Journal of the Operations Research Society of Japan
Vol. 57, No. 1, March 2014, pp. 45–61

ENUMERATING BOTTOM-LEFT STABLE POSITIONS FOR RECTANGLE PLACEMENTS WITH OVERLAP

Shinji Imahori   Yuyao Chien   Yuma Tanaka   Mutsunori Yagiura
Nagoya University   *Seikei University*   *Nagoya University*

(Received December 26, 2012; Revised November 25, 2013)

Abstract This paper presents an efficient algorithm that enumerates all the bottom-left stable positions for a given layout of rectangles. Its time complexity is $O((n+K)\log n)$, where n is the number of placed rectangles (i.e., input size) and K is the number of bottom-left stable positions (i.e., output size). This is the first non-trivial algorithm that works for layouts without bottom-left stability and with overlap. In addition to the theoretical analysis of the time complexity, we evaluate the computation time of the proposed algorithm via computational experiments and confirm that it is applicable to very large-scale instances having one million rectangles.

Keywords: Combinatorial optimization, cutting and packing, rectangle packing problem, bottom-left stable position

1. Introduction

Rectangle packing is an important problem with applications in steel, wood, glass, paper and many other industries. There are a number of variants of the problem with different objectives and constraints, but the essential task is to place a given set of rectangles in a given larger area without overlap so that the wasted space in the resulting layout is minimized. (See [19] for a typology of cutting and packing problems.) Almost all variants of the problem are known to be NP-hard, and many heuristic algorithms have been proposed in the literature. One of the typical frameworks of existing heuristic algorithms is the *bottom-left strategy*, which places rectangles one by one at *bottom-left stable positions* [2, 12, 16]. A fundamental problem to be solved for executing these algorithms is to enumerate all bottom-left stable positions (or to find a bottom-left stable position with some properties) for a set of already placed rectangles and one new rectangle to be placed next.

Bottom-left stable positions are defined for a given area (in this paper, we assume that the shape of the given area is rectangular), a set of rectangles placed in the area, and one new rectangle. A bottom-left stable position is a point in the area where the new rectangle can be placed without overlap with already placed rectangles and the new rectangle cannot move to the bottom or to the left. There are many bottom-left stable positions in general and the lowest one (if there are ties, the leftmost one among the lowest) is called the *bottom-left position.* We also define *bottom-left stability* for a layout: If there is no overlap among rectangles and no rectangle can move to the bottom or to the left, the layout satisfies bottom-left stability.

Some constructive heuristic algorithms for the rectangle packing problem place rectangles at a bottom-left stable position [2, 10, 12, 16], and hence any layouts constructed by these algorithms (including intermediate layouts) satisfy bottom-left stability. For layouts with bottom-left stability, Chazelle [4] showed that the number of bottom-left stable positions for a new rectangle is at most $n+1$ when the number of placed rectangles is n and proposed an algorithm to enumerate all the bottom-left stable positions in linear time.

Another common framework of heuristic algorithms for the rectangle packing problem is the improvement method, which places all the rectangles in the given area without overlap and iteratively improves the layout by some operations. These kinds of algorithms often place a rectangle at a bottom-left stable position, but in this case, they may need to solve the problem of finding such a position in a layout *without* bottom-left stability. For this case, Healy *et al.* [9] showed that the number of bottom-left stable positions for a new rectangle is $O(n)$ when n rectangles are placed in the area without overlap, and they proposed an $O(n \log n)$ time algorithm to enumerate all the bottom-left stable positions.

For some packing problems including the two-dimensional irregular packing problem, algorithms with *compaction and separation operations* were proposed [3, 8, 11, 15]. These algorithms generate layouts with overlap during their execution. However, efficient algorithms to enumerate bottom-left stable positions in layouts with overlap have not been proposed yet.

In this paper, we consider the problem of enumerating bottom-left stable positions for a new rectangle within a given layout that may not satisfy bottom-left stability and may have overlap between rectangles. We propose an enumeration algorithm that runs in $O((n + K) \log n)$ time, where n is the number of placed rectangles and K is the number of bottom-left stable positions. It is noted that if the given layout has no overlap between the placed rectangles, then $K = O(n)$ and the time complexity of our algorithm becomes $O(n \log n)$, which is the same as the result in [9]. We use no-fit polygons and a sweep line to enumerate bottom-left stable positions efficiently, where no-fit polygons are widely used in packing algorithms and the sweep line technique is used for many problems in computational geometry and other areas. Our algorithm enumerates bottom-left stable positions from bottom to top (from left to right for positions with an identical y-coordinate), and hence it outputs the bottom-left position first in $O(n \log n)$ time.

The bottom-left strategy can naturally be generalized to the three dimensional case [13]. An important consequence of the algorithm proposed in this paper is that it can be utilized to design an efficient algorithm to execute such a bottom-left algorithm for the three-dimensional packing problem. Kawashima *et al.* [14] showed that the time complexity was improved from the previous best-known $O(n^4)$ (in [17]) to $O(n^3 \log n)$. In their proof, our algorithm is used as a core part of the algorithm, and the applicability of our algorithm to the case with rectangles having overlap is crucial, i.e., existing algorithms for enumerating bottom-left stable positions such as those proposed in [4, 9] cannot be used for this purpose.

2. Problem Description

We are given a set of n rectangles $I = \{1, 2, \ldots, n\}$ and one large rectangular area, also called the container. The container has its width and height (W, H) and its bottom left point is placed at $(0, 0)$ in the plane. Each rectangle $i \in I$ has its width and height (w_i, h_i), and is placed orthogonally in the plane. Let (x_i, y_i) be the coordinate of the bottom left point of rectangle i. We note that the given rectangles may protrude from the container and may overlap each other. We are also given one new rectangle $j \notin I$ with size (w_j, h_j) that has not been placed yet. The objective is to enumerate all the bottom-left stable positions in the container for rectangle j. See Figure 1(a) for an example of bottom-left stable positions; black points in this figure denote bottom-left stable positions for rectangle j. Let K be the number of bottom-left stable positions for a given layout and one new rectangle. It is easy

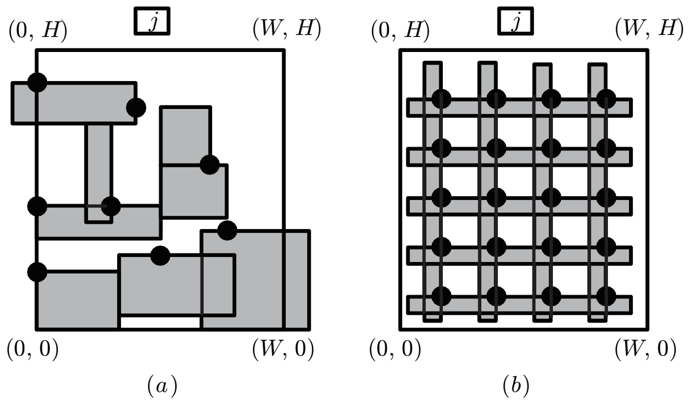

Figure 1: Bottom-left stable positions

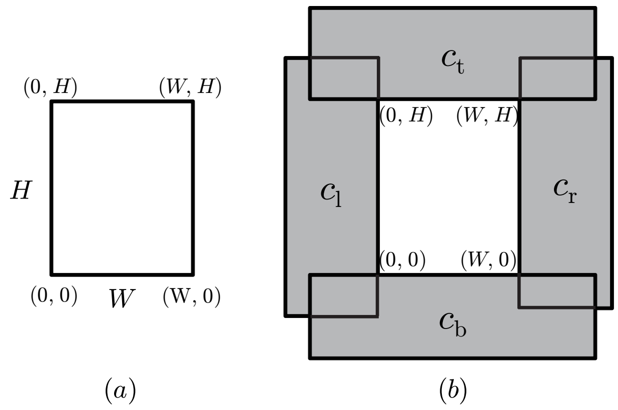

Figure 2: (a) The given area (container), and (b) container rectangles to represent the area

to see that $K = O(n^2)$ and K can be $\Theta(n^2)$ for some cases (see Figure 1(b) for an example).

3. Algorithms

In this section, we propose algorithms to enumerate bottom-left stable positions. We first introduce *no-fit polygon*, which is often used in packing algorithms to check overlap efficiently. In Section 3.2, we give a technique to compute for each point p in the plane the number of no-fit polygons containing p by using a *sweep line*. In Section 3.3, we propose an algorithm for enumerating bottom-left stable positions. We estimate the computational complexity of our algorithms in Section 3.4.

Instead of considering the constraint that requires a new rectangle to be placed in the container, we use a set of four sufficiently large virtual rectangles $C = \{c_l, c_r, c_t, c_b\}$ that satisfies the following condition: Rectangle j does not have overlap with rectangles $i' \in I \cup C$ if and only if it is placed in the container without overlap with rectangles $i \in I$. We call these virtual rectangles container rectangles; see Figure 2 for an example of container rectangles. We denote $I' = I \cup C$; then $|I'| = |I| + 4$ holds.

3.1. No-fit polygon

No-fit polygons (NFP) [1] are widely used as a geometric technique to check overlap of two polygons in two-dimensional space. An NFP is defined for an ordered pair of two polygons i and j, where the position of polygon i is fixed and polygon j can be moved. The no-fit polygon of i and j, denoted by $NFP(i, j)$, is the set of positions of polygon j having intersection with polygon i (more precisely, the interior of polygon j intersects with the interior of polygon i). In this paper, we only treat the situation such that both of two polygons i and j are rectangles, and hence it is easy to compute $NFP(i,j)$, whose shape is also rectangular. Suppose that rectangle i is placed at (x_i, y_i) and rectangle j is the new rectangle. Then $NFP(i,j)$ is defined as follows:

$$NFP(i,j) = \{(x,y) \mid x_i - w_j < x < x_i + w_i,\ y_i - h_j < y < y_i + h_i\}.$$

We also define the *overlap number* $B_j(x,y)$ of no-fit polygons at point (x,y) as follows:

$$B_j(x,y) = \left|\{i \in I' \mid (x,y) \in NFP(i,j)\}\right|.$$

Note that every no-fit polygon is an open set (i.e., the points on the boundary are not included in the NFP), and hence an NFP does not contribute to $B_j(x,y)$ if (x,y) is on its boundary. By using this overlap number, we can characterize bottom-left stable positions as follows:

$$\begin{aligned} &(x,y) \text{ is a bottom-left stable position for rectangle } j \\ &\iff B_j(x,y) = 0 \wedge B_j(x-\varepsilon, y) > 0 \wedge B_j(x, y-\varepsilon) > 0 \qquad (3.1)\\ &\quad \text{for any sufficiently small positive number } \varepsilon. \end{aligned}$$

In the next section, we will describe how to compute the overlap number of no-fit polygons.

3.2. Compute overlap numbers

The algorithm first computes all no-fit polygons $NFP(i,j)$ of rectangle j relative to placed and container rectangles $i \in I'$. In order to compute overlap numbers (of no-fit polygons) in the given area efficiently, the algorithm uses a sweep line parallel to the x-axis and moves it from bottom to top. When we need to specify the sweep line that is currently located at a y-coordinate y_s, we call it the sweep line at y_s. For every fixed y-coordinate y_s, the overlap numbers on the sweep line at y_s are defined as the set of overlap numbers $B_j(x, y_s)$ of all points (x, y_s) on the line. The overlap numbers on the sweep line can be treated as a function of x when the position y_s of the sweep line is fixed, i.e., $B_j(x, y_s)$ is a function of x for every fixed y_s. The algorithm (implicitly) keeps this function and modifies it as the sweep line moves from bottom to top; to be more precise, the function is modified whenever the sweep line encounters the top or bottom edge of a no-fit polygon.

Let N_t (resp., N_b) be the set of all the top (resp., bottom) edges of no-fit polygons and $N_\mathrm{tb} = N_\mathrm{t} \cup N_\mathrm{b}$. As the sweep line moves up, the function representing the overlap numbers on the sweep line changes only when the sweep line encounters a member of N_tb. Let N_l (resp., N_r) be the set of all the left (resp., right) edges of no-fit polygons and $N_\mathrm{lr} = N_\mathrm{l} \cup N_\mathrm{r}$. Because there are n placed rectangles and four container rectangles, $|N_\mathrm{t}| = |N_\mathrm{b}| = |N_\mathrm{l}| = |N_\mathrm{r}| = n+4$ and $|N_\mathrm{tb}| = |N_\mathrm{lr}| = 2n+8$ hold. We use the following rule to sort the elements in N_lr.

Rule A. The elements in N_lr are sorted in nondecreasing order of the x-coordinates of the elements, where ties are broken by giving higher priority to elements in N_r. Moreover, if the right edges of some no-fit polygons have the same x-coordinate, we give higher priority to those elements that correspond to no-fit polygons whose top edges have smaller y-coordinates.

The importance of the tie-breaking rules in Rule A is explained in the next section. Let $x_\mathrm{lr}^{(k)}$ be the x-coordinate of the kth element in the sorted list of N_lr, and define intervals

$$S_k = \left[x_\mathrm{lr}^{(k)}, x_\mathrm{lr}^{(k+1)}\right], \quad k = 1, 2, \ldots, 2n+7.$$

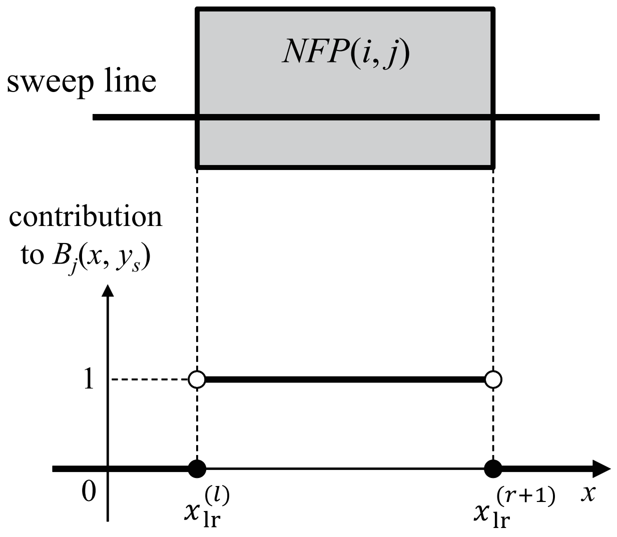

Figure 3: The contribution of $NFP(i, j)$ to $B_j(x, y_s)$

Suppose that for a no-fit polygon $NFP(i, j)$, its left (resp., right) edge is the lth (resp., $(r + 1)$st) element in the sorted list of $N_{\rm lr}$. When the sweep line is at a y-coordinate y_s that is between the top and bottom edges of $NFP(i, j)$, this no-fit polygon contributes one unit to $B_j(x, y_s)$ for all x satisfying $x_{\rm lr}^{(l)} < x < x_{\rm lr}^{(r+1)}$ and has no contribution to it for the remaining values of x as depicted in Figure 3. For this reason, for every fixed y-coordinate y_s, function $B_j(x, y_s)$ is the summation of such functions and hence is a lower semicontinuous (i.e., $B_j(x, y_s) \leq \lim_{\varepsilon \to +0} \min\{B_j(x - \varepsilon, y_s), B_j(x + \varepsilon, y_s)\}$) piecewise linear function whose break points are only on the boundaries of the intervals. That is, the overlap numbers on the sweep line at y_s are the same for all internal points of an interval S_k, i.e., $B_j(x, y_s) = B_j(x', y_s)$ holds for all x and x' that satisfy $x_{\rm lr}^{(k)} < x < x' < x_{\rm lr}^{(k+1)}$. Accordingly, it suffices to keep one value for every interval to maintain the whole shape of the function representing the overlap numbers on the sweep line at y_s.

The algorithm maintains a value $g(k)$ for each interval S_k during the computation, where $g(k)$ is maintained so that it represents the overlap numbers on the sweep line for internal points of S_k except for those intervals whose left and right boundaries have the same x-coordinate and both are in $N_{\rm l}$ or both in $N_{\rm r}$. We show in Section 3.3 that for every y-coordinate y_s of the sweep line, there is a moment when $B_j(x, y_s) = \min_{k:\, x \in S_k} g(k)$ holds for all x. Hence we can find any point (x, y) that satisfies $B_j(x, y) = 0$ by finding an interval S_k such that $x \in S_k$ and $g(k) = 0$ at an appropriate moment while the sweep line is at y. Initially, the sweep line is at a sufficiently low position, and it overlaps with no NFP. At this moment, the value $g(k)$ of every interval S_k is set to zero. We now consider the moment when the sweep line encounters a member in $N_{\rm tb}$. Let $NFP(i, j)$ be the no-fit polygon whose top or bottom edge is encountered by the sweep line, and assume that the left (resp., right) edge of $NFP(i, j)$ is the lth (resp., $(r + 1)$st) element in the sorted list of $N_{\rm lr}$. In this situation, we should increase (resp., decrease) the values of $g(k)$ by one for $k = l, l + 1, \ldots, r$ if the encountered edge is a member of $N_{\rm b}$ (resp., $N_{\rm t}$). See Figure 4 for an example; (a) for a y-coordinate of the sweep line, the value of $g(k)$ for every interval S_k is shown at the bottom, and (b) when the sweep line moves up and encounters the top edge of an NFP, the values of $g(k)$ for the third to eighth interval are decreased by one. In this example, the value of $g(k)$ for every k represents the overlap numbers on the sweep line for

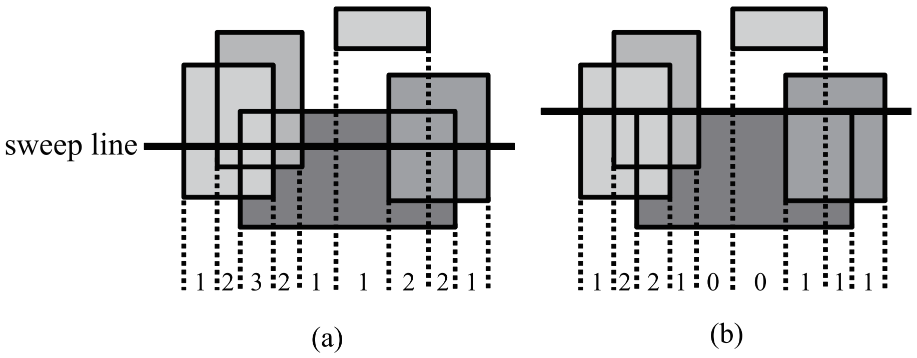

Figure 4: (a) The value of $g(k)$ for each interval S_k, and (b) the sweep line encounters the top edge of an NFP and the values of $g(k)$ for six intervals are updated (decreased by one)

all internal points in S_k.

If the values of $g(k)$ are stored simply in an array whose cells correspond to the intervals, it takes $O(n)$ time to update the $g(k)$ values of relevant cells when the sweep line encounters a member of N_{tb}. To update the $g(k)$ values efficiently, we use a complete binary tree whose leaves represent intervals $S_1, S_2, \ldots, S_{2n+7}$. Here, the kth leaf from the left corresponds to interval S_k, and the name of this leaf is k. We note that $2n+7$ is not a power of two for any n, and there are remaining leaves on the right side of the leaf corresponding to interval S_{2n+7}. Such remaining leaves are called dummy leaves. Corresponding to each dummy leaf k, we consider a dummy interval S_k $(k \geq 2n+8)$ such that $g(k) = 1$ during the entire computation and that the left boundary $x_{\text{lr}}^{(k)}$ is to the right of the container rectangle c_{r} on the right of the container (i.e., S_k is outside of the container). We use a complete binary tree with the minimum number of dummy leaves. Then the number of dummy leaves is less than $2n+7$ and the height of this tree is $O(\log n)$. Every node of this tree stores values p_{self}, p_{min} and p_{max}, whose roles are explained as follows.

For two nodes u and v of the tree, let $PATH(u,v)$ be the set of nodes in the path from u to v including u and v themselves. The algorithm maintains the values of p_{self} for all nodes of the tree so that

$$\sum_{u \in PATH(k,root)} p_{\text{self}}(u) = g(k)$$

is satisfied for every leaf k, where $root$ is the root node of the binary tree. Then it is possible to compute the value of $g(k)$ for each interval S_k in $O(\log n)$ time by using the values of p_{self} in the path from the corresponding leaf to the root node. We also define the values of $p_{\text{min}}(v)$ and $p_{\text{max}}(v)$ for each node v of the complete binary tree as follows:

$$p_{\text{min}}(v) = \min_{k \in Q(v)} \sum_{u \in PATH(k,v)} p_{\text{self}}(u), \tag{3.2}$$

$$p_{\text{max}}(v) = \max_{k \in Q(v)} \sum_{u \in PATH(k,v)} p_{\text{self}}(u), \tag{3.3}$$

where $Q(v)$ is the set of all leaves in the subtree rooted at node v. By using the value of $p_{\text{min}}(v)$ (resp., $p_{\text{max}}(v)$) and the values of $p_{\text{self}}(u)$ for nodes u in the path from the parent node of v to the root node, it is possible to check whether there exist leaves in $Q(v)$ whose $g(k)$ values are equal to zero (resp., positive). Let u and u' be the children of node v and

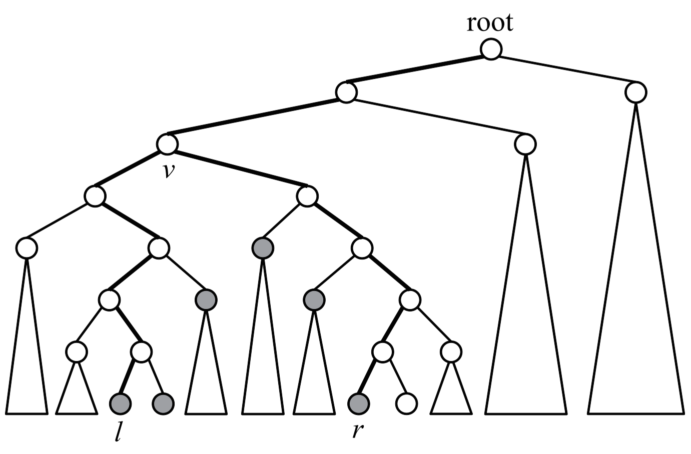

Figure 5: Complete binary tree to maintain the $g(k)$ values; instead of updating the values of all leaves between l and r, the values only on the shaded nodes are updated

assume that the values of $p_{\min}(u)$ and $p_{\min}(u')$ are known. In this situation, the value of $p_{\min}(v)$ can be computed in constant time by

$$p_{\min}(v) = p_{\text{self}}(v) + \min\{p_{\min}(u), p_{\min}(u')\}. \tag{3.4}$$

The value of $p_{\max}(v)$ can be computed similarly.

We now explain the algorithm to keep the values of $p_{\text{self}}, p_{\min}$ and $p_{\max}$ appropriately. Consider the moment when the sweep line encounters a member in N_{tb}. Let $NFP(i, j)$ be the no-fit polygon whose top or bottom edge is encountered by the sweep line, and let the left (resp., right) edge of $NFP(i, j)$ be the lth (resp., $(r + 1)$st) element in the sorted list of N_{lr}. Here we assume for simplicity that the encountered edge is the bottom edge of the NFP. The case when the top edge is encountered is similar; instead of increasing the values by one, the algorithm decreases the values by one. The algorithm first finds the leaves l and r that correspond to the lth and rth intervals and increases the values of $p_{\text{self}}, p_{\min}$ and $p_{\max}$ of these two leaves by one. It then traverses nodes in the paths from the leaves l and r to their least common ancestor v. During this traversal, whenever a node in the path from l (resp., r) to v is reached from its left (resp., right) child, the algorithm increases the values of $p_{\text{self}}, p_{\min}$ and $p_{\max}$ of the right (resp., left) child by one. It also updates $p_{\min}$ (by using (3.4)) and $p_{\max}$ for nodes in the paths from l and r to v so that the conditions (3.2) and (3.3) are satisfied. Finally, the algorithm updates the values of $p_{\min}$ and $p_{\max}$ for all nodes in the path from v to the root node of the tree. See Figure 5 for an example; instead of increasing the values of p_{self} for all leaves between l and r, the values only on the shaded nodes are updated. For every such shaded node u, all leaves in $Q(u)$ are between the leaves l and r, and for every leaf k between l and r, there exists exactly one such shaded node in the path from k to the root. For this reason, increasing the value of p_{self} for every shaded node by one is equivalent to increasing the $g(k)$ value of every leaf between l and r by one. To satisfy Equations (3.2) and (3.3), the values of $p_{\min}$ and $p_{\max}$ on every shaded node and those on every node connected to thick edges are updated from bottom to top.

The details of our procedure to update the values on the binary tree efficiently are summarized as Algorithm UpdateValues(λ, l, r). (The time complexity of this procedure is discussed in Section 3.4.)

```
Algorithm UPDATEVALUES($\lambda, l, r$)
Input: An increment $\lambda$ ($\lambda = 1$ when the sweep line encounters a bottom edge of NFP; $\lambda = -1$ when the sweep line encounters a top edge of NFP), two leaves $l$ and $r$ corresponding to the leftmost and rightmost intervals and current values of $p_{\mathrm{self}}$, $p_{\min}$ and $p_{\max}$.
Task: Update the values of $p_{\mathrm{self}}$, $p_{\min}$ and $p_{\max}$.
Step 1: Increase the values of $p_{\mathrm{self}}(l), p_{\min}(l), p_{\max}(l), p_{\mathrm{self}}(r), p_{\min}(r), p_{\max}(r)$ by $\lambda$ (this means that if $\lambda = -1$, these values are actually decreased by one).
Step 2: Let $l_{\mathrm{prev}} := l$ and $r_{\mathrm{prev}} := r$, and then let $l$ be the parent of $l_{\mathrm{prev}}$ and $r$ be the parent of $r_{\mathrm{prev}}$. If $l \neq r$ holds, proceed to Step 3; otherwise go to Step 4.
Step 3: If the right (resp., left) child $u$ of $l$ (resp., $r$) is different from $l_{\mathrm{prev}}$ (resp., $r_{\mathrm{prev}}$), increase the values of $p_{\mathrm{self}}(u), p_{\min}(u)$ and $p_{\max}(u)$ by $\lambda$.
    Let $p_{\min}(l) := p_{\mathrm{self}}(l) + \min\{p_{\min}(u), p_{\min}(u')\}$, where $u$ and $u'$ are the children of node $l$.
    Update the values of $p_{\max}(l), p_{\min}(r), p_{\max}(r)$ similarly. Return to Step 2.
Step 4: For each node $v$ in the path from $l$ ($= r$) to the root and the children $u$ and $u'$ of $v$, let $p_{\min}(v) := p_{\mathrm{self}}(v) + \min\{p_{\min}(u), p_{\min}(u')\}$ and update $p_{\max}(v)$ similarly. Then stop.
```

3.3. Enumerate bottom-left stable positions

We explain our algorithm that enumerates bottom-left stable positions. Observe that, while the sweep line parallel to the x-axis is moved from bottom to top, the overlap numbers on the sweep line decrease only if the top edge of a no-fit polygon is encountered. This means that bottom-left stable positions can be found only in this case, because a point (x, y) can be a bottom-left stable position only if $B_j(x, y) = 0$ and $B_j(x, y - \varepsilon) > 0$ for any sufficiently small positive ε. For this reason, when the sweep line encounters the bottom edge of a no-fit polygon, the algorithm just updates the values $g(k)$ (implicitly by updating the values on some nodes in the binary tree) of relevant intervals S_k according to the rule described in Section 3.2. On the other hand, when the sweep line encounters the top edge of a no-fit polygon, the algorithm updates the values of $g(k)$ of relevant intervals and then outputs bottom-left stable positions on the sweep line if such positions exist. To manage these events, the following rule is used to sort the elements in N_{tb}.

Rule B. The elements in N_{tb} are sorted in nondecreasing order of the y-coordinates of the elements, where ties are broken by giving higher priority to elements in N_{t}. If the top edges of some no-fit polygons have the same y-coordinate, we give higher priority to those elements that correspond to no-fit polygons whose right edges have smaller x-coordinates (more precisely, we give higher priority to top edges of no-fit polygons whose right edges are given higher priority in N_{lr}).

We show later in this section that every bottom-left stable position (x, y) appears as the intersection of the sweep line and the left boundary of an interval S_k such that when the sweep line encounters all top edges in N_{t} that contain (x, y), $g(k)$ changes from positive to zero and $g(k-1)$ is positive. Our algorithm enumerates all such points. For such points, the above properties correspond to the following facts: (1) $g(k) = 0 \implies B_j(x, y) = 0$, (2) $g(k) > 0$ just before the sweep line encounters such top edges $\implies (x, y)$ is on the top edge of a no-fit polygon and hence $B_j(x, y - \varepsilon) > 0$ for any sufficiently small $\varepsilon > 0$, and (3) $g(k-1) > 0 \implies (x, y)$ is on the right edge of a no-fit polygon and hence $B_j(x - \varepsilon, y) > 0$ for any sufficiently small $\varepsilon > 0$.

The details of our algorithm to enumerate all the bottom-left stable positions is formally described in Algorithm ENUMERATEBL(j, I). Step 1 initializes the data structures. In

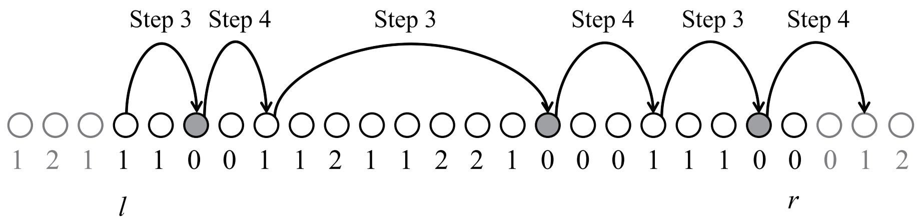

Figure 6: How to enumerate the bottom-left stable positions on the current top edge

Step 2, for the top or bottom edge e of a no-fit polygon encountered by the sweep line, the $g(k)$ values of relevant intervals are updated. If a top edge is encountered in Step 2 (we call it the current top edge), Steps 3 and 4 enumerate the bottom-left stable positions on the current top edge.

Algorithm ENUMERATEBL(j, I)

Input: Placed rectangles $i \in I$ in the given area and one new rectangle $j \notin I$.

Output: All the bottom-left stable positions for rectangle j.

Step 1: Compute all no-fit polygons of rectangle j relative to all placed and container rectangles $i \in I'$. Sort left and right edges $N_{\text{lr}} := N_{\text{l}} \cup N_{\text{r}}$ of no-fit polygons according to Rule A. Create the minimum complete binary tree with at least $2n+7$ leaves. Initialize the values $p_{\text{self}}(u) := 0$, $p_{\min}(u) := 0$ and $p_{\max}(u) := 0$ for all leaves u corresponding to intervals S_1 to S_{2n+7}, and initialize the values $p_{\text{self}}(u) := 1$, $p_{\min}(u) := 1$ and $p_{\max}(u) := 1$ for all dummy leaves u. For all internal nodes u, set $p_{\text{self}}(u) := 0$ and compute the values of $p_{\min}(u)$ and $p_{\max}(u)$. Sort top and bottom edges $N_{\text{tb}} := N_{\text{t}} \cup N_{\text{b}}$ of no-fit polygons according to Rule B.

Step 2: Choose the first element $e \in N_{\text{tb}}$ and let $N_{\text{tb}} := N_{\text{tb}} \setminus \{e\}$ (i.e., e is encountered by the sweep line). If the y-coordinate of element e is greater than $H - h_j$, stop. Let $NFP(i,j)$ be the no-fit polygon having the element e as its top or bottom edge, and assume that its left (resp., right) edge is the lth (resp., $(r+1)$st) element in N_{lr}. If $e \in N_{\text{t}}$ (resp., N_{b}), then set $\lambda := -1$ (resp., $\lambda := 1$). Call algorithm UPDATEVALUES(λ, l, r). If $e \in N_{\text{b}}$, return to Step 2; otherwise (i.e., $e \in N_{\text{t}}$) set $\alpha := l$. If $g(\alpha)$ is positive, go to Step 3. Otherwise, go to Step 4.

Step 3: By using the values of $p_{\min}$ and p_{self}, find the leftmost interval S_γ that satisfies $\gamma > \alpha$ and $g(\gamma) = 0$. If $\gamma > r$ or such a γ is not found, return to Step 2; otherwise, output a bottom-left stable position (x, y), where $x = x_{\text{lr}}^{(\gamma)}$ and y is the y-coordinate of the current top edge e. Let $\alpha := \gamma$ and go to Step 4.

Step 4: By using the values of $p_{\max}$ and p_{self}, find the leftmost interval S_γ that satisfies $\gamma > \alpha$ and $g(\gamma) > 0$. If $\gamma \geq r$ or such a γ is not found, return to Step 2; otherwise, let $\alpha := \gamma$ and go to Step 3.

To attain efficient enumeration of the bottom-left stable positions on the current top edge, Steps 3 and 4 are used instead of checking the $g(k)$ values for all leaves from l to r. Figure 6 shows how the algorithm traverses leaves by applying Step 3 and 4 alternately, where circles in the figure are the leaves of the tree and the numbers below the circles denote the $g(k)$ values of corresponding intervals. In Step 3, γ is found as follows. The algorithm first climbs the binary tree from the leaf α, and whenever a node v is reached from its left child, it checks whether the subtree rooted at the right child u of v has a leaf k such that $g(k) = 0$. When the first node u having such a leaf is found, the algorithm goes

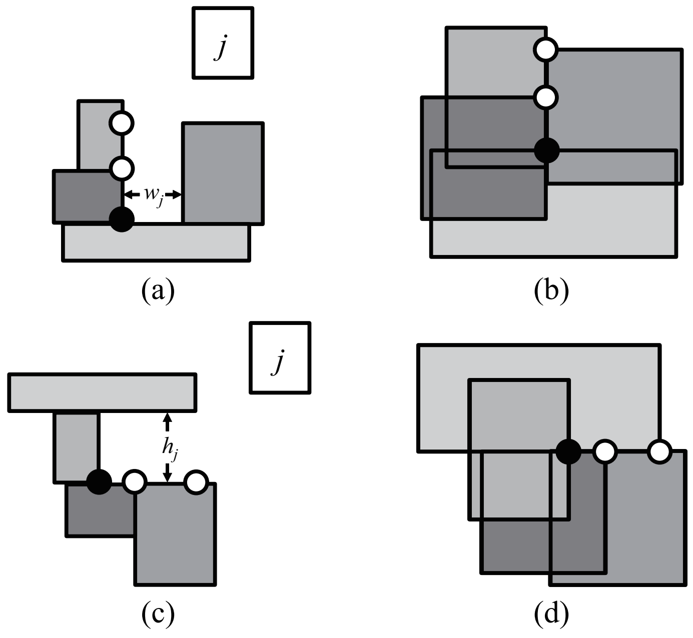

Figure 7: Tie-breaking rules are necessary to enumerate the bottom-left stable positions; black (resp., white) points must (resp., must not) be output

down the tree from u, choosing the left-most child including such a leaf. In Step 4, the leaf γ is found similarly.

We now explain the necessity of the tie-breaking rules in Rule A and B by giving some examples showing that without them, the output of the algorithm may become incorrect. Figure 7 (a) is a layout of four rectangles and we want to find bottom-left stable positions of a new rectangle j in this layout. (There is one bottom-left stable position (black point) in this layout.) Figure 7 (b) is the layout of corresponding NFPs. The tie-breaking rule in Rule A that gives higher priority to edges in N_r than those in N_l is necessary to output the bottom-left stable position (black point) in these figures. This tie-breaking rule is also crucial to avoid outputting the upper white point, which is not a bottom-left stable position. Moreover, the tie-breaking rule in Rule A for edges in N_r is necessary to avoid outputting the lower white point. Figure 7 (c) and (d) are similar; the tie-breaking rule in Rule B that gives higher priority to edges in N_t than those in N_b is necessary to output (resp., to avoid outputting) the black point (resp., the right white point). Moreover, as described in Rule B, if there are some no-fit polygons whose top edges have the same y-coordinate, the top edges that correspond to no-fit polygons whose right edges have smaller x-coordinates must leave the sweep line earlier. Without this tie-breaking rule, the algorithm may output positions (the left white point in Figure 7 (c)) from which rectangle j can move to the left.

In the following, we formally show that Algorithm EnumerateBL(j, I) enumerates all the bottom-left stable positions.

Lemma 3.1. *For every point (x, y) in the container, the following two properties hold: (1) For every interval S_k that satisfies $x \in S_k$, $g(k) \geq B_j(x, y)$ holds whenever the sweep line is at y-coordinate y. (2) There is an interval S_k that satisfies $x \in S_k$ and $g(k) = B_j(x, y)$ when the sweep line is at y-coordinate y and has encountered all of the top edges in N_t that contain (x, y) but none of the bottom edges in N_b that contain (x, y).*

Note that by the tie-breaking rule in Rule B that gives higher priority to edges in $N_{\rm t}$, there is always such a moment specified in Lemma 3.1 (2). This lemma implies that

$$B_j(x,y) = \min_{k:\, x\in S_k} g(k)$$

holds for all x when the sweep line has encountered all of the top edges in $N_{\rm t}$ whose y-coordinates are y or less but none of the bottom edges in $N_{\rm b}$ whose y-coordinates are y or more.

Proof. For every point (x,y) in the container, let $N'_{\rm l} \subseteq N_{\rm l}$, $N'_{\rm r} \subseteq N_{\rm r}$, $N'_{\rm t} \subseteq N_{\rm t}$ and $N'_{\rm b} \subseteq N_{\rm b}$ be the left, right, top and bottom edges that contain (x,y) (including those having (x,y) at their endpoints in the case of $N'_{\rm l}$ and $N'_{\rm r}$), and let $\tilde{I}_{xy}$ be the set of no-fit polygons that contain (x,y) in their interior regions (n.b., $\tilde{I}_{xy}$ does not contain any no-fit polygon that touches (x,y) on its boundary). We assume for simplicity that $N'_{\rm r}$ is nonempty; the proof for the opposite case is similar and is omitted. Because no-fit polygons are open sets (i.e., they do not contain points on their boundary), no-fit polygons having their edges in $N'_{\rm l}$, $N'_{\rm r}$, $N'_{\rm t}$ or $N'_{\rm b}$ do not contribute to the overlap number $B_j(x,y)$, and every no-fit polygon in $\tilde{I}_{xy}$ contributes one unit to it; hence we have $B_j(x,y) = |\tilde{I}_{xy}|$. It is not hard to see that for every interval S_k satisfying $x \in S_k$, when the y-coordinate of the sweep line is y, every no-fit polygon in $\tilde{I}_{xy}$ contributes one unit to $g(k)$, which implies $g(k) \geq |\tilde{I}_{xy}| = B_j(x,y)$. Indeed, the x-coordinate of the left (resp., right) boundary of such a no-fit polygon is strictly less (resp., greater) than x, and in the sorted list of edges in $N_{\rm lr}$, there is no edge between the two edges corresponding to the left and right boundaries of S_k. Hence in this sorted list, the left (resp., right) edge of this no-fit polygon is the same as or to the left (resp., right) of the edge corresponding to the left (resp., right) boundary of S_k. Moreover, the y-coordinate of the bottom (resp., top) boundary of this no-fit polygon is strictly less (resp., greater) than y. Accordingly, this no-fit polygon contributes one unit to $g(k)$ when the y-coordinate of the sweep line is y.

Then, all we have to do to prove (2) is to show that there is an interval S_k satisfying $x \in S_k$ for which at the moment specified in the lemma, none of the no-fit polygons having their edges in $N'_{\rm l}$, $N'_{\rm r}$, $N'_{\rm t}$ or $N'_{\rm b}$ contributes to $g(k)$. By the tie-breaking rule in Rule A that gives higher priority to elements in $N_{\rm r}$, there is a right edge $e \in N'_{\rm r}$ such that in the sorted list of elements in $N_{\rm lr}$, all edges in $N'_{\rm r} \setminus \{e\}$ are to the left of e and all edges in $N'_{\rm l}$ are to the right of e. Let S_k be the interval whose left boundary corresponds to e. Then all no-fit polygons having their right edges in $N'_{\rm r}$ and those having their left edges in $N'_{\rm l}$ cannot contribute to $g(k)$. At the moment specified in the lemma, the sweep line has encountered all of the top edges in $N'_{\rm t}$ but none of the bottom edges in $N'_{\rm b}$, and the y-coordinate of the sweep line is y. At this moment, none of the no-fit polygons having their top or bottom edges in $N'_{\rm t}$ or $N'_{\rm b}$ can contribute to $g(k)$, and hence $g(k) = B_j(x,y)$ holds. □

The following corollary is immediate from this lemma.

Corollary 3.1. *If $g(k) = 0$ holds for an interval S_k when the sweep line is at a y-coordinate y, then $B_j(x,y) = 0$ holds for every point (x,y) such that $x \in S_k$.*

Proof. Suppose that there is a point (x,y) such that $x \in S_k$ and $B_j(x,y) > 0$ (and y is the value specified in the claim). Then from Lemma 3.1 (1), $g(k)$ must be positive, which contradicts the assumption of this corollary. □

By these lemma and corollary, the following theorem holds.

Theorem 3.1. *The Algorithm* ENUMERATEBL(j, I) *enumerates all the bottom-left stable positions from bottom to top* (*from left to right for positions with an identical* y*-coordinate*).

Proof. Let $(\hat{x}, \hat{y})$ be a bottom-left stable position. Then, as stated in (3.1) in Section 3.1, $B_j(\hat{x}, \hat{y}) = 0$ and $B_j(\hat{x}, \hat{y} - \varepsilon) > 0$ hold. Thus, the point $(\hat{x}, \hat{y})$ is on the top edge (except the both end points) of at least one no-fit polygon. Among such no-fit polygons, let $NFP(i, j)$ be the no-fit polygon whose top edge appears last in the sorted list of top and bottom edges $N_{\rm tb}$. Let the left (resp., right) edge of $NFP(i, j)$ be the lth (resp., $(r + 1)$st) element in $N_{\rm lr}$. Because $(\hat{x}, \hat{y})$ is not on either end of the top edge of $NFP(i, j)$, the x-coordinate of the left (resp., right) edge of $NFP(i, j)$ is strictly smaller (resp., larger) than $\hat{x}$.

Consider the time when the algorithm removes the top edge of $NFP(i, j)$ from $N_{\rm tb}$. Lemma 3.1, together with the fact that $B_j(\hat{x}, \hat{y}) = 0$, ensures the existence of k' that satisfies $\hat{x} \in S_{k'}$ and $g(k') = 0$. Here, $B_j(\hat{x} - \varepsilon, \hat{y}) > 0$ for any small positive ε means that the x-coordinate of the left boundary of every such interval $S_{k'}$ is $\hat{x}$ (otherwise, we could choose a positive ε such that $\hat{x} - \varepsilon \in S_{k'}$; then $g(k') = 0$ implies $B_j(\hat{x} - \varepsilon, \hat{y}) = 0$ by Corollary 3.1). As mentioned above, the x-coordinate of the left (resp., right) edge of $NFP(i, j)$ is smaller (resp., larger) than $\hat{x}$, and hence k' must satisfy $l < k' \leq r$. Let k be the smallest k' that satisfies the two conditions $\hat{x} \in S_{k'}$ and $g(k') = 0$. The x-coordinate of the right boundary of the interval S_{k-1} is $\hat{x}$, which means $\hat{x} \in S_{k-1}$. Then, we must have $g(k - 1) > 0$ because k is the smallest among those satisfying the two conditions. In summary, this interval S_k satisfies the following four conditions: (1) $\hat{x}$ is the left boundary of the interval S_k, (2) the value of $g(k)$ for interval S_k is zero, (3) the value of $g(k - 1)$ for interval S_{k-1} immediately to the left of S_k is positive, and (4) $l < k \leq r$ holds. Then, it is not hard to see that the algorithm outputs the point $(\hat{x}, \hat{y})$ in Step 3 when the value of γ is k during the repetition of Steps 3 and 4 immediately after the execution of Step 2 in which the top edge of $NFP(i, j)$ is removed from $N_{\rm tb}$.

Assume that the algorithm outputs a point $(\tilde{x}, \tilde{y})$. Let k be the value of γ at the time this point is output in Step 3. Then interval S_k satisfies the following conditions: (1) the x-coordinate of its left boundary is $\tilde{x}$, (2) the value of $g(k)$ is zero and (3) $g(k-1)$ is positive at the time when $(\tilde{x}, \tilde{y})$ was output. Because $g(k - 1) > g(k)$ holds, there must be a no-fit polygon that contributes to $g(k - 1)$ at this moment but does not contribute to $g(k)$, and hence the boundary between the intervals S_{k-1} and S_k is formed by the right edge of such a no-fit polygon, which we call $NFP(i_{\rm r}, j)$. The algorithm outputs a point only when the top edge of a no-fit polygon is encountered by the sweep line in Step 2 (i.e., output occurs during the subsequent calls to Steps 3 and 4 after such a call to Step 2). Let $NFP(i_{\rm t}, j)$ be the no-fit polygon whose top edge is removed from $N_{\rm tb}$ in the latest call to Step 2 before $(\tilde{x}, \tilde{y})$ is output, and let l and r be their values in this call to Step 2. Then it is not hard to see from the rules in Steps 2–4 that $l < k \leq r$ is satisfied. This means that the point $(\tilde{x}, \tilde{y})$ is on the top edge of $NFP(i_{\rm t}, j)$ and the value of $g(k)$ decreased when this top edge encountered the sweep line. The inequality $k \leq r$ also implies that the right edge of $NFP(i_{\rm t}, j)$ is the same as or to the right of the right boundary of S_k in the sorted list of $N_{\rm lr}$, and hence this right edge is to the right of (i.e., it has lower priority than) the right edge of $NFP(i_{\rm r}, j)$ in the sorted list.

We first show that the point $(\tilde{x}, \tilde{y})$ is on the right edge of $NFP(i_{\rm r}, j)$ but not on its either end point. At the time the point $(\tilde{x}, \tilde{y})$ is output, the top edge of $NFP(i_{\rm t}, j)$ has just encountered the sweep line, and $NFP(i_{\rm r}, j)$ contributes to $g(k - 1)$, which means that the sweep line has encountered the bottom edge of $NFP(i_{\rm r}, j)$ but has not encountered its top edge. These facts imply that the top edge of $NFP(i_{\rm r}, j)$ has y-coordinate strictly greater than $\tilde{y}$, because otherwise by the tie-breaking rule in Rule B that requires top edges of no-fit polygons whose right edges have higher priority in $N_{\rm lr}$ leave the sweep line earlier, the top edge of $NFP(i_{\rm r}, j)$ would leave the sweep line before the top edge of $NFP(i_{\rm t}, j)$, and that the bottom edge of $NFP(i_{\rm r}, j)$ has y-coordinate strictly less than $\tilde{y}$, because of the tie-breaking rule in Rule B that requires bottom edges of no-fit polygons having the same y-coordinate with some top edges to enter the sweep line after all such top edges leave the sweep line.

We next show that the point $(\tilde{x}, \tilde{y})$ is not at either end of the top edge of $NFP(i_{\rm t}, j)$. It is not at the right end for the following reason. Such a situation would happen only if the right edge of $NFP(i_{\rm t}, j)$ has x-coordinate $\tilde{x}$ (i.e., the right edges of $NFP(i_{\rm r}, j)$ and $NFP(i_{\rm t}, j)$ have the same x-coordinate). As discussed above, the top edge of $NFP(i_{\rm r}, j)$ has y-coordinate greater than $\tilde{y}$, while that of $NFP(i_{\rm t}, j)$ has y-coordinate $\tilde{y}$. These imply that in the sorted list of left and right edges $N_{\rm lr}$, the right edge of $NFP(i_{\rm r}, j)$ must be to the right of the right edge of $NFP(i_{\rm t}, j)$ by the tie-breaking rule in Rule A, which requires the right edges of no-fit polygons having an identical x-coordinate to be sorted so that those corresponding to no-fit polygons whose top edges have smaller y-coordinates have higher priority (i.e., arranged to the left). This contradicts the above-mentioned order of the right edges of $NFP(i_{\rm r}, j)$ and $NFP(i_{\rm t}, j)$. Hence the point $(\tilde{x}, \tilde{y})$ is not at the right end of the top edge of $NFP(i_{\rm t}, j)$.

Similarly, the point $(\tilde{x}, \tilde{y})$ is not at the left end of this top edge. This situation would happen only if the left edge of $NFP(i_{\rm t}, j)$ has x-coordinate $\tilde{x}$, which is the same as that of the right edge of $NFP(i_{\rm r}, j)$. Then, the left edge of $NFP(i_{\rm t}, j)$ must be to the right of the right edge of $NFP(i_{\rm r}, j)$ in the sorted list of left and right edges $N_{\rm lr}$ by the tie-breaking rule in Rule A that requires the left edges having the same x-coordinate with some right edges be arranged to the right of such right edges. Recall that the left edge of $NFP(i_{\rm t}, j)$ forms the left boundary of interval S_l, and the right edge of $NFP(i_{\rm r}, j)$ forms the left boundary of interval S_k. Then the above-mentioned inequality $l < k$ implies that the left edge of $NFP(i_{\rm t}, j)$ is to the left of the right edge of $NFP(i_{\rm r}, j)$, and the above-mentioned order contradicts this fact. Hence, $(\tilde{x}, \tilde{y})$ cannot be at the left end of the top edge of $NFP(i_{\rm t}, j)$.

To summarize, the point $(\tilde{x}, \tilde{y})$ is on the top edge of $NFP(i_{\rm t}, j)$ but not at either end of this top edge. This implies $B_j(\tilde{x}, \tilde{y} - \varepsilon) > 0$. Because the value of $g(k)$ for the interval S_k is zero, $B_j(\tilde{x}, \tilde{y}) = 0$ holds by Corollary 3.1. We also have $B_j(\tilde{x} - \varepsilon, \tilde{y}) > 0$ because as mentioned above, the point $(\tilde{x}, \tilde{y})$ is on the right edge of $NFP(i_{\rm r}, j)$ but not at its either end point. Consequently, the point $(\tilde{x}, \tilde{y})$ is a bottom-left stable position. We note that the value of $g(k)$ for every dummy leaf k is initialized to one and is not changed during the execution of the algorithm; hence the dummy leaves have no influence on the output of the algorithm.

The algorithm uses a sweep line parallel to the x-axis, moves it from bottom to top, and outputs points on it. Thus, the algorithm outputs points from bottom to top. When the algorithm outputs two or more points with an identical y-coordinate, the algorithm outputs points from left to right for the following reason. Let (x, y) and (x', y) be bottom-left stable positions with an identical y-coordinate, where $x < x'$ holds. If the algorithm outputs both of these points when the top edge of a no-fit polygon was encountered by the sweep line (i.e., these points are on an identical top edge e and both are output during the subsequent calls to Steps 3 and 4 immediately after the call to Step 2 in which e encountered the sweep line), it is trivial to see that the algorithm outputs them from left to right. We consider the remaining case where the two points are output via different top edges having an identical y-coordinate. Let $NFP(i, j)$ be the no-fit polygon such that the point (x, y) is output when its top edge leaves the sweep line, and x_l (resp., x_r) be the x-coordinate of the left (resp., right) edge of $NFP(i,j)$. We similarly define $NFP(i',j)$, x'_l and x'_r for the point (x',y). Then, as discussed above, (x,y) (resp., (x',y)) is on the top edge of $NFP(i,j)$ (resp., $NFP(i',j)$) but not at its either end, and hence $x_l < x < x_r$ and $x'_l < x' < x'_r$ hold. To show that (x,y) is output before (x',y), assume the opposite, which would happen only if the top edge of $NFP(i',j)$ leaves the sweep line earlier than that of $NFP(i,j)$. Then $x'_r \leq x_r$ holds by the tie-breaking rule in Rule B. This means that $x_l < x < x' < x'_r \leq x_r$ holds and hence the point (x',y) is also on the top edge of $NFP(i,j)$ but not at its either end point. At the time when the top edge of $NFP(i',j)$ leaves the sweep line, the bottom edge of $NFP(i,j)$ has already encountered the sweep line but its top edge has not encountered the sweep line yet. This means that $NFP(i,j)$ contributes to the $g(k)$ values of all intervals S_k that satisfy $x' \in S_k$, and hence the $g(k)$ values of all such intervals are positive. However, for the algorithm to output (x',y), there must be an interval S_k that satisfies $g(k) = 0$ and $x' \in S_k$, which is a contradiction.

Therefore, the algorithm outputs bottom-left stable positions from bottom to top (from left to right for positions with an identical y-coordinate). This property also means that the algorithm outputs the bottom-left position first. □

3.4. Computational complexity

We estimate the time complexity of our algorithms described in Sections 3.2 and 3.3. Algorithm UpdateValues(λ, l, r) proposed in Section 3.2 runs in $O(\log n)$ time since the height of the complete binary tree is $O(\log n)$. Algorithm EnumerateBL(j, I) calls algorithm UpdateValues(λ, l, r) as a subroutine at most $2n+8$ times during the whole execution of the algorithm; then the total time for this part is $O(n \log n)$. The time complexity of Step 3 is $O(\log n)$ because $O(\log n)$ nodes are visited during the traversal from α to γ, and it is possible for each node u to check in constant time whether the subtree rooted at the node u has a leaf k that satisfies $g(k) = 0$ (this is not hard to see from the property explained in Section 3.2 just after Equation (3.3)). The time complexity of Step 4 is also $O(\log n)$ for a similar reason. Steps 3 and 4 are called at most $n + 4 + K$ times respectively, where K is the number of all bottom-left stable positions. In summary, the total time complexity of algorithm EnumerateBL(j, I) is $O((n + K) \log n)$. Furthermore, if our algorithms are utilized for finding the bottom-left position (i.e., the leftmost position among the lowest bottom-left stable positions), algorithm EnumerateBL(j, I) can be stopped immediately when it finds the first bottom-left stable position. In this case, the time complexity to find the bottom-left position is $O(n \log n)$.

4. Computational Results

In this section, we evaluate the proposed algorithms via computational experiments. All of the algorithms were coded in C and experiments were conducted on a PC (Intel Xeon 3GHz, 1GB memory). As a set of placed rectangles, we used test instances for the rectangle packing problem with $16, 32, 64, \ldots, 1048576$ rectangles (they are obtained electronically from http://www.na.cse.nagoya-u.ac.jp/˜imahori/packing/). These sets of rectangles were generated in [10] with a method proposed by Wang and Valenzela [18]. For each set of rectangles, a container whose area is equal to the sum of rectangles' areas was given and we place all the rectangles into the container randomly. The size of a new rectangle to be placed next is similar to the placed rectangles.

Table 1 shows the computational results. Column "n" shows the number of rectangles placed in a rectangular area. The number of bottom-left stable positions enumer-

Table 1: Number of bottom-left stable positions in a layout and time for enumeration

n	BLpts	time (s)
16	7	6.61×10^{-6}
32	9	1.71×10^{-5}
64	15	4.55×10^{-5}
128	39	1.18×10^{-4}
256	76	2.63×10^{-4}
512	138	5.84×10^{-4}
1024	266	1.25×10^{-3}
2048	504	2.66×10^{-3}
4096	636	5.72×10^{-3}
8192	1248	1.26×10^{-2}
16,384	2375	2.80×10^{-2}
32,768	2931	6.97×10^{-2}
65,536	6044	2.15×10^{-1}
131,072	6446	6.54×10^{-1}
262,144	12,901	1.73×10^{0}
524,288	16,036	4.29×10^{0}
1,048,576	33,483	1.04×10^{1}

ated is shown in column "BLpts," and computation time in seconds is reported in column "time (s)." We note that layouts were randomly generated many times (at least 10 times; it depends on the number of placed rectangles) and the average number of bottom-left stable positions and average computation time were reported. From the table, we can confirm that the proposed algorithm runs in near linear time. Moreover, we can observe that the proposed algorithm enumerates bottom-left stable positions in short time from a practical viewpoint. It can enumerate all the bottom-left stable positions among hundreds of rectangles within 0.001 seconds, it spends less than 0.1 seconds for instances with up to 32,768 rectangles, and it takes 10.4 seconds to enumerate all the bottom-left stable positions in layouts with about one million rectangles.

5. Extensions

In the previous sections, the problem of enumerating bottom-left stable positions for a new rectangle within a layout of rectangles were considered. By using the techniques introduced in this paper, it is possible to treat similar problems with different shapes.

One of the important candidates of different shapes is arbitrary rectilinear blocks, which are the shapes bounded by line segments parallel to the x- or y-axis. Packing problems of arbitrary rectilinear blocks have applications in VLSI design and have been studied in the literature [5, 7]. As a core part of designing (most of) heuristic algorithms for the rectilinear block packing problem, the problem of finding (or enumerating) bottom-left stable position(s) for a new rectilinear block within a layout of rectilinear blocks should be solved iteratively. For this problem, placed rectilinear blocks are replaced with a set of rectangles that covers the rectilinear blocks within the layout. For a new rectilinear block, another set of rectangles that covers the block is arranged, and these rectangles share a reference point to keep their relative positions. (Note that computing a minimum set of rectangles to cover a rectilinear area is NP-hard and some approximation algorithms were proposed [6].)

We compute all no-fit polygons $NFP(i,j)$, where i is a rectangle that represents a part of a placed block and j is a rectangle representing a part of the new rectilinear block. Then, the algorithm proposed in this paper can output all the bottom-left stable positions for the new rectilinear block.

6. Conclusions

In this paper, the problem of enumerating bottom-left stable positions for a given layout of rectangles was studied. We proposed an algorithm that enumerates all the bottom-left stable positions in $O((n+K)\log n)$ time, where n is the number of placed rectangles and K is the number of bottom-left stable positions (i.e., the size of the output). One of the important features of our algorithm is that it works for layouts without bottom-left stability and with overlap. We also evaluated the proposed algorithm via computational experiments. Even for instances having one million rectangles, the proposed algorithm works in short computation time.

A direction of future work is to propose faster algorithms: The proposed algorithm runs in $O((n+K)\log n)$ time, but the existence of algorithms that run in $O(n\log n+K)$ is open.

Acknowledgments

We thank Yannan Hu and the anonymous referees for valuable comments.

References

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

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

[3] J.A. Bennell and K.A. Dowsland: Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science*, **47** (2001), 1160–1172.

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

[5] D. Chen, J. Liu, Y. Fu, and M. Shang: An efficient heuristic algorithm for arbitrary shaped rectilinear block packing problem. *Computers and Operations Research*, **37** (2010), 1068–1074.

[6] D. Franzblau: Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles. *SIAM Journal on Discrete Mathematics*, **2** (1989), 307–321.

[7] K. Fujiyoshi and H. Murata: Arbitrary convex and concave rectilinear block packing using sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **19** (2000), 224–233.

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

[9] P. Healy, M. Creavin, and A. Kuusik: An optimal algorithm for rectangle placement. *Operations Research Letters*, **24** (1999), 73–80.

[10] S. Imahori and M. Yagiura: The best-fit heuristic for the rectangular strip packing problem: an efficient implementation and the worst-case approximation ratio. *Computers and Operations Research*, **37** (2010), 325–333.

[11] T. Imamichi, M. Yagiura, and H. Nagamochi: An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. *Discrete Optimization*, **6** (2009), 345–361.

[12] S. Jakobs: On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, **88** (1996), 165–181.

[13] K. Karabulut and M. İnceoğlu: A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method. In *Proceedings of ADVIS 2004* (LNCS 3261), 441–450.

[14] H. Kawashima, Y. Tanaka, S. Imahori, and M. Yagiura: An efficient implementation of a constructive algorithm for the three-dimensional packing problem. In *Proceedings of the 9th Forum on Information Technology 2010*, Issue 1, 31–38 (in Japanese).

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

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

[17] L. Wang, S. Guo, S. Chen, W. Zhu, and A. Lim: Two natural heuristics for 3D packing with practical loading constraints. In *Proceedings of PRICAI 2010* (LNAI 6230), 256–267.

[18] P.Y. Wang and C.L. Valenzela: Data set generation for rectangular placement problems. *European Journal of Operational Research*, **134** (2001), 378–391.

[19] G. Wäscher, H. Haussner, and H. Schumann: An improved typology of cutting and packing problems. *European Journal of Operational Research*, **183** (2007), 1109–1130.

Shinji Imahori
Nagoya University
Furo-cho, Chikusa-ku,
Nagoya 464-8603, Japan
E-mail: `imahori@na.cse.nagoya-u.ac.jp`