Improved Local Search Algorithms for the Rectangle Packing Problem with General Spatial Costs

S. Imahori * M. Yagiura T. Ibaraki

Department of Applied Mathematics and Physics, Graduate School of Informatics,
Kyoto University, Kyoto 606-8501, Japan.
E-mail: {imahori,yagiura,ibaraki}@amp.i.kyoto-u.ac.jp
Phone/Fax: +81-75-753-5494.
* corresponding author

Abstract: The rectangle packing problem with general spatial costs is to pack given rectangles without overlap in the plane so that the maximum cost of the rectangles is minimized. This problem is very general, and various types of packing problems and scheduling problems can be formulated in this form. For this problem, we have previously presented local search algorithms using a pair of permutations of rectangles to represent a solution. In this paper, we propose speed-up techniques to evaluate solutions in various neighborhoods. Computational results for the rectangle packing problem and a real-world scheduling problem exhibit good prospects of the proposed techniques.

Keywords: packing, sequence pair, general spatial cost, dynamic programming, local search.

1 Introduction

The two-dimensional rectangle packing problem (2RP) is to place a given set of rectangles in the plane without overlap so that the cost function is minimized. This is one of the representative combinatorial optimization problems and is known to be NP-hard. There are many industrial applications; e.g., wood, glass and cloth industries, newspapers paging, LSI designing and so on. Therefore many heuristic algorithms for this problem have been proposed [1, 5, 10, 12, 14, 15].

There are many options on packing rule and objective function; e.g., rotations are allowed (90°/free) or not, a bin of a fixed size to which rectangles are packed is given or not, constraints on the items' placement (such as the "guillotine cuts") are given or not. Many kinds of algorithms

have been proposed for each combination of these options.

In this paper, we consider the rectangle packing problem with general spatial costs (we call this problem RPGSC) [8]. Each rectangle is associated with its spatial cost function and a set of modes, where each mode specifies the width and height of the rectangle. The spatial cost is a general piecewise linear function which can be non-convex and discontinuous. The problem is to pack a given set of n rectangles without overlap so that the maximum cost of the rectangles is minimized. A solution, called a packing, is determined by specifying the mode and the location of each rectangle, which is feasible if no two rectangles overlap.

This problem is very general, and various types of packing problems and scheduling problems can be formulated in this form. For example, we can treat 2RP in which rotations of 90° are allowed as a special case of RPGSC by considering two modes for each rectangle corresponding to: (1) the original orientation and (2) the orientation rotated by 90°. A quite different problem taken from the scheduling problem of constructing large building blocks can also be handled by using appropriate spatial costs, as will be described in Section 6.

In this paper, we consider local search algorithms for RPGSC. If we search directly the x and y coordinates and the mode of each rectangle, then an effective search will be difficult, since the number of solutions is uncountably many and eliminating overlap of rectangles is not easy. Therefore many coding schemes of solutions have been proposed [1, 10, 12, 14, 15]. One of the most popular coding schemes is to represent a solution by a permutation of n rectangles [1, 7, 10, 12, 20], for which there are various decoding schemes to obtain a packing from a coded solution. Among such schemes are first-fit, bottom-left, floor-ceiling and so on, where each of these schemes takes $O(n \log n)$, $O(n^2)$ and $O(n^3)$ time, respectively, to decode a solution. The reader is referred to surveys [2, 3, 11] on cutting and packing problems to know numerous algorithms of this type. Nakatake, Fujiyoshi, Murata and Kajitani proposed a coding scheme called the bounded sliceline grid (BSG) [15], and obtained good results for 2RP in which rotations are allowed and the objective is to minimize the area of the bin containing all given rectangles. Murata, Fujiyoshi, Nakatake and Kajitani proposed a coding scheme called the sequence pair [14]. In the sequence pair representation, a solution is represented by a pair of permutations of n rectangles. They proposed an $O(n^2)$ time decoding algorithm to obtain a feasible packing (i.e., x and y coordinates of rectangles) from a pair of permutations. Takahashi [17] and Tang, Tian and Wong [18] improved the time complexity of the decoding algorithm to $O(n \log n)$; Tang and Wong [19] further improved it to $O(n \log \log n)$.

We also adopt the sequence pair [14] as the coding scheme in our algorithm for RPGSC. A

solution is coded as a pair of permutations of n rectangles and a vector specifying the modes of all rectangles. In our previous paper [8], we proposed a decoding algorithm based on dynamic programming to obtain an optimal packing under the constraint specified by the coded solution. This algorithm is a generalization of the algorithms proposed in [17, 18] so that general spatial costs can be handled. The running time of this algorithm is $O(n \log n)$ if applied to the cases of area minimization, strip packing, two-dimensional 0-1 knapsack and so on. The details of this algorithm are described in Section 5.1.

In this paper, we propose new algorithms to evaluate all coded solutions in various neighborhoods. The amortized computational time of these algorithms per one solution is $O(1)$ or $O(\log n)$ depending on the neighborhood, if applied to special cases including area minimization, strip packing, two-dimensional 0-1 knapsack and so on. These algorithms can not produce a packing, but only compute the objective value of the packing; however, the speed up of such evaluation is essentially important, since we must evaluate a large number of coded solutions in local search and metaheuristic algorithms. The details of the proposed algorithms are described in Section 5.2.

We consider three types of neighborhoods in Section 4, which are used in the local search algorithms for finding good coded solutions. We define the critical path that corresponds to the bottleneck of the current solution, and make use of it to reduce the sizes of neighborhoods. The local search algorithms based on these neighborhoods are then incorporated in metaheuristic algorithms such as the random multi-start local search (MLS) and the iterated local search (ILS).

The computational results are reported in Section 6. We examine the performance of proposed decoding algorithms to compare them with original decoding algorithms. We also compare our algorithms with other existing heuristic algorithms for the rectangle packing problem and a real-world scheduling problem.

2 Problem formulation

Let $I = \{1, 2, \ldots, n\}$ be a set of n rectangles. Each rectangle $i \in I$ has m_i modes, and each mode k ($k = 1, 2, \ldots, m_i$) of rectangle i specifies:

- a width $w_i^{(k)}$, a height $h_i^{(k)}$ ($w_i^{(k)}, h_i^{(k)} \geq 0$) and a cost $c_i^{(k)}$ of the mode,
- spatial cost functions $p_i^{(k)}(x)$ and $q_i^{(k)}(y)$ on the location (x, y) of the rectangle, where the location of a rectangle means the (x, y)-coordinate of its lower left corner.

We assume that $p_i^{(k)}(x)$ and $q_i^{(k)}(y)$ are piecewise linear and nonnegative (i.e., $p_i^{(k)}(x), q_i^{(k)}(y) \geq 0$ hold for all $x, y \in [-\infty, \infty]$). It is also assumed that if $x, y \to \pm\infty$, then $p_i^{(k)}(x), q_i^{(k)}(y) \to +\infty$. Moreover, we assume that these functions are lower semi-continuous; that is, any discontinuous point x (if exists) must satisfy $p_i^{(k)}(x) \leq \lim_{\varepsilon \to 0} \min\{p_i^{(k)}(x+\varepsilon), p_i^{(k)}(x-\varepsilon)\}$. The assumption on the discontinuous points of $q_i^{(k)}(y)$ is similar. These conditions are necessary to ensure the existence of an optimal solution, and most natural spatial cost functions satisfy them. Note that the spatial cost functions can be non-convex and discontinuous as long as they satisfy the above conditions. It is also assumed that the information of each linear piece of functions $p_i^{(k)}(x)$ and $q_i^{(k)}(y)$ are given explicitly, and each function is represented by the linked lists. In many applications, the number of linear pieces of each spatial cost function is small, and hence this assumption is natural.

A packing π of an instance of RPGSC is specified by the modes of n rectangles $\mu(\pi) = (\mu_1(\pi), \mu_2(\pi), \ldots, \mu_n(\pi))$ and locations $(x_i(\pi), y_i(\pi))$ of all rectangles i. We define $p_{\max}(\pi)$ and $q_{\max}(\pi)$ as follows:

$$p_{\max}(\pi) = \max_{i \in I} p_i^{(\mu_i(\pi))}(x_i(\pi)), \tag{1}$$

$$q_{\max}(\pi) = \max_{i \in I} q_i^{(\mu_i(\pi))}(y_i(\pi)). \tag{2}$$

Now we are given two cost functions $g(p_{\max}(\pi), q_{\max}(\pi))$ and $c(\mu(\pi))$. We assume that function g is nondecreasing in $p_{\max}(\pi)$ and $q_{\max}(\pi)$, and can be computed in $O(1)$ time for given $p_{\max}(\pi)$ and $q_{\max}(\pi)$. Moreover, we assume that $c(\mu(\pi))$ can be computed in $O(n)$ time for a given $\mu(\pi)$, and $c(\mu(\pi)) - c(\mu(\pi'))$ can be computed in $O(|\{i \in I \mid \mu_i(\pi) \neq \mu_i(\pi')\}|)$ time for given $\mu(\pi)$ and $\mu(\pi')$. Then the problem considered in this paper is defined as follows:

RPGSC:	minimize	$g(p_{\max}(\pi), q_{\max}(\pi)) + c(\mu(\pi))$	
	subject to	At least one of the next four inequalities	
		holds for every pair (i, j) of rectangles:	
		$x_i(\pi) + w_i^{(\mu_i(\pi))} \leq x_j(\pi)$,	(3)
		$x_j(\pi) + w_j^{(\mu_j(\pi))} \leq x_i(\pi)$,	(4)
		$y_i(\pi) + h_i^{(\mu_i(\pi))} \leq y_j(\pi)$,	(5)
		$y_j(\pi) + h_j^{(\mu_j(\pi))} \leq y_i(\pi)$.	(6)

The constraints from (3) to (6) tell that no two rectangles overlap in a packing π, and we call a packing satisfying all constraints feasible.

Defining $p_i^{(k)}(x)$, $q_i^{(k)}(y)$ and $c_i^{(k)}$ for each i and k appropriately, various types of packing problems and scheduling problems can be formulated in our form. For example, we can treat 2RP in which rotations of 90° are allowed as a special case of RPGSC by considering two modes of (1) the original orientation and (2) the orientation rotated by 90°. We can also treat both types of rectangle packing problems, where a bin of a fixed size to which rectangles should be packed is given or not, by defining $p_i^{(k)}(x)$ and $q_i^{(k)}(y)$ appropriately. Some examples are described in Section 6.

The problem RPGSC is NP-hard, since the one-dimensional bin packing problem, which is known to be strongly NP-hard [4], can be polynomially reducible to this problem. Note that problem RPGSC is not purely combinatorial, since the coordinates $x_i(\pi)$ and $y_i(\pi)$ of each rectangle i can be any real values, and hence there are continuously infinite number of solutions.

3 Coding scheme

As noted before, we adopt the sequence pair [14] as the coding scheme in our algorithm. A sequence pair is a pair of permutations $\sigma = (\sigma_+, \sigma_-)$ of $I = \{1, 2, \ldots, n\}$, where $\sigma_+(l) = i$ (equivalently $\sigma_+^{-1}(i) = l$) means that rectangle i is the lth rectangle in σ_+. σ_- is similarly defined. In a feasible packing π, every pair i and j of rectangles satisfies at least one of the four conditions from (3) to (6). A sequence pair determines which of the four conditions is satisfied in the packing as follows.

Given a sequence pair $\sigma = (\sigma_+, \sigma_-)$, we define two partial orders $\preceq_\sigma^x$ and $\preceq_\sigma^y$ by

$$\sigma_+^{-1}(i) \leq \sigma_+^{-1}(j) \text{ and } \sigma_-^{-1}(i) \leq \sigma_-^{-1}(j) \Longleftrightarrow i \preceq_\sigma^x j, \tag{7}$$

$$\sigma_+^{-1}(i) \geq \sigma_+^{-1}(j) \text{ and } \sigma_-^{-1}(i) \leq \sigma_-^{-1}(j) \Longleftrightarrow i \preceq_\sigma^y j, \tag{8}$$

for any pair i and j of rectangles. Note that $i \preceq_\sigma^x i$ and $i \preceq_\sigma^y i$ hold for all i, and the next property holds.

Property 1 [14]**:** Exactly one of the four relations $i \preceq_\sigma^x j$, $j \preceq_\sigma^x i$, $i \preceq_\sigma^y j$ and $j \preceq_\sigma^y i$ holds for any pair i and j of rectangles with $i \neq j$.

Then, given a sequence pair $\sigma = (\sigma_+, \sigma_-)$ and a vector of modes $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$, we define

$\Pi_{\sigma,\mu}$ as the set of packings π that satisfy the following three conditions for all i and $j \in I$:

$$\mu_i(\pi) = \mu_i, \tag{9}$$

$$i \preceq^x_\sigma j \text{ and } i \neq j \Longrightarrow x_i(\pi) + w_i^{(\mu_i)} \leq x_j(\pi), \tag{10}$$

$$i \preceq^y_\sigma j \text{ and } i \neq j \Longrightarrow y_i(\pi) + h_i^{(\mu_i)} \leq y_j(\pi). \tag{11}$$

This means that any packing $\pi \in \Pi_{\sigma,\mu}$ is feasible and satisfies the mode constraint. Conversely, it can be shown that, for any feasible packing π, there exists a pair of σ and μ that satisfies $\pi \in \Pi_{\sigma,\mu}$ [8, 14]. Moreover, it is possible to find a feasible packing π from a coded solution (σ, μ) efficiently; see decoding algorithms in Section 5.

4 Local search

In this section, we propose local search algorithms to find good coded solutions (σ, μ). In Section 4.1, we define critical paths which are used in our local search algorithms, and then we explain the framework of local search in Section 4.2. In Section 4.3, we propose three types of neighborhoods based on critical paths.

4.1 Critical paths

Critical paths are defined for both of the x and y directions. We explain the definition only for the x direction, as that for the y direction is similar.

Given a packing $\pi \in \Pi_{\sigma,\mu}$, define a directed graph $G = (V, E)$ and subsets $S, T, P \subseteq I$ as follows, where $p_{\max}(\pi)$ was defined in (1):

$$V = I,$$

$$(i,j) \in E \Longleftrightarrow x_i(\pi) + w_i^{(\mu_i(\pi))} = x_j(\pi) \text{ and } i \preceq^x_\sigma j,$$

$$S = \{i \in I \,|\, p_i(x_i(\pi) - \varepsilon) \geq p_{\max}(\pi) \text{ for an arbitrarily small } \varepsilon > 0\},$$

$$T = \{i \in I \,|\, p_i(x_i(\pi) + \varepsilon) \geq p_{\max}(\pi) \text{ for an arbitrarily small } \varepsilon > 0\},$$

$$P = \{i \in I \,|\, p_i(x_i(\pi)) = p_{\max}(\pi)\}.$$

We then define a *critical path* as a directed path in G, whose initial vertex s is in S, final vertex t is in T and at least one vertex $v \in P$ exists on this path (including the end vertices). For any packing π obtained by our decoding algorithms in Section 5.1, S, T and P are nonempty and there is at least one critical path for each direction. It is easy to find all rectangles which are on critical paths in $O(n^2)$ time. Critical paths have an important property: $p_{\max}(\pi)$ cannot be

decreased without breaking all critical paths of the x direction. Our local search algorithms are designed on the basis of critical paths.

4.2 Framework of local search

The local search (LS) starts from an initial solution (σ, μ) and repeats replacing (σ, μ) with a better solution in its *neighborhood* $N(\sigma, \mu)$ until no better solution is found in $N(\sigma, \mu)$, where $N(\sigma, \mu)$ is a set of solutions obtainable from (σ, μ) by slight perturbations (which will be defined later). A solution (σ, μ) is called *locally optimal*, if no better solution exists in $N(\sigma, \mu)$. The LS from an initial solution $(\sigma^{(0)}, \mu^{(0)})$, in which neighborhood N is used and solutions are evaluated by a function *eval*, is described as follows.

Algorithm LS$(N, (\sigma^{(0)}, \mu^{(0)}))$

Step 1 Let $\sigma := \sigma^{(0)}$ and $\mu := \mu^{(0)}$.

Step 2 If there is a solution $(\sigma', \mu') \in N(\sigma, \mu)$ such that $eval(\sigma', \mu') < eval(\sigma, \mu)$, let $\sigma := \sigma'$, $\mu := \mu'$ and return to Step 2. Otherwise output (σ, μ) and stop.

The search space of LS is the set of all coded solutions (σ, μ). A solution (σ, μ) is basically evaluated by the objective value of an optimal packing $\pi \in \Pi_{\sigma,\mu}$, which can be computed efficiently by decoding algorithms in Section 5. However, to break ties, we also compute the number of rectangles which are on the critical paths, and put a higher priority on a packing with a smaller number of rectangles on the critical paths. We use a pair of random permutations and a random mode vector as the initial solution, and adopt first admissible move strategy in Step 2 (i.e., as soon as we find a better solution in its neighborhood, we move to the solution).

In general, if LS is applied only once, many solutions of better quality may remain unvisited in the search space. Therefore, local search may be enhanced by various ideas. Metaheuristics can be viewed as a framework for such variations. We now explain two simple metaheuristic algorithms.

(1) The *random multi-start local search* (MLS). This is one of the simplest metaheuristic algorithms. In MLS, we randomly generate many initial solutions and apply LS to each initial solution independently. Then, the best of the obtained locally optimal solutions is output.

(2) The *iterated local search* (ILS) [9, 13]. ILS is a variant of MLS, in which initial solutions are generated by slightly perturbing a good solution $(\sigma_{\rm seed}, \mu_{\rm seed})$ found during the search so far. It is important to generate initial solutions which retain some features of good solutions and to

avoid a cycling of solutions in order to improve the performance of ILS.

In our ILS algorithm, we use the best locally optimal solution obtained so far (if there exists ties, we use the solution obtained most recently) as $(\sigma_{\rm seed}, \mu_{\rm seed})$, and apply random swap operations (i.e., exchange the positions of randomly chosen two rectangles in both permutations) ν times on $(\sigma_{\rm seed}, \mu_{\rm seed})$ to generate a new initial solution. Here, ν is a parameter and we set it one, two or three at random. In preliminary experiments, we observed that ILS was generally superior to MLS, and hence we will use ILS as the framework of our metaheuristic algorithm in Section 6.

4.3 Neighborhoods

In our LS, we use the following three types of neighborhoods, called *shift*, *two-shifts* and *change mode*. Shift and change mode are more or less standard neighborhoods, and two-shifts is an extension of the shift neighborhood which includes the swap neighborhood, another standard neighborhood.

4.3.1 Shift neighborhood

A shift operation changes the position of one rectangle i to another position in both σ_+ and σ_-. The *shift* neighborhood is defined to be the set of all solutions obtainable from the current solution by a shift operation. Its size is $O(n^3)$. To reduce the size of shift neighborhood, we use the information of critical paths, i.e., we shift only those rectangles i located on critical paths. It is easy to show that a solution is locally optimal in the original neighborhood if no improved solution is found in the reduced neighborhood. In this sense, we can reduce the size of shift neighborhood without sacrificing the solution quality. In our implementation, we further restrict the positions, where a shifted rectangle i is placed, in the following three manners.

(1) The position of the shifted rectangle is changed only in one permutation (σ_+ or σ_-). We call this a *single shift operation*.

(2) The positions, where the shifted rectangle i is inserted, are determined by the following rule: We choose one rectangle j ($\neq i$) arbitrarily and insert i before or after j in both σ_+ and σ_-, thereby four insertion positions of i are examined for each j (see Figure 1). Intuitively, in the packing space, we move rectangle i to the position just to the left of, right of, above or below the chosen j by these restricted operations. We call this a *limited double shift operation*. (Figure 1)

(3) We insert rectangle i to positions close to the current positions of i in two permutations.

To control the size of this neighborhood, we limit the distance from the current position to up to $\lceil a\sqrt{n} \rceil$ in each permutation, where a is a parameter. (We set $a = 1$ in our computational experiments.) The size of this neighborhood becomes $O(a^2 n)$ for each i. We call this a *near-place double shift operation.*

For convenience, we denote each of these operations a limited shift operation, and the set of solutions obtainable by a limited shift operation is called the *limited shift* neighborhood. The size of this neighborhood (based on three kinds of operations) is $O(cn)$, where c is the number of rectangles on the critical paths.

4.3.2 Two-shifts neighborhood

A two-shifts operation changes the positions of two rectangles i and j to another positions in σ_+ and σ_-. The *two-shifts* neighborhood defined by this operation is an extension of two kinds of standard neighborhoods (i.e., shift and swap neighborhoods). The size of two-shifts neighborhood without any reduction is $O(n^6)$. This is too large to search efficiently, and hence we propose various reduction methods for this neighborhood.

First, we restrict the rectangles i and j to be shifted. As we mentioned in Section 4.1, at least one rectangle must be chosen from the rectangles on critical paths to improve the current solution. If we choose both rectangles only from critical paths, the size of neighborhood becomes more small. However, the possibility of missing some improved solutions in the original neighborhood appears to be high. Hence, we choose one rectangle from critical paths and another arbitrarily.

Next, we restrict the insertion positions of the rectangles i and j to those determined by one of the following two rules:

(1) First, we exchange the positions of two rectangles i and j in both permutations. Then, we shift the position of only one rectangle i to another position in σ_+ and σ_- by a limited shift operation proposed in the previous section. We call this a *swap-and-shift operation.*

(2) We remove two rectangles i and j from permutations and insert them one by one in greedy fashion. First we choose a pair of permutations consisting of $n-1$ rectangles $I \setminus \{j\}$ among those obtainable by a limited shift operation of rectangle i, where $n-2$ rectangles in $I \setminus \{i, j\}$ are fixed. Then rectangle j is inserted into any position by a limited shift operation of rectangle j. We call this a *greedy two-shifts operation.*

The set of solutions obtainable by one of these operations is called the *limited two-shifts* neigh-

borhood, and its size is $O(cn^2)$.

4.3.3 Change mode neighborhood

A change mode operation changes the mode of one selected rectangle. This is the only operation applied to the vector μ. We combine this operation to shift and two-shifts operations, i.e., when we insert a rectangle to permutations, we may change its mode.

5 Decoding algorithm

In this section, we consider the following problem for a given coded solution (σ, μ):

$$\begin{array}{lll} \text{RPGSC}(\sigma,\mu)\text{:} & \text{minimize} & g(p_{\max}(\pi), q_{\max}(\pi)) \\ & \text{subject to} & \pi \in \Pi_{\sigma,\mu}, \end{array}$$

and propose dynamic programming algorithms to compute the objective value of an optimal packing in polynomial time. In Section 5.1, we review a basic algorithm proposed in our previous paper [8], and propose in Section 5.2 new efficient algorithms for various neighborhoods.

5.1 Basic algorithm

We describe the basic idea of algorithm CMPF (Compute-Minimum-Penalty-Function) proposed in [8] for solving RPGSC(σ, μ). This algorithm is also used in our local search and metaheuristic algorithms.

By Property 1 in Section 3, we can obtain a feasible packing by considering horizontal and vertical coordinates separately. Moreover, since the objective function $g(p_{\max}(\pi), q_{\max}(\pi))$ is assumed to be nondecreasing in $p_{\max}(\pi)$ and $q_{\max}(\pi)$, respectively, it can be minimized by minimizing $p_{\max}(\pi)$ and $q_{\max}(\pi)$ independently. We give below an algorithm to minimize $p_{\max}(\pi)$ only, as the minimization of $q_{\max}(\pi)$ can be similarly treated.

Let us define J_i^{f} and $f_i(x)$ for each i as follows:

$J_i^{\mathrm{f}} = \{j \in I \mid j \preceq_\sigma^x i\}$,

$f_i(x)$: the minimum value of $\max_{j \in J_i^{\mathrm{f}}} p_j^{(\mu_j)}(x_j(\pi))$ subject to $x_j(\pi) + w_j^{(\mu_j)} \leq x$ for all $j \in J_i^{\mathrm{f}}$, and $\pi \in \Pi_{\sigma,\mu}$.

We call $f_i(x)$ the minimum penalty function. This function is nonincreasing in x by definition, and the minimum penalty value $p_{\max}(\pi)$ of (1) can be obtained by

$$p_{\max}(\pi) = \max_{i \in I} \min_x f_i(x). \tag{12}$$

Then, by the idea of dynamic programming, $f_i(x)$ can be computed by

$$f_i(x) = \begin{cases} \min_{x_i \le x - w_i^{(\mu_i)}} p_i^{(\mu_i)}(x_i), & \text{if } J_i^{\mathrm{f}} = \{i\} \\ \min_{x_i \le x - w_i^{(\mu_i)}} \max\{p_i^{(\mu_i)}(x_i), \max_{j \in J_i^{\mathrm{f}} \setminus \{i\}} f_j(x_i)\}, & \text{otherwise,} \end{cases} \tag{13}$$

for all $i = \sigma_+(1), \sigma_+(2), \ldots, \sigma_+(n)$. The horizontal coordinates $x_i(\pi)$ of each rectangle $i = \sigma_+(n), \sigma_+(n-1), \ldots, \sigma_+(1)$ can be computed by

$$x_i(\pi) = \begin{cases} \max\{x_i \,|\, p_i^{(\mu_i)}(x_i) = \min_{x_i'}\{p_i^{(\mu_i)}(x_i') \,|\, f_i(x_i' + w_i^{(\mu_i)}) = \min_x f_i(x)\}\}, \\ \qquad\qquad \text{if } J_i^{\mathrm{b}} = \{i\} \\ \max\{x_i \,|\, p_i^{(\mu_i)}(x_i) = \min_{x_i'}\{p_i^{(\mu_i)}(x_i') \,|\, f_i(x_i' + w_i^{(\mu_i)}) = \min_x\{f_i(x) \,|\, x \le r_i\}\}\}, \\ \qquad\qquad \text{otherwise,} \end{cases} \tag{14}$$

where $J_i^{\mathrm{b}} = \{j \in I \,|\, i \preceq_\sigma^x j\}$ and $r_i = \min_{j \in J_i^{\mathrm{b}} \setminus \{i\}} x_j(\pi)$. We can minimize $p_{\max}(\pi)$ with these horizontal coordinates, and moreover, we can minimize $p_i^{(\mu_i)}(x_i(\pi))$ for all i locally.

Computational time of this algorithm depends on the space complexity (i.e., the number of linear pieces) of the minimum penalty functions $f_i(x)$, and we denote its upper bound by τ. In many important special cases, such as area minimization, strip packing, two-dimensional 0-1 knapsack and so on, the spatial cost functions $p_i^{(\mu_i)}(x)$ are simple and τ becomes $O(1)$. In more general cases of $p_i^{(\mu_i)}(x)$, τ is $O(n)$ in many realistic applications. More detailed analysis of τ is discussed in [8].

If we compute $f_i(x)$ by using (13) naively, the computational time to compute $f_i(x)$ is $O(\tau n^2)$, since $\sum_i |\, J_i^{\mathrm{f}} \,| = O(n^2)$. However, algorithm CMPF [8] utilizes a complete binary tree of height $\lceil \log_2 n \rceil$ with n leaves to compute $f_i(x)$, and the time complexity of CMPF is $O(\tau n \log n)$.

5.2 Improved decoding algorithms

In this section, we propose new algorithms to evaluate coded solutions (σ, μ) in those neighborhoods proposed in Section 4.3. We will explain algorithms to minimize $p_{\max}(\pi)$ only. Algorithms to minimize $q_{\max}(\pi)$ can be similarly defined and we can minimize the objective value of RPGSC(σ, μ) by applying these algorithms to the x and y coordinates independently.

5.2.1 Evaluating shift moves

We propose an algorithm to evaluate solutions in the shift neighborhood where the current solution is (σ, μ) and rectangle i will be shifted. We call this algorithm Evaluate-Shift-Moves (ESM), which is described as follows.

Let $\tilde{I} = I - \{i\}$ and $(\tilde{\sigma}, \tilde{\mu})$ be the coded solution obtained from the current solution (σ, μ) by removing a rectangle i, and $\tilde{\pi}$ be a packing of rectangles in $\tilde{I}$ such that $\tilde{\pi} \in \Pi_{\tilde{\sigma},\tilde{\mu}}$. We can compute an optimal packing $\tilde{\pi}^* \in \Pi_{\tilde{\sigma},\tilde{\mu}}$ from $(\tilde{\sigma}, \tilde{\mu})$ by algorithm CMPF in the previous section. If a new packing π does not have the rectangle i on its critical paths of x direction, the minimum penalty value $p_{\max}(\pi)$ is equal to $p_{\max}(\tilde{\pi}^*)$.

We then compute the minimum penalty value under the influence of rectangle i. For this, let us define $\tilde{J}^{\mathrm{f}}_{\alpha,\beta}$, $\tilde{J}^{\mathrm{b}}_{\alpha,\beta}$, $\tilde{f}_{\alpha,\beta}(x)$ and $\tilde{b}_{\alpha,\beta}(x)$ for each pair α and β such that $1 \leq \alpha, \beta \leq n$:

$\tilde{J}^{\mathrm{f}}_{\alpha,\beta} = \{j \in \tilde{I} \mid \tilde{\sigma}_+^{-1}(j) < \alpha, \tilde{\sigma}_-^{-1}(j) < \beta\}$,
$\tilde{J}^{\mathrm{b}}_{\alpha,\beta} = \{j \in \tilde{I} \mid \tilde{\sigma}_+^{-1}(j) \geq \alpha, \tilde{\sigma}_-^{-1}(j) \geq \beta\}$,
$\tilde{f}_{\alpha,\beta}(x)$: the minimum value of $\max_{j \in \tilde{J}^{\mathrm{f}}_{\alpha,\beta}} p_j^{(\tilde{\mu}_j)}(x_j(\tilde{\pi}))$ subject to $x_j(\tilde{\pi}) + w_j^{(\tilde{\mu}_j)} \leq x$ for all $j \in \tilde{J}^{\mathrm{f}}_{\alpha,\beta}$, and $\tilde{\pi} \in \Pi_{\tilde{\sigma},\tilde{\mu}}$,
$\tilde{b}_{\alpha,\beta}(x)$: the minimum value of $\max_{j \in \tilde{J}^{\mathrm{b}}_{\alpha,\beta}} p_j^{(\tilde{\mu}_j)}(x_j(\tilde{\pi}))$ subject to $x_j(\tilde{\pi}) \geq x$ for all $j \in \tilde{J}^{\mathrm{b}}_{\alpha,\beta}$, and $\tilde{\pi} \in \Pi_{\tilde{\sigma},\tilde{\mu}}$.

Then, by the idea of dynamic programming, $\tilde{f}_{\alpha,\beta}(x)$ (respectively, $\tilde{b}_{\alpha,\beta}(x)$) can be computed by

$$\tilde{f}_{\alpha,\beta}(x) = \begin{cases} 0, & \text{if } \alpha = 1 \text{ or } \beta = 1 \\ \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha,\beta-1}(x)\}, & \text{if } \tilde{\sigma}_+(\alpha-1) \neq \tilde{\sigma}_-(\beta-1) \\ \min_{t \leq x - w_j^{(\tilde{\mu}_j)}}\{\max(p_j^{(\tilde{\mu}_j)}(t), \tilde{f}_{\alpha-1,\beta-1}(t))\}, & \text{if } \tilde{\sigma}_+(\alpha-1) = \tilde{\sigma}_-(\beta-1) = j, \end{cases} \tag{15}$$

$$\tilde{b}_{\alpha,\beta}(x) = \begin{cases} 0, & \text{if } \alpha = n \text{ or } \beta = n \\ \max\{\tilde{b}_{\alpha+1,\beta}(x), \tilde{b}_{\alpha,\beta+1}(x)\}, & \text{if } \tilde{\sigma}_+(\alpha) \neq \tilde{\sigma}_-(\beta) \\ \min_{t \geq x}\{\max(p_j^{(\tilde{\mu}_j)}(t), \tilde{b}_{\alpha+1,\beta+1}(t + w_j^{(\tilde{\mu}_j)}))\}, & \text{if } \tilde{\sigma}_+(\alpha) = \tilde{\sigma}_-(\beta) = j, \end{cases} \tag{16}$$

for all pairs of $\alpha = 1, 2, \ldots, n$ and $\beta = 1, 2, \ldots, n$ (resp., all pairs of $\alpha = n, n-1, \ldots, 1$ and $\beta = n, n-1, \ldots, 1$). See Figure 2 as an example of computing $\tilde{f}_{\alpha,\beta}(x)$ and $\tilde{b}_{\alpha,\beta}(x)$. Each box in this figure corresponds to $\tilde{f}_{\alpha,\beta}(x)$ (resp., $\tilde{b}_{\alpha,\beta}(x)$) for each pair of α and β, and arrows mean how to compute each function. That is, the function of each shaded box is computed by the third formula of (15) (resp., (16)). After computing these, we can obtain the minimum penalty value $p_{\max}(\pi)$ of the coded solution obtained by inserting the removed rectangle i into the αth position of $\tilde{\sigma}_+$ and the βth position of $\tilde{\sigma}_-$ with mode k, by (Figure 2)

$$p_{\max}(\pi) = \max\{p_{\max}(\tilde{\pi}^*), \min_t \max(\tilde{f}_{\alpha,\beta}(t), p_i^{(k)}(t), \tilde{b}_{\alpha,\beta}(t + w_i^{(k)}))\}. \tag{17}$$

Note that, we can check all modes $k = 1, 2, \ldots, m_i$ for rectangle i at this stage. It takes $O(\tau)$ time to compute $p_{\max}(\pi)$ for each mode.

Now we evaluate the time complexity of ESM. It first takes $O(\tau n \log n)$ time to compute an optimal $\tilde{\pi}^* \in \Pi_{\tilde{\sigma},\tilde{\mu}}$ with algorithm CMPF. Time to compute $\tilde{f}_{\alpha,\beta}(x)$ and $\tilde{b}_{\alpha,\beta}(x)$ for all pairs of

α and β by (15) and (16) is $O(\tau n^2)$. Time to compute the minimum penalty value by (17) is $O(\tau)$ for each pair of α and β, and it becomes $O(\tau n^2)$ for all pairs of α and β. In summary, the total computational time of this algorithm, i.e., time to evaluate all solutions when rectangle i is shifted in the shift neighborhood, is $O(\tau n^2)$. That is, it takes $O(\tau)$ amortized time to evaluate a coded solution in the neighborhood, and this becomes $O(1)$ for special cases in which $\tau = O(1)$ (e.g., area minimization, strip packing, and two-dimensional 0-1 knapsack).

5.2.2 Evaluating limited shift moves

We propose an algorithm to evaluate solutions in the limited shift neighborhood where rectangle i will be shifted. It is clear that we can evaluate all solutions in this neighborhood with the previous algorithm, however, taking $O(\tau n)$ amortized time to evaluate each coded solution, since there are only $O(n)$ solutions in this case. We therefore introduce some ideas to compute $\tilde{f}_{\alpha,\beta}(x)$ and $\tilde{b}_{\alpha,\beta}(x)$ only for those α and β necessary to evaluate solutions in this neighborhood. Note that, the limited shift neighborhood is based on three kinds of operations defined in Section 4.3.1, and we propose ideas for each operation.

To evaluate solutions obtainable by a limited double shift operation, we can obtain $\tilde{f}_{\alpha,\beta}(x)$ from the computation of algorithm CMPF for $\tilde{\pi}^*$. That is, if we insert rectangle i before or after rectangle $j = \tilde{\sigma}_+(\alpha-1) = \tilde{\sigma}_-(\beta-1)$ in both permutations, we use $\tilde{f}_{\alpha-1,\beta-1}(x)$, $\tilde{f}_{\alpha-1,\beta}(x)$, $\tilde{f}_{\alpha,\beta-1}(x)$ or $\tilde{f}_{\alpha,\beta}(x)$ to evaluate a new coded solution (e.g., if i is inserted 'after' j in both $\tilde{\sigma}_+$ and $\tilde{\sigma}_-$, $\tilde{f}_{\alpha,\beta}(x)$ is necessary to compute (17)). Let us define $\tilde{J}^{\mathrm{f}}_j$ and $\tilde{f}_j(x)$ in a similar way as J^{f}_i and $f_i(x)$ (see Section 5.1). Function $\tilde{f}_j(x)$ is computed in the computation CMPF for $\tilde{\pi}^*$. Then, $\tilde{f}_{\alpha,\beta}(x)$ is equal to $\tilde{f}_j(x)$, since $\tilde{J}^{\mathrm{f}}_{\alpha,\beta} = \tilde{J}^{\mathrm{f}}_j$. Moreover, other functions $\tilde{f}_{\alpha-1,\beta-1}(x)$, $\tilde{f}_{\alpha-1,\beta}(x)$ and $\tilde{f}_{\alpha,\beta-1}(x)$ are equal to $\max_{j' \in \tilde{J}^{\mathrm{f}}_j \setminus \{j\}} \tilde{f}_{j'}(x)$, which is a part of (13) and has been already computed, since $\tilde{J}^{\mathrm{f}}_{\alpha-1,\beta-1} = \tilde{J}^{\mathrm{f}}_{\alpha-1,\beta} = \tilde{J}^{\mathrm{f}}_{\alpha,\beta-1} = \tilde{J}^{\mathrm{f}}_j \setminus \{j\}$. Hence, we can compute $\tilde{f}_{\alpha,\beta}(x)$ for all necessary α and β in $O(\tau n \log n)$ time, which is the computational time of CMPF. We can compute $\tilde{b}_{\alpha,\beta}(x)$ for all necessary α and β in a similar way.

To evaluate solutions obtainable by a single shift operation where rectangle i will be shifted in σ_+, we should compute $\tilde{f}_{\alpha,\beta}(x)$ for all $1 \leq \alpha \leq n$ and $\beta = \sigma_-^{-1}(i)$. $\tilde{f}_{\alpha,\beta}(x)$ can be computed for all necessary α and β by

$$\tilde{f}_{\alpha,\beta}(x) = \begin{cases} \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{j'}(x)\}, & \text{if } \tilde{\sigma}_-^{-1}(j') \leq \beta - 2 \\ \tilde{f}_{j'}(x), & \text{if } \tilde{\sigma}_-^{-1}(j') = \beta - 1 \\ \tilde{f}_{\alpha-1,\beta}(x), & \text{otherwise,} \end{cases} \tag{18}$$

where $j' = \tilde{\sigma}_+(\alpha - 1)$. This is because $\tilde{f}_{\alpha,\beta}(x)$ defined by (15) satisfy (see Figure 3):

$$\begin{aligned}
\tilde{f}_{\alpha,\beta}(x) &= \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha,\beta-1}(x)\} = \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha-1,\beta-1}(x), \tilde{f}_{\alpha,\beta-2}(x)\} = \cdots \\
&= \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha-1,\beta-1}(x), \ldots, \tilde{f}_{\alpha-1,l+1}(x), \tilde{f}_{\alpha,l}(x)\} \\
&= \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{j'}(x)\}, \qquad \text{if } \tilde{\sigma}_-^{-1}(j') = l - 1 \leq \beta - 2, \\
\tilde{f}_{\alpha,\beta}(x) &= \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha,\beta-1}(x)\} = \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha-1,\beta-1}(x), \tilde{f}_{\alpha,\beta-2}(x)\} = \cdots \\
&= \max\{\tilde{f}_{\alpha-1,\beta}(x), \tilde{f}_{\alpha-1,\beta-1}(x), \ldots, \tilde{f}_{\alpha-1,2}(x), \tilde{f}_{\alpha,1}(x)\} \\
&= \tilde{f}_{\alpha-1,\beta}(x), \qquad \text{if } \tilde{\sigma}_-^{-1}(j') \geq \beta.
\end{aligned}$$

Similarly, if i is shifted in σ_-, $\tilde{f}_{\alpha,\beta}(x)$ can be computed for all necessary α and β by (Figure 3)

$$\tilde{f}_{\alpha,\beta}(x) = \begin{cases} \max\{\tilde{f}_{\alpha,\beta-1}(x), \tilde{f}_{j''}(x)\}, & \text{if } \tilde{\sigma}_+^{-1}(j'') \leq \alpha - 2 \\ \tilde{f}_{j''}(x), & \text{if } \tilde{\sigma}_+^{-1}(j'') = \alpha - 1 \\ \tilde{f}_{\alpha,\beta-1}(x), & \text{otherwise,} \end{cases} \tag{19}$$

where $j'' = \tilde{\sigma}_-(\beta - 1)$. Time to compute $\tilde{f}_{\alpha,\beta}(x)$ for all necessary α and β by (18) or (19) is $O(\tau n)$ if the information from algorithm CMPF is retained. The computation of $\tilde{b}_{\alpha,\beta}(x)$ is similar and is omitted.

To evaluate solutions obtainable by a near-place double shift operation where rectangle i will be shifted, we should compute $\tilde{f}_{\alpha,\beta}(x)$ for all $\alpha_l \leq \alpha \leq \alpha_u$ and $\beta_l \leq \beta \leq \beta_u$, where $\alpha_u - \alpha_l \leq 2\lceil a\sqrt{n}\,\rceil$ and $\beta_u - \beta_l \leq 2\lceil a\sqrt{n}\,\rceil$ hold (a is a parameter explained in Section 4.3.1). We first compute $\tilde{f}_{\alpha,\beta_l}(x)$ for $1 \leq \alpha \leq \alpha_u$ by (18) and $\tilde{f}_{\alpha_l,\beta}(x)$ for $1 \leq \beta \leq \beta_u$ by (19). Then, we use (15) to compute $\tilde{f}_{\alpha,\beta}(x)$ for all $\alpha_l + 1 \leq \alpha \leq \alpha_u$ and $\beta_l + 1 \leq \beta \leq \beta_u$ (see Figure 4 as an example). Therefore, we can compute $\tilde{f}_{\alpha,\beta}(x)$ for all necessary α and β in $O(\tau a^2 n)$ time. The (Figure 4) computation of $\tilde{b}_{\alpha,\beta}(x)$ is similar and is omitted.

In summary, we could evaluate all solutions in the limited shift neighborhood where rectangle i is shifted in $O(\tau n \log n)$ time, i.e., amortized computational time to evaluate one coded solution becomes $O(\tau \log n)$. We call this algorithm Evaluate-Limited-Shift-Moves (ELSM). Note that we can evaluate solutions in the limited two-shifts neighborhood efficiently with similar ideas. The amortized computational time to evaluate one solution in the limited two-shifts neighborhood where rectangles i and j will be shifted is also $O(\tau \log n)$. This becomes $O(\log n)$ for special cases such as area minimization, strip packing and so on, since $\tau = O(1)$ holds for such cases as mentioned before.

6 Computational experiments

We conducted thorough computational experiments to evaluate the proposed algorithms. The algorithms were coded in the C language and run on a handmade PC (Intel Pentium III 1 GHz, 1 GB of memory). We used instances of three problems, whose data are obtainable electronically from `http://www-or.amp.i.kyoto-u.ac.jp/~imahori/packing/`.

6.1 Comparison with our previous algorithms

We conducted two kinds of computational experiments to compare the proposed algorithms with our previous algorithms [8].

6.1.1 Decoding algorithms

We examine the performance of four decoding algorithms: (1) an $O(\tau n^2)$ time naive algorithm in Section 5.1 (denoted NAIVE), (2) algorithm CMPF [8] whose time complexity is $O(\tau n \log n)$, (3) algorithm ESM in Section 5.2 whose time complexity is $O(\tau)$, and (4) algorithm ELSM in Section 5.2 whose time complexity is $O(\tau \log n)$. All time complexities are for one decoded solution. We tested instances of several sizes and several cost functions. Results are shown in Table 1. The figures in the table show the computational time to evaluate one given coded solution. Column "n" shows the size of instances and column "cost type" shows the type of cost functions. The type "special" means that all rectangles have some special cost functions and then τ is $O(1)$, and "general" means that rectangles have general cost functions, each with a few segments, and then τ is $O(n)$. (Table 1)

From the table, we can observe that proposed algorithms ESM and ELSM are much more efficient than previous algorithms even for small instances such as $n = 49$.

6.1.2 Local search and metaheuristic algorithms

We compare the performance of the iterated local search algorithm (denoted ILS), in which proposed decoding algorithms are incorporated, with our previous ILS algorithm [8] (denoted ILS-pre). Note that ILS-pre is based on a decoding algorithm CBP (Compute-Better-Packing), which was observed to be superior to another ILS algorithm based on CMPF if the same neighborhoods are used [8]. ILS-pre uses standard neighborhoods (swap, shift and change mode) and swap* neighborhood, where swap* is a subset of the two-shifts neighborhood. ILS uses the combination of limited shift, limited two-shifts and change mode neighborhoods, and evaluates solutions with ELSM. We tested instances of the area minimization problem (see the next section

for details) with up to 500 rectangles. We terminate the search when a prespecified computational time is reached even if we have not found any locally optimal solution, and output the best solution obtained during the search. Results are shown in Table 2. Column "value" shows (Table 2)
the average of the following ratio,

$$100 \times \frac{\text{(the lower bound of the objective function)}}{\text{(the objective value of the output optimum)}},$$

for ten trials (i.e., the larger the better). Column "time" shows the time limit (in seconds) for one instance. These notations are also used in Tables 3 and 5. Column "lopt" shows the average number of obtained locally optimal solutions (i.e., "0.0" means each run is forced to stop by the time limit before any locally optimal solution is found). Column "move" shows the average number of moves from a solution to a better solution. From the table, we can observe that ILS is superior to ILS-pre in quality for all instances. The numbers of obtained locally optimal solutions and moves by the proposed ILS algorithm are not so large, even though the time to evaluate a solution is much smaller than ILS-pre, since the size of the neighborhood we use is larger than the previous one.

6.2 Comparison with other existing algorithms on the area minimization

6.2.1 Definition of the area minimization problem

We are given a set of n rectangles $I = \{1, 2, \ldots, n\}$, where each rectangle $i \in I$ has a width w_i and a height h_i. The rotations of 90° are allowed and the objective is to minimize the area of the rectangle bin that contains all given rectangles. This problem was considered in many papers including [8, 14, 15, 17, 18, 19].

In our formulation as RPGSC instances, each rectangle has two modes of orientations: (1) the original orientation and (2) the orientation after 90° rotation. For each $i = 1, 2, \ldots, n$, we set

$$\begin{aligned} w_i^{(1)} &= w_i, \qquad h_i^{(1)} = h_i, \\ w_i^{(2)} &= h_i, \qquad h_i^{(2)} = w_i. \end{aligned} \tag{20}$$

For $i = 1, 2, \ldots, n$ and $k = 1, 2$, we use

$$p_i^{(k)}(x) = \begin{cases} +\infty & (x < 0) \\ x + w_i^{(k)} & (x \geq 0), \end{cases} \qquad q_i^{(k)}(y) = \begin{cases} +\infty & (y < 0) \\ y + h_i^{(k)} & (y \geq 0). \end{cases}$$

The objective of the resulting RPGSC instance is to minimize $p_{\max}(\pi) \cdot q_{\max}(\pi)$ (i.e., the area of the rectangle that covers all given rectangles).

We use five instances of this problem: ami49, rp100, pcb146, rp200 and pcb500, with 49, 100, 146, 200 and 500 rectangles, respectively. Since the optimal solutions of these instances are unknown, we use the sum of the areas of n rectangles as a lower bound of the objective function.

6.2.2 Computational results

We compared our algorithm ILS with two existing algorithms for the area minimizing problem: (1) A simulated annealing algorithm with the BSG (bounded sliceline grid) coding scheme by Nakatake et al. [15] (denoted SA-BSG) and (2) a simulated annealing algorithm with the sequence pair coding scheme by Murata et al. [14] (denoted SA-SP). Algorithms SA-BSG and SA-SP were coded in the C language and run on a PC (Intel Pentium III 910 MHz), where the speed of this CPU is similar to ours (slightly slower, see Appendix A for more detailed comparison). These algorithms are specially tailored to the rectangle packing problem of minimizing the area, and their results for rp200 are not available (denoted N.A.). All results are shown in Table 3. Notations of this table are the same as Table 2. From the table, we can observe that ILS is superior to SA-SP in both the solution quality and the computational time for all instances. We can also observe that ILS is superior to SA-BSG for almost all instances, and this tendency becomes clearer for larger instances. Note that our algorithm is designed to solve more general problems. Taking the generality of ILS into consideration, the performance of ILS appears to be quite satisfactory. (Table 3)

6.3 Comparison with other existing algorithms on the strip packing problem

6.3.1 Definition of the strip packing problem

We are given a set of n rectangles $I = \{1, 2, \ldots, n\}$ and the width W of a large bin (stock roll), where each rectangle $i \in I$ has a width w_i and a height h_i. The rotations of 90° are allowed and the objective is to minimize the height of the large bin that contains all given rectangles.

In our formulation as RPGSC instances, each rectangle has two modes of orientations. For $i = 1, 2, \ldots, n$ and $k = 1, 2$, we set the width and height as in (20), and use cost functions

$$p_i^{(k)}(x) = \begin{cases} +\infty & (x < 0) \\ 0 & (0 \le x \le W - w_i^{(k)}) \\ \alpha(x - W + w_i^{(k)}) + \beta & (x > W - w_i^{(k)}), \end{cases} \qquad q_i^{(k)}(y) = \begin{cases} +\infty & (y < 0) \\ y + h_i^{(k)} & (y \ge 0), \end{cases}$$

where α and β are nonnegative constants such that at least one of them is positive. We set $\alpha = 10$ and $\beta = 0$. The objective of the resulting RPGSC instance is to minimize $p_{\max}(\pi) + q_{\max}(\pi)$ (i.e., the height of the bin if $p_{\max}(\pi) = 0$). We use test instances given by Hopper and Turton [7].

There are seven different categories called C1, C2, ..., C7 with the number of rectangles ranging from 16 to 197, where each category has three instances. The optimal value is known for all categories (see Table 4). (Table 4)

6.3.2 Computational results

We compared our algorithm ILS with three existing heuristic algorithms for the strip packing problem: (1) A heuristic algorithm BLF-DW (bottom left fill with decreasing width) proposed by Chazelle [1] and implemented by Hopper and Turton [7] (denoted BLF), (2) a simulated annealing algorithm with BLF algorithm by Hopper and Turton [7] (denoted SA-BLF) and (3) an effective quasi-human based heuristic by Wu et al. [20] (denoted QH). The results of algorithms BLF and SA-BLF are taken from [7], where these algorithms were coded in the C++ language and run on a PC (Intel Pentium Pro 200 MHz, 65 MB memory). The results of algorithm QH are taken from [20], where algorithm QH was run on a SUN Sparc 20/71 (71 MHz SuperSparc CPU, 64 MB memory). Based on the benchmark results of SPECint from SPEC web page (`http://www.specbench.org/`), our CPU is about six times faster than Intel Pentium Pro 200 MHz and fourteen times faster than SuperSparc 71 MHz (see Appendix A for more detailed comparisons). Note that QH is not designed for solving the strip packing problem but for the rectangle packing problem with a bounding box. In order to solve the strip packing problem with QH, Wu et al. run their algorithm many times while increasing the size of bounding box by one unit per iteration (not only the height but the width of large bin may be increased) to find the minimum bounding box in which all rectangles can be packed. As they change W, the problem solved by QH is different from others.

Results are shown in Table 5 (see Appendix B for more detailed results of our ILS algorithm). Notations of this table are the same as Table 2, except for the number of trials; the results are the average of ten trials for SA-BLF and ILS, however, only one trial for BLF and QH, since these two algorithms are deterministic. Note that the result of QH for C7 is not available (denoted N.A.) and their computational time does not include the time of the trial iterations to find the minimum bounding box in which all rectangles can be packed with QH (i.e., the time is reported only for the "successful" iteration). From the table, we can observe that BLF is much faster than other algorithms ("< 0.1" in Table 5 means the computational time is less than 0.1 seconds), but the quality of solutions is a little worse than other algorithms. ILS is slightly superior to SA-BLF in terms of the solution quality with less computational time. It is not easy to compare ILS with QH, since the problem solved by QH is different from the strip packing problem. The (Table 5)

solution quality of QH is quite good, however, ILS seems very effective since it can deal with large instances such as $n = 196$ or more, and does not need any preliminary adjustments.

6.4 Comparison with other existing algorithms on the scheduling problem

6.4.1 Definition of the scheduling problem of large building blocks

This is a scheduling problem that arises in a factory producing large building blocks. The blocks produced in the factory are very large, and once the building block is placed in the factory, it cannot be moved until all processes on the building block are finished. Each building block i has a length l_i, a processing time t_i, a ready time s_i and a due date d_i. As the shape of the work space is long and narrow, building blocks can be regarded as the one-dimensional objects. Blocks must be placed without overlap. A schedule is determined by the place and the start time S_i of each block i. Let C_i be the completion time of the process for block i; i.e., $C_i = S_i + t_i$. Then the objective is to minimize the maximum absolute difference $\max\{0, s_i - S_i, C_i - d_i\}$.

This problem can also be formulated as RPGSC, in which each rectangle has only one mode. For $i = 1, 2, \ldots, n$, let

$$w_i^{(1)} = t_i, \qquad p_i^{(1)}(x) = \begin{cases} -x + s_i & (x < s_i) \\ 0 & (s_i \le x \le d_i - w_i^{(1)}) \\ x + w_i^{(1)} - d_i & (x > d_i - w_i^{(1)}), \end{cases}$$

$$h_i^{(1)} = l_i, \qquad q_i^{(1)}(y) = \begin{cases} +\infty & (y < 0) \\ 0 & (0 \le y \le H - h_i^{(1)}) \\ \alpha(y + h_i^{(1)} - H) + \beta & (y > H - h_i^{(1)}), \end{cases}$$

where H is the length of the factory and α, β are nonnegative constants (we set $\alpha = 10$ and $\beta = 0$). $p_i^{(1)}(x)$ represents the time window $[s_i, d_i - t_i]$ for the start time x of building block i, and $q_i^{(1)}(y)$ represents the work space constraint $0 \le y \le H - l_i$ for the location y of bottom edge of building block i. Given a packing π, $x_i(\pi)$ represents the start time and $y_i(\pi)$ represents the bottom edge of the position of block i. The objective is to minimize $p_{\max}(\pi) + q_{\max}(\pi)$ (i.e., maximum gap from time window if $q_{\max}(\pi) = 0$).

We use five test instances sp78, sp50-a, sp50-b, sp100-a and sp100-b, where sp78 is a test instance from a real world application with 78 building blocks, and sp50-a, sp50-b, sp100-a, sp100-b are random instances that have 50 and 100 building blocks, respectively. It is known that there is a schedule with objective value 0 for every instance (i.e., no violation of constraints with respect to start time, due date and work space for all building blocks).

6.4.2 Computational results

We compared the performance of our algorithm with a tabu search algorithm developed for the resource constrained project scheduling problem by Nonobe and Ibaraki [16] (denoted TS) and our previous algorithm [8] (denoted ILS-pre). To solve the problem by TS, we reformulated the problem as a project scheduling problem in the way as described in [6]. Note that TS is not designed for solving the packing problem, but can handle more complicated scheduling problems with precedence constraints and other resources (e.g., manpower, machines and equipments) as well as work space.

We terminate the search either when an optimal solution (i.e., the objective value is 0) is obtained or when a prespecified computational time (we set the time limit to 3600 seconds) is reached. If a search stops after finding an optimum solution, it is called successful. Results are shown in Table 6. Column "ratio" shows (the number of successful trials) / (the number of trials). (Table 6) Column "time" shows the average computational time (in seconds) to find an optimal solution in successful trials. From the table, we can observe that ILS is superior to TS and ILS-pre for some instances such as sp50-a and sp50-b, but ILS-pre seems to be the best among the three algorithms. By preliminary experiments, an iterated local search algorithm, which is similar to ILS-pre but decoding algorithm CMPF is incorporated, is inferior to ILS-pre and ILS. We therefore conclude that decoding algorithm CBP works well for this scheduling problem. It is our future work to propose another improved decoding algorithm maintaining good characteristics of CBP and ELSM.

7 Conclusion

In this paper, we tackled the rectangle packing problem with general spatial costs, which contains various types of packing problems and scheduling problems as special cases. We adopted the sequence pair as the coding scheme, which is a pair of permutations of the given n rectangles, and proposed speed-up techniques to evaluate solutions in three types of neighborhoods for this problem. We realized $O(\tau)$ or $O(\tau \log n)$ amortized computational time to evaluate one coded solution, where τ is the space complexity of the minimum penalty function. It becomes $O(1)$ or $O(\log n)$ for important special cases such as area minimization, strip packing, two-dimensional 0-1 knapsack and so on.

We also report computational results for the rectangle packing problem (area minimization and strip packing) and a real-world scheduling problem. The computational results were quite

encouraging for the ILS algorithm proposed in this paper.

Acknowledgments

The authors are grateful to Takeaki Uno, National Institute of Informatics, for valuable comments. This research was partially supported by Scientific Grant-in-Aid by the Ministry of Education, Culture, Sports, Science and Technology of Japan, by Informatics Research Center for Development of Knowledge Society Infrastructure (COE program of the Ministry of Education, Culture, Sports, Science and Technology, Japan) and by the Telecommunications Advancement Foundation of Japan.

References

[1] B. Chazelle, The bottom-left bin-packing heuristic: an efficient implementation, IEEE Transactions on Computers 32 (1983) 697–707.

[2] K.A. Dowsland and W.B. Dowsland, Packing problems, European Journal of Operational Research 56 (1992) 2–14.

[3] H. Dyckhoff and U. Finke, Cutting and Packing in Production and Distribution, Physica Verlag, Heidelberg, 1992.

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

[5] P.C. Gilmore and R.E. Gomory, Multistage cutting stock problems of two and more dimensions, Operations Research 13 (1965) 94–119.

[6] S. Hartmann, Packing problem and project scheduling models: an integrating perspective, Journal of the Operational Research Society 51 (2000) 1083–1092.

[7] E. Hopper and B.C.H. Turton, An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, European Journal of Operational Research 128 (2001) 34–57.

[8] S. Imahori, M. Yagiura and T. Ibaraki, Local search algorithms for the rectangle packing problem with general spatial costs, Mathematical Programming 97 (2003) 543–569.

[9] D.S. Johnson, Local optimization and the traveling salesman problem, in: M.S. Peterson (Ed.), Automata, Languages and Programming, Lecture Notes in Computer Science 443, Springer, 1990, pp. 446–461.

[10] D. Liu and H. Teng, An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, European Journal of Operational Research 112 (1999) 413–420.

[11] A. Lodi, S. Martello and M. Monaci, Two-dimensional packing problems: a survey, European Journal of Operational Research 141 (2002) 241–252.

[12] A. Lodi, S. Martello and D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, Informs Journal on Computing 11 (1999) 345–357.

[13] H.R. Lourenço, O.C. Martin and T. Stützle, Iterated local search, in: G. Glover and G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, 2003, pp. 321–353.

[14] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, VLSI module placement based on rectangle-packing by the sequence-pair, IEEE Transactions on Computer Aided Design 15-12 (1996) 1518–1524.

[15] S. Nakatake, K. Fujiyoshi, H. Murata and Y. Kajitani, Module placement on BSG-structure and IC layout applications, Proceedings of International Conference on Computer Aided Design 1996, 484–491.

[16] K. Nonobe and T. Ibaraki, Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: C.C. Ribeiro and P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, 2001, pp. 557–588.

[17] T. Takahashi, An algorithm for finding a maximum-weight decreasing sequence in a permutation, motivated by rectangle packing problem (in Japanese), Technical Report of IEICE VLD96-30 (1996) 31–35.

[18] X. Tang, R. Tian and D.F. Wong, Fast evaluation of sequence pair in block placement by longest common subsequence computation, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 20 (2001) 1406–1413.

[19] X. Tang and D.F. Wong, Fast-SP: a fast algorithm for block placement based on sequence pair, Proceedings of Asia and South Pacific Design Automation Conference 2001, 521–526.

[20] Y.L. Wu, W. Huang, S. Lau, C.K. Wong and G.H. Young, An effective quasi-human based heuristic for solving the rectangle packing problem, European Journal of Operational Research 141 (2002) 341–358.

Appendix A Detailed comparisons of various CPUs

We compare various CPUs in order to compare the performance of various algorithms, ILS-pre, SA-BSG, SA-SP, BLF, SA-BLF, QH, TS and our ILS, as fairly as possible. Table 7 shows the benchmark results of SPECint2000 and SPECint95 from SPEC web page for related CPUs (`http://www.specbench.org/`). In case the data of a related CPU is not available, the data of similar CPUs are shown.

Based on these data, we compute rough estimates of the speed of the related CPUs and show them in Table 7, where the speed of the Pentium III 1 GHz (CPU for ILS-pre, TS and ILS) is normalized as one, and a larger value means the speed is faster. (Table 7)

Appendix B Detailed results for the strip packing problem

The details of the results on the Hopper and Turton's instances are explained. There are seven different sized categories called C1, C2, ..., C7 and each category has three instances called P1, P2 and P3. We ran our algorithm ILS ten times for each instance, and results are shown in Table 8. Column "average" (resp., "best" and "worst") shows the average (resp., best and worst) of the following ratio,

$$100 \times \frac{\text{(the lower bound of the objective function)}}{\text{(the objective value of the output optimum)}},$$

for ten trials. We terminated the search with time limit (specified in column "time"), and column "lopt" shows the average number of obtained locally optimal solutions. (Table 8)

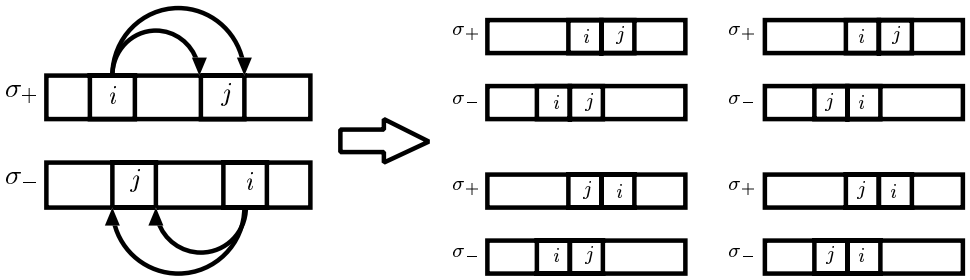
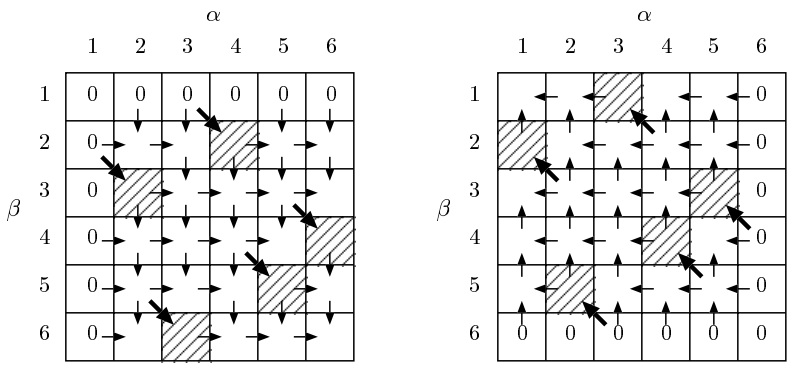


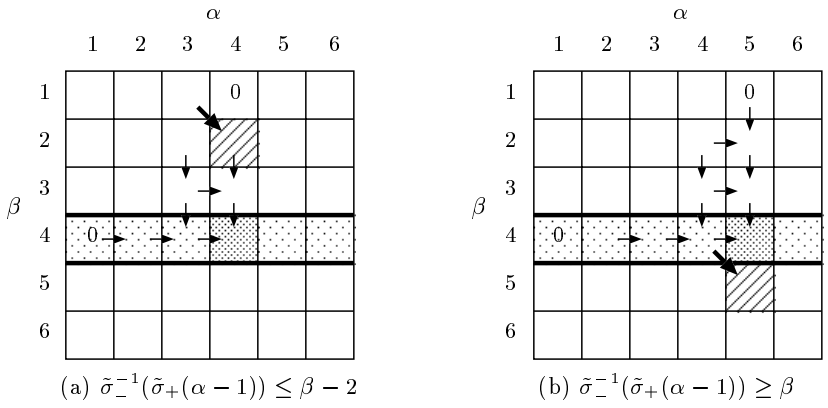
Figure 1: An example of the limited double shift operation



(a) computation of $\tilde{f}_{\alpha,\beta}(x)$ (b) computation of $\tilde{b}_{\alpha,\beta}(x)$

$(\tilde{\sigma}_+ : 1, 2, 3, 4, 5, \quad \tilde{\sigma}_- : 3, 1, 5, 4, 2.)$

Figure 2: An example of computing $\tilde{f}_{\alpha,\beta}(x)$ and $\tilde{b}_{\alpha,\beta}(x)$



(a) $\tilde{\sigma}_-^{-1}(\tilde{\sigma}_+(\alpha-1)) \leq \beta-2$ (b) $\tilde{\sigma}_-^{-1}(\tilde{\sigma}_+(\alpha-1)) \geq \beta$

Figure 3: An example of evaluating solutions obtainable by a single shift operation

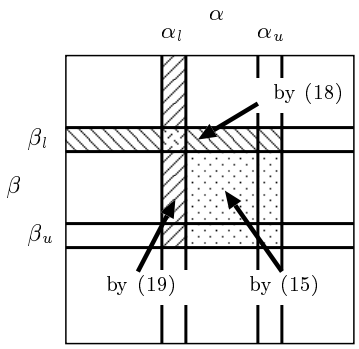


Figure 4: An example of evaluating solutions obtainable by a near-place double shift operation

Table 1: Computational time of the decoding algorithms in seconds

n	cost type	NAIVE	CMPF	ESM	ELSM
49	special	1.01×10^{-4}	3.80×10^{-5}	2.85×10^{-7}	7.18×10^{-7}
100	special	4.42×10^{-4}	9.12×10^{-5}	4.26×10^{-7}	6.91×10^{-7}
146	special	9.12×10^{-4}	1.38×10^{-4}	4.96×10^{-7}	8.47×10^{-7}
200	special	1.93×10^{-3}	1.92×10^{-4}	5.55×10^{-7}	7.84×10^{-7}
500	special	1.13×10^{-2}	7.30×10^{-4}	5.39×10^{-7}	1.20×10^{-6}
49	general	9.06×10^{-4}	3.70×10^{-4}	7.66×10^{-6}	9.81×10^{-6}
100	general	4.22×10^{-3}	8.66×10^{-4}	7.44×10^{-6}	1.14×10^{-5}
146	general	8.48×10^{-3}	1.90×10^{-3}	7.50×10^{-6}	1.42×10^{-5}
200	general	2.06×10^{-2}	2.54×10^{-3}	7.49×10^{-6}	1.37×10^{-5}
500	general	1.19×10^{-1}	1.23×10^{-2}	8.44×10^{-6}	1.90×10^{-5}

Table 2: Comparison with our previous algorithm for the area minimization problem

	ILS-pre				ILS			
instance	value	time	lopt	move	value	time	lopt	move
ami49	96.30	100.0	52.7	2987.5	97.37	100.0	61.9	5064.8
rp100	95.76	200.0	8.1	1176.5	96.78	200.0	15.7	2526.4
pcb146	95.63	300.0	11.8	1305.8	96.71	300.0	18.5	2347.2
rp200	95.67	400.0	0.3	495.3	96.30	400.0	3.3	1249.4
pcb500	92.27	1000.0	0.0	457.8	96.28	1000.0	0.4	800.0

Table 3: Comparison with other methods for the area minimization problem

	SA-BSG		SA-SP		ILS	
instance	value	time	value	time	value	time
ami49	97.10	69.0	96.29	176.0	97.29	65.0
rp100	97.08	68.2	88.54	248.7	96.40	65.0
pcb146	94.87	100.2	94.42	678.7	96.16	100.0
rp200	N.A.	N.A.	N.A.	N.A.	95.75	150.0
pcb500	94.10	334.6	90.82	7802.9	95.47	300.0

Table 4: Test instances of the strip packing problem

category	#rectangles n	bin width W	optimal height
C1	16 or 17	20	20
C2	25	40	15
C3	28 or 29	60	30
C4	49	60	60
C5	73	60	90
C6	97	80	120
C7	196 or 197	160	240

Table 5: Comparison with other methods for the strip packing problem

	BLF		SA-BLF		QH†		ILS	
category	value	time	value	time	value	time	value	time
C1	89	< 0.1	96	42	95.24	1.63	97.56	10.0
C2	84	< 0.1	94	144	97.92	6.19	93.75	15.0
C3	88	< 0.1	95	240	96.77	17.17	96.67	20.0
C4	95	< 0.1	97	1980	97.29	221.3	96.88	150.0
C5	95	< 0.1	97	6900	98.36	905.3	97.02	500.0
C6	95	< 0.1	97	22920	98.36	4581	96.85	1000.0
C7	95	0.64	96	250860	N.A.	N.A.	96.55	3600.0

†QH may increase the width of large bin not only the height.

Table 6: Comparison with other methods for the scheduling problem

	TS		ILS-pre		ILS	
instance	ratio	time	ratio	time	ratio	time
sp50-a	10/10	394.8	10/10	44.98	10/10	41.63
sp50-b	1/10	2730.6	10/10	125.0	10/10	47.60
sp78	10/10	427.2	10/10	405.6	7/10	1680.4
sp100-a	10/10	1851.9	10/10	362.6	3/10	1954.5
sp100-b	9/10	1230.1	10/10	47.03	10/10	574.9

Table 7: Speed of various CPUs

CPU	Algorithms	SPECint2000	SPECint95	estimated speed
Pentium III 1 GHz	ILS, ILS-pre, TS	418		1
Pentium III 933 MHz		398		0.97
Pentium III 910 MHz	SA-BSG, SA-SP			0.95
Pentium III 866 MHz		380		0.91
Pentium III 800 MHz		361	38.7	0.86
Pentium Pro 200 MHz	BLF, SA-BLF		8.08	0.18
Sun Sparc 20/71	QH		3.11	0.07

Table 8: Detailed results by ILS for Hopper and Turton's instances

instance	n	average	best	worst	time	lopt
C1-P1	16	99.50	100.0	95.24	10.0	411.0
C1-P2	17	95.69	100.0	95.24	10.0	297.6
C1-P3	16	97.56	100.0	95.24	10.0	322.4
C2-P1	25	93.75	93.75	93.75	15.0	155.5
C2-P2	25	93.75	93.75	93.75	15.0	167.6
C2-P3	25	93.75	93.75	93.75	15.0	195.9
C3-P1	28	96.77	96.77	96.77	20.0	194.3
C3-P2	29	96.77	96.77	96.77	20.0	220.3
C3-P3	28	96.46	96.77	93.75	20.0	224.6
C4-P1	49	96.93	98.36	96.77	150.0	233.0
C4-P2	49	96.77	96.77	96.77	150.0	268.2
C4-P3	49	96.93	98.36	96.77	150.0	265.6
C5-P1	73	96.98	97.83	96.77	500.0	199.8
C5-P2	73	96.77	97.83	95.74	500.0	221.8
C5-P3	73	97.30	98.90	96.77	500.0	195.7
C6-P1	97	96.93	97.56	96.77	1000.0	158.1
C6-P2	97	96.46	96.77	96.00	1000.0	200.3
C6-P3	97	97.16	97.56	96.77	1000.0	181.6
C7-P1	196	96.42	96.77	96.00	3600.0	54.5
C7-P2	197	96.70	97.17	96.00	3600.0	62.4
C7-P3	196	96.54	96.77	96.00	3600.0	64.7