Computers & Operations Research 53 (2015) 206–222

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Efficient implementations of construction heuristics for the rectilinear block packing problem

Y. Hu [a,*], H. Hashimoto [a], S. Imahori [b], M. Yagiura [a]

[a] Department of Computer Science and Mathematical Informatics, Graduate School of Information Science, Nagoya University, Furocho, Chikusa, Nagoya 464-8601, Japan

[b] Department of Computational Science and Engineering, Graduate School of Engineering, Nagoya University, Furocho, Chikusa, Nagoya 464-8603, Japan

ARTICLE INFO

Available online 18 July 2014

Keywords:
Strip packing
Rectilinear blocks
Construction heuristics
Efficient implementation

ABSTRACT

The rectilinear block packing problem is a problem of packing a set of rectilinear blocks into a larger rectangular container, where a rectilinear block is a polygonal block whose interior angle is either 90° or 270°. There exist many applications of this problem, such as VLSI design, timber/glass cutting, and newspaper layout. In this paper, we design efficient implementations of two construction heuristics for rectilinear block packing. The proposed algorithms are tested on a series of instances, which are generated from nine benchmark instances. The computational results show that the proposed algorithms are especially efficient for large instances with repeated shapes.

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/glass cutting, and newspaper layout. It is among the classical packing problems and is known to be NP-hard [3].

The rectilinear block packing problem is a special case of the problem of packing general polygons, called the irregular packing problem or nesting problem, for which a series of approaches have been developed [1,4–6,11,14,19,26]. A special case of the rectilinear block packing problem is the rectangle packing problem. Many efficient algorithms have been proposed to solve the rectangle packing problem, including simulated annealing [10], hybrid algorithm [15], and quasi-human heuristic algorithm [29]. The *bottom-left algorithm* [3] and the *best-fit algorithm* [7] are known as the most remarkable works among existing construction heuristics. Compared with the rectangle packing problem, the rectilinear block packing problem is more complicated, and it is difficult to design efficient data structures to represent the relationships among the rectilinear blocks. Several heuristic methods have been proposed for the rectilinear block packing problem based on different data structures to represent the relationships among the blocks, e.g., BSG (bounded sliceline grid) [20,24], sequence-pairs [12,13,21,30], O-tree [25], B*-tree [28], TCG (transitive closure graph) [22], CBL (corner block list) [23], etc.

This paper proposes construction heuristics for the rectilinear block packing problem by generalizing representative construction heuristics for rectangle packing. It then shows how to reduce the time complexity of these algorithms. One of the basic ideas of our algorithms is that we regard each rectilinear block as a set of rectangles whose relative positions are fixed. This leads to efficient implementations of the construction heuristics, and it also allows us to deal with the problem having noncontinuous items, each of which consists of a set of (separate) rectilinear blocks whose relative positions are fixed.

The main strategy of our algorithms is the *bottom-left strategy*, which derives from the bottom-left algorithm for rectangle packing [3]. In this strategy, starting from an empty layout, items are packed into the container one by one, and 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.

Various algorithms are possible under the bottom-left strategy, and if we consider two standard rules for choosing the new item from the remaining items, the resulting algorithms become the *bottom-left algorithm* and the *best-fit algorithm*. We explain how we generalize these representative construction heuristics for

* Corresponding author.
E-mail addresses: yannanhu@nagoya-u.jp (Y. Hu), hasimoto@nagoya-u.jp (H. Hashimoto), imahori@na.cse.nagoya-u.ac.jp (S. Imahori), yagiura@nagoya-u.jp (M. Yagiura).

http://dx.doi.org/10.1016/j.cor.2014.06.021
0305-0548/

rectangle packing to solve the rectilinear block packing problem in Section 4. Observe that, it is necessary to find a method to calculate BL positions for implementing these algorithms. We generalize the *Find2D-BL* algorithm [17], which was proposed to calculate the BL position of a new rectangle relative to a rectangular container and rectangles placed in the container, to the case of rectilinear blocks. We also analyze the time complexity of the bottom-left and best-fit algorithms when they are implemented based on the generalized Find2D-BL algorithm.

We then design more efficient implementations of the bottom-left and best-fit algorithms for the rectilinear block packing problem in Section 5. The basic idea is to design sophisticated data structures that keep the information dynamically so that the BL position of each item can be found in sub-linear amortized time. We then analyze the time complexity of the resulting implementations of the two construction heuristics. In this paper, the methods of processing the rectilinear block packing problem with rotation is also proposed as the extension of our algorithms in Section 6.

We perform a series of experiments on a set of instances that are generated from nine benchmark instances. The computational results are shown in Section 7. The computational results show that the proposed algorithms are effective for large-scale instances of the rectilinear block packing problem and are especially efficient for those instances having many repeated shapes. Even for instances with 10,000 distinct shapes, our algorithms run in about one and a half hours on a PC with a 2.3 GHz Intel Core i5 processor. For instances with more than 10,000 rectilinear blocks with up to 60 distinct shapes, the algorithms run in less than 12 s, often obtaining layouts with occupation rate higher than 90%.

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 the items orthogonally without overlap into the container. We assume that the bottom left corner of the container is located at the origin $O=(0,0)$ with its four sides parallel to the x- or y-axis. The objective is to minimize height H of the container that is necessary to pack all the given items. Note that the minimization of height H is equivalent to the maximization of the occupation rate defined by $\sum_{i=1}^{n}A(R_i)/WH$, where $A(R_i)$ denotes the area of a rectilinear shape R_i. This type of problem is often called the strip packing problem (e.g., the rectangular strip packing problem for the rectangular case and the irregular strip packing problem for the case of general polygons), and according to the improved typology of Wäscher et al. [27], strip packing problems are categorized into the two-dimensional open dimension problem (2D ODP) with a single variable dimension. Fig. 1 shows an example of the rectilinear block packing problem. The layout on the right is an example packing layout after packing all the rectilinear blocks into the container given on the left of the figure. The number of rectilinear blocks n is 7, and that of shapes t is 5. The task is to pack these seven items into the rectangular container so as to minimize the height of the container.

We define the bounding box of an item R_i as the smallest rectangle that encloses R_i, and its width and height are denoted as w_i and h_i. We call the area of the bounding box, w_ih_i, the *bounding area* of R_i. The location of an item R_i is described by the coordinate (x_i,y_i) of its reference point, where the reference point is the bottom-left corner of its bounding box. For convenience, each rectilinear block and the container C are regarded as the set of points (including both interior and boundary points) whose coordinates are determined from the origin $O=(0,0)$. Then, a rectilinear block R_i placed at $v_i=(x_i,y_i)$ is represented as the Minkowski sum $R_i\oplus v_i=\{p+v_i \mid p\in R_i\}$. For a rectilinear block R_i, let $\mathrm{int}(R_i)$ be the interior of R_i. Then the rectilinear block packing problem is formally described as follows:

$$\text{minimize} \quad H$$

$$\text{subject to} \quad 0\le x_i\le W-w_i,\quad 1\le i\le n \tag{1}$$

$$0\le y_i\le H-h_i,\quad 1\le i\le n \tag{2}$$

$$\mathrm{int}(R_i\oplus v_i)\cap(R_j\oplus v_j)=\varnothing,\quad i\ne j. \tag{3}$$

The constraints (1) and (2) require that all rectilinear blocks be packed inside the container. The constraint (3) ensures that there exists no item overlapping with others.

Two cases of this problem are often considered in the literature: (1) all the items are not allowed to be rotated and (2) all the items can be rotated 90°, 180° or 270°. However, the case without rotations is assumed in this paper unless otherwise stated, because it is easy to apply the results in this paper to the case with rotations as discussed in Section 6.

3. Basic knowledge

In this section, we explain some important techniques and definitions used in our algorithms. As a crucial technique for the packing problem, the concept of no-fit polygon is explained in Section 3.1. The definition of BL positions in general and the Find2D-BL algorithm [17] to calculate the BL positions for rectangles are introduced in Sections 3.2 and 3.3. Two construction heuristics for the rectangle packing problem are then introduced in Sections 3.4 and 3.5. We first explain the bottom-left algorithm, which is one of the simplest forms among the algorithms based on the bottom-left strategy. Then we explain the best-fit algorithm, which is slightly more complicated than the bottom-left algorithm but it is known to be more effective.

3.1. No-fit polygon

The no-fit polygon (*NFP*) is a geometric technique to check overlaps of two polygons in a two-dimensional space. This concept was introduced by Art [2] in 1960s, who used the term “shape envelope” to describe the positions where two polygons can be placed without intersection. It is defined for an ordered pair of two polygons P_i and P_j, where the position of polygon P_i is fixed and polygon P_j can be moved. The NFP of P_j relative to P_i, $NFP(P_i,P_j)$ denotes the set of positions of polygon P_j having an intersection with polygon P_i, which is formally defined as follows:

$$\begin{aligned} NFP(P_i,P_j) &= \mathrm{int}(P_i)\oplus(-\mathrm{int}(P_j)) \\ &= \{u-w \mid u\in\mathrm{int}(P_i), w\in\mathrm{int}(P_j)\}. \end{aligned} \tag{4}$$

$NFP(P_i,P_j)$ is illustrated through the example in Fig. 2.

When the two polygons are clear from the context, we may simply use *NFP* instead of $NFP(P_i,P_j)$. Assume that $\partial NFP(P_i,P_j)$ denotes the boundary of $NFP(P_i,P_j)$, and $\mathrm{cl}(NFP(P_i,P_j))$ denotes the closure of $NFP(P_i,P_j)$. The no-fit polygon has the following important properties, where v_i and v_j are the positions of P_i and P_j:

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