 0 \leq a \leq \lfloor \frac{W - w_m}{2} \rfloor\}$$

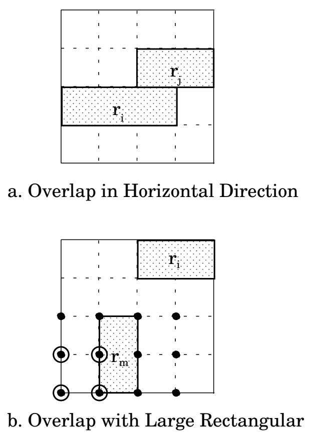
Figure 3. shows the original $D(r_m)$ with dots and reduced one with circles, that is, the domain $D(x_m)$ becomes from $\{0, 1, 2, 3\}$ to $\{0, 1\}$.

[1]That is, we use ph_H to restrict the upper bound of y-coordinate of each rectangle instead of restricting the domain of each rectangle that was defined in Section 2.2.



r_m
a. Original
r_m
b. Point Symmetry
r_m
c. Reflective Symmetry (horizontal)
r_m
d. Reflective Symmetry (vertical)

Figure 3: Domain Removal



r_j
r_i
a. Overlap in Horizontal Direction
r_i
r_m
b. Overlap with Large Rectangular

Figure 4: Overlap Condition

By applying this reduction, if w_i satisfies $w_i > \lfloor \frac{W - w_m}{2} \rfloor$, we can assign $lr_{i,m} = false$, that is, we can remove literals $lr_{i,m}$ and the clauses including $\neg lr_{i,m}$ from the SAT problem. In Figure 4b, we can see that the r_i, which has $w_i = 2$, cannot be packed to the left to r_m. This reduction is also available in the vertical direction.

Large Rectangles. If we are given large rectangles r_i and r_j which satisfy $w_i + w_j > W$, we can assign $lr_{i,j} = false$ and $lr_{j,i} = false$ by using a size relation of the pair of rectangles. Thereby, we can remove literals $lr_{i,j}, lr_{j,i}$ and the clauses including either $\neg lr_{i,j}$ or $\neg lr_{j,i}$ from the SAT problem. The condition $w_i + w_j > W$ means we cannot pack rectangles r_i and r_j in the horizontal direction (see Fig. 4a). This reduction technique is also available in the vertical direction.

Same Rectangles. If we are given rectangles r_i and r_j which have the same dimension $(w_i, h_i) = (w_j, h_j)$, we can fix the positional relation of rectangles. Thereby, we can assign $lr_{j,i} = false$ and add $lr_{i,j} \vee \neg ud_{j,i}$.

One Pair of Rectangles. We can fix the positional relation between only one pair of rectangles. See Figure 4, by using symmetry, we can restrict the positional relation between r_i and r_j. Hereby, we can assign $lr_{j,i} = false$ and $ud_{j,i} = false$. Note that, this technique cannot use with *domain reduction* simultaneously.

4.3 Reusing Clauses for Incremental Solving

To solve the 2SPP, we use Minisat [6] which is the one of the state-of-the-art solvers. Minisat is based on the DPLL algorithm. On the basis of DPLL, Minisat efficiently implements *conflict learning* [12]. When the current assignment leads to a conflict, a new clause indicating the incompatible assignment is generated as a *learned clauses*. For example, when $(x_1, x_2, x_3) = (true, true, false)$ is the source of a conflict, the clause $\neg(x_1 \wedge x_2 \wedge \neg x_3) = \neg x_1 \vee \neg x_2 \vee x_3$ is generated. Such a learned clause is utilized to prevent the solver from retrying the same portion of assignments. Note that all learned clauses can be deduced from the initial set of clauses.

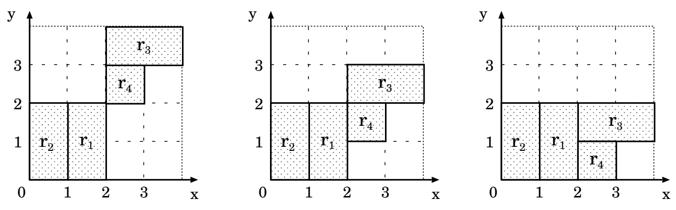
Our SAT encoding approach generates a sequence of SAT problems. These SAT problems are similar to each other, that is, P_{o+1} includes P_o except for some clauses. This kind of problem is called an *incremental SAT problem* [7]. Now we propose methods to reuse learned clauses, assumptions, and models for solving incremental SAT problems efficiently.

Learning Clauses. Nabeshima *et.al.* [14] show the effectiveness of reusing learned clauses for solving a job-shop scheduling problem. Eén and Sörensson also report a similar technique in solving incremental SAT [7]. To solve an incremental SAT problem efficiently, we here use their technique to reuse learned clauses for solving the 2SPP as follows. Let P and Q be SAT problems such that "all non-unit clauses of P are included in Q" (*lemma-reusability condition*). Then by the lemma-reusability theorem by [14], the set S of all learned clauses generated in solving P can be used to solve Q. That is, instead of solving Q, we can solve $Q \cup S$, which results in pruning a large portion of the search space in many cases. In solving a 2SPP, two 2OPPs in the form (8) differ only in their unit clauses ph_H, which satisfies the lemma-reusability condition above. By this way, we can reuse learned clauses produced in solving previous 2OPPs in subsequent 2OPPs.

Reusing Assumptions. In Section 4.1, we show how to decide the satisfiability at any height of 2SPP, and we need to add unit clause ph_H to the problem. If $\Psi \cup \{ph_H\}$ is unsatisfied, we cannot continue the bisection method without removing ph_H from the problem.

To resolve this issue, Eén and Sörensson proposed a particular set of a unit clauses called *assumptions* [7]. An assumption is added before solving the problem, and then removed from the problem. Adding ph_H as an assumption, we can do bisection method until the optimal height is obtained.

To solve incremental SAT efficiently, we propose to reuse assumptions. Generally, we solve the problems which are the conjunction of Ψ and at most one ph_H. Using our method, we can reuse assumptions to next subproblems in bisection method. For example, let Ψ be an encoded 2SPP with $lb = 4, ub = 10$ and the optimal height 6. First we give Ψ and $\{ph_7\}$ as an



a. H=4 b. H=3 c. H=2

Figure 5: Reusing Models

assumption to the solver and it returns `SAT`. Now we reuse assumptions by adding $\{ph_7\}$ to Ψ. Next the SAT solver is given $\Psi \cup \{ph_7\}$ and $\{ph_5\}$ as an assumption, and returns `UNSAT`. Finally, the SAT solver is given $\Psi \cup \{ph_7\} \cup \{\neg ph_5\}$ and $\{ph_6\}$ as an assumption, and returns `SAT`. Using this way, we avoid redundant search space.

Reusing Models. When a SAT-encoded 2OPP is satisfiable, a SAT solver outputs a model which represents a placement of all rectangles within a given height H. In solving a 2SPP, a part of the previously obtained model might correspond to a model of next sub-problem. Let us consider a simple example of a 2SPP which has the optimal height $H = 2$. Suppose that we found a model of the 2OPP with $H = 4$ as that shown in Figure 5a. This model partially corresponds to a model with $H = 3$ because the two models share the truth assignment that $lr_{2,1} = true$, $lr_{1,4} = true$, $ud_{4,3} = true$.

We thus propose to reuse a partial model to solve the next sub-problem efficiently. A SAT solver assigns either $true$ or $false$ to a variable chosen by some heuristic if there is no unit clause, for example, Minisat [6] tries to assign $false$ to such a decision variable preferentially. On the other hand, our proposed method assigns a decision variable the value which is found in a model of a previous satisfiable problem. By this reusing method, we can help the decision making of a SAT solver.

5 Experimental Results

The presented methods are run on a Xeon 2.6GHz with 2GB of memory within 3600 seconds and we use Minisat 2.0 [6] as a SAT solver. We use the benchmark set and those lower bounds shown in the literature by Martello *et al.* [13]: HT01-HT09, BENG01-BENG10, CGCUT01-CGCUT03, GCUT01-GCUT04, and NGCUT01-NGCUT12. All 38 instances are available at "DEIS - Operations Research Group Library of Instances" [17]. These benchmark sets include some problems which are very hard to solve. In

Table 1: Number of Solved Optimal Values with Reducing Techniques

Instance	normal	*domain*	*large*	*same*	*pair*
HT01-09	7	9	7	8	8
CGCUT01-03	1	1	1	1	1
GCUT01-04	2	2	2	2	2
NGCUT01-12	12	12	12	12	12
BENG01-10	2	2	2	4	2
Total # of Opt.	24	26	24	27	25

Table 2: Comparison with previous studies

Instance				Proposed method			Optimum or best known value		
Name	n	W	LB	normal	C1	C2	Incomp. [1, 15, 19]	Exact. [13]	Prop.
HT01(c1p1)	16	20	20	20	20	20	op [1, 15]	op	op
HT02(c1p2)	17	20	20	20	20	20	op [1, 15]	op	op
HT03(c1p3)	16	20	20	20	20	20	op [1, 15]	op	op
HT04(c2p1)	25	40	15	15	15	15	op [1, 15]	op	op
HT05(c2p2)	25	40	15	15	15	15	op [1, 15]	op	op
HT06(c2p3)	25	40	15	15	15	15	op [1, 15]	op	op
HT07(c3p1)	28	60	30	31	30	30	op [1]	31	op
HT08(c3p2)	29	60	30	31	31	30	31 [1, 15]	31	**op**
HT09(c3p3)	28	60	30	30	30	30	op [1, 15]	op	op
CGCUT01	16	10	23	23	23	23	op [1, 15, 19]	op	op
CGCUT02	23	70	63	67	67	65	65 [1, 15, 19]	67	65
CGCUT03	62	70	636	671	671	671	658 [15]	670	671
GCUT01	10	250	1016	1016	1016	1016	op [1, 15, 19]	op	op
GCUT02	20	250	1133	1196	1196	1196	1187 [15]	1208	1196
GCUT03	30	250	1803	1803	1803	1803	op [1, 15, 19]	op	op
GCUT04	50	250	2934	3056	3056	3056	3002 [1]	3077	3056
NGCUT01	10	10	23	23	23	23	op [1, 15, 19]	op	op
NGCUT02	17	10	30	30	30	30	op [1, 15, 19]	op	op
NGCUT03	21	10	28	28	28	28	op [1, 15, 19]	op	op
NGCUT04	7	10	20	20	20	20	op [1, 15, 19]	op	op
NGCUT05	14	10	36	36	36	36	op [1, 15, 19]	op	op
NGCUT06	15	10	31	31	31	31	op [1, 15, 19]	op	op
NGCUT07	8	20	20	20	20	20	op [1, 15, 19]	op	op
NGCUT08	13	20	33	33	33	33	op [1, 15]	op	op
NGCUT09	18	20	49	50	50	50	50 [1, 15, 19]	50	**op**
NGCUT10	13	30	80	80	80	80	op [1, 15, 19]	op	op
NGCUT11	15	30	50	52	52	52	op [1, 15]	op	op
NGCUT12	22	30	79	87	87	87	op [1, 15, 19]	op	op
BENG01	20	25	30	30	30	30	op [1, 15, 19]	op	op
BENG02	40	25	57	58	57	57	op [1, 15]	58	op
BENG03	60	25	84	85	84	85	op [1, 15, 19]	85	op
BENG04	80	25	107	108	108	108	op [1, 15, 19]	108	108
BENG05	100	25	134	135	134	134	op [1, 15, 19]	op	op
BENG06	40	40	36	36	36	36	op [1, 15, 19]	37	op
BENG07	80	40	67	68	68	68	op [1, 15, 19]	op	68
BENG08	120	40	101	102	102	102	op [1, 15, 19]	op	102
BENG09	160	40	126	130	130	130	op [1, 15, 19]	op	130
BENG10	200	40	156	158	158	158	op [1, 15, 19]	op	158
Total # of Opt.				24	28	28	32	27	29

particular, HT08, CGCUT02, 03, GCUT02, 04, NGCUT09 are still open.

We evaluate the reducing techniques shown in Section 4.2. Table 1 shows the number of solved optimal values with each reducing technique. *normal* represents the method without reducing techniques. *domain* represents a domain reduction technique. *large* represents a reduction technique with large rectangles. *same* represents a reduction technique with same rectangles. *pair* represents a reduction technique with the one pair of rectangles. Table 1 shows that *same* computes the largest number of optimal values. Furthermore, all techniques compute the instance group "NGCUT" completely. This means that all these techniques can close open problem NGCUT09.

To compute problems more efficiently, we combine reducing and reusing techniques. Table 2 shows a comparison with our combined methods with previous methods. Due to space limitation, we show top two of 36 combinations that we tried. Columns 1–4 show the characteristics of each instance such as the instance name, the number of input rectangles, the strip width W, and lower bounds from the literature [13]. Columns 5–7 show the best value obtained by the normal method and the top two combined methods. "C1" denotes results of the following combination: reducing with *domain*, *large* and *same*, reusing *learned clauses* and *assumptions*. "C2" denotes results of the following combination: reducing with *large* and *pair*, reusing *learned clauses* and *assumptions*. Columns 8–10 show a comparison with previous methods. "Incomp." describes the best value by incomplete methods from the literature [1, 15, 19]. All of those methods are reported recently in 2008. "Exact." describes the best value by Martello *et al.* [13]. "Prop." describes the best value by our methods. "op" denotes that the method obtain the optimal value.

Table 1, 2 show effectiveness of our proposed methods. Our SAT-based method with no reducing and no reducing techniques, can solve 24 optimums including the one open problem. Moreover, the combined methods can solve 28 optimums and close the open problem HT08. Table 2 also shows that our exact methods are competitive with the state-of-the-art incomplete methods. As a result, our methods compute a optimal solution of the instance HT08 (see Figure 7) and proves the optimum of NGCUT09 to be 50. In other words, we prove that the height of 49 has no solution in NGCUT09.

6 Discussion

There are a few report of methods which solve an optimization problem through encoding which translates a problem to decision problems. Bekrar *et al.* [3] reported an approach to the *two-dimensional guillotine strip packing problem* which is a variant of the 2SPP. They represent the problem as CSPs and solve the optimum with bisection method. The difference from our work is that they directly solve CSPs. On the other hand, we propose

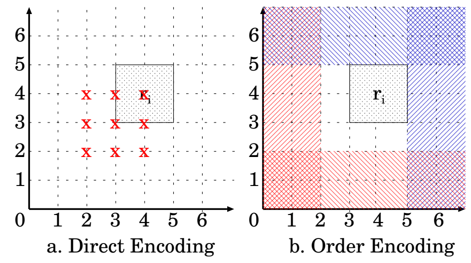


Figure 6: Representation of Conflicts

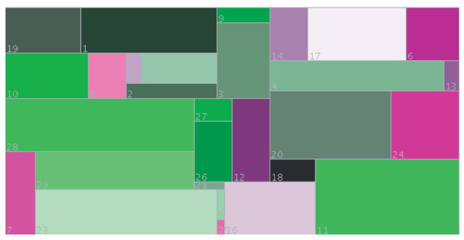


Figure 7: New Solution of HT08

the SAT-based method for the 2SPP. Thereby we can apply several incremental SAT techniques and use the state-of-the-art SAT solver that have been widely studied. There are several works on a SAT-based method for other optimization problems. Inoue *et al.* [11] propose “Multisat” which can execute several SAT solvers in parallel and apply Multisat to *SAT planning* and the *job-shop scheduling problem*. The solvers included in Multisat exchange lemmas derived by conflict analysis among different SAT solvers for solving the problems efficiently. Nabeshima *et al.* [14] report another approach to the problems. Their method shares a learned clauses between sub-problems. Our SAT-based approach enhances these works with reusing assumptions and models, along with reusing learned clauses and is newly applied to the 2SPP.

In our method, we use the order encoding as a SAT encoding. However, there have been well studied about SAT encoding. Here, we consider a difference between direct encoding and order encoding. Let $(w_i, h_i) = (2, 2), (w_j, h_j) = (2, 2)$ and place r_i at $(x_i, y_i) = (3, 3)$ (see Figure 6). We can represent overlap constraint of CSP between r_i and r_j as follows:

$$(x_j \leq 1) \lor (x_j \geq 5) \lor (y_j \leq 1) \lor (y_j \geq 5)$$

To see difference between order encoding and the others, we compare the SAT clauses encoded with direct encoding [18] and those with order encoding. In the direct encoding, we assign to a SAT variable as $p_{xa} = true$ if and only if the CSP variable x has the domain value a, and constraints are encoded to conflict clauses. The encoded clauses are as follows:

$$\begin{array}{rlll}
CSP: & (x_j \leq 1) \lor (x_j \geq 5) \lor (y_j \leq 1) \lor (y_j \geq 5) & & \\
SAT(direct): & \neg p_{x_j 2} \lor \neg p_{y_j 2} & \neg p_{x_j 2} \lor \neg p_{y_j 3} & \neg p_{x_j 2} \lor \neg p_{y_j 4} \\
 & \neg p_{x_j 3} \lor \neg p_{y_j 2} & \neg p_{x_j 3} \lor \neg p_{y_j 3} & \neg p_{x_j 3} \lor \neg p_{y_j 4} \\
 & \neg p_{x_j 4} \lor \neg p_{y_j 2} & \neg p_{x_j 4} \lor \neg p_{y_j 3} & \neg p_{x_j 4} \lor \neg p_{y_j 4} \\
SAT(order): & p_{x_j 1} \lor \neg p_{x_j 4} \lor p_{y_j 1} \lor \neg p_{y_j 4} & &
\end{array}$$

In direct encoding, constraints are represented as conflict points (see Figure 6a). On the other hand, order encoding represents constraints as a conflict region (see Figure 6b). This indicates SAT-based approach with order encoding is suitable not only 2SPP but also geometric problems such as shop scheduling problem.

7 Conclusion

We presented a SAT-based exact method to solve the two-dimensional strip packing problem. Our method solves the problem through order encoding and the bisection method. As far as the authors know, this is the first article that solves the 2SPP with a SAT solver. Our approach solved two open problems in 2SPP. This indicates that, while SAT-based approaches have been widely studied, there is still remaining challenging problems, especially OR problems like 2SPP.

There are several important future topics. Comparing other SAT-encoding methods is important to evaluate the effect of the order encoding. It is also interesting to compare this method with CSP solvers. Consideration of rotation of input rectangles and applying to other packing problems are worthwhile. There is a possibility that a hybrid system which includes incomplete methods as well as exact methods to solve larger problems.

Acknowledgments

This work was supported in part by Grant-in-Aid for Scientific Research No.20240003 in Japan Society for the Promotion of Science and the Joint Research Project funded by National Institute of Informatics in Japan. We would like to thank Ateet Bhalla and Koji Iwanuma for their kind suggestion and comments.

References

[1] R. Alvarez-Valdés, F. Parreno, and J. M. Tamarit. Reactive GRASP for the strip-packing problem. *Computers and Operations Research*, 35(4):1065–1083, 2008.

[2] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal of Computing*, 9(4):846–855, 1980.

[3] A. Bekrar, I. Kacem, C. Chu, and C. Sadfi. A dichotomical algorithm for solving the 2D guillotine strip packing problem. In *Proceedings of CIE'07*, pages 1216–1224, 2007.

[4] E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[6] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, volume 2919 of *LNCS*, pages 502–518, 2003.

[7] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[8] M. R. Garey and D. S. Johnson, editors. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* FREEMAN, 1979.

[9] M. Gavanelli. The log-support encoding of CSP into SAT. In *Proceedings of CP'07*, volume 4741 of *LNCS*, pages 815–822. 2007.

[10] I. P. Gent. Arc consistency in SAT. In *Proceedings of ECAI'02*, pages 121–125, 2002.

[11] K. Inoue, T. Soh, S. Ueda, Y. Sasaura, M. Banbara, and N. Tamura. A competitive and cooperative approach to propositional satisfiability. *Discrete Applied Mathematics*, 154(16):2291–2306, 2006.

[12] J. P. Marques-Silva and K. A. Sakallah. GRASP—A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[13] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *Journal on Computing*, 15(3):310–319, 2003.

[14] H. Nabeshima, T. Soh, K. Inoue, and K. Iwanuma. Lemma reusing for sat based planning and scheduling. In *Proceedings of ICAPS'06*, pages 103–112, June 2006.

[15] B. Neveu and G. Trombettoni. Strip packing based on local search and a randomized Best-Fit. In *Workshop on BPPC'08*, May 2008.

[16] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. In *Proceedings of CP'06*, volume 4204 of *LNCS*, pages 590–603. 2006.

[17] `http://www.or.deis.unibo.it/research_pages/ORinstances.htm`.

[18] T. Walsh. SAT v CSP. In *Proceedings of CP'02*, volume 1894 of *LNCS*, pages 441–456. 2000.

[19] L. Wei, D. Zhang, and Q. Chen. A least wasted first heuristic algorithm for the rectangular packing problem. *Computers and Operations Research*, 2008.