MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Solving the Irregular Strip Packing Problem via Guided Local Search for Overlap Minimization

Shunji UMETANI, Mutsunori YAGIURA,
Shinji IMAHORI, Takashi IMAMICHI,
Koji NONOBE, Toshihide IBARAKI

METR 2008–22 May 2008

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.keisu.t.u-tokyo.ac.jp/research/techrep/index.html

Solving the irregular strip packing problem via guided local search for overlap minimization

Shunji Umetani* Mutsunori Yagiura† Shinji Imahori‡ Takashi Imamichi§
Koji Nonobe¶ Toshihide Ibaraki‖

May 6, 2008

Abstract

The irregular strip packing problem (ISP) requires a given set of non-convex polygons to be placed without overlap within a rectangular container having a given width and a variable length which is to be minimized. As a core subproblem to solve ISP, we consider an overlap minimization problem (OMP) whose objective is to place all polygons into a container with given width and length so that the total amount of overlap between polygons is made as small as possible. We propose to use directional penetration depths to measure the amount of overlap between a pair of polygons, and present an efficient algorithm to find a position with the minimum overlap for each polygon when it is translated in a specified direction. Based on this, we develop a local search algorithm for OMP that translates a polygon in horizontal and vertical directions alternately. Then we incorporate it in our algorithm for OMP, which is a variant of the guided local search algorithm. Computational results show that our algorithm improves the best known values of some well known benchmark instances.

Keywords: Cutting, Packing, Irregular strip packing, Nesting, Guided local search.

1 Introduction

The *irregular strip packing problem* (ISP), or often called the *nesting problem*, is one of the cutting and packing problems, which deals with polygons (or arbitrary shapes) that can be neither rectangular nor convex. Given a set of polygons and a rectangular container called a *strip* with a constant width and a variable length, this problem requires a feasible placement of the polygons into the container such that its length is minimized. A placement is feasible if no two polygons overlap with each other and no polygon protrudes from the container (Figure 1). This problem has three variations depending on rotations of polygons: (1) rotations of any angles are allowed, (2) rotations of finite number of angles are allowed, and (3) no rotation is allowed. (Note that case (3) is a special case of (2) in which the number of given orientations of each polygon is one.) Among them, we deal with case (2) in this paper. In many practical applications such as textile industry, rotations are usually restricted to 0 or 180 degrees because textiles have the grain and may have a drawing pattern.

The ISP is much harder than the rectangle strip packing problem because the intersection test between polygons is considerably more complex. However, due to rapid innovation in computing

*Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Japan
†Graduate School of Information Science, Nagoya University, Nagoya 464-8603, Japan
‡Graduate School of Information Science and Technology, University of Tokyo, Tokyo 113-8656, Japan
§Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
¶Faculty of Engineering and Design, Hosei University, Tokyo 102-8160, Japan
‖School of Science and Technology, Kwansei Gakuin University, Sanda 669-1337, Japan

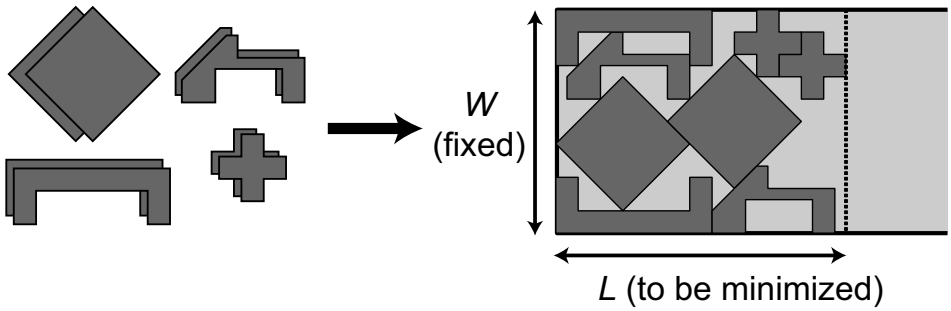

Figure 1: An instance of the irregular strip packing problem and a feasible solution.

power and in theory of computational geometry, many approaches to ISP have been developed in recent years.

A standard approach for designing heuristics is to specify a construction method that places polygons one by one into the container in a given sequence. Albano and Sapuppo (1980) proposed a construction algorithm of this type, which places each polygon to the left-most feasible position on the right envelope of the set of already placed polygons, breaking ties by preferring the bottom-most one. Błażewicz et al. (1993) proposed an extension of this algorithm by allowing to place a polygon into a hole surrounded by placed polygons, in addition to the right envelope of the set of placed polygons. Dowsland et al. (2002) and Gomes and Oliveira (2002) independently developed heuristic algorithms that place each polygon at the left-most feasible position in the whole container (i.e., candidate positions are not restricted to those on the right envelope of the set of already placed polygons), breaking ties by preferring the bottom-most one. In these algorithms, they used a geometric technique called the *no-fit polygon* to find such positions efficiently. Gomes and Oliveira (2002) then applied local search to find a good sequence of given polygons for their construction algorithm. Burke et al. (2006) considered a generalized problem that can deal with irregular shapes having circular edges. They proposed an efficient construction algorithm that searches for the left-most feasible position of each shape approximately by restricting the search on vertical lines with sufficiently small gaps between them, and they incorporated it with hill climbing and tabu search algorithms to find a good sequence of given objects.

Another heuristic approach is to resort to an improvement method that relocates polygons by solving one of the following subproblems: (1) the *overlap minimization problem*, (2) the *compaction problem*, and (3) the *separation problem*. The overlap minimization problem (OMP) requires a placement (not necessarily feasible) of given polygons to minimize the overlap penalty for all pairs of polygons under the constraint that they are placed in the container with given width and length. The compaction problem relocates polygons from a given feasible placement so as to minimize the container length, and the separation problem relocates polygons from a given infeasible placement so as to make it feasible while minimizing the total amount of polygon translations. Li and Milenkovic (1995) developed heuristic algorithms to solve the compaction and separation problems based on linear programming (LP). Bennell et al. (2001) and Gomes and Oliveira (2006) combined tabu search and simulated annealing algorithms with LP based compaction and separation algorithms, respectively.

Egeblad et al. (2007) proposed a heuristic algorithm for ISP via OMP, where they used the intersection area for each pair of polygons as the overlap penalty. They developed a guided local search algorithm for OMP, in which the neighborhood consists of placements obtainable by any horizontal or vertical translation of a polygon from its current position. Imamichi et al. (2007) proposed another OMP algorithm by using a measure of overlap amount called the *penetration depth*, which is the minimum translational distance to separate a given pair of polygons, and they incorporated a nonlinear programming technique into their iterated local search algorithm.

Algorithms for the compaction, separation and overlap minimization problems are surveyed in Ibaraki et al. (2008).

In this paper, we first propose a new OMP algorithm which is based on another measure of overlap amount called the *directional penetration depth*, which is the minimum translational distance in a specified direction to separate a given pair of polygons. Exploiting characteristics of this penalty function, we develop an efficient algorithm to find a position with the minimum overlap penalty for each polygon when it is translated in a specified direction. Based on this, we devise a local search algorithm for OMP that translates a polygon in horizontal and vertical directions alternately until no better position is found in either direction. This is incorporated into a variant of the guided local search algorithm for OMP. Using this as a main component, we propose a heuristic algorithm for ISP, which we call the *fast iterative translation search* (FITS).

This paper is organized as follows. We first formulate ISP and OMP in Section 2. We then explain the intersection test and the no-fit polygon in Section 3. We illustrate the outline of our algorithm FITS for ISP in Section 4. We explain our construction algorithm to generate an initial placement for ISP in Section 5, and our guided local search algorithm for OMP in Section 6. Finally, we report computational results in Section 7 and make concluding remarks in Section 8.

2 Formulation

2.1 Irregular strip packing problem

We are given a list of polygons $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ with a list of their possible orientations $\mathcal{O} = (O_1, O_2, \ldots, O_n)$, where O_i $(1 \leq i \leq n)$ is the set of all orientations which a polygon P_i can be rotated (i.e., P_i can be rotated by o degrees for all $o \in O_i$), and a rectangular container $C = C(W, L)$ with a width W and a length L, where W is a nonnegative constant and L is a nonnegative variable. We denote a polygon P_i rotated by $o \in O_i$ degree by $P_i(o)$, which may be written as P_i for simplicity when its orientation is not specified or clear from the context. We describe a position of a polygon P_i by a coordinate $\boldsymbol{v}_i = (x_i, y_i)$ of its *reference point*, which is a point of P_i (e.g., a vertex or the center of gravity of P_i). For convenience, we regard all polygons P_i and the container C as closed regions. To be more precise, we regard P_i or C as the set of all interior points and those points on the boundary, when its reference point is put at the origin $(0, 0)$. For a polygon S, let $\mathrm{int}(S)$ be the interior of S, $\mathrm{cl}(S)$ be the closure of S, ∂S be the boundary of S and $\bar{S}$ be the complement of S. For each polygon P_i, we assume without loss of generality that zero degree is included in the set of orientations O_i, and its width

$$w_i(o) = \max\{y \mid (x, y) \in P_i(o)\} - \min\{y \mid (x, y) \in P_i(o)\} \tag{1}$$

satisfies $w_i(o) \leq W$ for all orientations $o \in O_i$. We describe a polygon P_i placed at $\boldsymbol{v}_i$ by the Minkowski sum

$$P_i \oplus \boldsymbol{v}_i = \{\boldsymbol{p} + \boldsymbol{v}_i \mid \boldsymbol{p} \in P_i\}. \tag{2}$$

Then the *irregular strip packing problem* (ISP) is formally described as follows:

$$\begin{array}{llll} \text{(ISP)} & \text{minimize} & L & \\ & \text{subject to} & \mathrm{int}(P_i(o_i) \oplus \boldsymbol{v}_i) \cap (P_j(o_j) \oplus \boldsymbol{v}_j) = \emptyset, & (1 \leq i < j \leq n), \\ & & (P_i(o_i) \oplus \boldsymbol{v}_i) \subseteq C(W, L), & (1 \leq i \leq n), \\ & & L \geq 0, & \\ & & o_i \in O_i, & (1 \leq i \leq n), \\ & & \boldsymbol{v}_i \in \mathbb{R}^2, & (1 \leq i \leq n). \end{array} \tag{3}$$

We describe a solution of ISP by lists of positions $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$ and orientations $\boldsymbol{o} = (o_1, o_2, \ldots, o_n)$ of all polygons P_i $(1 \leq i \leq n)$. Note that a solution $(\boldsymbol{v}, \boldsymbol{o})$ uniquely determines a placement of the polygons. The length L of the container C is determined by

$$L(\boldsymbol{v}, \boldsymbol{o}) = \max\{x \mid (x, y) \in (P_i(o_i) \oplus \boldsymbol{v}_i), 1 \leq i \leq n\} - \min\{x \mid (x, y) \in (P_i(o_i) \oplus \boldsymbol{v}_i), 1 \leq i \leq n\} \tag{4}$$

and $(P_i(o_i) \oplus \boldsymbol{v}_i) \subseteq C(W, L(\boldsymbol{v}, \boldsymbol{o}))$ holds for all polygons P_i $(1 \leq i \leq n)$ if and only if

$$W \geq \max\{y \mid (x, y) \in (P_i(o_i) \oplus \boldsymbol{v}_i), 1 \leq i \leq n\} - \min\{y \mid (x, y) \in (P_i(o_i) \oplus \boldsymbol{v}_i), 1 \leq i \leq n\}. \quad (5)$$

2.2 Overlap minimization problem

We use the *overlap minimization problem* (OMP) as a subproblem of ISP to find a feasible placement $(\boldsymbol{v}, \boldsymbol{o})$ of given polygons for the container C with a given length L_{UB}. In this problem, a solution may have a number of overlapping polygons, and the total amount of overlap is penalized in such a way that a solution with no penalty gives a feasible placement. Let $f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j)$ be a function that measures the overlap amount of a pair of polygons $P_i(o_i)$ and $P_j(o_j)$ placed at $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, respectively. The objective of OMP is to find a solution $(\boldsymbol{v}, \boldsymbol{o})$ that minimizes the total amount of the overlap penalty $F(\boldsymbol{v}, \boldsymbol{o}) = \sum_{1 \leq i < j \leq n} f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j)$ under the constraint that all polygons P_i $(1 \leq i \leq n)$ are placed inside the container C with given width W and length L_{UB}. The problem is formally described as follows:

$$\begin{array}{lllll} (\mathrm{OMP}(L_{\mathrm{UB}})) & \text{minimize} & F(\boldsymbol{v}, \boldsymbol{o}) = \displaystyle\sum_{1 \leq i < j \leq n} f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j) & & \\ & \text{subject to} & (P_i(o_i) \oplus \boldsymbol{v}_i) \subseteq C(W, L_{\mathrm{UB}}), & (1 \leq i \leq n), & (6) \\ & & o_i \in O_i, & (1 \leq i \leq n), & \\ & & \boldsymbol{v}_i \in \mathbb{R}^2, & (1 \leq i \leq n). & \end{array}$$

In this paper, we use the *directional penetration depth* to define the overlap penalty function $f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j)$ of a pair of polygons P_i and P_j. The *penetration depth* (also known as the *intersection depth*) is an important measure used in robotics, computer vision and so on (Agarwal et al., 2000; Dobkin et al., 1993; Kim et al., 2004). The penetration depth $\delta(P_i(o_i) \oplus \boldsymbol{v}_i, P_j(o_j) \oplus \boldsymbol{v}_j)$ of a pair of overlapping polygons $P_i(o_i)$ and $P_j(o_j)$ placed at $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ is defined as the minimum translational distance to separate them. If they do not overlap, then their penetration depth is zero. The formal definition of the penetration depth is given by

$$\delta(P_i(o_i) \oplus \boldsymbol{v}_i, P_j(o_j) \oplus \boldsymbol{v}_j) = \min\{||\boldsymbol{u}|| \mid \mathrm{int}(P_i(o_i) \oplus \boldsymbol{v}_i) \cap (P_j(o_j) \oplus \boldsymbol{v}_j \oplus \boldsymbol{u}) = \emptyset, \boldsymbol{u} \in \mathbb{R}^2\}, \quad (7)$$

where $||\cdot||$ denotes the Euclidean norm.

The directional penetration depth $\rho(P_i(o_i) \oplus \boldsymbol{v}_i, P_j(o_j) \oplus \boldsymbol{v}_j, \boldsymbol{d})$ of a pair of overlapping polygons $P_i(o_i)$ and $P_j(o_j)$ placed at $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ is defined as the minimum translational distance in a given direction $\boldsymbol{d} = (d_x, d_y)$ ($||\boldsymbol{d}|| = 1$, $\boldsymbol{d} \in \mathbb{R}^2$) to separate them (Dobkin et al., 1993). If they do not overlap, then their directional penetration depth is zero. The formal definition of the directional penetration depth is given by

$$\rho(P_i(o_i) \oplus \boldsymbol{v}_i, P_j(o_j) \oplus \boldsymbol{v}_j, \boldsymbol{d}) = \min\{|t| \mid \mathrm{int}(P_i(o_i) \oplus \boldsymbol{v}_i) \cap (P_j(o_j) \oplus \boldsymbol{v}_j \oplus t\boldsymbol{d}) = \emptyset, t \in \mathbb{R}\}. \quad (8)$$

In this paper, we define the overlap penalty $f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j)$ for a pair of polygons $P_i(o_i)$ and $P_j(o_j)$ placed at $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ by

$$f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j) = \min\{\rho(P_i(o_i) \oplus \boldsymbol{v}_i, P_j(o_j) \oplus \boldsymbol{v}_j, \boldsymbol{d}) \mid \boldsymbol{d} \in \{\boldsymbol{e}_x, \boldsymbol{e}_y\}\}, \quad (9)$$

where $\boldsymbol{e}_x = (1, 0)$ and $\boldsymbol{e}_y = (0, 1)$; i.e., they are unit vectors of the horizontal and vertical directions, respectively.

Egeblad et al. (2007) used as the overlap penalty the intersection area for each pair of polygons P_i and P_j. Imamichi et al. (2007) considered another formulation of OMP that allows in addition to overlap of polygons the protrusion of polygons from the container. For this problem, they used the penetration depth of each pair of polygons P_i and P_j as the overlap penalty, and the penetration depth of P_i to the outer region $\bar{C}$ as the protrusion penalty.

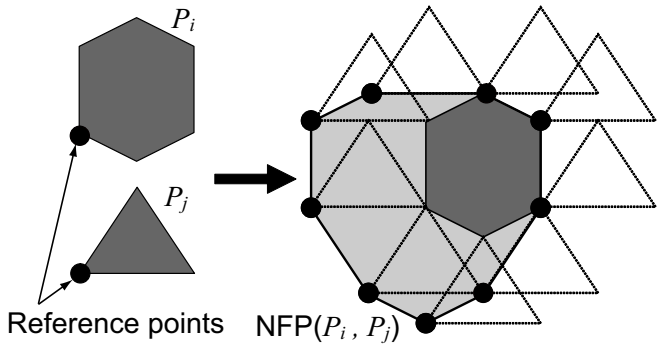

Figure 2: The no-fit polygon $\mathrm{NFP}(P_i, P_j)$ of two convex polygons P_i and P_j.

3 Intersection test and no-fit polygon

One of the geometric techniques commonly used for the intersection test is the *no-fit polygon* (NFP), which is often used in algorithms for ISP (Adamowicz and Albano, 1976; Albano and Sapuppo, 1980; Bennell et al., 2001; Gomes and Oliveira, 2002; Gomes and Oliveira, 2006; Imamichi et al., 2007; Oliveira et al., 2000). It is also used for other applications such as robot motion planning and image analysis, and has various names such as the Minkowski difference and the configuration-space obstacle.

The no-fit polygon $\mathrm{NFP}(P_i, P_j)$ of an ordered pair of polygons P_i and P_j is defined by

$$\mathrm{NFP}(P_i, P_j) = \mathrm{int}(P_i) \oplus (-P_j) = \{\boldsymbol{u} - \boldsymbol{w} \mid \boldsymbol{u} \in \mathrm{int}(P_i), \boldsymbol{w} \in P_j\}, \tag{10}$$

and has the following important properties:

- $P_j \oplus \boldsymbol{v}_j$ overlaps with $P_i \oplus \boldsymbol{v}_i$ if and only if $\boldsymbol{v}_j - \boldsymbol{v}_i \in \mathrm{NFP}(P_i, P_j)$.
- $P_j \oplus \boldsymbol{v}_j$ touches $P_i \oplus \boldsymbol{v}_i$ if and only if $\boldsymbol{v}_j - \boldsymbol{v}_i \in \partial\mathrm{NFP}(P_i, P_j)$.
- $P_i \oplus \boldsymbol{v}_i$ and $P_j \oplus \boldsymbol{v}_j$ are separated if and only if $\boldsymbol{v}_j - \boldsymbol{v}_i \notin \mathrm{cl}(\mathrm{NFP}(P_i, P_j))$,

where ∂S and $\mathrm{cl}(S)$ denote the boundary and the closure of a polygon S, respectively. Hence the problem of checking whether a pair of polygons overlap or not becomes an easier problem of checking whether a point is included in a polygon or not.

Let p_i and p_j be the number of edges of non-convex polygons P_i and P_j, respectively. Although it takes $O(p_i^2 p_j^2)$ time to compute $\mathrm{NFP}(P_i, P_j)$ in the worst case (de Berg et al., 2000), several practical algorithms to compute it have been proposed (Bennell and Dowsland, 2001; Burke et al., 2007; Dean et al., 2006). When polygons P_i and P_j are both convex, $\partial\mathrm{NFP}(P_i, P_j)$ can be computed by the following simple procedure: We first place the reference point of the polygon P_i at the origin $(0, 0)$, and slide the other polygon P_j around the polygon P_i having it keep touching with the polygon P_i. Then the trajectory of the reference point of the polygon P_j is $\partial\mathrm{NFP}(P_i, P_j)$. This procedure takes $O(p_i+p_j)$ time. Figure 2 shows an example of $\mathrm{NFP}(P_i, P_j)$ for two convex polygons. Even if polygons P_i and P_j are non-convex, they can often be decomposed into a small number (i.e., $O(1)$) of convex polygons in practical applications, and in such cases, $\mathrm{NFP}(P_i, P_j)$ can still be computed in $O(p_i + p_j)$ time.

We can also check whether a polygon P_i protrudes from the container C or not similarly by using

$$\mathrm{NFP}(\bar{C}, P_i) = \mathrm{int}(\bar{C}) \oplus (-P_i) = \{\boldsymbol{v} - \boldsymbol{w} \mid \boldsymbol{v} \in \mathbb{R}^2 \setminus C, \boldsymbol{w} \in P_i\}, \tag{11}$$

which is the complement of a rectangle whose boundary is the trajectory of the reference point of the polygon P_i when we slide it inside the container C having it keep touching with the container C.

To check whether $P_i \oplus \boldsymbol{v}_i$ protrudes from the container C or not, Gomes and Oliveira (2002; 2006) introduced the *inner-fit rectangle* $\mathrm{IFR}(C, P_i)$, which is equivalent to $\overline{\mathrm{NFP}(\bar{C}, P_i)}$ (i.e., polygon $P_i \oplus \boldsymbol{v}_i$ is contained in C if and only if $\boldsymbol{v}_i \in \mathrm{IFR}(C, P_i)$).

4 Outline of the entire algorithm for the irregular strip packing problem

In this section, we give the outline of our algorithm for ISP, which we call the *fast iterative translation search* (FITS). It first generates an initial feasible solution $(\boldsymbol{v}, \boldsymbol{o})$ by a construction algorithm, which we call Construct (it is explained in Section 5), and computes the container length L so that it contains all polygons P_i $(1 \le i \le n)$ while keeping both vertical sides touching some polygons. The algorithm then repeats the following procedures until the time limit is reached or the best feasible solution is proved to be optimal.

It first changes the container length L by shrinking or extending the right side of the container C, where the rates of shrinkage and extension are controlled by parameters r_{dec} and r_{inc}, respectively. If the current solution $(\boldsymbol{v}, \boldsymbol{o})$ is feasible, then it shrinks the container length L to $(1 - r_{\mathrm{dec}})L$ and relocates protruding polygons P_i at random positions in the container $C(W, L)$; otherwise it extends the container length L to $(1 + r_{\mathrm{inc}})L$. If there is at least one overlapping pair of polygons in the solution, then it tries to resolve overlap by a variant of the guided local search algorithm for the overlap minimization problem $\mathrm{OMP}(L)$, which we call MinimizeOverlap. The details of MinimizeOverlap is explained in Section 6. Algorithm FITS uses the above basic rule with slight modifications so that the current container length L satisfies $L_{\mathrm{LB}} \le L < L^*$ in the search, where L^* is the minimum container length of feasible placements obtained so far, and L_{LB} is a lower bound of L defined by

$$L_{\mathrm{LB}} = \max\left\{ \frac{\sum_{i=1}^{n} (\text{area of } P_i)}{W}, \max_{1 \le i \le n} l_i^{\min} \right\}, \tag{12}$$

where

$$l_i^{\min} = \min_{o \in O_i} l_i(o), \tag{13}$$

$$l_i(o) = \max\{x \mid (x, y) \in P_i(o)\} - \min\{x \mid (x, y) \in P_i(o)\}. \tag{14}$$

Figure 3 illustrates the behavior of algorithm FITS. The algorithm is formally described as follows.

Algorithm FITS

Input: A list of polygons $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ with a list of their possible orientations $\mathcal{O} = (O_1, O_2, \ldots, O_n)$ and a rectangular container C with a width W.

Output: The container length L, and lists of positions $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$ and orientations $\boldsymbol{o} = (o_1, o_2, \ldots, o_n)$ of all polygons P_i $(1 \le i \le n)$.

Step 1: Generate an initial solution $(\boldsymbol{v}, \boldsymbol{o})$ by Construct, and set L_{LB} by (12).

Step 2: If the current solution $(\boldsymbol{v}, \boldsymbol{o})$ is feasible, then set $L \leftarrow L(\boldsymbol{v}, \boldsymbol{o})$, $L^* \leftarrow L$ and $(\boldsymbol{v}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{v}, \boldsymbol{o})$; otherwise set $L \leftarrow (1 + r_{\mathrm{inc}})L$. If the time limit is reached or $L^* = L_{\mathrm{LB}}$ holds, then output L^* and $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ and halt.

Step 3: If $L \ge L^*$ holds, then set $L \leftarrow \max\{(1 - r_{\mathrm{dec}})L^*, L_{\mathrm{LB}}\}$ and $(\boldsymbol{v}, \boldsymbol{o}) \leftarrow (\boldsymbol{v}^*, \boldsymbol{o}^*)$, and relocate protruding polygons at random positions in the container $C(W, L)$.

Step 4: If the current solution $(\boldsymbol{v}, \boldsymbol{o})$ is infeasible, then compute a new solution $(\boldsymbol{v}', \boldsymbol{o}')$ by MinimizeOverlap$(L, \boldsymbol{v}, \boldsymbol{o})$ and set $(\boldsymbol{v}, \boldsymbol{o}) \leftarrow (\boldsymbol{v}', \boldsymbol{o}')$. Return to Step 2.

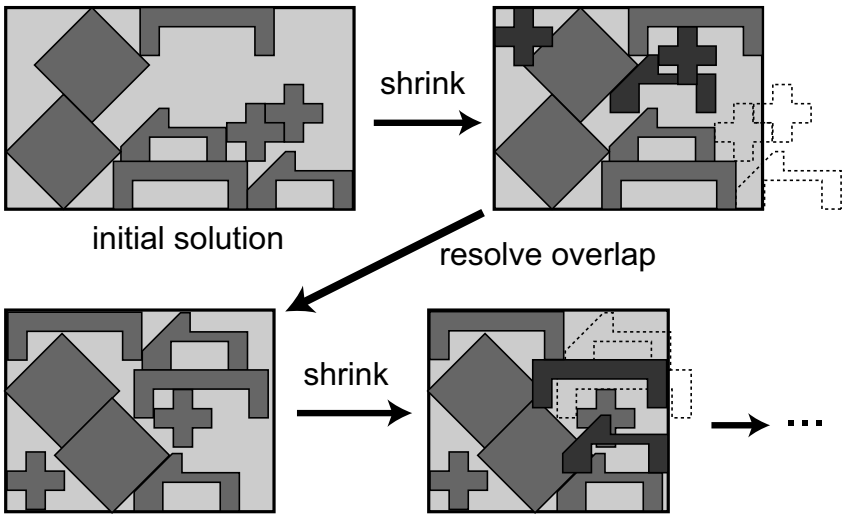

Figure 3: The behavior of algorithm FITS.

5 Construction algorithm

We present a construction algorithm called Construct that places polygons one by one into the container according to a specified order, where the position of each polygon is determined by iterative translations to the left and to the bottom. The algorithm first sets orientations $o_i \in O_i$ of all polygons P_i $(1 \leq i \leq n)$ so as to minimize their lengths $l_i(o_i)$ and sorts them in the descending order of their lengths. Let the container length L be sufficiently long (e.g., $L = \sum_{i=1}^{n} l_i(o_i)$). It then locates all polygons P_i $(1 \leq i \leq n)$ one by one into the container C according to the above order. It first places each polygon P_i at the top right corner of C and then translates it to the left and to the bottom alternately; each time translating P_i to the farthest feasible position while allowing to pass through the polygons already placed. Figure 4 shows an example of this procedure. The algorithm is formally described as follows.

Algorithm Construct

Input: A list of polygons $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ with a list of their possible orientations $\mathcal{O} = (O_1, O_2, \ldots, O_n)$ and a rectangular container C with a width W.

Output: A feasible solution $(\boldsymbol{v}, \boldsymbol{o})$.

Step 1: Set the orientation o_i of each polygon P_i $(1 \leq i \leq n)$ to the one that satisfies $l_i(o_i) = l_i^{\min}$ (see (13)), and sort the polygons in the descending order of their lengths $l_i^{\min}$. Let $\sigma(k)$ be the kth polygon in this order. Set $k \leftarrow 1$.

Step 2: If $k > n$ holds, then output $(\boldsymbol{v}, \boldsymbol{o})$ and halt; otherwise set $\boldsymbol{d} \leftarrow \boldsymbol{e}_x$ and initialize the position $\boldsymbol{v}_{\sigma(k)}$ of the polygon $P_{\sigma(k)}$ so that $P_{\sigma(k)}$ is placed at the top right corner of C.

Step 3: Find the feasible position $\boldsymbol{v}'_{\sigma(k)} = \boldsymbol{v}_{\sigma(k)} + t\boldsymbol{d}$ $(t \leq 0)$ of $P_{\sigma(k)}$ with the minimum t. If $\boldsymbol{v}'_{\sigma(k)} = \boldsymbol{v}_{\sigma(k)}$ holds, then set $k \leftarrow k + 1$ and return to Step 2; otherwise set $\boldsymbol{v}_{\sigma(k)} \leftarrow \boldsymbol{v}'_{\sigma(k)}$.

Step 4: If $\boldsymbol{d} = \boldsymbol{e}_x$ holds, then set $\boldsymbol{d} \leftarrow \boldsymbol{e}_y$; otherwise set $\boldsymbol{d} \leftarrow \boldsymbol{e}_x$. Return to Step 3.

We now give the details of the core part of Construct, where we assume that $\sigma(k) = k$ $(1 \leq k \leq n)$ for simplicity. Let polygons $P_1, P_2, \ldots, P_{k-1}$ be already placed in the container C, and let $\boldsymbol{v}_k$ be the current position of P_k to be translated. We now explain how the algorithm finds the

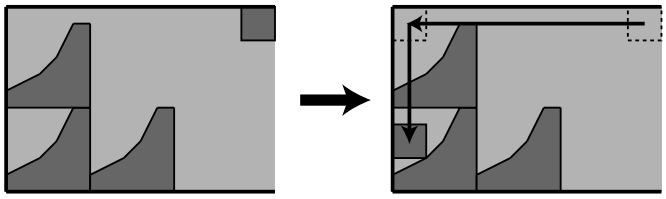

Figure 4: An example of alternating translations in algorithm CONSTRUCT.

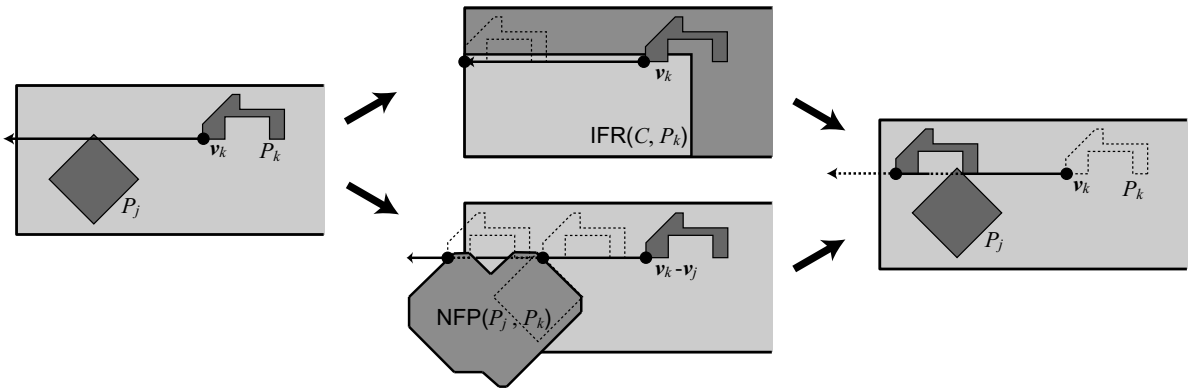

Figure 5: Finding the left-most feasible position of a polygon P_k.

new feasible position $\boldsymbol{v}'_k = \boldsymbol{v}_k + t\boldsymbol{d}$ $(t \leq 0)$ with the minimum t when it is translated to the left (i.e., $\boldsymbol{d} = \boldsymbol{e}_x$). The algorithm for $\boldsymbol{d} = \boldsymbol{e}_y$ is similar and is omitted. Among all positions on the half-line $\boldsymbol{v}_k + t\boldsymbol{d}$ $(t \leq 0)$, let

$$N_0^- = \{t \mid \boldsymbol{v}_k + t\boldsymbol{d} \in \mathrm{IFR}(C, P_k), t \leq 0\} \tag{15}$$

be the set of valid t (i.e., P_k is contained in C), and for $j = 1, 2, \ldots, k-1$, let

$$N_j^- = \{t \mid \boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j \notin \mathrm{NFP}(P_j, P_k), t \leq 0\} \tag{16}$$

be the set of t inducing no overlap with polygon P_j. Then, the left-most feasible position of the polygon P_k is given by the minimum $t \in \bigcap_{j=0}^{k-1} N_j^-$. The computation time for this is $O(q_k \log q_k)$ if it is implemented naively, where

$$q_k = \sum_{j=1}^{k-1} q_{kj} \tag{17}$$

and q_{kj} is the number of edges of no-fit polygon $\mathrm{NFP}(P_j, P_k)$. It can be reduced to $O(q_k \log k)$ time using a sorted list of intervals of t for each set N_j^- $(1 \leq j \leq k-1)$ that is computed in $O(q_{kj})$ time using trapezoidal partition of no-fit polygon $\mathrm{NFP}(P_j, P_k)$ (it is explained in Section 6.2). Figure 5 shows an example of the above procedure.

6 Local search algorithm for overlap minimization

6.1 Outline of the local search algorithm

The *local search* (LS) is a basic component of metaheuristics, which starts from an initial solution and repeatedly replaces the current solution with a better solution in its neighborhood until no better solution is found in the neighborhood.

We first explain the neighborhood of our LS for OMP. Let $(\boldsymbol{v}, \boldsymbol{o})$ be the current solution. Its neighborhood $\mathrm{NB}(\boldsymbol{v}, \boldsymbol{o})$ is defined as the set of solutions obtainable by setting a new orientation

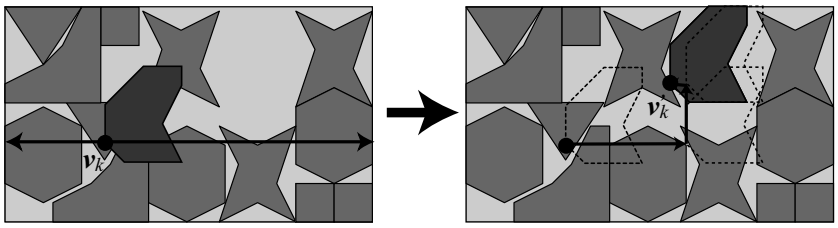

Figure 6: An example of the operation SearchNeighbor.

$o'_k \in O_k$ of each P_k $(1 \leq k \leq n)$ and applying an operation called SearchNeighbor to find a new position $\boldsymbol{v}'_k$.

Before presenting the local search algorithm, we explain how SearchNeighbor computes a solution. The quality of each solution $(\boldsymbol{v}, \boldsymbol{o})$ is measured by the following weighted overlap penalty function

$$\widetilde{F}(\boldsymbol{v}, \boldsymbol{o}) = \sum_{1 \leq i < j \leq n} w_{ij} \cdot f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j), \tag{18}$$

where $w_{ij} > 0$ are penalty weights (it is explained in Section 6.3 how to determine w_{ij}). For a polygon $P_k(o'_k)$, SearchNeighbor finds a new position $\boldsymbol{v}'_k$ in C such that the following weighted overlap penalty function

$$\widetilde{F}_k(\boldsymbol{v}'_k, o'_k) = \sum_{1 \leq j \leq n, j \neq k} w_{kj} \cdot f_{kj}(\boldsymbol{v}'_k, \boldsymbol{v}_j, o'_k, o_j) \tag{19}$$

is as small as possible. For this, SearchNeighbor repeats translating $P_k(o'_k)$ in horizontal and vertical directions alternately until no better position is found in either direction. Figure 6 illustrates how SearchNeighbor proceeds. For each translation of the polygon $P_k(o'_k)$ in a specified direction $\boldsymbol{d} \in \{\boldsymbol{e}_x, \boldsymbol{e}_y\}$, let

$$N_0 = \{t \mid \boldsymbol{v}_k + t\boldsymbol{d} \in \mathrm{IFR}(C, P_k(o'_k)), t \in \mathbb{R}\} \tag{20}$$

be the set of valid t (i.e., $P_k(o'_k)$ is contained in C) among all positions on the line $\boldsymbol{v}_k + t\boldsymbol{d}$ $(t \in \mathbb{R})$. The SearchNeighbor finds a new valid position $\boldsymbol{v}'_k = \boldsymbol{v}_k + t\boldsymbol{d}$ $(t \in N_0)$ that minimizes the overlap penalty function $\widetilde{F}_k(\boldsymbol{v}_k + t\boldsymbol{d}, o'_k)$ while breaking ties by preferring the nearest one from the current position $\boldsymbol{v}_k$. The algorithm is formally described as follows.

Algorithm SearchNeighbor(P_k, o'_k)

Input: A list of polygons $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ and a rectangular container C with a width W and a length L. A solution $(\boldsymbol{v}, \boldsymbol{o})$ and a polygon P_k to be translated and its new orientation o'_k.

Output: The new position $\boldsymbol{v}'_k$ of polygon $P_k(o'_k)$.

Step 1: Set $\boldsymbol{v}'_k \leftarrow \boldsymbol{v}_k$. Find the valid position $\boldsymbol{v}''_k = \boldsymbol{v}'_k + t\boldsymbol{e}_x$ $(t \in N_0)$ of $P_k(o'_k)$ that minimizes the overlap penalty function $\widetilde{F}_k(\boldsymbol{v}'_k + t\boldsymbol{e}_x, o'_k)$. If $\widetilde{F}_k(\boldsymbol{v}''_k, o'_k) < \widetilde{F}_k(\boldsymbol{v}'_k, o'_k)$ holds, then set $\boldsymbol{v}'_k \leftarrow \boldsymbol{v}''_k$. Set $\boldsymbol{d} \leftarrow \boldsymbol{e}_y$.

Step 2: Find the valid position $\boldsymbol{v}''_k = \boldsymbol{v}'_k + t\boldsymbol{d}$ $(t \in N_0)$ of $P_k(o'_k)$ that minimizes the overlap penalty function $\widetilde{F}_k(\boldsymbol{v}'_k + t\boldsymbol{d}, o'_k)$. If $\widetilde{F}_k(\boldsymbol{v}''_k, o'_k) < \widetilde{F}_k(\boldsymbol{v}'_k, o'_k)$ holds, then set $\boldsymbol{v}'_k \leftarrow \boldsymbol{v}''_k$; otherwise output $\boldsymbol{v}'_k$ and halt.

Step 3: If $\boldsymbol{d} = \boldsymbol{e}_x$ holds, then set $\boldsymbol{d} \leftarrow \boldsymbol{e}_y$; otherwise set $\boldsymbol{d} \leftarrow \boldsymbol{e}_x$. Return to Step 2.

We now give the outline of our LS for OMP, which we call IMPROVE. The algorithm starts from an initial solution $(\boldsymbol{v}, \boldsymbol{o})$ with some overlapping polygons, and repeatedly replaces the current solution $(\boldsymbol{v}, \boldsymbol{o})$ with a better solution $(\boldsymbol{v}', \boldsymbol{o}')$ (obtained by SEARCHNEIGHBOR) in the neighborhood $\mathrm{NB}(\boldsymbol{v}, \boldsymbol{o})$ with the first admissible move strategy. That is, if the algorithm finds an improved solution $(\boldsymbol{v}', \boldsymbol{o}') \in \mathrm{NB}(\boldsymbol{v}, \boldsymbol{o})$ with $\widetilde{F}(\boldsymbol{v}', \boldsymbol{o}') < \widetilde{F}(\boldsymbol{v}, \boldsymbol{o})$, then it immediately replaces the current solution $(\boldsymbol{v}, \boldsymbol{o})$ with $(\boldsymbol{v}', \boldsymbol{o}')$. If no overlapping polygon exists in the current solution $(\boldsymbol{v}, \boldsymbol{o})$ or no better solution is found in the neighborhood $\mathrm{NB}(\boldsymbol{v}, \boldsymbol{o})$, then it outputs the current solution $(\boldsymbol{v}, \boldsymbol{o})$ (as a locally optimal solution) and the best solution $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ obtained so far, measured by the original overlap penalty function F, and halts.

To facilitate the efficiency of IMPROVE, we incorporate the *fast local search* strategy (Voudouris and Tsang, 1999). This strategy decomposes the neighborhood into a number of sub-neighborhoods, which are labeled with active or inactive depending on whether they are being searched or not; i.e., it skips the evaluations of all neighbor solutions in inactive sub-neighborhoods.

We define the sub-neighborhood $\mathrm{NB}_k(\boldsymbol{v}, \boldsymbol{o})$ $(1 \leq k \leq n)$ of the current solution $(\boldsymbol{v}, \boldsymbol{o})$ as the set of solutions obtainable by trying each orientation $o'_k \in O_k$ and applying SEARCHNEIGHBOR to P_k. The algorithm first sets all sub-neighborhoods $\mathrm{NB}_k(\boldsymbol{v}, \boldsymbol{o})$ $(1 \leq k \leq n)$ to be active. If no improvement has been made in an active sub-neighborhood $\mathrm{NB}_k(\boldsymbol{v}, \boldsymbol{o})$, then the algorithm inactivates it. If it finds an improved neighbor solution $(\boldsymbol{v}', \boldsymbol{o}')$ with $\widetilde{F}(\boldsymbol{v}', \boldsymbol{o}') < \widetilde{F}(\boldsymbol{v}, \boldsymbol{o})$ in an active sub-neighborhood $\mathrm{NB}_k(\boldsymbol{v}, \boldsymbol{o})$ (i.e., by moving P_k), then it activates all sub-neighborhoods $\mathrm{NB}_j(\boldsymbol{v}, \boldsymbol{o})$ of those P_j $(j \neq k)$ which overlap with P_k before or after moving P_k. The algorithm IMPROVE is formally described as follows, where A denotes the set of indices k $(1 \leq k \leq n)$ corresponding to the active sub-neighborhoods $\mathrm{NB}_k(\boldsymbol{v}, \boldsymbol{o})$, and $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}})$ denotes the locally optimal solution measured by the weighted overlap penalty function $\widetilde{F}$.

Algorithm IMPROVE$(L, \boldsymbol{v}, \boldsymbol{o})$

Input: A list of polygons $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ with a list of their possible orientations $\mathcal{O} = (O_1, O_2, \ldots, O_n)$ and a rectangular container C with a width W and a length L. A solution $(\boldsymbol{v}, \boldsymbol{o})$.

Output: The best solution $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ by the measure of F, and the best solution $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}})$ by the measure of $\widetilde{F}$.

Step 1: Set $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}}) \leftarrow (\boldsymbol{v}, \boldsymbol{o})$, $(\boldsymbol{v}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{v}, \boldsymbol{o})$ and $A \leftarrow \{1, 2, \ldots, n\}$.

Step 2: If $A = \emptyset$ holds, then output $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ and $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}})$ and halt; otherwise select an index $k \in A$ and set $O \leftarrow O_k$.

Step 3: Apply SEARCHNEIGHBOR(P_k, o'_k) for an orientation $o'_k \in O$ to obtain a neighbor solution $(\boldsymbol{v}', \boldsymbol{o}')$. If $F(\boldsymbol{v}', \boldsymbol{o}') < F(\boldsymbol{v}^*, \boldsymbol{o}^*)$ holds, then set $(\boldsymbol{v}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{v}', \boldsymbol{o}')$. If $\widetilde{F}(\boldsymbol{v}', \boldsymbol{o}') < \widetilde{F}(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}})$ holds, then set $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}}) \leftarrow (\boldsymbol{v}', \boldsymbol{o}')$ and go to Step 4; otherwise set $O \leftarrow O \setminus \{o'_k\}$. If $O = \emptyset$ holds, then set $A \leftarrow A \setminus \{k\}$ and return to Step 2; otherwise return to Step 3.

Step 4: If $F(\boldsymbol{v}^*, \boldsymbol{o}^*) = 0$ (i.e., no overlapping polygon exists), then output $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ and $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}})$ and halt; otherwise set $A \leftarrow A \cup \{j\}$ for all polygons P_j $(j \neq k)$ overlapping with the polygon P_k before or after applying SEARCHNEIGHBOR(P_k, o'_k) and return to Step 2.

6.2 Fast neighborhood search

In this section, we describe the core part of SEARCHNEIGHBOR, i.e., how it finds a new position when a polygon $P_k(o'_k)$ is translated in a specified direction $\boldsymbol{d}$. Recall that the new position $\boldsymbol{v}'_k = \boldsymbol{v}_k + t\boldsymbol{d}$ minimizes the overlap penalty function $\widetilde{F}_k(\boldsymbol{v}_k + t\boldsymbol{d}, o'_k)$ while the polygon is in the container C. We consider below the case when $P_k(o'_k)$ is translated in the horizontal direction (i.e., $\boldsymbol{d} = \boldsymbol{e}_x$). The case of $\boldsymbol{d} = \boldsymbol{e}_y$ is similar and is omitted. By definition (19), the overlap penalty function

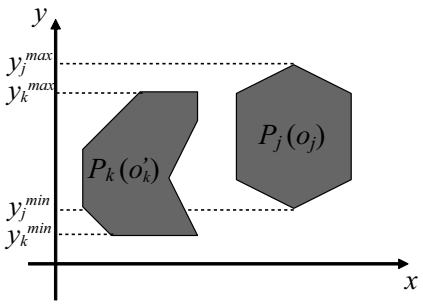

Figure 7: Detecting the overlap between $P_k(o'_k)$ and $P_j(o_j)$ when $P_k(o'_k)$ moves horizontally.

$\widetilde{F}_k(\boldsymbol{v}_k + t\boldsymbol{d}, o'_k)$ is decomposed into $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ for pairs of polygons $P_k(o'_k)$ and $P_j(o_j)$ $(j \neq k)$. Let

$$I_{kj} = \{t \mid \boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j \in \mathrm{NFP}(P_j(o_j), P_k(o'_k))\} \tag{21}$$

be the set of positions (in terms of t) of $P_k(o'_k)$ such that it overlaps with polygon $P_j(o_j)$. If $I_{kj} = \emptyset$ holds, then the overlap penalty $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ is zero for all $t \in \mathbb{R}$. The algorithm first determines whether $I_{kj} = \emptyset$ or not by checking the overlap between projections of $P_k(o'_k)$ and $P_j(o_j)$ onto the y-axis: $I_{kj} = \emptyset$ holds if and only if the intersection of two intervals satisfies $(y_k^{\min}, y_k^{\max}) \cap (y_j^{\min}, y_j^{\max}) = \emptyset$, where $y_l^{\min}$ and $y_l^{\max}$ $(1 \leq l \leq n)$ of $P_l(o_l)$ at location $\boldsymbol{v}_l$ are defined by

$$y_l^{\min} = \min\{y \mid (x, y) \in P_l(o_l) \oplus \boldsymbol{v}_l\}, \tag{22}$$

$$y_l^{\max} = \max\{y \mid (x, y) \in P_l(o_l) \oplus \boldsymbol{v}_l\}, \tag{23}$$

respectively. Figure 7 illustrates an example of detecting the overlap between $P_k(o'_k)$ and $P_j(o_j)$ when $P_k(o'_k)$ moves horizontally (in this example projections of two polygons overlap and $I_{kj} \neq \emptyset$ holds).

We now consider how to compute the overlap penalty function $f_{ij}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ (defined in (9)) for a given t, where $I_{kj} \neq \emptyset$ is assumed. See Figure 8. If $t \notin I_{kj}$ holds, then $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j) = 0$. If $t \in I_{kj}$ holds, then the overlap penalty $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ is decomposed into the horizontal and vertical penetration depths; furthermore, the vertical penetration depth is decomposed into the upward and downward penetration depths. Figure 8 (b) illustrates the computation of these three directional penetration depths. We denote the horizontal penetration depth for a given t by

$$\rho_{kj}^{\mathrm{H}}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j) = \min\{|s| \mid \mathrm{int}(P_j(o_j) \oplus \boldsymbol{v}_j) \cap (P_k(o'_k) \oplus (\boldsymbol{v}_k + t\boldsymbol{d}) \oplus s\boldsymbol{e}_x) = \emptyset\}, \tag{24}$$

where $\mathrm{int}(S)$ denotes the interior of a polygon S. The right hand side is equivalent to the minimum horizontal distance $|s|$ from the point $\boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j$ to the nearest boundary of no-fit polygon $\partial\mathrm{NFP}(P_j(o_j), P_k(o'_k))$.

The set I_{kj} is in general decomposed into a number of separated intervals $(t_{kj1}^{\min}, t_{kj1}^{\max})$, $(t_{kj2}^{\min}, t_{kj2}^{\max})$, $\ldots$, $(t_{kjm_{kj}}^{\min}, t_{kjm_{kj}}^{\max})$ (since polygons may not be convex), where $m_{kj} (\leq \frac{q_{kj}}{2})$ is the number of distinct intervals belonging to I_{kj}. The I_{kj} in the example of Figure 8 (a) is decomposed into two intervals. If the value t belongs to an interval $(t_{kjl}^{\min}, t_{kjl}^{\max})$, then $\rho_{kj}^{\mathrm{H}}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ is computed by

$$\rho_{kj}^{\mathrm{H}}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j) = \min\{t - t_{kjl}^{\min}, t_{kjl}^{\max} - t\}. \tag{25}$$

We also denote upward and downward penetration depths by

$$\rho_{kj}^{\mathrm{U}}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j) = \min\{|s| \mid \mathrm{int}(P_j(o_j) \oplus \boldsymbol{v}_j) \cap (P_k(o'_k) \oplus (\boldsymbol{v}_k + t\boldsymbol{d}) \oplus s\boldsymbol{e}_y) = \emptyset, s \geq 0\}, \tag{26}$$

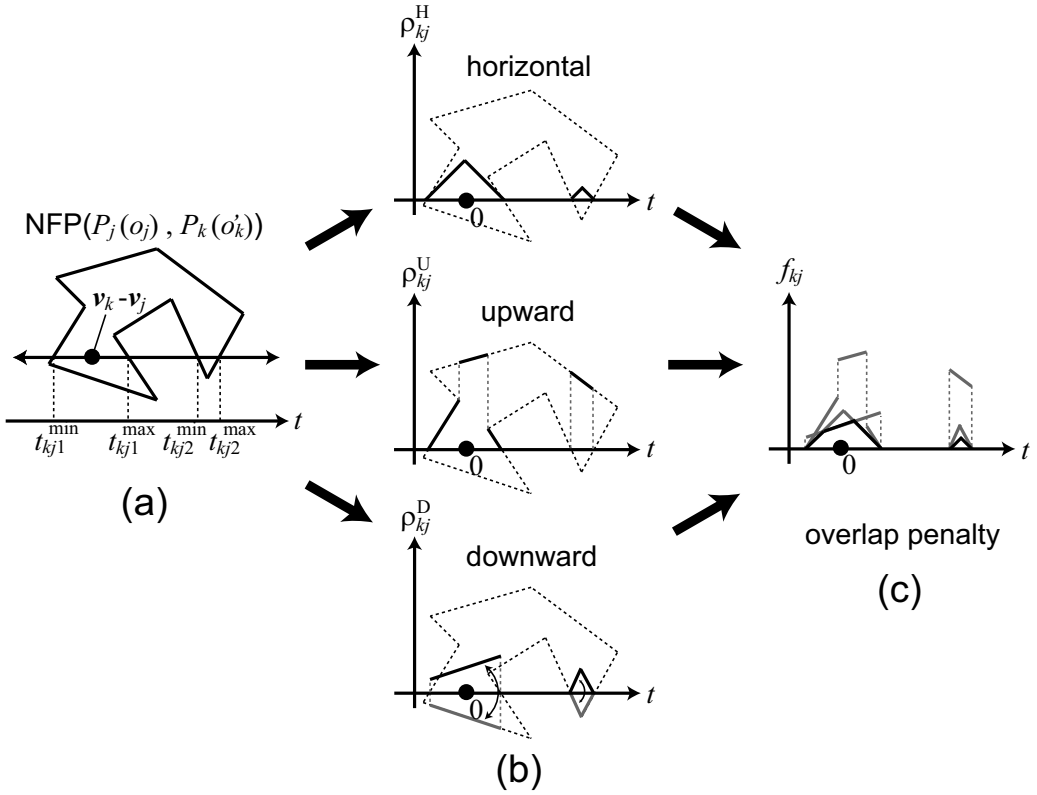

Figure 8: Computing the overlap penalty function $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ when $P_k(o'_k)$ moves horizontally.

$$\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j) = \min\{|s| \mid \mathrm{int}(P_j(o_j) \oplus \boldsymbol{v}_j) \cap (P_k(o'_k) \oplus (\boldsymbol{v}_k + t\boldsymbol{d}) \oplus s\boldsymbol{e}_y) = \emptyset, s \leq 0\}, \quad (27)$$

respectively, which is equivalent to the minimum distances $|s|$ from the point $\boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j$ to the nearest boundary of no-fit polygon $\partial\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ in upward and downward directions, respectively.

The algorithm first divides $\partial\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ into the upper and lower parts by the horizontal line $\boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j$ ($t \in \mathbb{R}$) (see Figure 8 (a)). It then computes $\rho^{\mathrm{U}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ by taking the lowest edges of $\partial\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ above the horizontal line for $t \in I_{kj}$; it also computes $\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ by taking the negative of the closest edges of $\partial\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ below the horizontal line for $t \in I_{kj}$. These are illustrated in Figure 8 (b). The overlap penalty function $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ for $t \in I_{kj}$ is eventually computed by the following formula:

$$\begin{aligned} f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j) = \min\{&\rho^{\mathrm{H}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j), \rho^{\mathrm{U}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j), \\ &\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)\}, \end{aligned} \quad (28)$$

which is given by the lower envelope of the three directional penetration depths $\rho^{\mathrm{H}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$, $\rho^{\mathrm{U}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ and $\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ (Figure 8 (c)).

The horizontal penetration depth $\rho^{\mathrm{H}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ is a piecewise linear function over the sorted lists of intervals $(t^{\min}_{kjl}, t^{\max}_{kjl})$ $(1 \leq l \leq m_{kj})$ of I_{kj} along the t-axis. The upward and downward penetration depths $\rho^{\mathrm{U}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ and $\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ are also piecewise linear functions over the sorted lists of edges of no-fit polygon $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ along the t-axis. The computation time of each directional penetration depth is accordingly $O(q_{kj} \log q_{kj})$ if it is implemented naively. However, as explained below, it can be reduced to $O(q_{kj})$ by using two trapezoidal partitions of no-fit polygon $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$.

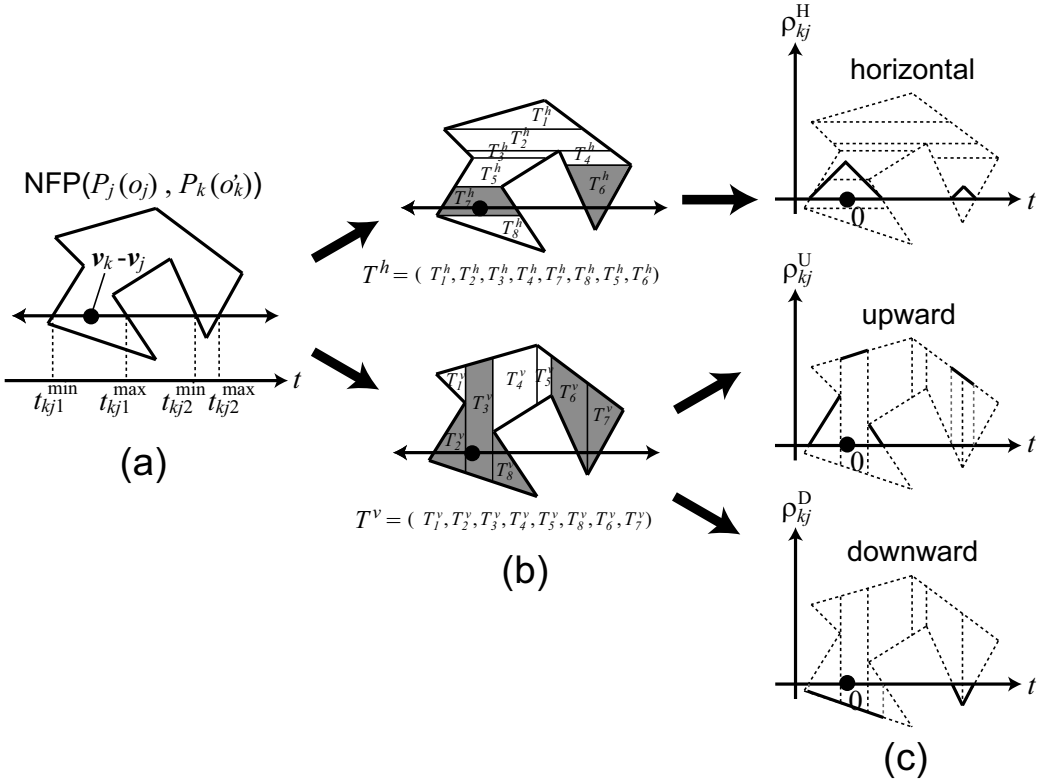

Figure 9: Computing directional penetration depths $\rho^{\mathrm{H}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$, $\rho^{\mathrm{U}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ and $\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ using two trapezoidal partitions T^h and T^v of a no-fit polygon.

The computation of three directional penetration depths $\rho^{\mathrm{H}}_{kj}(\boldsymbol{v}_k+t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$, $\rho^{\mathrm{U}}_{kj}(\boldsymbol{v}_k+t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ and $\rho^{\mathrm{D}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ is illustrated in Figure 9. The algorithm first computes the trapezoidal partition $T^h = \{T^h_1, T^h_2, \ldots, T^h_{r_h}\}$ of $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ with horizontal lines, which we call the horizontal partition, and another trapezoidal partition $T^v = \{T^v_1, T^v_2, \ldots, T^v_{r_v}\}$ with vertical lines, which we call the vertical partition. Each T^h_i (resp., T^v_i) is a trapezoid with two parallel horizontal (resp., vertical) lines, and the number of trapezoids r_h (resp., r_v) is at most the number of edges q_{kj} of $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$. These two partitions are illustrated in Figure 9 (b). We note that T^h satisfies $\bigcup_{i=1}^{r_h} T^h_i = \mathrm{NFP}(P_j(o_j), P_k(o'_k))$ and $T^h_a \cap T^h_b = \emptyset$ $(1 \le a < b \le r_h)$, and T^v satisfies analogous conditions. The computation time of $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ is $O(q_{kj})$ (see Section 3), and that of its trapezoidal partitions is also $O(q_{kj})$ because it can be done in linear time to the number of edges of $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$ (Chazelle, 1991).

We define a partial order $\preceq_x$ for any pair of trapezoids T^h_a and T^h_b as follows: $T^h_a \preceq_x T^h_b$ holds if and only if (i) there exists a horizontal line $\boldsymbol{w} + t\boldsymbol{e}_x$ ($\boldsymbol{w} \in \mathbb{R}^2$, $t \in \mathbb{R}$) crossing both T^h_a and T^h_b, and (ii) $\min\{t \mid \boldsymbol{w} + t\boldsymbol{e}_x \in T^h_a\} \le \min\{t \mid \boldsymbol{w} + t\boldsymbol{e}_x \in T^h_b\}$ holds. This partial order $\preceq_x$ can be defined for any pair of T^v_a and T^v_b in a similar manner. For each of T^h and T^v, the algorithm extends $\preceq_x$ to a total order $\preceq^*_x$ by topological sorting, which is called a *linear extension* of $\preceq_x$. We note that the above computation is done only once as preprocessing for each no-fit polygon.

The algorithm then computes these three directional penetration depths using the trapezoidal partitions T^h and T^v of $\mathrm{NFP}(P_j(o_j), P_k(o'_k))$. See Figure 9 (c). For the horizontal penetration depth $\rho^{\mathrm{H}}_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$, the algorithm first constructs the list of intervals $(t^{\min}_{kj1}, t^{\max}_{kj1}), (t^{\min}_{kj2}, t^{\max}_{kj2}), ...,$ $(t^{\min}_{kjm_{kj}}, t^{\max}_{kjm_{kj}})$ of I_{kj} by scanning intersections of the horizontal line $\boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j$ $(t \in \mathbb{R})$ with trapezoids T^h_i $(1 \le i \le r_h)$, according to the total order $\preceq^*_x$ on $\{T^h_i\}$. This computation can be done in $O(q_{kj})$ time because checking the intersection between horizontal line $\boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j$ $(t \in \mathbb{R})$

and a trapezoid T_i^h is done in $O(1)$ time. It then computes function (25) on all the intervals. For the upward penetration depth $\rho_{kj}^{\mathrm{U}}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$, it scans all the trapezoids T_i^v $(1 \leq i \leq r_v)$, which cross the horizontal line $\boldsymbol{v}_k + t\boldsymbol{d} - \boldsymbol{v}_j$ $(t \in \mathbb{R})$ according to the total order $\preceq_x^*$ on $\{T_i^v\}$. It then computes function (26) by taking the upper edge of each crossing trapezoid for $t \in I_{kj}$. Similarly, it computes the downward penetration depth $\rho_{kj}^{\mathrm{D}}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$. In this way, the computation for each directional penetration depth is done in $O(q_{kj})$ time. Then the overlap penalty function $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ can be computed in $O(q_{kj})$ time by taking the lower envelope of these three directional penetration depths along the t-axis.

Finally, the algorithm computes the overlap penalty function $\widetilde{F}_k(\boldsymbol{v}_k + t\boldsymbol{d}, o'_k)$ of (19) by summing up the overlap penalty functions $f_{kj}(\boldsymbol{v}_k + t\boldsymbol{d}, \boldsymbol{v}_j, o'_k, o_j)$ for all polygons $P_j(o_j)$ $(j \neq k)$ satisfying $I_{kj} \neq \emptyset$. Note that the number of such $P_j(o_j)$ is $O(n)$. The time for this computation is $O((\sum_j q_{kj}) \log n) = O(q_k \log n)$, since it is carried out on the sorted list of $O(q_k)$ breakpoints obtained by merging $O(n)$ lists, each keeping $O(q_{kj})$ number of already sorted breakpoints.

6.3 Guided local search

It is often the case that LS alone may not attain a sufficiently good solution. To improve the situation, many variants have been developed, and their framework is called *metaheuristics*. The *guided local search* is one of the representative metaheuristic approaches (Voudouris and Tsang, 1999), which repeatedly applies local search algorithm while updating the penalty weights of the objective function adaptively and resumes the search from the previous locally optimal solution.

To solve OMP, we develop a guided local search algorithm based on the *weighting method* (Selman and Kautz, 1993), which we call MinimizeOverlap. The algorithm starts from an initial solution $(\boldsymbol{v}, \boldsymbol{o})$ with some overlapping polygons, where the penalty weights w_{ij} are initialized to 1.0. Whenever Improve stops at a locally optimal solution $(\boldsymbol{v}, \boldsymbol{o})$, the algorithm updates the penalty weights w_{ij} by the following rule. If a pair of polygons $P_i(o_i)$ and $P_j(o_j)$ placed at $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ overlaps (i.e., $f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j) > 0$), the algorithm increases the penalty weight w_{ij} as follows:

$$w_{ij} \leftarrow w_{ij} + \frac{f_{ij}(\boldsymbol{v}_i, \boldsymbol{v}_j, o_i, o_j)}{\max\limits_{1 \leq k < l \leq n} f_{kl}(\boldsymbol{v}_k, \boldsymbol{v}_l, o_k, o_l)}. \tag{29}$$

By applying (29) repeatedly, the current solution $(\boldsymbol{v}, \boldsymbol{o})$ becomes no longer locally optimal in the updated overlap penalty function, and it resumes the search from the current solution.

The algorithm MinimizeOverlap is formally described as follows, where *iter* denotes the number of calls to Improve after the last improvement of the best solution, *max_iter* (an input parameter given by users) specifies the upper bound on *iter*, and $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ is the best solution obtained so far measured by the original overlap penalty function F.

Algorithm MinimizeOverlap$(L, \boldsymbol{v}, \boldsymbol{o})$

Input: A list of polygons $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ with a list of their possible orientations $\mathcal{O} = (O_1, O_2, \ldots, O_n)$ and a rectangular container C with a width W and a length L. A solution $(\boldsymbol{v}, \boldsymbol{o})$.

Output: The best solution $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ by the measure of F.

Step 1: Set $iter \leftarrow 0$ and initialize $w_{ij} \leftarrow 1.0$ for all pairs of polygons P_i and P_j $(1 \leq i, j \leq n)$. Set $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}}) \leftarrow (\boldsymbol{v}, \boldsymbol{o})$ and $(\boldsymbol{v}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{v}, \boldsymbol{o})$.

Step 2: Apply Improve$(L, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}})$ to obtain the best solution $(\boldsymbol{v}', \boldsymbol{o}')$ in F and the locally optimal solution $(\tilde{\boldsymbol{v}}', \tilde{\boldsymbol{o}}')$ in $\widetilde{F}$. Set $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{o}}) \leftarrow (\tilde{\boldsymbol{v}}', \tilde{\boldsymbol{o}}')$.

Step 3: Update the penalty weights w_{ij} for all overlapping pairs of polygons P_i and P_j $(1 \leq i, j \leq n)$ by (29).

Table 1: The benchmark instances of ISP (cited from Gomes and Oliveira (2006)).

Instance	#shapes	#pieces	avg.#vertices	degrees
Albano	8	24	7.25	0,180
Dagli	10	30	6.30	0,180
Dighe1	16	16	3.87	0
Dighe2	10	10	4.70	0
Fu	12	12	3.58	0,90,180,270
Jakobs1	25	25	5.60	0,90,180,270
Jakobs2	25	25	5.36	0,90,180,270
Mao	9	20	9.22	0,90,180,270
Marques	8	24	7.37	0,90,180,270
Shapes0	4	43	8.75	0
Shapes1	4	43	8.75	0,180
Shapes2	7	28	6.29	0,180
Shirts	8	99	6.63	0,180
Swim	10	48	21.90	0,180
Trousers	17	64	5.06	0,180

Step 4: If $F(\boldsymbol{v}', \boldsymbol{o}') = 0$ holds, then set $(\boldsymbol{v}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{v}', \boldsymbol{o}')$, output $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ and halt. If $F(\boldsymbol{v}', \boldsymbol{o}') < F(\boldsymbol{v}^*, \boldsymbol{o}^*)$ holds, then set $(\boldsymbol{v}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{v}', \boldsymbol{o}')$ and $iter \leftarrow 0$ and return to Step 2.

Step 5: If $iter \geq max\_iter$ holds, then output $(\boldsymbol{v}^*, \boldsymbol{o}^*)$ and halt; otherwise set $iter \leftarrow iter + 1$ and return to Step 2.

7 Computational results

In this section, we report computational results for 15 well known benchmark instances, which are available online at EURO special interest group on cutting and packing (ESICUP) web site[1]. Table 1 summarizes the information of the instances. The second column "#shapes" shows the number of different shapes, the third column "#pieces" shows the total number of polygons, the fourth column "avg.#vertices" shows the average number of the vertices of different shapes, and the fifth column "degrees" shows the permitted orientations.

We tested our algorithm FITS on an IBM-compatible personal computer (Intel Xeon 2.8GHz, 2GB memory) and compared our results with those reported by Burke et al. (2006) (denoted as "BLF"), Gomes and Oliveira (2006) (denoted as "SAHA"), Egeblad et al. (2007) (denoted as "2DNest"), and Imamichi et al. (2007) (denoted as "ILSQN"). Table 2 shows the best efficiency in % of the algorithms and the average efficiency in % of the algorithms except for BLF, where the efficiency of a solution $(\boldsymbol{v}, \boldsymbol{o})$ with the container length $L = L(\boldsymbol{v}, \boldsymbol{o})$ is measured by the ratio $\sum_{i=1}^{n} (\text{area of } P_i)/WL$. The best results among these algorithms are marked with asterisks. Table 3 shows the computational environment and the computation time (in seconds) of the algorithms.

We set the input parameters $r_{\text{dec}} = 0.02$, $r_{\text{inc}} = 0.005$ and $max\_iter = 200$ for our algorithm. We executed our algorithm 10 times for each instance with the time limit of 1200 seconds for each run. Burke et al. (2006) tested four variations of their algorithm, and each variation were run 10 times. Their results in Table 2 are the best results of 40 runs. They did not use time limit but stopped their algorithm by other criteria, and their computation time in Table 3 is the time spent to find the best solution reported in Table 2 in the run that found it (i.e., the time for only one run is reported). Since they tested their algorithm for instances Albano, Dighe1 and Dighe2 with

[1]ESICUP: `http://paginas.fe.up.pt/~esicup/`

Table 2: The efficiency in % of the solutions obtained by FITS and other algorithms.

Instance	BLF	SAHA		2DNest		ILSQN		FITS	
	best	best	avg.	best	avg.	best	avg.	best	avg.
Albano	–	87.43	84.70	87.44	86.96	∗88.16	87.14	87.56	∗87.28
Dagli	83.7	87.15	85.38	85.98	85.31	∗87.40	∗85.80	86.35	85.71
Dighe1	–	∗100.00	82.13	99.86	93.93	99.89	90.49	99.89	∗99.84
Dighe2	–	∗100.00	84.17	99.95	93.11	99.99	84.21	∗100.00	∗99.99
Fu	86.9	90.96	87.17	∗91.84	∗90.93	90.67	87.57	91.23	90.31
Jakobs1	82.6	78.89	75.79	89.07	∗88.90	86.89	84.78	∗89.09	88.74
Jakobs2	74.8	77.28	74.66	80.41	80.28	∗82.51	∗80.50	80.84	80.26
Mao	79.5	82.54	80.72	∗85.15	82.67	83.44	81.31	83.73	∗82.79
Marques	86.5	88.14	86.88	89.17	88.73	89.03	86.81	∗89.21	∗88.80
Shapes0	60.5	66.50	63.20	67.09	65.42	∗68.44	∗66.49	66.50	66.20
Shapes1	66.5	71.25	68.63	73.84	71.74	73.84	∗72.83	∗73.88	72.60
Shapes2	77.7	83.60	81.41	81.21	79.89	∗84.25	∗81.72	81.68	80.87
Shirts	84.6	86.79	85.67	86.33	85.73	∗88.78	∗88.12	86.92	86.13
Swim	68.4	74.37	72.28	71.53	70.27	∗75.29	∗74.62	74.54	72.97
Trousers	88.5	∗89.96	89.02	89.84	∗89.29	89.79	88.69	89.40	88.78

different set of orientations from others, we do not show their results on these instances. Gomes and Oliveira (2006) ran their algorithm 20 times for each instance and the best and average results of 20 runs are shown in Table 2. They did not use time limit but stopped their algorithm by other criteria, and the average computation time of 20 runs is shown in Table 3. Egeblad et al. (2007) and Imamichi et al. (2007) executed their algorithms 20 times and 10 times for each instance, respectively, where they set the time limit of each run as shown in Table 3.

Our algorithm FITS obtained the best results for four and five instances out of 15 instances in the best and average efficiencies of all runs, respectively, within a reasonable amount of computation time. It also obtained the results with almost equivalent efficiency to the best results for some instances. The computation time of BLF is much shorter than that of FITS, but FITS obtained better results in efficiency than BLF for all instances. Figure 10 shows the best placements obtained by FITS for these instances.

8 Conclusion

In this paper, we proposed a heuristic algorithm for the irregular strip packing problem (ISP). As a core subproblem to solve ISP, we presented an overlap minimization problem (OMP) whose objective is to place all polygons into a container with fixed width and length so that the total amount of overlap between polygons is made as small as possible. We proposed to use directional penetration depths to measure the amount of overlap between a pair of polygons, and presented an efficient algorithm to find a position with the minimum overlap penalty for each polygon when it is translated in a specified direction. Based on this, we developed a local search algorithm for OMP using the iterative translation search that translates a polygon in horizontal and vertical directions alternately, and then we incorporated it into a variant of the guided local search algorithm. Computational results show that our algorithm is quite promising and improves the best known values of some well known benchmark instances.

Table 3: Computation time of FITS and other algorithms (in seconds).

Instance	BLF Pentium4 2.0GHz 4×10 runs best	SAHA Pentium4 2.4GHz 20 runs avg.	2DNest Pentium4 3.0GHz 20 runs limit	ILSQN Xeon 2.8GHz 10 runs limit	FITS Xeon 2.8GHz 10 runs limit
Albano	–	2257	600	1200	1200
Dagli	188.80	5110	600	1200	1200
Dighe1	–	83	600	600	1200
Dighe2	–	22	600	600	1200
Fu	20.78	296	600	600	1200
Jakobs1	43.49	332	600	600	1200
Jakobs2	81.41	454	600	600	1200
Mao	29.74	8245	600	1200	1200
Marques	4.87	7507	600	1200	1200
Shapes0	21.33	3914	600	1200	1200
Shapes1	2.19	10,314	600	1200	1200
Shapes2	21.00	2136	600	1200	1200
Shirts	58.36	10,391	600	1200	1200
Swim	607.37	6937	600	1200	1200
Trousers	756.15	8588	600	1200	1200

Acknowledgment

This research was partially supported by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

Adamowicz, M., Albano, A, 1976. Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design* 8, 27–33.

Agarwal, P. K., Guibas, L. J., Har-Peled, S., Rabinovitch, A., Sharir, M., 2000. Penetration depth of two convex polytopes in 3D. *Nordic Journal of Computing* 7, 227–240.

Albano, A., Sapuppo, G., 1980. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Transactions on Systems, Man and Cybernetics* 10, 242–248.

Bennell, J. A., Dowsland, K. A., 2001. Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science* 47, 1160–1172.

Bennell, J. A., Dowsland, K. A., Dowsland, W. B., 2001. The irregular cutting-stock problem — A new procedure for deriving the no-fit polygon. *Computers and Operations Research* 28, 271–287.

Błażewicz, J., Hawryluk, P., Walkowiak, R., 1993. Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Annals of Operations Research* 41, 313–325.

Burke, E., Hellier, R., Kendall, G., Whitwell, G., 2006. A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations Research* 54, 587–601.

Burke, E. K., Hellier, R. S. R., Kendall, G., Whitwell, G., 2007. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research* 179, 27–49.

de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O., 2000. Computational Geometry: Algorithms and Applications (2nd edition). Springer, Berlin.

Chazelle, B., 1991. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry* 6, 485–524.

Dean, H. T., Tu, Y., Raffensperger, J. F., 2006. An improved method for calculating the no-fit polygon. *Computers and Operations Research* 33, 1521–1539.

Dobkin, D., Hershberger, J., Kirkpatrick, D., Suri, S., 1993. Computing the intersection-depth of polyhedra. *Algorithmica* 9, 518–533.

Dowsland, K. A., Vaid, S., Dowsland, W. B., 2002. An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research* 141, 371–381.

Egeblad, J., Nielsen, B. K., Odgaard, A., 2007. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research* 183, 1249–1266.

Gomes, A. M., Oliveira, J. F., 2002. A 2-exchange heuristic for nesting problems. *European Journal of Operational Research* 141, 359–370.

Gomes, A. M., Oliveira, J. F., 2006. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research* 171, 811–829.

Ibaraki, T., Imahori, S., Yagiura, M., 2008. Hybrid metaheuristics for packing problems. In: Blum, C., Aguilera, M. J. B., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics: An Emergent Approach for Optimization. Springer, Berlin.

Imamichi, T., Yagiura, M., Nagamochi, H., 2007. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. Technical Report 2007-009, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University.

Kim, Y. J., Lin, M. C., Manocha, D., 2004. Incremental penetration depth estimation between convex polytopes using dual-space expansion. *IEEE Transactions on Visualization and Computer Graphics* 10, 152–163.

Li, Z., Milenkovic, V., 1995. Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research* 84, 539–561.

Oliveira, J. F., Gomes, A. M., Ferreira, J. S., 2000. TOPOS — A new constructive algorithm for nesting problems. *OR Spektrum* 22, 263–284.

Selman, B., Kautz, H., 1993. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. *Proceedings of 13th International Joint Conference on Artificial Intelligence*, 290–295.

Voudouris, C., Tsang, E., 1999. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research* 113, 469–499.

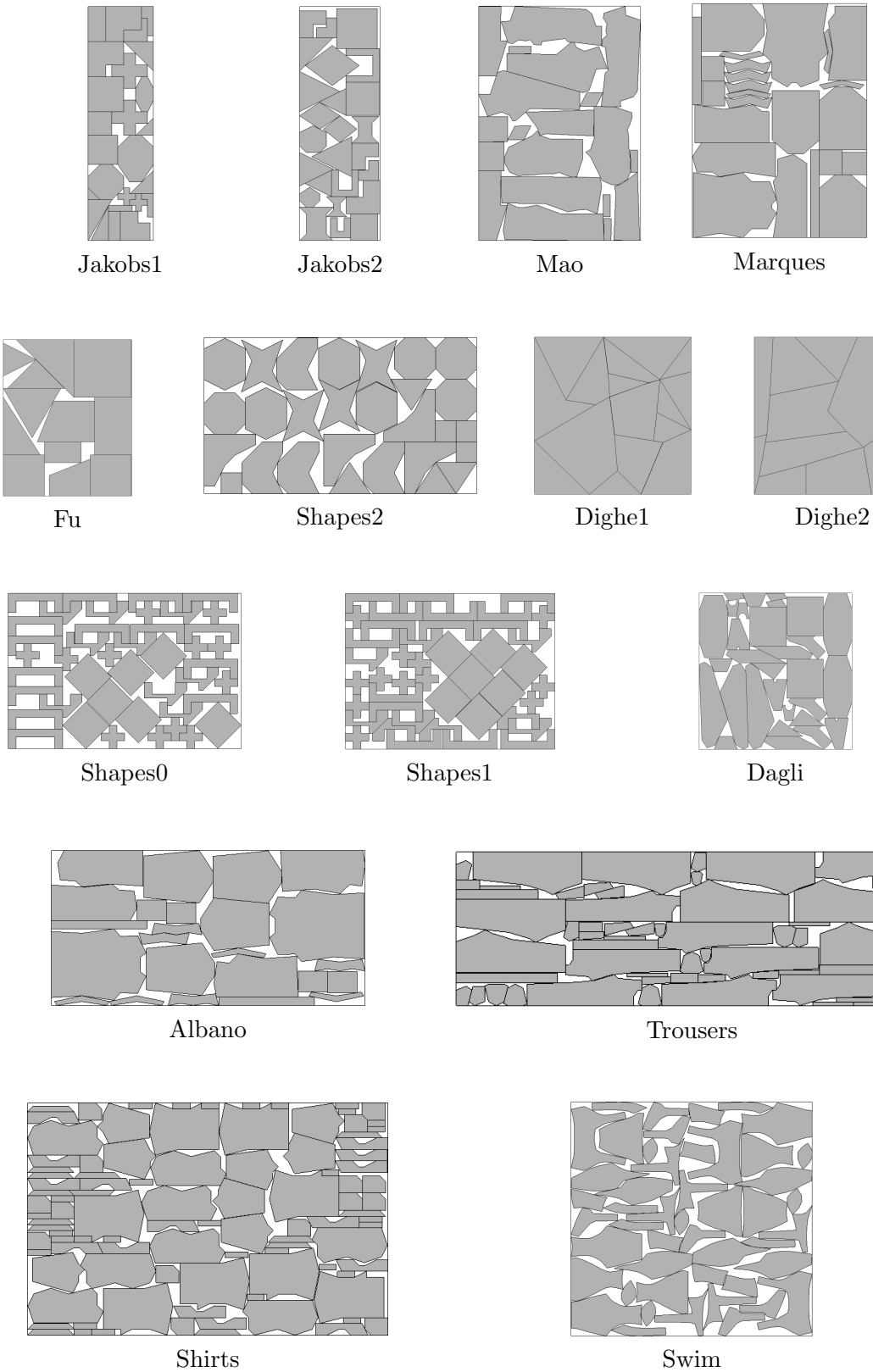

Figure 10: The best solutions obtained by FITS.