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
The METR technical reports are published as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author’s copyright. These works may not be reposted without the explicit permission of the copyright holder.
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
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. Blázewicz 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. Dowland 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.
Then, the Minkowski sum \( \text{we describe a solution of ISP by lists of positions} \)
\( \text{w}_0 \text{degree is included in the set of orientations} \)
\( \text{For a polygon} \)
\( \text{interior points and those points on the boundary, when its reference point is put at the origin} \)
\( \text{denote a polygon} \)
\( \text{O} = (\text{P}_1, \text{P}_2, \ldots, \text{P}_n) \)
\( \text{with a list of their possible orientations} \)
\( \text{O} = (\text{O}_1, \text{O}_2, \ldots, \text{O}_n), \text{where} \)
\( \text{O}_i \text{ (1} \leq \text{i} \leq \text{n) is the set of all orientations which a polygon} \)
\( \text{can be rotated (i.e.,} \text{P}_i \text{can be rotated by} \text{o degrees for all} \text{o} \in \text{O}_i) \)
\( \text{and a rectangular container} \)
\( \text{C} = \text{C(W, L)} \text{with a width} \text{W} \text{and a length} \text{L}, \text{where} \text{W} \text{is} \)
\( \text{a nonnegative constant and} \text{L is a nonnegative variable.} \)
\( \text{We denote a polygon} \)
\( \text{P}_i \text{rotated by} \text{o} \in \text{O}_i \text{degree by} \text{P}_i(o), \text{which may be} \)
\( \text{written as} \text{P}_i \text{for simplicity when its orientation is not specified or clear from the context.} \)
\( \text{We describe a position of a polygon} \)
\( \text{P}_i \text{by a coordinate} \text{v}_i = (x_i, y_i) \text{of its} \text{reference point}, \text{which is a} \)
\( \text{point of} \text{P}_i \text{(e.g., a vertex or the center of gravity of} \text{P}_i) \text{.} \)
\( \text{For convenience, we regard all polygons} \)
\( \text{P}_i \text{and the container} \text{C} \text{as closed regions. To be more precise, we regard} \text{P}_i \text{or} \text{C} \text{as the set of all} \)
\( \text{interior points and those points on the boundary, when its reference point is put at the origin} \)
\( \text{For a polygon} \text{S}, \text{let} \text{int(S) be the interior of} \text{S, cl(S) be the closure of} \text{S,} \partial \text{S be the boundary of} \text{S} \text{and} \)
\( \text{S is the complement of} \text{S.} \)
\( \text{For each polygon} \text{P}_i, \text{we assume without loss of generality that zero degree is included in the set of orientations} \text{O}_i, \text{and its width} \)
\( w_i(o) = \max \{y \mid (x, y) \in P_i(o)\} - \min \{y \mid (x, y) \in P_i(o)\} \) \hspace{1cm} (1) \)
\( \text{satisfies} \ w_i(o) \leq W \text{ for all orientations} \text{o} \in \text{O}_i. \)
\( \text{We describe a polygon} \)
\( \text{P}_i \text{placed at} \text{v}_i \text{by the Minkowski sum} \)
\( P_i \oplus v_i = \{p + v_i \mid p \in P_i\}. \) \hspace{1cm} (2) \)
\( \text{Then the irregular strip packing problem} \text{(ISP)} \text{is formally described as follows:} \)
\( \text{(ISP)} \text{ minimize} \)
\( \text{subject to} \)
\( \text{int}(P_i(o_i) \oplus v_i) \cap (P_j(o_j) \oplus v_j) = \emptyset, \quad (1 \leq i < j \leq n), \) \( \text{(1)} \)
\( P_i(o_i) \oplus v_i \subseteq C(W, L), \quad (1 \leq i \leq n), \) \( \text{(2)} \)
\( L \geq 0, \) \( \text{(3)} \)
\( o_i \in O_i, \quad (1 \leq i \leq n), \) \( \text{(4)} \)
\( v_i \in \mathbb{R}^2, \quad (1 \leq i \leq n). \) \( \text{We describe a solution of ISP by lists of positions} \text{v} = (v_1, v_2, \ldots, v_n) \text{and orientations} \text{o} = (o_1, o_2, \ldots, o_n) \text{of all polygons} \text{P}_i \text{(1} \leq \text{i} \leq \text{n). Note that a solution} \text{(v, o)} \text{uniquely determines a} \)
\( \text{placement of the polygons.} \) \( \text{The length} \text{L of the container} \text{C is determined by} \)
\( L(v, o) = \max \{x \mid (x, y) \in (P_i(o_i) \oplus v_i), 1 \leq i \leq n\} - \min \{x \mid (x, y) \in (P_i(o_i) \oplus v_i), 1 \leq i \leq n\} \) \hspace{1cm} (4) \)
and \((P_i(o_i) \oplus v_i) \subseteq C(W, L(v, o))\) holds for all polygons \(P_i\) \((1 \leq i \leq n)\) if and only if
\[
W \geq \max\{y \mid (x, y) \in (P_i(o_i) \oplus v_i), 1 \leq i \leq n\} - \min\{y \mid (x, y) \in (P_i(o_i) \oplus v_i), 1 \leq i \leq n\}. \tag{5}
\]

2.2 Overlap minimization problem

We use the overlap minimization problem (OMP) as a subproblem of ISP to find a feasible placement \((v, o)\) 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 \(f_{ij}(v_i, v_j, o_i, o_j)\) be a function that measures the overlap amount of a pair of polygons \(P_i(o_i)\) and \(P_j(o_j)\) placed at \(v_i\) and \(v_j\), respectively. The objective of OMP is to find a solution \((v, o)\) that minimizes the total amount of the overlap penalty \(F(v, o) = \sum_{1 \leq i < j \leq n} f_{ij}(v_i, v_j, o_i, o_j)\) 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:

\[
\text{(OMP}(L_{UB})\text{)} \quad \begin{array}{ll}
\text{minimize} & F(v, o) = \sum_{1 \leq i < j \leq n} f_{ij}(v_i, v_j, o_i, o_j) \\
\text{subject to} & (P_i(o_i) \oplus v_i) \subseteq C(W, L_{UB}), \\
& o_i \in O_i, \\
& v_i \in \mathbb{R}^2, \\
& (1 \leq i \leq n), \\
& (1 \leq i \leq n), \\
& (1 \leq i \leq n).
\end{array} \tag{6}
\]

In this paper, we use the directional penetration depth to define the overlap penalty function \(f_{ij}(v_i, v_j, o_i, o_j)\) 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 \(\delta(P_i(o_i) \oplus v_i, P_j(o_j) \oplus v_j)\) of a pair of overlapping polygons \(P_i(o_i)\) and \(P_j(o_j)\) placed at \(v_i\) and \(v_j\) 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

\[
\delta(P_i(o_i) \oplus v_i, P_j(o_j) \oplus v_j) = \min\{||u|| \mid \text{int}(P_i(o_i) \oplus v_i) \cap (P_j(o_j) \oplus v_j + u) = \emptyset, u \in \mathbb{R}^2\}, \tag{7}
\]

where \(|| \cdot ||\) denotes the Euclidean norm.

The directional penetration depth \(\rho(P_i(o_i) \oplus v_i, P_j(o_j) \oplus v_j, d)\) of a pair of overlapping polygons \(P_i(o_i)\) and \(P_j(o_j)\) placed at \(v_i\) and \(v_j\) is defined as the minimum translational distance in a given direction \(d = (d_x, d_y)\) \((||d|| = 1, d \in \mathbb{R}^2)\) 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

\[
\rho(P_i(o_i) \oplus v_i, P_j(o_j) \oplus v_j, d) = \min\{|t| \mid \text{int}(P_i(o_i) \oplus v_i) \cap (P_j(o_j) \oplus v_j + td) = \emptyset, t \in \mathbb{R}\}. \tag{8}
\]

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

\[
f_{ij}(v_i, v_j, o_i, o_j) = \min\{\rho(P_i(o_i) \oplus v_i, P_j(o_j) \oplus v_j, d) \mid d \in \{e_x, e_y\}\}, \tag{9}
\]

where \(e_x = (1, 0)\) and \(e_y = (0, 1)\); 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 \(C\) as the protrusion penalty.
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 NFP($P_i, P_j$) of an ordered pair of polygons $P_i$ and $P_j$ is defined by

$$NFP(P_i, P_j) = \text{int}(P_i) \oplus (-P_j) = \{u - w \mid u \in \text{int}(P_i), w \in P_j\},$$

and has the following important properties:

- $P_j \oplus v_j$ overlaps with $P_i \oplus v_i$ if and only if $v_j - v_i \in \text{NFP}(P_i, P_j)$.
- $P_j \oplus v_j$ touches $P_i \oplus v_i$ if and only if $v_j - v_i \in \partial \text{NFP}(P_i, P_j)$.
- $P_i \oplus v_i$ and $P_j \oplus v_j$ are separated if and only if $v_j - v_i \notin \text{cl}(\text{NFP}(P_i, P_j))$.

where $\partial S$ and $\text{cl}(S)$ 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 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, $\partial \text{NFP}(P_i, P_j)$ 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 $\partial \text{NFP}(P_i, P_j)$. This procedure takes $O(p_i + p_j)$ time. Figure 2 shows an example of 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, 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

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

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 $C$.

![Figure 2: The no-fit polygon NFP($P_i, P_j$) of two convex polygons $P_i$ and $P_j.$](image-url)
To check whether \( P_i \oplus \mathbf{v}_i \) protrudes from the container \( C \) or not, Gomes and Oliveira (2002; 2006) introduced the inner-fit rectangle \( \text{IFR}(C, P_i) \), which is equivalent to \( \text{NFP}(C, P_i) \) (i.e., polygon \( P_i \oplus \mathbf{v}_i \) is contained in \( C \) if and only if \( \mathbf{v}_i \in \text{IFR}(C, P_i) \)).

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 \( (\mathbf{v}, \mathbf{o}) \) by a construction algorithm, which we call \textsc{Construct} (it is explained in Section 5), and computes the container length \( L \) so that it contains all polygons \( P_i \) \( (1 \leq i \leq n) \) 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 \( r_{\text{dec}} \) and \( r_{\text{inc}} \), respectively. If the current solution \( (\mathbf{v}, \mathbf{o}) \) is feasible, then it shrinks the container length \( L \) to \( (1 - r_{\text{dec}})L \) and relocates protruding polygons \( P_i \) at random positions in the container \( C(W, L) \); otherwise it extends the container length \( L \) to \( (1 + r_{\text{inc}})L \). 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 \( \text{OMP}(L) \), which we call \textsc{MinimizeOverlap}. The details of \textsc{MinimizeOverlap} is explained in Section 6. Algorithm FITS uses the above basic rule with slight modifications so that the current container length \( L \) satisfies \( L_{\text{LB}} \leq L < L^* \) in the search, where \( L^* \) is the minimum container length of feasible placements obtained so far, and \( L_{\text{LB}} \) is a lower bound of \( L \) defined by

\[
L_{\text{LB}} = \max \left\{ \frac{\sum_{i=1}^{n} (\text{area of } P_i)}{W}, \max_{1 \leq i \leq n} l_{i_{\text{min}}} \right\}, \tag{12}
\]

where

\[
l_{i_{\text{min}}} = \min_{o \in O_i} l_i(o), \tag{13}
\]

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

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 \( \mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n) \) and orientations \( \mathbf{o} = (o_1, o_2, \ldots, o_n) \) of all polygons \( P_i \) \( (1 \leq i \leq n) \).

**Step 1:** Generate an initial solution \( (\mathbf{v}, \mathbf{o}) \) by \textsc{Construct}, and set \( L_{\text{LB}} \) by (12).

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

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

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

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

**Step 2:** If \( k > n \) holds, then output \((v, o)\) and halt; otherwise set \( d \leftarrow e_x \) and initialize the position \( v_{\sigma(k)} \) 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 \( v'_{\sigma(k)} = v_{\sigma(k)} + td \ (t \leq 0) \) of \( P_{\sigma(k)} \) with the minimum \( t \). If \( v'_{\sigma(k)} = v_{\sigma(k)} \) holds, then set \( k \leftarrow k + 1 \) and return to Step 2; otherwise set \( v_{\sigma(k)} \leftarrow v'_{\sigma(k)} \).

**Step 4:** If \( d = e_x \) holds, then set \( d \leftarrow e_y \); otherwise set \( d \leftarrow e_x \). 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 \( v_k \) be the current position of \( P_k \) to be translated. We now explain how the algorithm finds the
new feasible position \( v'_k = v_k + td \) \((t \leq 0)\) with the minimum \( t \) when it is translated to the left (i.e., \( d = e_x \)). The algorithm for \( d = e_y \) is similar and is omitted. Among all positions on the half-line \( v_k + td \) \((t \leq 0)\), let

\[
N_0^- = \{t \mid v_k + td \in \text{IFR}(C, P_k), t \leq 0\}
\]

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

\[
N_j^- = \{t \mid v_k + td - v_j \notin \text{NFP}(P_j, P_k), t \leq 0\}
\]

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

\[
q_k = \sum_{j=1}^{k-1} q_{kj}
\]

and \( q_{kj} \) is the number of edges of no-fit polygon \( \text{NFP}(P_j, P_k) \). 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 \( \text{NFP}(P_j, P_k) \) (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 \((v, o)\) be the current solution. Its neighborhood \( \text{NB}(v, o) \) is defined as the set of solutions obtainable by setting a new orientation
$o_k' \in O_k$ of each $P_k$ $(1 \leq k \leq n)$ and applying an operation called \textsc{SearchNeighbor} to find a new position $v_k'$.

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

$$\tilde{F}(v, o) = \sum_{1 \leq i < j \leq n} w_{ij} \cdot f_{ij}(v_i, v_j, o_i, o_j), \quad (18)$$

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

$$\tilde{F}_k(v_k', o_k') = \sum_{1 \leq i \leq n, j \neq k} w_{ij} \cdot f_{ij}(v_k', v_j, o_k', o_j) \quad (19)$$

is as small as possible. For this, \textsc{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 \textsc{SearchNeighbor} proceeds. For each translation of the polygon $P_k(o_k')$ in a specified direction $d \in \{e_x, e_y\}$, let

$$N_0 = \{t \mid v_k + t d \in \text{IFR}(C, P_k(o_k')), t \in \mathbb{R}\} \quad (20)$$

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

\textbf{Algorithm \textsc{SearchNeighbor}}($P_k, o_k'$)

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

\textbf{Output:} The new position $v_k'$ of polygon $P_k(o_k')$.

\textbf{Step 1:} Set $v_k' \leftarrow v_k$. Find the valid position $v_k'' = v_k' + t e_x$ ($t \in N_0$) of $P_k(o_k')$ that minimizes the overlap penalty function $\tilde{F}_k(v_k' + t e_x, o_k')$. If $\tilde{F}_k(v_k'', o_k') < \tilde{F}_k(v_k', o_k')$ holds, then set $v_k' \leftarrow v_k''$.  

Set $d \leftarrow e_y$.

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

\textbf{Step 3:} If $d = e_x$ holds, then set $d \leftarrow e_y$; otherwise set $d \leftarrow e_x$. 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 \((v, o)\) with some overlapping polygons, and repeatedly replaces the current solution \((v, o)\) with a better solution \((v', o')\) (obtained by SEARCHNEIGHBOR) in the neighborhood NB\((v, o)\) with the first admissible move strategy. That is, if the algorithm finds an improved solution \((v', o') \in \text{NB}(v, o)\) with \(\bar{F}(v', o') < \bar{F}(v, o)\), then it immediately replaces the current solution \((v, o)\) with \((v', o')\). If no overlapping polygon exists in the current solution \((v, o)\) or no better solution is found in the neighborhood \(\text{NB}(v, o)\), then it outputs the current solution \((v, o)\) (as a locally optimal solution) and the best solution \((v^*, o^*)\) 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-neighbors, 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-neighbors.

We define the sub-neighborhood \(\text{NB}_k(v, o)\) \((1 \leq k \leq n)\) of the current solution \((v, o)\) as the set of solutions obtainable by trying each orientation \(o'_k \in O_k\) and applying \(\text{SEARCHNEIGHBOR}\) to \(P_k\). The algorithm first sets all sub-neighbors \(\text{NB}_k(v, o)\) \((1 \leq k \leq n)\) to be active. If no improvement has been made in an active sub-neighborhood \(\text{NB}_k(v, o)\), then the algorithm inactivates it. If it finds an improved neighbor solution \((v', o')\) with \(\bar{F}(v', o') < \bar{F}(v, o)\) in an active sub-neighborhood \(\text{NB}_k(v, o)\) (i.e., by moving \(P_k\)), then it activates all sub-neighbors \(\text{NB}_j(v, o)\) 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-neighbors \(\text{NB}_k(v, o)\), and \((\tilde{v}, \tilde{o})\) denotes the locally optimal solution measured by the weighted overlap penalty function \(\bar{F}\).

**Algorithm IMPROVE \((L, v, o)\)**

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

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

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

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

**Step 3:** Apply \(\text{SEARCHNEIGHBOR}(P_k, o'_k)\) for an orientation \(o'_k \in O\) to obtain a neighbor solution \((v', o')\). If \(F(v', o') < F(v^*, o^*)\) holds, then set \((v^*, o^*) \leftarrow (v', o')\). If \(\bar{F}(v', o') < \bar{F}(\tilde{v}, \tilde{o})\) holds, then set \((\tilde{v}, \tilde{o}) \leftarrow (v', o')\) 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 \(F(v^*, o^*) = 0\) (i.e., no overlapping polygon exists), then output \((v^*, o^*)\) and \((\tilde{v}, \tilde{o})\) 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 \(\text{SEARCHNEIGHBOR}(P_k, o'_k)\) and return to Step 2.

### 6.2 Fast neighborhood search

In this section, we describe the core part of \(\text{SEARCHNEIGHBOR}\), i.e., how it finds a new position when a polygon \(P_k(o'_k)\) is translated in a specified direction \(d\). Recall that the new position \(v'_k = v_k + td\) minimizes the overlap penalty function \(\bar{F}_k(v_k + td, o'_k)\) 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., \(d = e_x\)). The case of \(d = e_y\) is similar and is omitted. By definition (19), the overlap penalty function
\[ F_k(v_k + td, o'_j) \] is decomposed into \( f_{kj}(v_k + td, v_j, o'_k, o_j) \) for pairs of polygons \( P_k(o'_k) \) and \( P_j(o_j) \) \((j \neq k)\). Let 

\[ I_{kj} = \{ t \mid v_k + td - v_j \in \text{NFP}(P_j(o_j), P_k(o'_k)) \} \]  

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 \( f_{kj}(v_k + td, v_j, o'_k, o_j) \) is zero for all \( t \in \mathbb{R} \). The algorithm first determines whether \( I_{kj} \neq \emptyset \) 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 \( y_k^{\min} \) and \( y_k^{\max} \) \((1 \leq l \leq n)\) of \( P_l(o_l) \) at location \( v_l \) are defined by 

\[
 y_l^{\min} = \min \{ y \mid (x, y) \in P_l(o_l) \oplus v_l \}, \tag{22} 
\]

\[
 y_l^{\max} = \max \{ y \mid (x, y) \in P_l(o_l) \oplus v_l \}, \tag{23} 
\]

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 \( f_{kj}(v_k + td, v_j, o'_k, o_j) \) (defined in (9)) for a given \( t \), where \( I_{kj} \neq \emptyset \) is assumed. See Figure 8. If \( t \not\in I_{kj} \) holds, then \( f_{kj}(v_k + td, v_j, o'_k, o_j) = 0 \). If \( t \in I_{kj} \) holds, then the overlap penalty \( f_{kj}(v_k + td, v_j, o'_k, o_j) \) 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 

\[
 \rho_{kj}^{\text{H}}(v_k + td, v_j, o'_k, o_j) = \min \{ |s| \mid \text{int}(P_j(o_j) \oplus v_j) \cap (P_k(o'_k) \oplus (v_k + td) \oplus se_x) = \emptyset \}, \tag{24} 
\]

where \( \text{int}(S) \) denotes the interior of a polygon \( S \). The right hand side is equivalent to the minimum horizontal distance \(|s|\) from the point \( v_k + td - v_j \) to the nearest boundary of no-fit polygon \( \partial\text{NFP}(P_j(o_j), P_k(o'_k)) \).

The set \( I_{kj} \) is in general decomposed into a number of separated intervals \( (t_{kj1}^{\text{min}}, t_{kj1}^{\text{max}}), (t_{kj2}^{\text{min}}, t_{kj2}^{\text{max}}), \ldots, (t_{kjm_{kj1}}^{\text{min}}, t_{kjm_{kj1}}^{\text{max}}) \) \((\text{since polygons may not be convex})\), where \( m_{kj}(\leq \frac{mn}{2}) \) 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 \( (t_{kj1}^{\text{min}}, t_{kj1}^{\text{max}}) \), then \( \rho_{kj}^{\text{H}}(v_k + td, v_j, o'_k, o_j) \) is computed by 

\[
 \rho_{kj}^{\text{H}}(v_k + td, v_j, o'_k, o_j) = \min \{ t - t_{kj1}^{\text{min}}, t_{kj1}^{\text{max}} - t \}. \tag{25} 
\]

We also denote upward and downward penetration depths by 

\[
 \rho_{kj}^{\text{U}}(v_k + td, v_j, o'_k, o_j) = \min \{ |s| \mid \text{int}(P_j(o_j) \oplus v_j) \cap (P_k(o'_k) \oplus (v_k + td) \oplus se_y) = \emptyset, s \geq 0 \}, \tag{26} 
\]
The overlap penalty function $f_{kj}(v_k + td, v_j, o'_k, o_j)$ when $P_k(o'_k)$ moves horizontally.

$$
\rho_{kj}^H(v_k + td, v_j, o'_k, o_j) = \min\{|s| \mid \text{int}(P_j(o_j) \oplus v_j) \cap (P_k(o'_k) \oplus (v_k + td) \oplus se_y) = \emptyset, s \leq 0\},
$$

respectively, which is equivalent to the minimum distances $|s|$ from the point $v_k + td - v_j$ to the nearest boundary of no-fit polygon $\partial NFP(P_j(o_j), P_k(o'_k))$ in upward and downward directions, respectively.

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

$$
f_{kj}(v_k + td, v_j, o'_k, o_j) = \min\{\rho_{kj}^H(v_k + td, v_j, o'_k, o_j), \rho_{kj}^U(v_k + td, v_j, o'_k, o_j)\},
$$

which is given by the lower envelope of the three directional penetration depths $\rho_{kj}^H(v_k + td, v_j, o'_k, o_j)$, $\rho_{kj}^U(v_k + td, v_j, o'_k, o_j)$ and $\rho_{kj}^D(v_k + td, v_j, o'_k, o_j)$ (Figure 8 (c)).

The horizontal penetration depth $\rho_{kj}^H(v_k + td, v_j, o'_k, o_j)$ is a piecewise linear function over the sorted lists of intervals $(t_{kj}^{min}, t_{kj}^{max})$ ($1 \leq l \leq m_{kj}$) of $I_{kj}$ along the $t$-axis. The upward and downward penetration depths $\rho_{kj}^U(v_k + td, v_j, o'_k, o_j)$ and $\rho_{kj}^D(v_k + td, v_j, o'_k, o_j)$ are also piecewise linear functions over the sorted lists of edges of no-fit polygon $NFP(P_j(o_j), P_k(o'_k))$ 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 $NFP(P_j(o_j), P_k(o'_k))$. 

Figure 8: Computing the overlap penalty function $f_{kj}(v_k + td, v_j, o'_k, o_j)$ when $P_k(o'_k)$ moves horizontally.
Figure 9: Computing directional penetration depths \( \rho_{kj}^H(v_k + td, v_j, o'_k, o_j) \) and \( \rho_{kj}^D(v_k + td, v_j, o'_k, o_j) \) using two trapezoidal partitions \( T^h \) and \( T^v \) of a no-fit polygon.

The computation of three directional penetration depths \( \rho_{kj}^H(v_k + td, v_j, o'_k, o_j) \), \( \rho_{kj}^U(v_k + td, v_j, o'_k, o_j) \) and \( \rho_{kj}^D(v_k + td, v_j, o'_k, o_j) \) 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 \( NFP(P_j(o_j), P_k(o'_k)) \) 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 \( q_{k,j} \) of \( NFP(P_j(o_j), P_k(o'_k)) \). These two partitions are illustrated in Figure 9 (b). We note that \( T^h \) satisfies \( \bigcup_{i=1}^{r_h} T^h_i = NFP(P_j(o_j), P_k(o'_k)) \) and \( T^h \cap T^v = \emptyset \) (1 \( \leq a < b \leq r_h \)), and \( T^v \) satisfies analogous conditions. The computation time of \( NFP(P_j(o_j), P_k(o'_k)) \) is \( O(q_{k,j}) \) (see Section 3), and that of its trapezoidal partitions is also \( O(q_{k,j}) \) because it can be done in linear time to the number of edges of \( NFP(P_j(o_j), P_k(o'_k)) \) (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 \( w + te_x \ (w \in \mathbb{R}^2, t \in \mathbb{R}) \) crossing both \( T^h_a \) and \( T^h_b \), and (ii) \( \min\{t \mid w + te_x \in T^h_a\} \leq \min\{t \mid w + te_x \in T^h_b\} \). 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 \( NFP(P_j(o_j), P_k(o'_k)) \). See Figure 9 (c). For the horizontal penetration depth \( \rho_{kj}^H(v_k + td, v_j, o'_k, o_j) \), the algorithm first constructs the list of intervals \( (t^{'min}_{kj1}, t^{max}_{kj1}), (t^{'min}_{kj2}, t^{max}_{kj2}), \ldots, (t^{'min}_{kj_{m_{kj}}}, t^{max}_{kj_{m_{kj}}} \) of \( I_{kj} \) by scanning intersections of the horizontal line \( v_k + td - v_j \) (\( t \in \mathbb{R} \)) with trapezoids \( T^h_i \) (1 \( \leq i \leq r_h \)), according to the total order \( \preceq_{x*} \) on \( \{T^h_i\} \). This computation can be done in \( O(q_{k,j}) \) time because checking the intersection between horizontal line \( v_k + td - v_j \) (\( t \in \mathbb{R} \))
and a trapezoid \( T^+_i \) is done in \( O(1) \) time. It then computes function (25) on all the intervals. For the upward penetration depth \( \rho^+_i(v_k + td, v_j, o'_k, o_j) \), it scans all the trapezoids \( T^+_i \) (1 \( \leq i \leq r_v \)), which cross the horizontal line \( v_k + td - v_j \) (\( t \in \mathbb{R} \)) according to the total order \( \leq^*_v \) on \( \{T^+_i\} \). 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 \( \rho^-_i(v_k + td, v_j, o'_k, o_j) \). In this way, the computation for each directional penetration depth is done in \( O(q_{kj}) \) time. Then the overlap penalty function \( f_{kj}(v_k + td, v_j, o'_k, o_j) \) 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 \( \tilde{F}_k(v_k + td, o'_k) \) of (19) by summing up the overlap penalty functions \( f_{kj}(v_k + td, v_j, o'_k, o_j) \) 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 \( O(\sum q_{kj}) \log n = O(q_k \log n) \), 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 \( (v, o) \) with some overlapping polygons, where the penalty weights \( w_{ij} \) are initialized to 1.0. Whenever **IMPROVE** stops at a locally optimal solution \( (v, o) \), 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 \( v_i \) and \( v_j \) overlaps \( (i.e., f_{ij}(v_i, v_j, o_i, o_j) > 0) \), the algorithm increases the penalty weight \( w_{ij} \) as follows:

\[
 w_{ij} \leftarrow w_{ij} + \frac{f_{ij}(v_i, v_j, o_i, o_j)}{\max_{1 \leq k < l \leq n} f_{kl}(v_k, v_l, o_k, o_l)}.
\]  

(29)

By applying (29) repeatedly, the current solution \( (v, o) \) 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 \( (v^*, o^*) \) is the best solution obtained so far measured by the original overlap penalty function \( F \).

**Algorithm MinimizeOverlap** \((L, v, o)\)

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

**Output:** The best solution \( (v^*, o^*) \) 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 \( (\tilde{v}, \tilde{o}) \leftarrow (v, o) \) and \( (v^*, o^*) \leftarrow (v, o) \).

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

**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)).

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

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

Step 5: If \( \text{iter} \geq \text{max}\_\text{iter} \) holds, then output \((v^*,o^*)\) and halt; otherwise set \( \text{iter} \leftarrow \text{iter} + 1 \) 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. 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 \((v,o)\) with the container length \( L = L(v,o) \) 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 \( r_{dec} = 0.02, r_{inc} = 0.005 \) and \( \text{max}\_\text{iter} = 200 \) 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

---

1ESICUP: http://paginas.fe.up.pt/~esicup/
Table 2: The efficiency in % of the solutions obtained by FITS and other algorithms.

<table>
<thead>
<tr>
<th>Instance</th>
<th>BLF</th>
<th>SAHA</th>
<th>2DNest</th>
<th>ILSQN</th>
<th>FITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albano</td>
<td>–</td>
<td>87.43</td>
<td>84.70</td>
<td>87.44</td>
<td>86.96</td>
</tr>
<tr>
<td>Dagli</td>
<td>83.7</td>
<td>87.15</td>
<td>85.38</td>
<td>85.98</td>
<td>85.31</td>
</tr>
<tr>
<td>Dighe1</td>
<td>–</td>
<td>*100.00</td>
<td>82.13</td>
<td>99.86</td>
<td>93.93</td>
</tr>
<tr>
<td>Dighe2</td>
<td>–</td>
<td>*100.00</td>
<td>84.17</td>
<td>99.95</td>
<td>93.11</td>
</tr>
<tr>
<td>Fu</td>
<td>86.9</td>
<td>90.96</td>
<td>87.17</td>
<td>*91.84</td>
<td>*90.93</td>
</tr>
<tr>
<td>Jakobs1</td>
<td>82.6</td>
<td>78.89</td>
<td>75.79</td>
<td>89.07</td>
<td>*88.90</td>
</tr>
<tr>
<td>Jakobs2</td>
<td>74.8</td>
<td>77.28</td>
<td>74.66</td>
<td>80.41</td>
<td>80.28</td>
</tr>
<tr>
<td>Mao</td>
<td>79.5</td>
<td>82.54</td>
<td>80.72</td>
<td>*85.15</td>
<td>82.67</td>
</tr>
<tr>
<td>Marques</td>
<td>86.5</td>
<td>88.14</td>
<td>86.88</td>
<td>89.17</td>
<td>88.73</td>
</tr>
<tr>
<td>Shapes0</td>
<td>60.5</td>
<td>66.50</td>
<td>63.20</td>
<td>67.09</td>
<td>65.42</td>
</tr>
<tr>
<td>Shapes1</td>
<td>66.5</td>
<td>71.25</td>
<td>68.63</td>
<td>73.84</td>
<td>71.74</td>
</tr>
<tr>
<td>Shapes2</td>
<td>77.7</td>
<td>83.60</td>
<td>81.41</td>
<td>81.21</td>
<td>79.89</td>
</tr>
<tr>
<td>Shirts</td>
<td>84.6</td>
<td>86.79</td>
<td>85.67</td>
<td>86.33</td>
<td>85.73</td>
</tr>
<tr>
<td>Swim</td>
<td>68.4</td>
<td>74.37</td>
<td>72.28</td>
<td>71.53</td>
<td>70.27</td>
</tr>
<tr>
<td>Trousers</td>
<td>88.5</td>
<td>*89.96</td>
<td>89.02</td>
<td>89.84</td>
<td>*89.29</td>
</tr>
</tbody>
</table>

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).

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

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


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:

Geometry* 6, 485–524.

Dean, H. T., Tu, Y., Raffensperger, J. F., 2006. An improved method for calculating the no-fit

Dobkin, D., Hershberger, J., Kirkpatrick, D., Suri, S., 1993. Computing the intersection-depth of


Egeblad, J., Nielsen, B. K., Odgaard, A., 2007. Fast neighborhood search for two- and three-

of Operational Research* 141, 359–370.

Gomes, A. M., Oliveira, J. F., 2006. Solving irregular strip packing problems by hybridising simu-
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

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 Uni-
versity.

Kim, Y. J., Lin, M. C., Manocha, D., 2004. Incremental penetration depth estimation between con-
vex polytopes using dual-space expansion. *IEEE Transactions on Visualization and Computer

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


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

Voudouris, C., Tsang, E., 1999. Guided local search and its application to the traveling salesman
Figure 10: The best solutions obtained by FITS.