MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

The Best-fit Heuristic for the Rectangular Strip Packing Problem: An Efficient Implementation

Shinji IMAHORI, Mutsunori YAGIURA

METR 2007–53 September 2007

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

Shinji Imahori, Mutsunori Yagiura

Shinji Imahori
Department of Mathematical Informatics,
Graduate School of Information Science and Technology, University of Tokyo,
Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
email: imahori@mist.i.u-tokyo.ac.jp
phone: +81-3-5841-6907

Mutsunori Yagiura
Department of Computer Science and Mathematical Informatics,
Graduate School of Information Science, Nagoya University,
Furocho, Chikusa-ku, Nagoya 464-8603, Japan
email: yagiura@nagoya-u.jp

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 $\mathrm{O}(n \log n)$ 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. We also give a lower bound on the worst-case approximation ratio of the best-fit heuristic.

Keywords Cutting and packing, Rectangular strip packing, Best-fit heuristic, Time complexity, Approximation ratio

1 Introduction

Cutting and packing problems are representative combinatorial optimization problems with many applications in various areas such as steel and garment industry and VLSI design. For several decades, the field of cutting and packing has been attracting 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., placements are not restricted to those obtainable by guillotine cuts only, where a guillotine cut is a full horizontal or vertical cut from one sheet edge to another. As for the rotation of rectangles, the following two cases are considered: (1) Each rectangle has a fixed orientation, and (2) rotations of 90 degrees are allowed. A formal definition of the problem is as follows: We are given a set of n rectangles $I = \{1, 2, \ldots, n\}$. 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). When rectangles can be rotated by 90 degrees, the size of each rectangle i is (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. The objective is to place all the rectangles in I into the strip without overlap so as 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 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), and 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. Such heuristic algorithms are often 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), where metaheuristics are usually 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 specify 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 problem. In this paper, we propose an efficient implementation of the best-fit heuristic that requires linear space and $\mathrm{O}(n \log n)$ time, and we 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 computational experiments to confirm the efficiency in practical sense.

We also analyse theoretical guarantees on the approximation ratio of the best-fit heuristic. It is known that some simple heuristics attain good approximation guarantees; e.g., the next-fit heuristic with decreasing height attains three (Coffman Jr. et al. 1980), the bottom-left-fill heuristic with decreasing width also attains three (Baker et al. 1980). In this paper, we show a negative aspect of the best-fit heuristic: It cannot guarantee a constant approximation ratio in the worst case.

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

[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.

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. Section 5 discusses the approximation ratio of the best-fit heuristic. In Section 6, this paper will be concluded.

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 a good-quality placement by examining an available space as low as possible 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.

During the computation, the algorithm keeps a *skyline*, which consists of a sequence of line segments satisfying the following properties: (1) Each line segment is parallel to the x-axis, (2) two adjacent line segments have different y-coordinates and have exactly one common x-coordinate (i.e., the x-coordinate of the right end point of a line segment is the same as that of the left end point of its right neighbour), (3) viewed from an infinitely high position, no point in the line segments is hidden by already placed rectangles, and (4) each line segment touches the top edge of an already placed rectangle or the bottom edge of the strip. See examples of skylines shown in thick lines in Figure 1(a)–(c). Among all the line segments in a skyline, the *lowest available segment* is the one that has the smallest y-coordinate.

The best-fit algorithm repeats the following two operations until all the rectangles are placed: (1) Find the lowest available segment of the current skyline, and (2) place a rectangle on the segment. At the beginning of the packing process, no rectangles are placed in the strip, and the skyline consists of the bottom edge of the strip (see Figure 1(a)). Whenever a rectangle is placed, a part of the lowest available segment moves upward in such a way that the part of the segment hidden by the bottom edge of the placed rectangle is replaced by the top edge of the rectangle (see Figure 1(d) and (e)). If there are several segments on the lowest level (i.e., they have the same y-coordinate) as in Figure 1(c), the algorithm selects the leftmost one as the lowest available segment. For the lowest available segment, the *best fit rectangle*, which is to be placed on the segment, is defined to be 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 needs to decide where to place the rectangle among those positions on the segment. For this purpose the following three strategies are 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). 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 as far as possible from the strip edge (i.e., next to the adjacent line segment). When the best-fit algorithm is executed, it uses one of these strategies throughout its execution. If there are no rectangles that can be placed on the lowest available segment, the segment is raised to the level of the lower segment adjacent to it, and the two segments are merged (see

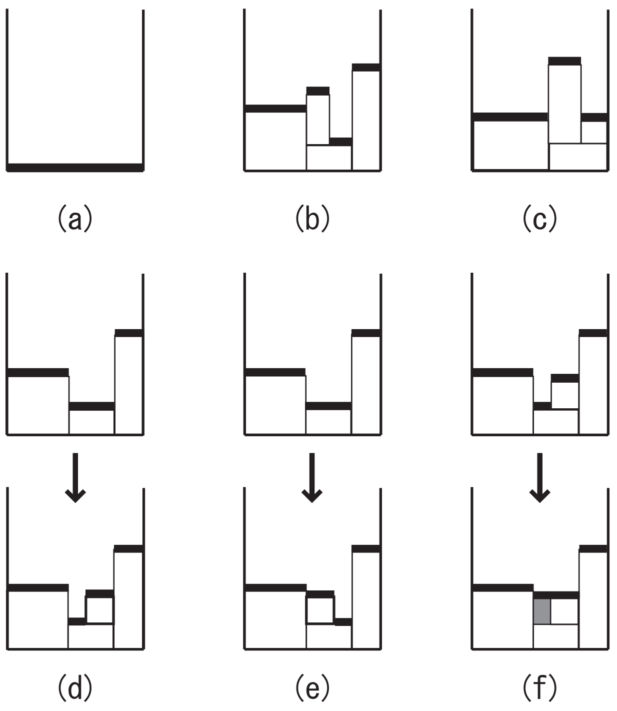

Figure 1: Description of the best-fit heuristic: (a) Initial state of the skyline, (b) line segments in the skyline, (c) two segments on the lowest level, (d) placing a rectangle with the high strategy, (e) placing a rectangle with the low strategy, and (f) raising the lowest available segment without putting a rectangle (discarding the shaded area).

Figure 1(f)). In this case, the space below the raised segment becomes waste. When all the rectangles are placed in the strip, the main packing stage of the best-fit heuristic algorithm ends.

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 operations are applied to improve the placement 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 skyline related to this rectangle. The removed rectangle is then rotated by 90 degrees and placed on the lowest available segment. If the quality is improved by this operation, the algorithm looks for the next rectangle which touches the top of the strip and performs the same operation again. This operation is repeated until there is no improvement in solution quality. This additional postprocessing stage is invoked only if rotations of rectangles are allowed.

3 Efficient implementation

As described in the previous section, the best-fit heuristic repeatedly searches for the lowest available segment 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

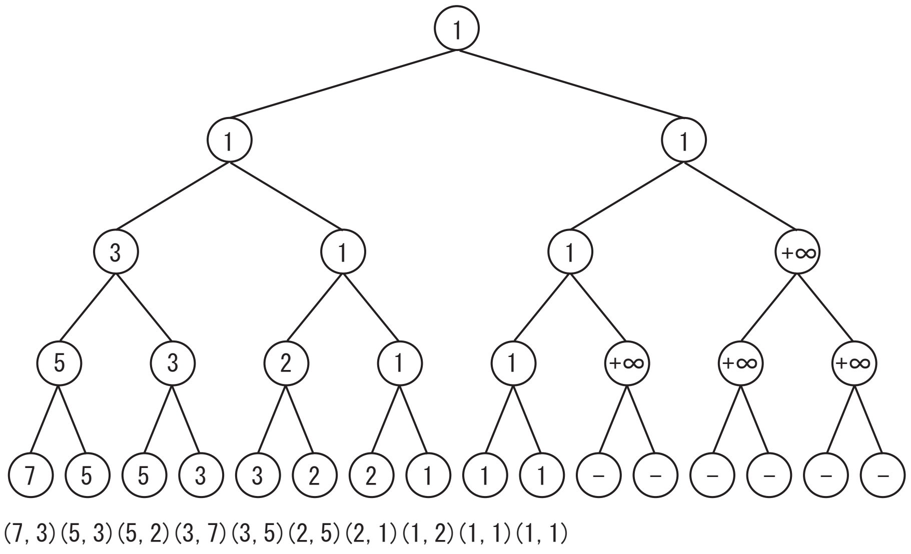

Figure 2: An example of the binary search tree to find the best fit rectangle.

skyline and a sorted list of rectangles by decreasing width to find the best fit rectangle. Their implementation requires $\mathrm{O}(n+W)$ space and $\mathrm{O}(n^2+nW)$ computation time. In this section, we give an efficient implementation of the best-fit heuristic algorithm. Our implementation stores the skyline (i.e., line segments) using a heap and a doubly linked list connected by bi-directional pointers, and it uses a binary search tree to find the best fit rectangle. Below we describe how we maintain these data structures in the following three stages: Preprocessing, packing and postprocessing stages. In the preprocessing stage, these data structures are initialized. The packing stage is the main part of the best-fit heuristic algorithm; i.e., all the rectangles are placed in the strip one by one. The postprocessing stage is the phase in which the algorithm tries to improve the placement by applying the operations of removing towers.

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, the proposed implementation creates two rectangles with size (w_i, h_i) and (h_i, w_i), respectively, corresponding to the two orientations. Hence, $2n$ rectangles are created 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 with sizes $(3,5), (5,2), (1,1), (7,3)$ and $(1,2)$. The proposed implementation creates 10 rectangles of sizes $(3,5), (5,3), (5,2), (2,5), (1,1), (1,1), (7,3), (3,7), (1,2)$ and $(2,1)$, and it sorts them by decreasing width as follows: $(7,3), (5,3), (5,2), (3,7), (3,5), (2,5), (2,1), (1,2), (1,1), (1,1)$. It then constructs a binary search tree as in Figure 2. This binary search tree requires linear space and its construction takes $\mathrm{O}(n \log n)$ time. If each rectangle has its fixed orientation, the proposed implementation constructs a binary search tree in a similar manner just with the given n rectangles.

In order to store the skyline (i.e., a set of line segments), our implementation uses a heap and a doubly linked list connected by bi-directional pointers. The heap stores the line segments

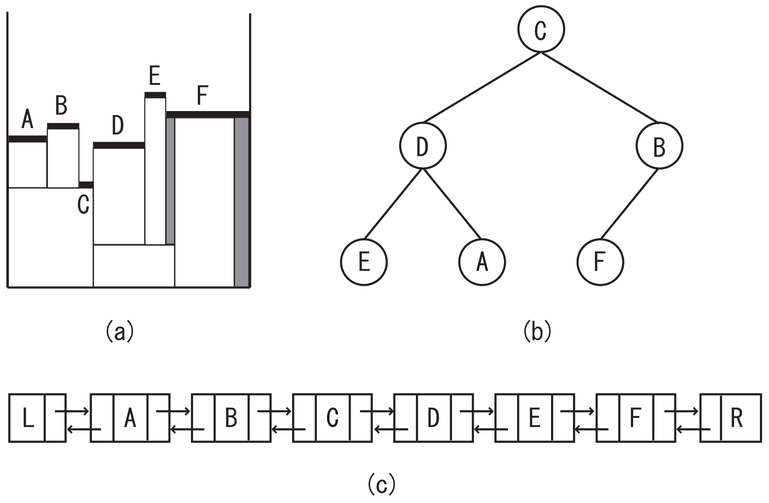

Figure 3: A skyline of the strip: (a) A skyline consisting of six line segments, (b) a heap storing the segments using their y-coordinates as keys, and (c) a doubly linked list storing the segments sorted by their x-coordinates.

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. Cells with labels L and R in Figure 3(c) correspond to the left and right boundaries, respectively. 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 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 can be found in constant time by looking at the root node of the heap. Our implementation searches the best fit rectangle to this segment using the binary search tree. Starting from the root node, it goes down the tree according to the following rule until a leaf node is reached: (1) Go to the left child if the value stored in the left child is at most the width of the lowest available segment, (2) go to the right child otherwise. This search takes time proportional 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 line segments adjacent to the current segment are checked using 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 (actually, the values of the leaves are changed to $+\infty$) and the values stored in internal nodes are updated. These updating operations on the binary search tree take $O(\log n)$ time in total, because only the internal nodes on the paths from the leaves to the root are candidates, and it takes constant time to update the value stored in an internal node if the updating operations are done from the leaves to the root. According to the changes in the skyline, the heap and the doubly linked list are also updated. The top edge of the best fit rectangle is inserted as a new line segment of the skyline, and the width of the lowest available segment is shrunk, where the lowest available segment is deleted when its width becomes 0 (this happens if the original width of the lowest available segment is the same as that of the best fit rectangle), and the new segment is merged with its adjacent segment(s) if their y-coordinates are the same. As such changes are necessary only to a constant number of segments around the lowest available segment, updating the doubly linked list takes constant time, and updating the heap takes time proportional to the height of the heap; that is, $\mathrm{O}(\log n)$ time.

If the best fit rectangle is not found (in other words, the lowest available segment is too narrow to place a remaining rectangle), then the segment is raised to the level of the lower segment adjacent to it, and the two segments are merged (if the two adjacent segments have the same height, then the three segments are merged). We call this a segment-raising operation. According to such changes in the skyline, the doubly linked list is updated in constant time, and the heap is updated in $\mathrm{O}(\log n)$ time. We note that a pointer from the linked list to the heap is needed in the case where the two adjacent segments to the current segment have the same y-coordinate and three segments are merged to one. The segment-raising operation is applied at most $n - 1$ times throughout an execution of the algorithm for the following reason. There is one line segment at the beginning of the execution. The number of line segments in the skyline is increased by at most one when a new rectangle is placed in the strip, while number of line segments is decreased by at least one when a segment-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. 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 $\mathrm{O}(n \log n)$ 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 heap and the linked list are also utilized. Hence, we require $\mathrm{O}(n \log n)$ time for the postprocessing stage. We note that the postprocessing stage is an optional part of the 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. It uses a binary search tree of size $\mathrm{O}(n)$ to find the best fit rectangle, a combination of heap and doubly linked list of size $\mathrm{O}(n)$ to store all the line segments and to find the lowest available segment, and some information of 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 $\mathrm{O}(n \log n)$ time in the preprocessing stage to construct a binary search tree of rectangles, and $\mathrm{O}(1)$ time to initialize the heap and doubly linked list. In the packing stage, the number of iterations is $\mathrm{O}(n)$ and each iteration requires $\mathrm{O}(\log n)$ time; i.e., $\mathrm{O}(n \log n)$ time is spent in total. The postprocessing stage also requires $\mathrm{O}(n \log n)$ time. In total, our implementation of the best-fit heuristic requires $\mathrm{O}(n \log n)$ 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 $\mathrm{O}(n \log n)$ computation time.

Lemma 1. The worst-case computation time of the best-fit heuristic is $\Omega(n \log n)$.

Proof: We show that the best-fit heuristic can sort given numbers in the decreasing order. 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 W of the strip to $2x_{\max}$. 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 $\mathrm{O}(n)$ time to construct this instance of the strip packing problem.

In the resulting strip packing instance, any rectangle cannot be rotated by 90 degrees, and any two rectangles cannot be placed at the same height. When the best-fit heuristic is applied to this instance, it 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 in the worst case, and the result follows. □

Putting together Lemma 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 needs linear space and runs in $\Theta(n \log n)$ time.

4 Computational results

We evaluate the proposed implementation of the best-fit heuristic via computational experiments. The algorithm was coded in C language and run on a PC with a 3.2 GHz CPU and 1 GB RAM. In Section 4.1, new test instances are introduced. In Sections 4.2 and 4.3, computational results (execution time and solution quality) on those test instances are reported.

4.1 Benchmark instances

In the literature, many benchmark instances of the rectangular strip packing problem have been introduced (e.g., Burke et al. 2004, Hopper and Turton 2001, Valenzuela and Wang 2001), and it may seem natural to use these existing instances for computational experiments. However, in this paper, new benchmark instances are generated and computational experiments are conducted on them for the following reasons: (1) The sizes of the existing test instances are not large enough to evaluate the efficiency of the proposed implementation since it can place thousands of rectangles within 0.1 seconds, and (2) the solution quality of the best-fit heuristic on representative benchmark instances were already reported by Burke et al. (2004).

We generated random test instances of the rectangular strip packing problem by two different methods: (1) A large rectangle is given and it is cut to a specified number of rectangles by a series of randomly generated guillotine cuts, and (2) a specified number of rectangles are independently and randomly generated. For every test instance generated by the first method, an optimal solution is known and it satisfies the guillotine cut constraint. Optimal solutions to the second type instances are unknown, and a simple lower bound on the height of the strip

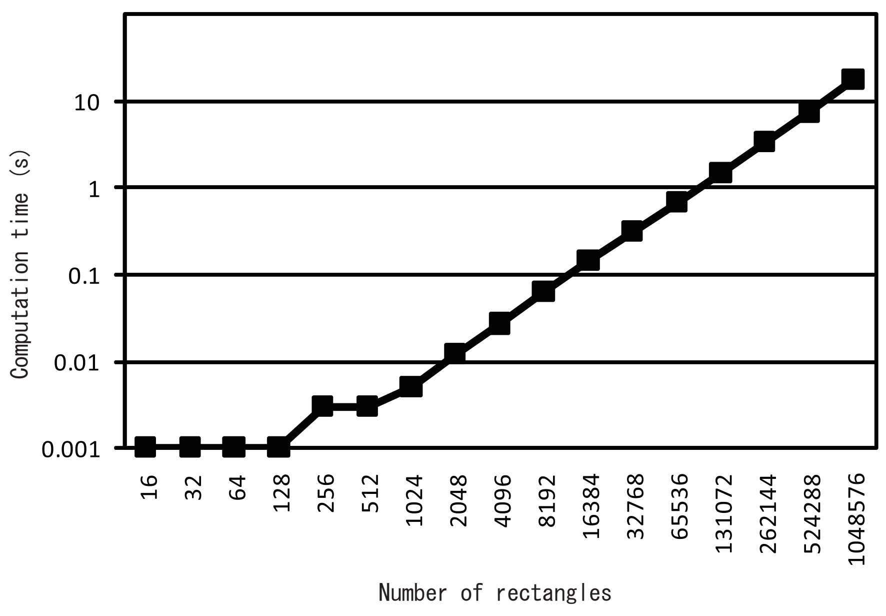

Figure 4: Computation time (in seconds) for instances with various sizes.

(i.e., $\sum_i w_i h_i / W$) is used in evaluating solution quality for instances of this type. In preliminary experiments, we observed that the difference of instance generators did not affect the execution time nor solution quality of our implementation of the best-fit heuristic. Thus, we only report the computational results on test instances generated by the first method.

The detailed settings of our test instances are as follows: (1) Each instance is generated from a large square whose size is $(5050\sqrt{n}, 5050\sqrt{n})$, (2) the aspect ratio of each rectangle is at most 5; i.e., $1/5 \leq w_i/h_i \leq 5$ holds for every rectangle i, and (3) the width and height of each rectangle is at least 100. As for the number of rectangles, there are 17 classes of instances: The smallest instance has $2^4 = 16$, the next one has $2^5 = 32$, and the largest one has $2^{20} = 1,048,576$ rectangles. For each number of rectangles, ten instances are generated with different seeds for random numbers. All the test instances are electronically available from our web site (`http://www.simplex.t.u-tokyo.ac.jp/~imahori/packing/`).

4.2 Execution time

We evaluate the execution time of the proposed implementation via computational experiments. In our computational experiments, three different placements with the left, high and low strategies are computed and the algorithm outputs the best one for each instance. That is, computation time on a test instance consists of one input operation, three executions of the placement algorithm and one output operation. As for the rotations of rectangles, each rectangle can be rotated by 90 degrees. Computation time strongly depends on the number of rectangles, and the results for various sizes are shown in Figure 4.

In this figure, horizontal axis shows the number of rectangles and vertical axis shows the computation time (average of ten different instances) 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. It can place thousands of rectangles within 0.1 seconds, it spends less than one second for test instances with up to 100,000 rectangles, and it takes less than 20 seconds for instances with about one million rectangles.

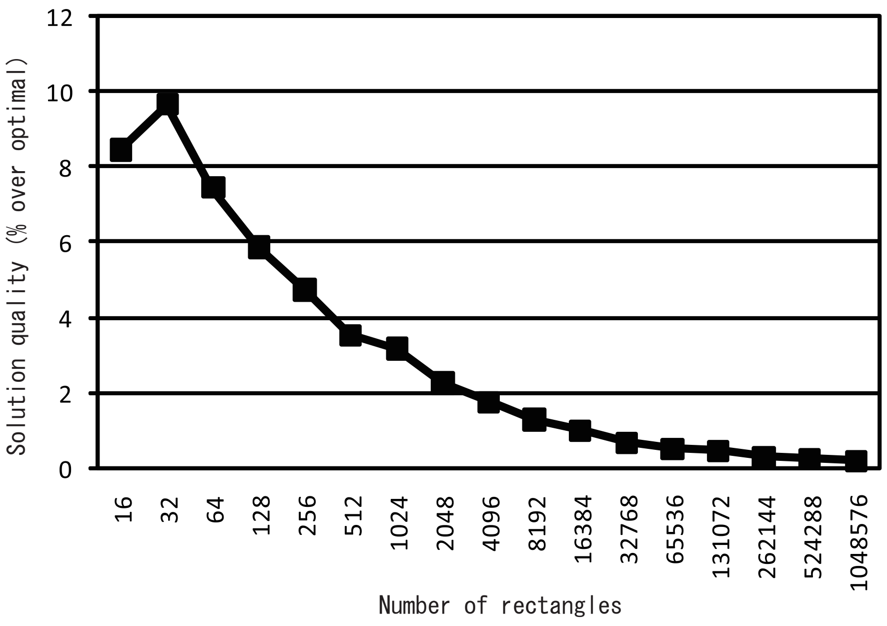

Figure 5: Solution quality (% over optimal) for instances with various sizes.

4.3 Solution quality

As denoted in the paper by Burke et al. (2004), the best-fit heuristic works very effectively for many benchmark instances. In this paper, we investigate the relationship between the number of rectangles and solution quality of the best-fit heuristic. We utilize test instances whose numbers of rectangles are from $n = 16$ to $1,048,576$. The computational results are shown in Figure 5.

In this figure, horizontal axis shows the number of rectangles and vertical axis shows the solution quality (average of ten instances), where the quality is measured by the deviation in % from the optimal height H^*, i.e., $100(H - H^*)/H^*$, 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. When the number of rectangles becomes larger, solution quality becomes better.

5 Worst-case approximation ratio

In the previous section, the solution quality of the best-fit heuristic via computational experiments was reported. This section gives a worst-case approximation ratio of the best-fit heuristic. It is known that some simple heuristic algorithms for the rectangular strip packing problem attain good approximation guarantees; e.g., the next-fit heuristic with decreasing height attains three (Coffman Jr. et al. 1980), the bottom-left-fill heuristic with decreasing width also attains three (Baker et al. 1980). To our knowledge, the best-known approximation algorithms (with respect to the worst-case approximation ratio) for the rectangular strip packing problem without rotations are (1) an approximation algorithm by Steinberg (1997) whose absolute bound on approximation ratio is two, and (2) an asymptotic FPTAS by Kenyon and Remila (2000).

We show a negative aspect of the best-fit heuristic: It cannot guarantee a constant approximation ratio in the worst case. We first consider the case with rotations by 90 degrees of rectangles. We then show another result for the case without rotations, where the left strategy

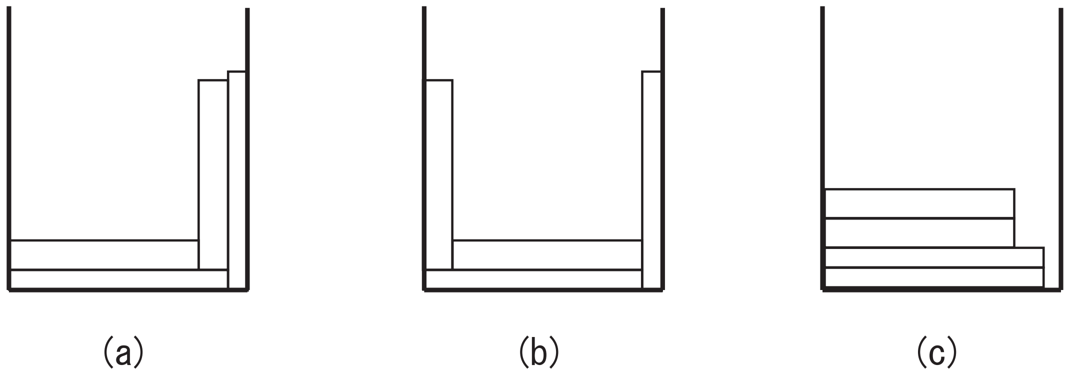

Figure 6: An adverse instance to the best-fit heuristic with rotations by 90 degrees: (a) A placement by the best-fit heuristic with the left or high strategies, (b) a placement by the best-fit heuristic with the low strategy, and (c) an optimal placement.

is used to place rectangles. Let $h_{\rm OPT}$ be the optimal height of the strip for a given instance and $h_{\rm BF}$ be the height computed by the best-fit heuristic.

Theorem 2. For any constant $M > 0$, there exist instances such that $h_{\rm BF}/h_{\rm OPT} > M$, where each rectangle can be rotated by 90 degrees.

Proof: We consider the following instance with 4 rectangles: The sizes of rectangles are $(1-2\varepsilon, 2\varepsilon), (1-2\varepsilon, 2\varepsilon), (1-5\varepsilon, 3\varepsilon)$ and $(1-5\varepsilon, 3\varepsilon)$ $(0 < \varepsilon \ll 1)$, and the strip width is $W = 1$. If the best-fit heuristic with the left or high strategies is applied to this instance, the resulting placement has height $1-2\varepsilon$ as shown in Figure 6(a). If the best-fit heuristic with the low strategy is applied, the resulting placement also has height $1-2\varepsilon$ as shown in Figure 6(b). The optimal height of this instance is 10ε as shown in Figure 6(c). By setting ε to $0 < \varepsilon < 1/(10M+2)$, the result follows. □

Theorem 3. For any constant $M > 0$, there exist instances such that $h_{\rm BF}/h_{\rm OPT} > M$, where each rectangle has a fixed orientation and the left strategy is used to place rectangles.

Proof: We design instances of the rectangular strip packing problem for which the best-fit heuristic cannot work effectively. Let $k\,(\geq 4)$ be an even number and ε be a small positive (more precise definition of ε is given later). A strip with width $W = k^k$ and a set of rectangles $I = I_1 \cup I_2 \cup \cdots \cup I_{k/2}$ are given, where each subset of rectangles I_j for $j = 1, 2, \ldots, k/2$ is defined as follows: Let $p_j = k^{k-j} - k^{k-j-1}$. For each $i = 1, 2, \ldots, p_j - 1$, there are $2k^{j-1} + 2$ rectangles with four different sizes $(k^{k-j+1} - ik + 1, \varepsilon), (k^j - k^{j-1}, (2p_j - 2i - 1)\varepsilon), (k^{k-j+1} - ik, \varepsilon)$ and $(k^{j-1}, 1 - (2i-1)\varepsilon)$, where there are k^{j-1} congruent rectangles with the first and third sizes, respectively. There are also k^{j-1} congruent rectangles whose size is $(k^{k-j} + 1, 1 - 2(p_j - 1)\varepsilon)$. In total, the set I_j consists of $2k^{k-1} - 2k^{k-2} + 2k^{k-j} - 2k^{k-j-1} - k^{j-1} - 2$ rectangles. Each rectangle in I_j has width w that satisfies $k^{j-1} \leq w < k^j$ or $k^{k-j} < w < k^{k-j+1}$.

Let us estimate $h_{\rm BF}$ on this instance. When the best-fit heuristic is applied to this instance, the widths of the lowest available segments in the first $4k^{k-1} - 4k^{k-2} - 3$ steps are sufficient for all the remaining rectangles (i.e., the widest rectangle is selected to place) or less than k (i.e., too narrow to place rectangles belonging to $I_2 \cup I_3 \cup \cdots \cup I_{k/2}$). Hence, rectangles belonging to the set I_1 are placed in the strip first. After $4k^{k-1} - 4k^{k-2} - 3$ steps, all the rectangles in I_1 have already been placed and no other rectangles have been placed yet. In the skyline at this moment,

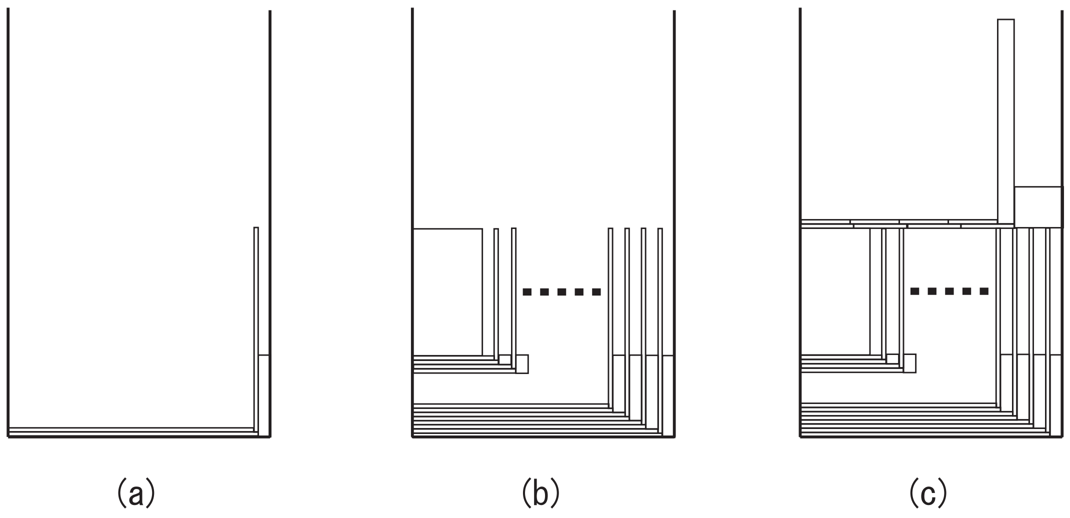

Figure 7: An adverse instance ($k = 4$) to the best-fit heuristic without rotations: (a) First four rectangles are placed in the strip, (b) $4k^{k-1} - 4k^{k-2} - 3$ rectangles that belong to I_1 are placed in the strip, and (c) next $2k + 2$ rectangles are placed in the strip.

line segments whose y-coordinates are 1 and those whose y-coordinates are less than 1 appear alternately, and any line segment whose y-coordinate is less than 1 cannot accommodate any remaining rectangle. Thus, all these segments are merged with their adjacent segments, and the resulting skyline consists of a line segment whose width is W and y-coordinate is 1. Rectangles belonging to I_2 are placed in the strip next, and a similar situation happens. The segments whose y-coordinates are less than 2 are narrower than the width of any remaining rectangle, and the skyline becomes a line segment with width W at $y = 2$. The remaining rectangles are placed similarly for $j = 3, 4, \ldots, k/2$. Finally, the height of the strip h_{BF} becomes $k/2$. See Figure 7 as an example of the placement by the best-fit heuristic.

We then consider a feasible placement for this instance, which is obtained by modifying the above placement. All the rectangles whose heights are more than $k^k\varepsilon$ are removed from the placement generated by the best-fit heuristic, and the rectangles left in the strip are moved as downward as possible. The height of the strip becomes $(2k^{k-1} - 2k^{k/2-1} - k)\varepsilon$ at this moment. The removed rectangles (i.e., rectangles whose heights are more than $k^k\varepsilon$) are then placed in the strip; since the total width of such rectangles is less than the width of the strip, all of them can be placed on a horizontal line (in other words, on the same level). The maximum height of the rectangles is $1 - \varepsilon$. We now have a feasible placement whose strip height is $1 + (2k^{k-1} - 2k^{k/2-1} - k - 1)\varepsilon$, and this implies that $h_{\mathrm{OPT}} < 1 + 2k^{k-1}\varepsilon$ holds.

Based on these, $h_{\mathrm{BF}}/h_{\mathrm{OPT}} > k/2(1+2k^{k-1}\varepsilon)$ holds for this instance. By setting k to $k > 2M$ and ε to $0 < \varepsilon < (k - 2M)/4k^{k-1}M$, the result follows. □

6 Conclusions

In this paper we have proposed an efficient implementation of the best-fit heuristic for the rectangular strip packing problem. The proposed implementation requires linear space and $\mathrm{O}(n \log n)$ time, which is optimal for the best-fit heuristic. The proposed implementation is simple, and hence it is easy to implement. We confirmed through computational experiments that our implementation required very small execution time: It spent less than one second for test instances with up to 100,000 rectangles. We also showed that the best-fit heuristic cannot guarantee a constant approximation ratio in the worst case.

Acknowledgements This work was partially supported by Grants-in-Aid for Scientific Research, by the Ministry of Education, Culture, Sports, Science and Technology of Japan.

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.

Kenyon, C., and E. Remila. (2000). "A Near-Optimal Solution to a Two-Dimensional Cutting Stock Problem." *Mathematics of Operations Research*, 25, 645–656.

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.

Steinberg, A. (1997). "A Strip-Packing Algorithm with Absolute Performance Bound 2." *SIAM Journal on Computing*, 26, 401–409.

Valenzuela, C. L., and P. Y. Wang. (2001). "Heuristics for Large Strip Packing Problems with Guillotine Patterns: An Empirical Study." *Proceedings of the 4th Metaheuristics International Conference*, 417–421.

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