International Symposium on Scheduling 2006
July 18 - 20, 2006, Tokyo, Japan
5B2

AN ITERATED LOCAL SEARCH ALGORITHM BASED ON NONLINEAR PROGRAMMING FOR THE IRREGULAR STRIP PACKING PROBLEM

Takashi Imamichi and Hiroshi Nagamochi
Department of Applied Mathematics and Physics,
Kyoto University
Sakyo-ku, Kyoto, 606-8501, Japan
{ima, nag}@amp.i.kyoto-u.ac.jp

Mutsunori Yagiura
Department of Computer Science and Mathematical Informatics,
Nagoya University,
Furocho, Chikusaku, Nagoya, 464-8603, Japan
yagiura@nagoya-u.jp

Abstract

The irregular strip packing problem is a combinatorial optimization problem that asks to place a set of 2-dimensional polygons within a rectangular container so that no two polygons overlap each other and no polygon protrudes from the container, where each polygon is not necessarily convex. The container has a fixed width, while its length can change so that all polygons are placed in it. The objective is to find a layout that minimizes the length of the container. This problem has many applications in material industry such as paper and textile industries, where raw materials are usually given in rolls.

We propose a new separation algorithm based on nonlinear programming, and an algorithm that swaps two polygons in a sophisticated way; it tries to find their positions with the least overlap. We incorporate these algorithms as components in an iterated local search algorithm for the overlap minimization problem and then develop an algorithm for the irregular strip packing problem using the iterated local search algorithm. Computational comparisons on representative instances disclose that our algorithm is competitive with other existing algorithms. Moreover, our algorithm updates several best known results.

Keywords: irregular strip packing problem, iterated local search, unconstrained nonlinear programming.

1. Introduction

The *irregular strip packing problem* is a combinatorial optimization problem that asks to place a set of 2-dimensional polygons within a rectangular container so that no two polygons overlap each other and no polygon protrudes from the container, where each polygon is not necessarily convex. The container has a fixed width, while its length can change so that all polygons are placed in it. The objective is to find a layout that minimizes the length of the container. This problem has a few variations depending on rotations of polygons: (1) rotations of any angle are allowed, (2) finite number of angles are allowed, (3) no rotation is allowed. Among them, we deal with case (2). Note that case (3) is a special case of (2) in which the number of given orientations for each polygon is one. The irregular strip packing problem has many applications in material industry such as paper and textile industries, where raw materials are usually given in rolls. In textile industry, rotations are usually restricted to 180 degrees because textiles have the grain and may have a drawing pattern. The irregular strip packing problem is known to be NP-hard even without rotation.

Adamowicz and Albano (1976) proposed an algorithm that first partitions a given polygons into several subsets of polygons, then generates for each subset a rectangle enclosure in which the polygons in the subset are placed compactly (i.e., being with a little wasted space), and finally finds a layout of these enclosures. Albano and Sapuppo (1980) gave an algorithm that places polygons one by one at the bottom-left position according to a sequence of input polygons, where they used tree search to obtain a good sequence. Mathematical programming was also used for this problem. Based on linear programming, Li and Milenkovic (1995) proposed *compaction* and *separation* algorithms that reduce the overlap and the length of the container by perturbing the current positions of all polygons simultaneously. Afterwards, Bennell and Dowsland (2001) combined the bottom-left method and *compaction* algorithm to obtain a better algorithm. Gomes and Oliveira (2006) hybridised the bottom-left heuristic and linear programming based *compaction* and *separation* algorithms. They further incorporated the method with simulated annealing. Burke, *et al.* (2005) developed a bottom-left-fill heuristic algorithm and utilized it with hill climbing or tabu search to obtain good solutions quickly. Egeblad, *et al.* (2006) developed an

efficient method that finds a layout with minimum overlap position when a polygon is translated in a given direction, and they utilized it in guided local search. See a review by Hopper and Turton (2001) for more on the strip packing problem including the irregular strip packing problem.

In this paper, we propose a new *separation* algorithm based on nonlinear programming. We also give an algorithm that swaps two polygons in a sophisticated way; it tries to find their positions with the least overlap provided that the positions of other polygons in a given layout are fixed. We incorporate these algorithms as components in an iterated local search algorithm whose objective is to minimize the total amount of overlap and protrusion of a layout, where a layout may not be completely contained in the container during the algorithm. We then develop an algorithm for the irregular strip packing problem using the iterated local search algorithm, which we call ILSQN because we use the quasi-Newton method in the iterated local search algorithm. Computational comparisons on representative benchmark instances disclose that our algorithm is competitive with other existing algorithms. Moreover, our algorithm updates several best known results.

2. Formulation

For the irregular strip packing problem, we are given a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of polygons, a set $\mathcal{O} = \{O_1, \ldots, O_n\}$ of orientations where O_i $(1 \le i \le n)$ is a set of possible orientations of P_i, and a rectangular container $C = C(W, L)$ with width W and length L, where W is a constant and $L \ge 0$ is a variable. Polygons in $\mathcal{P}$ may not be convex.

We denote polygon $P_i \in \mathcal{P}$ rotated by $o \in O_i$ degrees by $P_i(o)$, which may be written as P_i for simplicity when the orientation is not specified or clear from the context. Let S be polygon $P_i(o)$ or rectangle C. For convenience, we regard each of polygons $P_i(o)$ $(i = 1, \ldots, n)$ and rectangle C as the set of points inside it including the points on the boundary. Let $\mathrm{int}(S)$ be the interior of S, ∂S be the boundary of S, $\overline{S}$ be the complement of S, and $\mathrm{cl}(S)$ be the closure of S. We describe translations of polygons by Minkowski sums. Let $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ $(i = 1, \ldots, n)$ be a translation vector for P_i. The polygon obtained by translating polygon P_i by $\boldsymbol{x}_i$ is $P_i \oplus \boldsymbol{x}_i = \{\boldsymbol{p} + \boldsymbol{x}_i \mid \boldsymbol{p} \in P_i\}$. Recall that $L \ge 0$ is the length of the container C, which is a decision variable to be minimized. Then the irregular strip packing problem is formally described as follows:

$$
\begin{aligned}
\text{minimize} \quad & L \\
\text{subject to} \quad & \mathrm{int}(P_i(o_i) \oplus \boldsymbol{x}_i) \cap (P_j(o_j) \oplus \boldsymbol{x}_j) = \emptyset, \\
& \qquad\qquad 1 \le i < j \le n, \\
& P_i(o_i) \oplus \boldsymbol{x}_i \subseteq C(W, L), \quad 1 \le i \le n, \\
& L \in \mathbb{R}_+, \quad \boldsymbol{x}_i \in \mathbb{R}^2, \quad 1 \le i \le n, \\
& o_i \in O_i, \quad 1 \le i \le n.
\end{aligned} \tag{1}
$$

We represent a solution of this problem with two n-tuples $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\boldsymbol{o} = (o_1, \ldots, o_n)$. Note that a solution $(\boldsymbol{x}, \boldsymbol{o})$ uniquely determines the layout of the polygons; i.e., the minimum length L is the x-coordinate of the rightmost point of the polygons placed by $(\boldsymbol{x}, \boldsymbol{o})$.

3. Overlap minimization based on nonlinear program

Our *separation* algorithm is based on nonlinear program. We move all polygons simultaneously to minimize the total amount of overlap and protrusion, where the length of the container is fixed. In this section, we formulate the overlap minimization problem as an unconstrained nonlinear programming problem, and gives the way of computing the objective function and its gradient.

3.1 The overlap minimization problem

Our objective is to find a feasible solution of the problem (1) with a given length L of the container. For this purpose, we allow solutions to have some polygons which overlap and/or protrude from the container, and penalize the amount of overlap and protrusion so that a solution with penalty zero gives a feasible layout for the irregular strip packing problem.

In this section, we fix the orientations $\boldsymbol{o} = (o_1, \ldots, o_n)$ of all polygons, and omit writing it explicitly for simplicity. Let $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ be a list of translation vectors of all polygons, $f_{ij}(\boldsymbol{x})$ be a function evaluating the amount of overlap of P_i and P_j, and $g_i(\boldsymbol{x})$ be a function evaluating the amount of protrusion of P_i from the container. Various choices of functions are possible for $f_{ij}(\cdot)$ and $g_i(\cdot)$, and we will choose suitable functions for $f_{ij}(\cdot)$ and $g_i(\cdot)$ in Section 3.4. Now we formulate the overlap minimization problem by

$$
\begin{aligned}
\text{minimize} \quad & F(\boldsymbol{x}) = \sum_{1 \le i < j \le n} f_{ij}(\boldsymbol{x}) + \sum_{1 \le i \le n} g_i(\boldsymbol{x}) \\
\text{subject to} \quad & \boldsymbol{x} \in \mathbb{R}^{2n}.
\end{aligned} \tag{2}
$$

This is an unconstrained nonlinear programming problem. For defining suitable functions $f_{ij}(\cdot)$ and $g_i(\cdot)$, we introduce some notions such as the no-fit polygon and the penetration depth in the next subsections. Our separation algorithm invokes the quasi-Newton method to compute a locally optimal solution to problem (2) by using the current layout as an initial solution.

When we need to clarify the orientation $\boldsymbol{o}$ explicitly as in Section 4, we write the objective function $F(\boldsymbol{x})$ as $F(\boldsymbol{x}, \boldsymbol{o})$.

3.2 The no-fit polygon

The *no-fit polygon* (NFP) is a data structure that is often used in algorithms for the irregular strip packing problem. It is also used for other problems such as robotics, in which the no-fit polygon is called configuration-space obstacle. Practical algorithms to calculate an NFP of two non-convex poly-

gons have been proposed, e.g., by Bennell, *et al.* (2001) and Ramkumar (1996).

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

$$\begin{aligned}\mathrm{NFP}(P_i, P_j) &= \mathrm{int}(P_i) \oplus \mathrm{int}(-P_j) \\ &= \{\boldsymbol{v} - \boldsymbol{w} \mid \boldsymbol{v} \in \mathrm{int}(P_i),\ \boldsymbol{w} \in \mathrm{int}(P_j)\}.\end{aligned}$$

The NFP have the following important properties:

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

Hence the problem of checking whether two polygons overlap or not becomes an easier problem of checking whether a point is in a polygon or not. Figure 1 shows an example of $\mathrm{NFP}(P_i, P_j)$ of two polygons P_i and P_j.

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

$$\begin{aligned}\mathrm{NFP}(\overline{C}, P_i) &= \mathrm{int}(\overline{C}) \oplus \mathrm{int}(-P_i) \\ &= \{\boldsymbol{v} - \boldsymbol{w} \mid \boldsymbol{v} \in \mathbb{R}^2 \setminus C,\ \boldsymbol{w} \in \mathrm{int}(P_i)\},\end{aligned}$$

which is the complement of a rectangle whose boundary is the trajectory of the reference point of P_i when we slide P_i inside the container C.

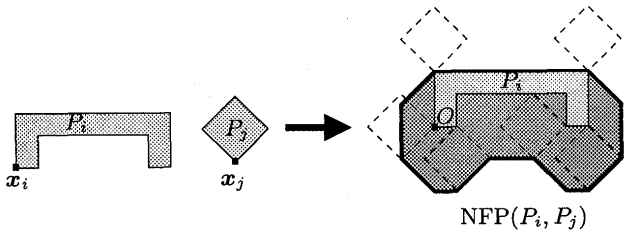

Fig. 1 Illustration of $\mathrm{NFP}(P_i, P_j)$

3.3 The penetration depth

The *penetration depth* (also known as the intersection depth) is a function used for robotics, computer vision and so on (Agarwal, *et al.*, 2000; Dobkin, *et al.*, 1993). The penetration depth $\delta(P_i, P_j)$ of two overlapping polygons P_i and P_j is defined to be the minimum translational distance to separate them. If two polygons do not overlap, their penetration depth is zero. Formally, the penetration depth of two polygons P_i and P_j is defined by

$$\delta(P_i, P_j) = \min\{\|\boldsymbol{z}\| \mid \mathrm{int}(P_i) \cap (P_j \oplus \boldsymbol{z}) = \emptyset,\ \boldsymbol{z} \in \mathbb{R}^2\},$$

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

We can separate two polygons P_i and P_j by translating the reference point of P_j to a point on $\partial\mathrm{NFP}(P_i, P_j)$. Hence $\delta(P_i \oplus \boldsymbol{x}_i, P_j \oplus \boldsymbol{x}_j)$ is the minimum distance from $\boldsymbol{x}_j - \boldsymbol{x}_i$ to $\partial\mathrm{NFP}(P_i, P_j)$.

3.4 The amount of overlap

We define functions $f_{ij}(\cdot)$ and $g_i(\cdot)$ using the penetration depth. To represent the amount of overlap between P_i and P_j, $f_{ij}(\cdot)$ is defined by

$$f_{ij}(\boldsymbol{x}) = \delta(P_i \oplus \boldsymbol{x}_i, P_j \oplus \boldsymbol{x}_j)^m,\quad 1 \le i < j \le n,$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and m is a positive parameter. Similarly $g_i(\boldsymbol{x})$ is defined by

$$g_i(\boldsymbol{x}) = \delta(\mathrm{cl}(\overline{C}), P_i \oplus \boldsymbol{x}_i)^m,\quad 1 \le i \le n,$$

where $\mathrm{cl}(\overline{C})$ is the exterior region of C and its boundary.

In order to apply efficient algorithms for solving the nonlinear program to the overlap minimization problem, we need to compute the values of $f_{ij}(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$ and their gradients for a given solution $(\boldsymbol{x}, \boldsymbol{o})$. We explain below how we realize such computation, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the translation vectors of P_i and P_j respectively and we denote $\boldsymbol{v} = \boldsymbol{x}_j - \boldsymbol{x}_i$ for convenience. We consider how to compute $f_{ij}(\boldsymbol{x})$ and its gradient. We compute $g_i(\boldsymbol{x})$ and $\nabla g_i(\boldsymbol{x})$ similarly as in the case of $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$. There are three cases for the computation of $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$.

Case 1: the case in which the two polygons P_i and P_j do not overlap. This case is easy; we just return $f_{ij}(\boldsymbol{x}) = 0$ and $\nabla f_{ij}(\boldsymbol{x}) = \boldsymbol{0}$.

Case 2: the case in which the two polygons overlap (i.e., $f_{ij}(\boldsymbol{x}) > 0$) and the nearest point on $\partial\mathrm{NFP}(P_i, P_j)$ from $\boldsymbol{v}$ is unique. See an example in Figure 2. Let $\boldsymbol{w}$ be the nearest point and let $\boldsymbol{z} = \boldsymbol{w} - \boldsymbol{v}$. Because the variable $\boldsymbol{x}$ is a list of n 2-dimensional vectors, $\nabla f_{ij}(\boldsymbol{x})$ is also such a list; hence we denote $\nabla f_{ij}(\boldsymbol{x}) = (\nabla_1 f_{ij}(\boldsymbol{x}), \ldots, \nabla_n f_{ij}(\boldsymbol{x}))$, where $\nabla_k = (\partial/\partial x_{k1}, \partial/\partial x_{k2})$ for all $1 \le k \le n$. Then, $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$ for $1 \le i < j \le n$ are given by

$$\begin{aligned} f_{ij}(\boldsymbol{x}) &= \|\boldsymbol{z}\|^m, \\ \nabla_i f_{ij}(\boldsymbol{x}) &= -\nabla_j f_{ij}(\boldsymbol{x}) = m\|\boldsymbol{z}\|^{m-2}\boldsymbol{z}, \\ \nabla_k f_{ij}(\boldsymbol{x}) &= 0,\quad k \in \{1, \ldots, n\} \setminus \{i, j\}. \end{aligned} \tag{3}$$

Every $\nabla_k f_{ij}(\boldsymbol{x})$ except $\nabla_i f_{ij}(\boldsymbol{x})$ and $\nabla_j f_{ij}(\boldsymbol{x})$ is zero because only P_i and P_j have influence on their overlap.

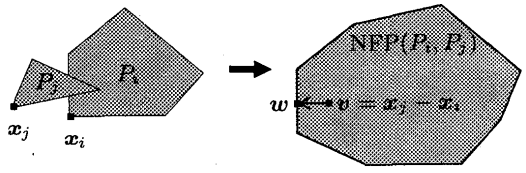

Fig. 2 The computation of $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$

Case 3: the case in which $f_{ij}(\boldsymbol{x}) > 0$ and the nearest point from $\boldsymbol{v}$ to $\partial\mathrm{NFP}(P_i, P_j)$ is not unique. In this case, $\nabla f_{ij}(\cdot)$ is not differentiable at $\boldsymbol{x}$; however, we choose one of the nearest points arbitrarily as $\boldsymbol{w}$ and calculate $\nabla f_{ij}(\boldsymbol{x})$ with (3) as in Case 2. Then this is one of the subgradients

of $f_{ij}(\boldsymbol{x})$. Note that Case 3 can occur only when $\boldsymbol{v}$ is on the medial axis of $\mathrm{NFP}(P_i, P_j)$.

The positive parameter m determines the differentiability of $f_{ij}(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$. When $\boldsymbol{v}$ is on $\partial\,\mathrm{NFP}(P_i, P_j)$, $f_{ij}(\boldsymbol{x})$ is differentiable for $m > 1$, while it is indifferentiable for $m \leq 1$. It is preferred that the objective function is differentiable for the quasi-Newton method. Moreover, $\nabla f_{ij}(\boldsymbol{x})$ in (3) becomes simpler for $m = 2$ because $\|z\|^{m-2}$ disappears. The situation is the same for $g_i(\boldsymbol{x})$, and hence we let $m = 2$ in our experiments.

4. Iterated local search

This section proposes an iterated local search algorithm for the overlap minimization problem (2) defined in the previous section. Iterated local search is one of the representative metaheuristic algorithms that repeats local search many times, where initial solutions for local search are generated by perturbing promising solutions obtained by then. The local search of our algorithm, which can be regarded as a separation algorithm, moves all polygons (usually) slightly to reduce overlap. For this purpose, we use the quasi-Newton method for the overlap minimization problem. For the perturbation in the iterated local search, we adopt the operation of swapping two polygons described in the next section.

4.1 The operation of swapping two polygons

We swap two polygons in our iterated local search algorithm to perturb locally optimal solutions. Instead of just exchanging two polygons P_i and P_j in their reference points, we attempt to find their positions with the least overlap. FINDBESTPOSITION(P_i) is a heuristic algorithm to find a minimum overlap position of a polygon P_i, without changing the positions of the other polygons, while considering all possible orientations $o \in O_i$ of P_i. For each point $\boldsymbol{v}$ of all vertices and intersections of $\partial\,\mathrm{NFP}(P_k \oplus \boldsymbol{x}_k, P_i(o))$ $k \in \{1, \ldots, n\} \setminus \{i\}$ and $\partial\,\mathrm{NFP}(\overline{C}, P_i(o))$, the heuristics computes the overlap of $P_i(o) \oplus \boldsymbol{v}$ with the other polygons, and finds the position with the least overlap, where $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is a list of the translation vectors of polygons, $\boldsymbol{o} = (o_1, \ldots, o_n)$ is a list of the orientations of polygons, and the amount of overlap is computed by the objective function $F(\boldsymbol{x}, \boldsymbol{o})$ of the overlap minimization problem (2). It repeats these operations for all orientations $o \in O_i$ of P_i and seeks the best position and orientation.

FINDBESTPOSITION has an important property: FINDBESTPOSITION(P_i) always finds a point $\boldsymbol{v}^\star \in \mathbb{R}^2$ and an orientation $o^\star \in O_i$ of polygon P_i such that $P_i(o^\star) \oplus \boldsymbol{v}^\star$ neither overlaps with the other polygons nor protrudes from the container C if there exists such a pair of a point and an orientation. However, FINDBESTPOSITION may miss the globally optimal position if there is no position whose overlap is zero.

SWAPTWOPOLYGONS(P_i, P_j) is an algorithm to swap two polygons P_i and P_j by using FINDBESTPOSITION. We first remove a polygon P_i from the container C, which results in making a hole in the layout. We next place a new polygon $P'(o_j) \oplus \boldsymbol{x}_j$, where $P' = P_j$, to prevent P_j from staying at the same place. Then we move a polygon P_j to a position computed by FINDBESTPOSITION(P_j), and remove P', where we expect that P_j moves into the hole. Finally, we place the removed polygon P_i by FINDBESTPOSITION(P_i).

4.2 Iterated local search for the overlap minimization problem

In this section, we formally describe the framework of our iterated local search algorithm called MINIMIZEOVERLAP for the overlap minimization problem (2).

We choose the first solution that minimizes the objective function of (2) among those searched by then as the incumbent solution that will be used for generating the next initial solution. MINIMIZEOVERLAP perturbs the incumbent solution by SWAPTWOPOLYGONS(P_i, P_j), where polygons P_i and P_j are randomly chosen, then calls the quasi-Newton method starting from the perturbed solution, and obtains a locally optimal solution. If the locally optimal solution has less overlap than the incumbent solution does, the locally optimal solution updates the incumbent solution. MINIMIZEOVERLAP stops these operations after failing to update the incumbent solution N_{om} (a parameter) consecutive calls to local search.

4.3 Initial solution

We generate an initial feasible layout of our algorithm for the irregular strip packing problem as follows. We assume that the length L of the container is long enough to place all polygons. We prepare a sequence of polygons in descending order of area and place polygons one by one in the order of the sequence, where the position of each polygon P_i is decided by FINDBESTPOSITION. If there are several positions with no overlap, we choose the bottom-left position (i.e., the position with the minimum x_{i1}, breaking ties with the minimum x_{i2}, where $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ is the translation vector of polygon P_i). After placing all polygons, we let L be the x-coordinate of the right most point of the resulting layout. Gomes and Oliveira (2006) also generated initial solutions in a similar way using a different sequence of polygons.

4.4 The entire algorithm for the strip packing problem

In this section, we describe the entire algorithm for the irregular strip packing problem, which we call ILSQN. It consists of two layers of computations. The inner layer is to find a feasible layout for a tentatively fixed length L of the container, which corresponds to MINIMIZEOVERLAP. The outer layer is to search the minimum feasible length L by shrinking or extending its left and/or right sides. We control the outer layer using parameters $r_{\mathrm{dec}}, r_{\mathrm{inc}} \in (0, 1)$, where r_{dec} and r_{inc} are the ratio we shrink or extend the length L of the container, respectively.

ILSQN first generates an initial solution, sets the length L of the container so that it contains all polygons and the both sides touch some polygons, and then shorten L by $L := (1 - r_{\text{dec}})L$. Then, ILSQN repeats the following two operations until a time limit is reached. ILSQN tries to reduce the overlap of the current solution $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$ by MINIMIZEOVERLAP. If ILSQN obtains a layout with no overlap, it shortens the length L of the container by $L := (1 - r_{\text{dec}})L$; otherwise, ILSQN extends the length by $L := (1 + r_{\text{inc}})L$. The entire algorithm of ILSQN is formally described in Algorithm 1.

Algorithm 1 : ILSQN($\mathcal{P}, \mathcal{O}, W, r_{\text{dec}}, r_{\text{inc}}$)

Generate an initial solution $(\boldsymbol{x}, \boldsymbol{o})$ by the algorithm in Section 4.3.
Let L be the minimum feasible length of the container for the initial solution $(\boldsymbol{x}, \boldsymbol{o})$.
Let $L_{\text{best}} := L$ and $(\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}}) := (\boldsymbol{x}, \boldsymbol{o})$.
Let $L := (1 - r_{\text{dec}})L$ and $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) := (\boldsymbol{x}, \boldsymbol{o})$.
while within a time limit **do**
  Let $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) :=$ MINIMIZEOVERLAP$(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$.
  if $F(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) = 0$ **then**
    Let $L_{\text{best}} := L$ and $(\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}}) := (\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$.
    Let $L := (1 - r_{\text{dec}})L$.
  else
    Let $L := (1 + r_{\text{inc}})L$.
    if $L > L_{\text{best}}$ **then**
      Let $L := (1 - r_{\text{dec}})L_{\text{best}}$.
      Let $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) := (\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}})$.
    end if
  end if
end while
Return L_{best} and $(\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}})$.

5. Computational results

5.1 Environment

Benchmark instances for the irregular strip packing problem are available online at EURO Special Interest Group on Cutting and Packing (ESICUP) website[1].

We implemented our algorithm in the C++ language, compiled it by GCC 4.0.2 and conducted computational experiments on a PC with a Pentium4 2.8GHz processor and 1GB memory. We adopt a quasi-Newton method package L-BFGS by Liu and Nocedal (1989) for the overlap minimization problem. L-BFGS has a parameter m_{BFGS} that is the number of BFGS corrections in L-BFGS. We set $m_{\text{BFGS}} = 6$ because $3 \leq m_{\text{BFGS}} \leq 7$ is recommended by Liu and Nocedal (1989).

A layout is judged to have no overlap when the objective function of (2) is less than $\epsilon = 10^{-10}W^2$ due to limited precision. Thus, our algorithm may generate layouts that have slight overlap.

[1]ESICUP: `http://www.apdio.pt/sicup/`

We set $r_{\text{dec}} = 0.04$, $r_{\text{inc}} = 0.01$ and $N_{\text{om}} = 200$ for the computational experiments of all benchmark instances in Section 5.2.

5.2 Results

In this section, we show the computational results of our algorithm ILSQN and compare it with other existing algorithms. We run algorithm ILSQN ten times for each instance and compare our results with those reported by Gomes and Oliveira (2006), Burke, *et al.* (2005) and Egeblad, *et al.* (2006). Table 1 shows the best and average length and efficiency in % of ILSQN and the best efficiency in % of the other algorithms. ILSQN is our algorithm, SAHA is the algorithm of Gomes and Oliveira (2006), BLF is the algorithm of Burke, *et al.* (2005), and 2DNest is the algorithm of Egeblad, *et al.* (2006). The column EF shows the efficiency in %. The best results among these algorithms are written in bold typeface. Table 2 shows the computation time (in seconds) of the algorithms.

Gomes and Oliveira (2006) did not use time limit but stop their algorithm by other criteria. They conducted 20 runs for each instance and the best results of the 20 runs are shown in Table 1, while their computation times in Table 2 are the average computation time of the 20 runs.

Burke, *et al.* (2005) conducted four variations of their algorithms, and conducted 10 runs for each variation. Their results in Table 1 are the best results of the 40 runs, which are taken from Table 8 in Burke, *et al.* (2005). They limited the number of iterations for each run, and their computation time in Table 2 is the time spent to find the best solution reported in Table 1 in the run that found it (i.e., the time for only one run is reported). Since they conducted experiments for instances ALBANO, DIGHE1 and DIGHE2 with different orientations from the other papers, we do not include the results.

Egeblad, *et al.* (2006) and we conducted experiments using the time limits for each run shown in Table 2. Although our total computation time of all runs for each instance is not so long compared with SAHA and 2DNest, ILSQN obtained the best results for 5 instances out of the 15 instances in efficiency of the resulting layouts and 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 ILSQN, and ILSQN obtained better results in efficiency than those BLF obtained for all instances.

6. Conclusions

We proposed an iterated local search algorithm for overlap minimization based on nonlinear program and the operation of swapping two polygons in a sophisticated way, and incorporated it in our algorithm for the irregular strip packing problem. We showed through computational experiments that our algorithm is competitive with existing algorithms, updating the best known solutions for several benchmark instances.

Table 1 The best efficiency in % of the four algorithms

Instance	ILSQN	SAHA	BLF	2DNest
ALBANO	**87.76**	87.43	–	87.44
DAGLI	85.62	**87.15**	83.7	85.98
DIGHE1	99.83	**100.00**	–	99.86
DIGHE2	80.52	**100.00**	–	99.95
FU	90.83	90.96	86.9	**91.84**
JAKOBS1	**89.09**	†*78.89	82.6	89.07
JAKOBS2	**80.68**	77.28	74.8	80.41
MAO	84.60	82.54	79.5	**85.15**
MARQUES	89.09	88.14	86.5	**89.17**
SHAPES0	**67.62**	66.50	60.5	67.09
SHAPES1	**73.90**	71.25	66.5	73.84
SHAPES2	82.86	**83.60**	77.7	81.21
SHIRTS	86.18	†**86.79**	84.6	86.33
SWIM	74.02	**74.37**	68.4	71.53
TROUSERS	88.73	**89.96**	88.5	89.84

* The value has been corrected according to the information sent from Gomes and Oliveira (2006).

† Better results were obtained by a simpler greedy approach (GLSHA) (Gomes and Oliveira, 2006): 81.67% for JAKOBS1 and 86.80% for SHIRTS.

Table 2 The computation time in seconds of the four algorithms

Instance	*ILSQN §2.8GHz 10 runs	†SAHA §2.4GHz 20 runs	‡BLF §2.0GHz 40 runs	*2DNest §3.0GHz 20 runs
ALBANO	1200	2257	–	600
DAGLI	1200	5110	188.80	600
DIGHE1	600	83	–	600
DIGHE2	600	22	–	600
FU	600	296	20.78	600
JAKOBS1	600	332	43.49	600
JAKOBS2	600	454	81.41	600
MAO	1200	8245	29.74	600
MARQUES	1200	7507	4.87	600
SHAPES0	1200	3914	21.33	600
SHAPES1	1200	10314	2.19	600
SHAPES2	1200	2136	21.00	600
SHIRTS	1200	10391	58.36	600
SWIM	1200	6937	607.37	600
TROUSERS	1200	8588	756.15	600

* Computation time is the time limit.

† Computation time is the average computation time.

‡ Computation time is the time spent to find the best solution in the run that found it.

§ The experiments are conducted on a PC with a Pentium4 processor.

References

Adamowicz, M. and A. Albano. (1976). Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design*, Vol. 8, No. 1, pp. 27–33.

Agarwal, P. K., L. J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir. (2000). Penetration depth of two convex polytopes in 3D. *Nordic Journal of Computing*, Vol. 7, No. 3, pp. 227–240.

Albano, A. and G. Sapuppo. (1980). Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 10, No. 5, pp. 242–248.

Bennell, J. A. and K. A. Dowsland. (2001). Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science*, Vol. 47, No. 8, pp. 1160–1172.

Bennell, J. A., K. A. Dowsland, and W. B. Dowsland. (2001). The irregular cutting-stock problem – a new procedure for deriving the no-fit polygon. *Computers & Operations Research*, Vol. 28, No. 3, pp. 271–287.

Burke, E. K., R. Hellier, G. Kendall, and G. Whitwell. (2005). A new bottom-left-fill heuristic algorithm for the 2D irregular packing problem. *Operations Research*, to appear.

Dobkin, D., J. Hershberger, D. Kirkpatrick, and S. Suri. (1993). Computing the intersection-depth of polyhedra. *Algorithmica*, Vol. 9, No. 6, pp. 518–533.

Egeblad, J., B. K. Nielsen, and A. Odgaard. (2006). Fast neighborhood search for two- and three-dimensional nesting problem. *European Journal of Operational Research*, to appear.

Gomes, M. A. and J. F. Oliveira. (2006). Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, Vol. 171, No. 3, pp. 811–829.

Hopper, E. and B. C. H. Turton. (2001). A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review*, Vol. 16, No. 4, pp. 257–300.

Li, Z. and V. Milenkovic. (1995). Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research*, Vol. 84, No. 3, pp. 539–561.

Liu, D. C. and J. Nocedal. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, Vol. 45, No. 3, pp. 503–528.

Ramkumar, G. D. (1996). An algorithm to compute the Minkowski sum outer-face of two simple polygons. In: *Proceedings of the twelfth annual symposium on computational geometry*, pp. 234–241, ACM Press, New York.