■学生論文賞受賞論文 要約■

Heuristic Algorithms for the Rectilinear Block Packing Problem

胡 艷楠

名古屋大学大学院情報科学研究科計算機数理科学専攻
指導教員：柳浦睦憲 教授，今堀慎治 准教授，橋本英樹 助教

1. Introduction

The rectilinear block packing problem involves packing a set of arbitrarily shaped rectilinear blocks into a larger rectangular container without overlap so as to minimize or maximize a given objective function. A rectilinear block is a polygonal block whose interior angle is either 90° or 270°. This problem is important for many industrial applications, such as VLSI design, timber cutting, and newspaper layout.

A special case of the rectilinear block packing problem is the rectangle packing problem. The *bottom-left algorithm* and the *best-fit algorithm* are known as representative works among the typical frameworks of existing construction heuristics. The main strategy of these two algorithms is the *bottom-left strategy*. In this strategy, whenever a new item is packed into the container, it is placed at the *BL position* relative to the current layout. The BL position of a new item relative to the current layout is defined as the leftmost location among the lowest *bottom-left stable feasible positions*, where a bottom-left stable feasible position is a location where the new item can be placed without overlap and cannot be moved leftward or downward.

2. Problem Description

A set of n items $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ of rectilinear blocks are given, where each rectilinear block takes a deterministic shape and size from a set of t shapes $\mathcal{T} = \{T_1, T_2, \ldots, T_t\}$. Also given is a rectangular container C with fixed width W and unrestricted height H. The task is to pack all of the items orthogonally without overlap into the container so as to minimize height H of the container that is necessary to pack all of the given items.

3. Heuristic Algorithms

For a new rectilinear block to be placed next and a set of placed items, the *overlap number* of a point (x, y) is defined as the number of items that overlap with the new block when it is placed at (x, y). In this paper, we use no-fit polygons (abbreviated as NFPs) as a crucial technique to calculate overlap numbers.

We assume that each rectilinear block is represented with a set of rectangles whose relative positions are fixed. The Find2D-BL-R algorithm is generalized from the Find2D-BL [1] algorithm to find the BL position for a rectilinear block. The number of rectangles that represent a rectilinear block R_i is denoted by m_i and $M = \sum_{i=1}^{n} m_i$. The Find2D-BL-R algorithm computes the BL position of a rectilinear block R_i in $O(m_i M \log(m_i M)) = O(m_i M \log M)$ time.

3.1 Bottom-Left Algorithm

The bottom-left algorithm can be generally explained as follows: Given a set of n rectilinear blocks $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ and an order of items (e.g., decreasing order of area), the algorithm packs all of the items one by one according to the given order, where each item is placed at its BL position relative to the current layout (i.e., the layout at the time just before it is placed).

3.2 Best-Fit Algorithm

The best-fit algorithm is explained as follows: Given a set of n rectilinear blocks $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ and a priority among them (e.g., an item with a wider bounding box has higher priority), the algorithm packs all of the items one by one into the container, where each item is placed at its BL position relative to the current layout. At the beginning of the packing process, no item is placed in the container. Whenever an item is to

be packed into the container, the algorithm calculates the BL positions of all of the remaining items relative to the current layout. In this iteration, the rectilinear block whose BL position takes the smallest x-coordinate among those with the lowest y-coordinate is packed. If there exist ties, the block with the highest priority is chosen.

3.3 Time Complexity

In addition to m_i and M, we also define m_j^{T} as the number of rectangles that represent T_j, and m is the sum of m_j^{T} for all of the shapes in $\mathcal{T}$.

The bottom-left algorithm calls the Find2D-BL-R algorithm once for every item R_i in $\mathcal{R}$ to pack it into the container. Thus, the running time of the bottom-left algorithm is $O(M^2 \log M)$.

Whenever the best-fit algorithm packs a new item into the container, the Find2D-BL-R algorithm will be called at most once for each shape. The time complexity of the best-fit algorithm is $O(nmM \log M)$.

4. Efficient Implementations

4.1 Basic Idea

In this section, we explain more efficient implementations of the bottom-left and best-fit algorithms than those proposed in Section 3.

The basic idea is to dynamically keep the layouts of NFPs called NFP layouts with respect to the current packing layout for all shapes in $\mathcal{T}$ during the packing process. Note that for each shape T_j, after packing an item R_i into the container, the NFPs in the container do not change, and we can obtain the new NFP layout for T_j by simply inserting the NFP of T_j relative to R_i. For each shape T_j, we maintain a heap $HEAP_j$ for events of the sweep line and a balanced search tree $TREE_j$ to keep the information of overlap numbers on the sweep line. We design an algorithm *FindBL* to find the BL position under this dynamic situation. The FindBL algorithm checks the events on the top of $HEAP_j$ and then deletes it from the heap one by one until it finds the BL position.

4.2 Time Complexity

For every iteration, our algorithms pack an item into the container. When an item R_i is placed, the time for modifying the balanced search trees and heaps for all shapes is $\sum_{j=1}^{t} O(m_i m_j^{\mathrm{T}} \log M) = O(m_i m \log M)$. Therefore, the total computation time for such modifications is $\sum_{i=1}^{n} O(m_i m \log M) = O(mM \log M)$ during the entire computation. As explained in Section 4.1, for each shape T_j, the sweep line moves from bottom to top only once (i.e., it never goes down), and hence $O(m_j^{\mathrm{T}} M)$ events are processed by the FindBL algorithm during the entire computation of the bottom-left or best-fit algorithm. Consequently, the total running time for computing BL positions of shape T_j is $O(m_j^{\mathrm{T}} M \log M)$. Therefore, the total computation time of this part for all shapes is $\sum_{j=1}^{t} O(m_j^{\mathrm{T}} M \log M) = O(mM \log M)$. These computations dominate the running time of the other parts of the algorithms. As a result, both the bottom-left and best-fit algorithms run in $O(mM \log M)$ time.

5. Computational Result

We performed a series of experiments based on benchmark instances. The occupation rate of the packing layouts obtained by our algorithms often reached higher than 95%. Even for instances with more than 10,000 rectilinear blocks, the proposed algorithms run in less than 10 seconds.

6. Conclusions

In this paper, we proposed a bottom-left and a best-fit algorithm for the rectilinear block packing problem. We also designed efficient implementations of these algorithms by using sophisticated data structures to dynamically keep the information.

We performed a series of experiments, and the computational results show that our algorithms are especially efficient for large-scale instances of the rectilinear block packing problem.

References

[1] S. Imahori, Y. Chien, Y. Tanaka, and M. Yagiura, Enumerating bottom-left stable positions for rectangles with overlap, In Proceedings of the 9th Forum on Information Technology 2010 (FIT2010), Issue 1, 25–30, 2010.