2007－AL－114 (9)
2007／9／21

長方形配置問題に対するbest-fit法の効率的な実現

今堀 慎治 東京大学
柳浦 睦憲 名古屋大学

概要 長方形配置問題は，与えられた長方形を重なりなく，できる限り密に配置することを目的とする問題である．本問題に対する発見的解法の一つとして，Burkeらによって提案されたbest-fit法 [Operations Research 52 (2004) 655–671] があり，その性能が広く注目されている．本研究では，best-fit法の効率的な実現法を提案し，計算量の観点での最適性を示す．我々の実現法は，長方形数を n とすると，$O(n)$ の領域と，$O(n \log n)$ の計算量を必要とする．また，数値実験を通して，best-fit法の実用性について論じる．

The best-fit heuristic for the rectangular strip packing problem: an efficient implementation

Shinji Imahori University of Tokyo
Mutsunori Yagiura Nagoya University

Abstract We investigate the best-fit heuristic algorithm by Burke et al. [Operations Research 52 (2004) 655–671] for the rectangular strip packing problem. For its simplicity and good performance, the best-fit heuristic has become one of the most significant algorithms for the rectangular strip packing. In this paper, we propose an efficient implementation of the best-fit heuristic that requires linear space and $O(n \log n)$ time, where n is the number of rectangles. We prove that this time complexity is optimal, and show practical usefulness of our implementation via computational experiments.

1 Introduction

Cutting and packing problems are representative combinatorial optimization problems with many applications in various areas such as VLSI design and steel garment industry. For several decades, the field of cutting and packing has attracted the attention of many researchers and practitioners. Depending on applications, different types of cutting and packing problems need to be solved, and hence many variants of cutting and packing problems have been studied in the literature. Dyckhoff (1990) presented a typology of cutting and packing problems and categorized existing studies in this field. Wäscher et al. (2007) recently presented an improved typology of cutting and packing problems. This paper addresses the problem of placing rectangles in a larger rectangular object with a fixed width in order to minimise its height. This problem is widely called the *rectangular strip packing problem*. According to the improved typology of Wäscher et al. (2007), the rectangular strip packing problem is categorized into the two-dimensional regular open dimension problem (2D regular ODP) with a single variable dimension. We allow non-guillotine placements; i.e., we are not restricted to only performing full horizontal or vertical cuts from one sheet edge to another. As for the rotation of rectangles, we consider the following two cases: (1) Each rectangle has a fixed orientation, and (2) we allow rotations of 90 degrees. A formal definition of the problem is as follows: We are given a set of n rectangles I. Each rectangle $i \in I$ has its width w_i and height h_i, and the size of this rectangle is denoted by (w_i, h_i). If each rectangle i can be rotated by 90 degrees, this rectangle can have a size (w_i, h_i) or (h_i, w_i). We are also given a rectangular object (called *strip*), which has a fixed width W and a variable height H. Our objective is to place all the rectangles in the strip without overlap in order to minimize the height of the strip.

Almost all cutting and packing problems (including the rectangular strip packing) are known to be NP-hard, and hence it is impossible to solve them exactly in polynomial time unless P = NP. Therefore heuristics and metaheuristics are important in designing practical algorithms for cutting and packing problems. In early stages, Coffman Jr. et al. (1980) presented some level-oriented algorithms and Baker et al. (1980) proposed the bottom-left-fill algorithm for the rectangular strip packing problem. Many papers related to these heuristic algorithms have been appeared; e.g., Chazelle (1983) gave an efficient implementation of the bottom-left-fill algorithm, Jakobs (1996) and Liu and Teng (1999) presented variants of the bottom-left-fill algorithm (Jakobs' algorithm is called the bottom-left algorithm), Lodi et al. (1999) proposed new level-oriented heuristics. These algorithms have a common characteristic: Each algorithm first decides a sequence of rectangles to place, and then it places rectangles one by one in this order at an appropriate position in the strip. Some heuristic algorithms have been incorporated in metaheuristics in order to improve the quality of solutions (Hopper and Turton 2001, Jakobs 1996, Liu and Teng 1999, Lodi et al. 1999). In almost all cases, metaheuristics are utilized to find good sequences (i.e., packing orders) of rectangles.

Burke et al. (2004) proposed a different type heuristic algorithm (called the *best-fit heuristic*[1]) that does not have a sequence of rectangles to place. Instead of using a sequence of rectangles, the best-fit heuristic dynamically selects the next rectangle to place during the packing stage. Because of its simplicity and good performance, the best-fit heuristic has become one of the most significant algorithms for the rectangular strip packing. In this paper, we propose an efficient implementation of the best-fit heuristic that requires linear space and $O(n \log n)$ time, where n is the number of rectangles, and show that this time complexity is optimal. In addition to such theoretical advantage, our implementation has also practical merits; it is easy to implement and it runs very fast even for relatively small values of n. We conduct some computational experiments to confirm the efficiency in practical sense.

The remaining part of this paper is organized as follows: Section 2 describes the best-fit heuristic proposed by Burke et al. (2004). Section 3 presents an efficient implementation of the best-fit heuristic and gives a proof for the optimality of our implementation. Section 4 shows computational results on various instances of the rectangular strip packing problem.

2 Description of the best-fit heuristic

In this section, we describe the best-fit heuristic for the rectangular strip packing problem. This heuristic algorithm was proposed by Burke et al. (2004), and has been widely known with its simplicity and good performance. The best-fit heuristic is a greedy algorithm that attempts to produce good-quality placements by examining the lowest available space in the strip and then placing the rectangle that best fits the space. Unlike most heuristic algorithms (e.g., the bottom-left and bottom-left-fill methods) that have a sequence of rectangles to place, the best-fit heuristic dynamically selects the next rectangle to place during the packing stage. This enables the algorithm to make informed decisions about which rectangle should be placed next.

In order to represent the configuration in the strip, the algorithm stores a *skyline* of the strip; it consists of a set of line segments satisfying the following properties: (1) Each segment is parallel to the x-axis, (2) no line segments are hidden by other line segments or already placed rectangles, (3) each line segment touches the top edge of an already placed rectangle or the bottom edge of the strip, and (4) two adjacent line segments have different y-coordinates and have a common x-coordinate at the end points of their intervals. See Figure 1 as examples of line segments in the

[1] A level-oriented algorithm based on the best-fit heuristic for the (one-dimensional) bin packing problem is sometimes referred to by the same name. However, in this paper, we use this name to denote the algorithm by Burke et al. as in their original paper.

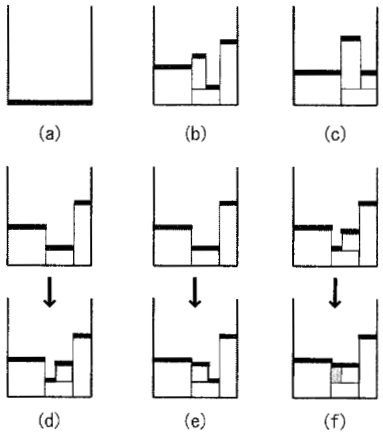

Figure 1: Description of the best-fit heuristic: (a) initial state of the strip, (b) line segments in the strip, (c) two segments on the lowest level, (d) place a rectangle with the high strategy, (e) place a rectangle with the low strategy, and (f) regard a segment as wastage.

strip. Among all the line segments in the strip, the *lowest available segment* is the line segment that has the smallest y-coordinate.

The best-fit algorithm repeats two operations until all the rectangles are placed: (1) Find the lowest available space in the strip, and (2) place a rectangle on the bottom of the space. At the beginning of the packing process no rectangles are placed in the strip, and the skyline of the strip corresponds to the bottom edge of the strip (see Figure 1(a)). As rectangles are placed, the skyline moves upward and the lowest available segment is changed with respect to both location and width (see Figure 1(b)). If there are several segments on the lowest level (i.e., they have the same y-coordinates), the algorithm selects the leftmost one as the lowest available segment (see Figure 1(c)). For the lowest available segment, the algorithm selects and places the *best fit rectangle*, where the best fit rectangle means the widest rectangle (resolving equal widths by the largest height) that has not been placed yet and can be placed on the segment without overlap (i.e., its width is not larger than that of the segment). If the width of the segment is larger than that of the best fit rectangle, the algorithm should decide where to place the rectangle among those positions on the segment. For this purpose the following three strategies have been used: (1) Place the rectangle at the left-most position on the segment (called the left strategy), (2) place the rectangle next to the higher segment adjacent to the current segment (called the high strategy) as shown in Figure 1(d), and (3) place the rectangle next to the lower segment adjacent to the current segment (called the low strategy) as shown in Figure 1(e). We note that if the lowest available segment is adjacent to the left (resp., right) edge of the strip, the high strategy places the rectangle in touch with the left (resp., right) edge of the strip and the low strategy places the rectangle next to the adjacent line segment. When the algorithm is executed, it uses one of these strategies throughout its execution. If there are no rectangles that can be placed on the segment, the relevant segment is regarded as wastage. In this case, the segment is raised to the level of the lower segment adjacent to the current segment, and the two segments are merged (see Figure 1(f)). These are the main part of the best-fit heuristic algorithm.

As mentioned in the original paper by Burke et al. (2004), a drawback of using the above greedy algorithm is that it may create a poor quality placement due to *towers*, where towers are produced when long thin rectangles are placed in portrait orientation near to the top of the strip. In solving an instance in which each rectangle can be rotated by 90 degrees, the following improving

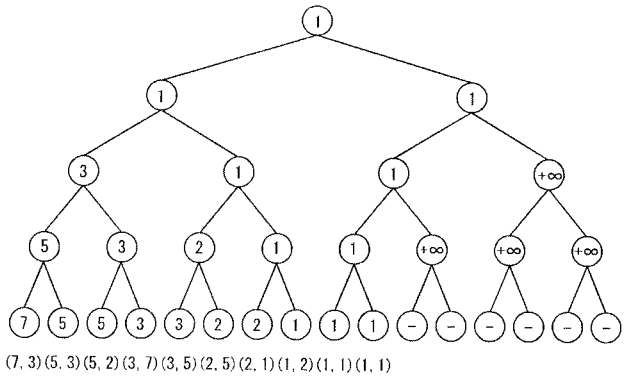

Figure 2: An example of the binary search tree to find the best fit rectangle.

operations are applied after all the rectangles have been placed in the strip. The algorithm finds a rectangle which touches the top edge of the strip (i.e., the y-coordinate of the top edge of the rectangle is H). If the rectangle is oriented in such a way that its height is greater than its width, the algorithm removes it from the strip and updates the information of the line segments related to this rectangle. The removed rectangle is then rotated by 90 degrees and placed on the lowest available segment in the strip. If the quality is improved by this operation, the algorithm looks for the next rectangle that touches the top of the strip and performs the same operation again. This operation is repeated until there is no improvement in solution quality.

3 Efficient implementation

As described in the previous section, the best-fit heuristic repeatedly searches for the lowest available space and the best fit rectangle until all the rectangles are placed in the strip. In order to implement this algorithm, Burke et al. (2004) used an array of size W to store the skyline of the strip and a sorted list of rectangles by decreasing width to find the best fit rectangle. Their implementation requires $O(n + W)$ space and $O(n^2 + nW)$ computation time. In this section, we give an efficient implementation of the best-fit heuristic algorithm. Our implementation stores the line segments (i.e., the skyline) using a heap and a doubly linked list connected by bi-directional pointers, and uses a binary search tree to find the best fit rectangle. In the preprocessing stage, these data structures are constructed. The packing stage is the main part of the best-fit heuristic; i.e., all the rectangles are placed in the strip one by one. The improving operations by removing towers are conducted in the postprocessing stage.

Preprocessing stage Our implementation constructs a binary search tree that stores the given rectangles. We first explain the case where each rectangle can be rotated by 90 degrees. For each rectangle i with its width w_i and height h_i, our implementation creates two rectangles with its size (w_i, h_i) and (h_i, w_i), respectively. Hence, we have $2n$ rectangles in total. These rectangles are sorted by decreasing width (resolving equal widths by decreasing height), and then they are stored in the leaves of a complete binary search tree with height $\lceil \log 2n \rceil$ from left to right. Each internal node of the tree keeps the value of the minimum width among its descendants. If an internal node has no rectangles as its descendants, this node keeps $+\infty$. Let us see an example with five rectangles $\{(3,5),(5,2),(1,1),(7,3),(1,2)\}$. Our implementation creates 10 rectangles $\{(3,5),(5,3),(5,2),(2,5),(1,1),(1,1),(7,3),(3,7),(1,2),(2,1)\}$, and sorts them by decreasing width as follows: $\{(7,3),(5,3),(5,2),(3,7),(3,5),(2,5),(2,1),(1,2),(1,1),(1,1)\}$. The implementation

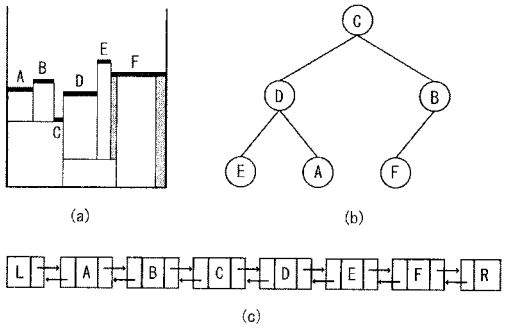

Figure 3: A skyline of the strip: (a) a skyline consists of six line segments, (b) a heap stores the segments using their y-coordinates as keys, and (c) a doubly linked list stores the segments sorted by their x-coordinates.

then constructs a binary search tree as in Figure 2. This binary search tree requires linear space and its construction takes $O(n \log n)$ time. If each rectangle has its fixed orientation, our implementation constructs a binary search tree in similar manner just with the given n rectangles.

In order to store line segments, our implementation uses a heap and a doubly linked list connected by bi-directional pointers. The heap stores the line segments using their y-coordinates as keys, where a segment with smaller y-coordinate has more priority (resolving equal y-coordinates by smaller x-coordinate). As a result, the lowest available segment can be found at the root node of the heap. The doubly linked list also stores the segments sorted by their x-coordinates. Figure 3 shows examples of the heap and the doubly linked list. Each segment appears in both of the heap and the linked list; they are connected by a bi-directional pointer. At the beginning of the execution of the algorithm, no rectangles are placed in the strip and the skyline of the strip consists of only one line segment. Hence, the heap and the doubly linked list contain only one element in the preprocessing stage; that is, their construction is done in constant time.

Packing stage The algorithm repeats two operations until all the rectangles are placed: (1) Find the lowest available segment, and (2) place the best fit rectangle on the segment. The lowest available segment in the strip can be found in constant time by looking at the root node of the heap. For this line segment, our implementation tries to find the best fit rectangle with the binary search tree; this search takes proportional time to the height of the binary search tree; i.e., $O(\log n)$ time.

If the best fit rectangle is found, the x-coordinate of this rectangle should be determined. (The y-coordinate is automatically decided to that of the lowest available segment.) For this purpose, the segments adjacent to the current segment are checked with the doubly linked list, and the x-coordinate of the best fit rectangle is determined along the chosen placement strategy (i.e., the left, high or low strategy). This is done in $O(1)$ time. The placed rectangle is then removed from the binary search tree. The leaf nodes corresponding to the rectangle (i.e., two leaves for the case with rotation or one leaf for the case without rotation) are removed and the values stored in internal nodes which are on the paths from the leaf nodes to the root node are updated; these updating operations on the binary search tree take $O(\log n)$ time. The heap and the doubly linked list should also be updated: Updating the doubly linked list takes constant time, and updating the heap takes proportional time to the height of the heap; that is, $O(\log n)$ time.

If the best fit rectangle is not found (in other words, the lowest available segment is too narrow to place a rectangle), the segment is raised to the level of the lower segment adjacent to the current segment and the two segments are merged. These operations are done in constant time with the

doubly linked list. Updating the data structures for the skyline takes $O(\log n)$ time. We note that a pointer from the linked list to the heap is needed in the case where the adjacent segments have the same y-coordinates and three segments should be merged. This gap-raising operation is applied at most $n-1$ times throughout an execution of the algorithm by the following reason: There is one line segment at the beginning of the execution. The number of segments in the strip is increased at most one when a rectangle is placed in the strip. The number of segments in the strip is decreased at least one when a gap-raising operation is applied.

Postprocessing stage In solving an instance where each rectangle can be rotated by 90 degrees, the operations to remove towers from the placement constructed in the packing stage are also considered. For this purpose, the following operations are repeated until there is no improvement in solution quality: (1) Find a rectangle that touches the top of the strip and check its orientation, and (2) rotate it by 90 degrees and place it on the lowest available segment in the strip. At the beginning of the postprocessing stage, the rectangles are sorted by decreasing order of y-coordinates of those top edges; this is done in $O(n \log n)$ time. By using this sorted list, a rectangle that touches the top of the strip can be found in constant time throughout the postprocessing stage. In order to place a removed rectangle on the lowest available segment, the algorithm check all the segments in the strip from the bottom to the top of the strip. If a segment is too narrow to place the rectangle, this segment is raised and merged; and hence the number of segments in the strip is decreased at least one. When the packing stage is done, the number of line segments in the strip is at most $n+1$, and it can be increased at most $O(n)$ throughout the postprocessing stage. Hence, we require $O(n \log n)$ time for the postprocessing stage. We note that the postprocessing stage is an optional part of the best-fit algorithm, and its execution time is negligible for most of practical instances.

Complexity of our implementation We analyze the space and time complexities of our implementation. Our implementation uses a binary search tree of size $O(n)$ to find the best fit rectangle, a combination of heap and doubly linked list of size $O(n)$ to store all the line segments and find the lowest available segment, and some information for each rectangle (size, coordinates, pointers to two leaves in the binary search tree). Therefore, our implementation requires linear space to the input size in total. As for the time complexity our implementation requires $O(n \log n)$ time in the preprocessing stage to construct a binary search tree of rectangles. In packing stage, the number of iterations is $O(n)$ and each iteration requires $O(\log n)$ time; i.e., $O(n \log n)$ time is needed. The postprocessing stage also requires $O(n \log n)$ time. In total, our implementation of the best-fit heuristic requires $O(n \log n)$ time.

Optimality of our implementation At the end of this section, we show the optimality of our implementation. As we have seen in this section, our implementation of the best-fit heuristic algorithm requires linear space and $O(n \log n)$ computation time.

Proposition 1. A lower bound on the computation time of the best-fit heuristic is $\Omega(n \log n)$.

Proof: We show that the best-fit heuristic can sort given numbers by decreasing size. Let $X = \{x_1, x_2, \ldots, x_n\}$ be an input of the sorting problem, where X is a set of unsorted positive numbers. Let us define $x_{\max} = \max\{x_1, x_2, \ldots, x_n\}$ and set the width of the strip $W = 2x_{\max}$ for an instance of the strip packing problem. A set of n rectangles is generated as follows: For each x_i, a rectangle i has its width $w_i = x_i + x_{\max}$ and height $y_i = W + 1$. It takes $O(n)$ time to construct an instance of the strip packing problem.

The best-fit heuristic is applied to the constructed strip packing instance. Any rectangle cannot be rotated by 90 degrees, any two rectangles cannot be placed at the same level, and the best-

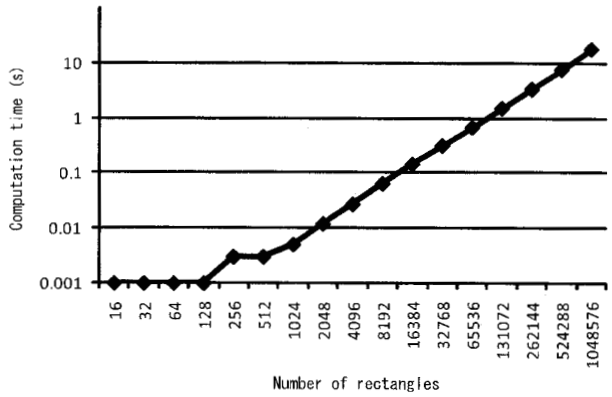

Figure 4: Computation time in seconds of our implementation with various size instances.

fit heuristic places rectangles in the strip from the bottom to the top with decreasing width of the rectangles. It is known that any sorting algorithm without particular assumptions requires $\Omega(n \log n)$ time, and the result follows. □

Putting together Proposition 1 and our implementation of the best-fit heuristic, we have the following result. We note that any implementation of the best-fit heuristic requires at least linear space.

Theorem 1. The optimal implementation of the best-fit heuristic runs in linear space and $\Theta(n \log n)$ time.

4 Computational results

We evaluate our implementation of the best-fit heuristic by some computational experiments. The algorithm was coded in C language and run on a PC with a 2.8 GHz CPU and 2 GB RAM. Test instances for these computational experiments are generated by ourselves at random. We treat instances where each rectangle can be rotated by 90 degrees. The optimal height for each instance is known: There exists a placement without any empty space for each instance. The smallest instance has 16 rectangles and the largest one has $2^{20} = 1,048,576$ rectangles. There are 10 instances for each size.

For each instance, our algorithm computes three different placements with the left, high and low strategies, and outputs the best one. The results are shown in Figures 4 and 5, where average results on 10 instances are appeared. In Figure 4, horizontal axis shows the number of rectangles and vertical axis shows the computation time in seconds. We note that this is a double logarithmic chart. From Figure 4, we can observe that our implementation runs very fast for every instance. In Figure 5, vertical axis shows the solution quality; that is, $100(H - H^*)/H^*$, where H^* is the optimal height, and hence the smaller value means a better solution. From Figure 5, we can observe that the number of rectangles has a substantial influence on the solution quality of the best-fit heuristic. We note that, in almost all cases, the solution quality of the best-fit heuristic is better than or equivalent to the existing heuristic and metaheuristic algorithms for the rectangular strip packing problem.

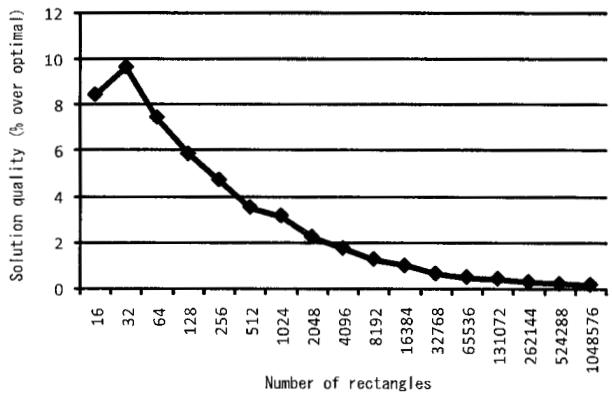

Figure 5: Solution quality of the best-fit heuristic.

References

Baker, B. S., E. G. Coffman Jr., and R. L. Rivest. (1980). "Orthogonal Packings in Two Dimensions." *SIAM Journal on Computing*, 9, 846–855.

Burke, E. K., G. Kendall, and G. Whitwell. (2004). "A New Placement Heuristic for the Orthogonal Stock-Cutting Problem." *Operations Research*, 52, 655–671.

Coffman Jr., E. G., M. R. Garey, D. S. Johnson, and R. E. Tarjan. (1980). "Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms." *SIAM Journal on Computing*, 9, 808–826.

Dyckhoff, H. (1990). "A Typology of Cutting and Packing Problems." *European Journal of Operational Research*, 44, 145–159.

Hopper, E., and B. C. H. Turton. (2001). "An Empirical Investigation of Meta-Heuristic and Heuristic Algorithms for a 2D Packing Problem." *European Journal of Operational Research*, 128, 34–57.

Jakobs, S. (1996). "On Genetic Algorithms for the Packing of Polygons." *European Journal of Operational Research*, 88, 165–181.

Liu, D., and H. Teng. (1999). "An Improved BL-Algorithm for Genetic Algorithm of the Orthogonal Packing of Rectangles." *European Journal of Operational Research*, 112, 413–420.

Lodi, A., S. Martello, and D. Vigo. (1999). "Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems." *INFORMS Journal on Computing*, 11, 345–357.

Wäscher, G., H. Haußner, and H. Schumann. (2007). "An Improved Typology of Cutting and Packing Problems." *European Journal of Operational Research*, 183, 1109–1130.