Practical Algorithms for Two-dimensional Packing

Shinji IMAHORI, Mutsunori YAGIURA
and Hiroshi NAGAMOCHI

METR 2006–19 March 2006

Practical Algorithms for Two-dimensional Packing

Shinji Imahori
Department of Mathematical Informatics
Graduate School of Information Science and Technology
University of Tokyo, Tokyo 113-8656, Japan
E-mail: imahori@simplex.t.u-tokyo.ac.jp

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

Hiroshi Nagamochi
Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University, Kyoto 606-8501, Japan
E-mail: nag@i.kyoto-u.ac.jp

1 Introduction

Cutting and packing problems consist of placing a given set of (small) items into one or more (larger) objects without overlap so as to minimize/maximize a given objective function. This is a combinatorial optimization problem with many important applications in the wood, glass, steel and leather industries, as well as in LSI and VLSI design, newspaper paging, and container and truck loading. For several decades, cutting and packing has attracted the attention of researchers in various areas including operations research, computer science, manufacturing, etc.

Cutting and packing problems can be classified using different criteria. The dimensionality of a problem is one of such criteria, and most problems are defined over one, two or three dimensions. In this chapter we consider two-dimensional problems. The next criterion to classify two-dimensional packing problems is the shape of items to pack. We focus on the *rectangle packing problem* in this chapter. This problem has been actively studied the past few decades. When the shapes of the items to be packed are polygons or arbitrary shapes, the problem is called *irregular packing.* We also discuss in this chapter recent research in this area.

Almost all two-dimensional packing problems 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 very important to design practical algorithms for these problems. We survey practical algorithms for two-dimensional packing problems in this chapter. We also survey various schemes used to represent solutions to rectangle packing problem, and introduce algorithms based on these *coding schemes.*

The remainder of this chapter is organized as follows: Section 2 defines the rectangle packing problem and its variations. Section 3 introduces coding schemes for the rectangle packing problem, which are used to represent solutions. Section 4 presents heuristic algorithms, from the traditional to the latest ones, for the rectangle packing problem. Section 5 discusses practical algorithms based on metaheuristics. In Sections 4 and 5, discusses computational results for the various algorithms on benchmark instances. Section 6 defines the irregular packing problem and practical algorithms for this problem are presented.

2 Rectangle Packing Problem

We consider the following two-dimensional rectangle packing problem. We are given n items (small rectangles) $I = \{1, 2, \ldots, n\}$, each of which has width w_i and height h_i, and one or many large objects (rectangles). We are required to place the items orthogonally without any overlap (an edge of each item is parallel to an edge of the object) so as to minimize (or maximize) a given objective function. The rectangle packing problem arises in many industrial applications, often with slightly different constraints, and many variants of this problem have been considered in the literature. The following characteristics are important to classify the problems [1, 2]: type of assignment, assortment of objects, and assortment of items. We will review some specific variations of the rectangle packing problem in this section. We should mention two more important constraints for the rectangle packing problem: orientation and guillotine cut constraint. As for the orientation of the items, we usually assume that "each rectangle has a given fixed orientation" or "each rectangle can be rotated by 90°." Rotation of items is not allowed in newspaper paging or when the items to be cut are decorated or corrugated, whereas orientation is free in the case of plain materials etc. Guillotine cut constraint signifies that the items must be obtained through a sequence of edge-to-edge cuts parallel to the edges of the large object (see Figure 1 for an example), which is usually imposed by technical limitations of the automated cutting machines or the material.

We introduce six types of rectangle packing problems that have been

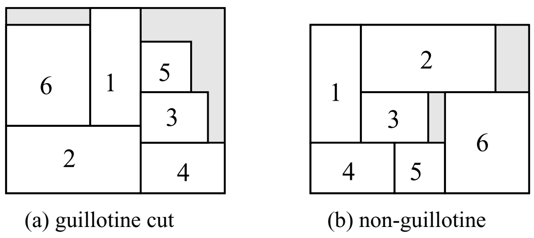

Figure 1: Examples of placements with/without guillotine cut constraint

actively studied. For simplicity, we define the problems assuming that each item has a fixed orientation and the guillotine cut constraint is not imposed unless otherwise stated. It is straightforward to extend our definitions for other cases where each item can be rotated by 90° and/or the guillotine cut constraint is imposed. We first consider two types of typical rectangle packing problems with one large rectangular object, which may grow in one or two dimensions, where all the items are to placed disjointly. The problems are called *strip packing* and *area minimization*.

Strip packing problem: We are given n items (small rectangles) each having width w_i and height h_i, and one large object (called a strip) whose width W is fixed, but its height H is variable. The objective is to minimize the height H of the strip such that all items can be packed into the strip.

Area minimization problem: We are given n items each having width w_i and height h_i, and one large rectangular object, where both its width W and height H are variables. The objective is to minimize the area WH of the object such that all items can be packed into the object.

In Sections 3, 4 and 5, we focus mainly on the strip packing (and area minimization) problem, which is formally formulated as the following mathematical program:

minimize the height of the strip H (or the area of the large object WH)

subject to

$$0 \leq x_i \leq W - w_i, \quad \text{for all } i \in I \tag{1}$$

$$0 \leq y_i \leq H - h_i, \quad \text{for all } i \in I \tag{2}$$

At least one of the next four inequalities holds for every pair i and j:

$$x_i + w_i \leq x_j, \tag{3}$$

$$x_j + w_j \leq x_i, \tag{4}$$

$$y_i + h_i \leq y_j, \tag{5}$$

$$y_j + h_j \leq y_i, \tag{6}$$

where (x_i, y_i) is the coordinate of the lower left corner of an item i. The constraints (1) and (2) mean that all items must be placed into the large object, and the constraints (3) to (6) mean that no two items overlap (that is, each inequality signifies one of the four relative locations: left-of, right-of, above and below).

Two other rectangle packing problems are the *two-dimensional bin packing* and *knapsack* problems which have (many or one) fixed sized objects.

Two-dimensional bin packing problem: We are given a set of items, where each item i has width w_i and height h_i, and an unlimited number of large objects (rectangular bins) having identical width W and height H. The objective is to minimize the number of rectangular bins used to place all the items.

Two-dimensional knapsack problem: We are given a set I of items, where each item $i \in I$ has width w_i, height h_i and value c_i. We are also given a rectangular knapsack with fixed width W and height H. The objective is to find a subset $I' \subseteq I$ of items with the maximum total value $\sum_{i \in I'} c_i$ such that all items $i \in I'$ can be packed into the knapsack.

For the two-dimensional bin packing problem, Lodi et al. [3] proposed practical heuristic and metaheutistic algorithms and performed computational experiments on various benchmark instances. For the two-dimensional knapsack problem, Wu et al. [4] proposed heuristic algorithms that are effective for many test instances.

We should also mention the following two problems, *two-dimensional cutting stock* and *pallet loading*. For some industrial applications, such as mass production manufacturing, many small items of an identical shape or relatively few classes of shapes are packed into the objects. The following two problems are useful for modeling these situations.

Two-dimensional cutting stock problem: We are given a set of items each with width w_i, height h_i and demand d_i. We are also given an unlimited number of objects having identical width W and height H. The objective is to minimize the number of objects used to place all the items (i.e., for each i, we place d_i copies of item i into the objects).

Pallet loading problem: We are given sufficiently large number of items with identical size (w, h), and one large rectangular object with size (W, H). The objective is to place the maximum number of items into the object, where each item can be rotated by 90°.

Note that, the pallet loading problem with a fixed orientation of items is trivial to solve. Among many studies on the two-dimensional cutting stock problem, Gilmore and Gomory [5] provided one of the earliest solution method. They proposed a column generation scheme in which new cutting patterns are produced by solving a generalized knapsack problem. Recently, some practical algorithms for the two-dimensional cutting stock problem have been proposed [6, 7]. Morabito and Morales [8] proposed a simple but effective algorithm for the pallet loading problem.

The complexity of the pallet loading problem is open (this problem is not known to be in class NP, because of the compact input description), whereas the other problems we defined in this section are known to be NP-hard.

3 Coding Schemes for Rectangle Packing

In this section, we review coding schemes for rectangle packing problems. For simplicity, we focus on the strip packing problem. An effective search will be difficult if we search the x and y coordinates of each item directly, since the number of solutions is uncountable and eliminating the overlap between items is not easy. To overcome this difficulty, various coding schemes have been proposed and many algorithms for rectangle packing problems are based on some coding schemes.

A *coding scheme* consists of a set of coded solutions, a mapping from coded solutions to placements, and a *decoding algorithm* that computes for a given coded solution the corresponding placement using the mapping. (The mapping is sometimes defined by the decoding algorithm.) Properties of a coding scheme (and a decoding algorithm) are given below.

1. There exists a coded solution that corresponds to an optimal placement.
2. The number of all possible coded solutions is finite, where a small total number is preferable
   provided that property 1 is satisfied.
3. Every coded solution corresponds to a feasible placement.
4. Decoding is possible in polynomial time. Fast algorithms are more desirable.

Some of the coding schemes in the literature satisfy all of the above four properties, but others do not.

One of the most popular coding schemes is to represent a solution by a permutation of the n items, where a coded solution (i.e., a permutation of n items) specifies the placement order. The number of all possible coded solutions is $O(n!)$, which is smaller than other coding schemes in the literature, and every permutation corresponds to a packing without items overlapping. A decoding algorithm computes a placement from a given coded solution by specifying the locations of the items one by one, which defines the mapping from coded solutions to placements and is sometimes called a *placement rule*. The decoding time complexity and the existence of a coded solution that corresponds to an optimal placement depend on the decoding algorithms. In the literature, a number of decoding algorithms for permutation coding schemes have been proposed. We will explain some typical decoding algorithms in Section 4, and compare them theoretically and experimentally.

We now explain different types of coding schemes for rectangle packing problems. The schemes we explain hereafter specify the relative locations for each pair of items by a coded solution. In other words, for every pair of items, a coded solution determines one of the four inequalities (3) to (6) that must be satisfied to avoid item overlap. The placement corresponding to a coded solution is the best one among those that satisfy the relative locations specified by the coded solution.

One of the most popular coding schemes of this type is to represent a solution by an n-leaf binary tree [9]. This coding scheme can represent only slicing structures (in other words, each placement obtained by this representation always satisfies the guillotine cut constraint). The leaves of a binary tree correspond to items, and each internal node has a label 'h' or 'v', where h stands for horizontal and v stands for vertical. This coding scheme uses $O(n)$ space to represent a solution, and the number of all possible coded solutions is $O(n!2^{5n-3}/n^{1.5})$ [9]. In this scheme, one of the four relative locations are assigned for each pair i and j of items as follows: If i is a left descendant of an internal node u with 'h' label and j is a right descendant of the same internal node u (i.e., u is the least common ancestor of i and j), then we must place i to the left of j (i.e., $x_i + w_i \leq x_j$). If the label of the least common ancestor is 'v', then we place i below j (i.e., $y_i + h_i \leq y_j$). Figure 2 shows an example of a binary tree representation and a placement that satisfies these constraints. For example, in the figure, there is an internal node with 'h' label for which the node for item 6 is a left descendant and the node for item 1 is a right descendant, and hence item 6 is placed to the left of item 1; the node for item 4 is a left descendant and the node for item 3 is a right descendant of the least common ancestor with 'v' label, and hence item 4 is placed below item 3 and so forth. For a given binary tree τ, let Π_τ denote the set of all placements that satisfy the above horizontal/vertical constraints. The placement corresponding to a coded solution τ is one of the best (i.e., the most compact) placements in Π_τ. Though Π_τ contains infinitely many placements for any τ, natural decoding algorithms for this coding scheme runs in linear time of the number of items, and compute one of the best placements among Π_τ. Moreover, for any placement π that satisfy the guillotine cut constraint, there exists a binary tree τ that satisfies $\pi \in \Pi_\tau$. That is, the binary tree coding scheme satisfies all of the four desirable properties of a coding scheme if the guillotine cut constraint is imposed.

Murata et al. [10] proposed a coding scheme called *sequence pair*. For

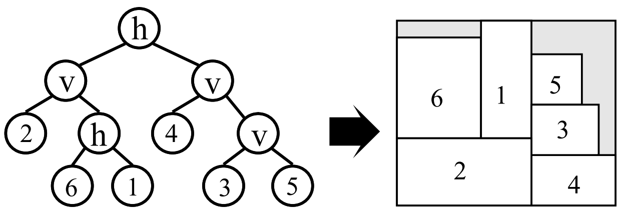

Figure 2: A binary tree representation τ and a solution $\pi \in \Pi_\tau$

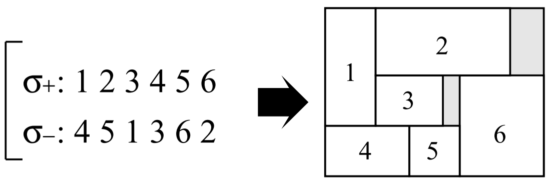

Figure 3: A sequence pair representation σ and a solution $\pi \in \Pi_\sigma$

the sequence pair representation, a solution is represented by a pair of permutations $\sigma = (\sigma_+, \sigma_-)$ of the n items (see Figure 3 for an example). Based on this coded solution, we assign relative locations for each pair of items i and j as follows: If item i is before item j in both permutations σ_+ and σ_-, then item i must be placed to the left of j. If i is before j in σ_+ and after j in σ_-, then we place i above j. For example, in Figure 3, element 1 is before element 2 in both permutations, and hence item 1 is placed to the left of item 2; element 2 is before element 3 in permutation σ_+ and after element 3 in σ_-, and hence item 2 is placed above item 3 and so on. For a given pair of permutations $\sigma = (\sigma_+, \sigma_-)$, let Π_σ be the set of placements that satisfy the above constraints. The placement corresponding to a coded solution σ is one of the best placements in Π_σ. Murata et al. [10] proposed an $O(n^2)$ time decoding algorithm to obtain one of the best placements $\pi \in \Pi_\sigma$ for a given coded solution σ. Takahashi [11] improved the time complexity of the decoding algorithm to $O(n \log n)$; Tang et al. [12] further improved it to $O(n \log \log n)$. Moreover, for any feasible placement π, there exists a coded solution σ that satisfy $\pi \in \Pi_\sigma$ (such a σ can be computed in $O(n \log n)$ time [13]). That is, the sequence pair coding scheme satisfies all of the four desirable properties of a coding scheme.

Nakatake et al. [14] proposed a coding scheme called *bounded sliceline grid* (in short, BSG). BSG consists of a set of small rooms that are separated by horizontal and vertical segments, where the number of rooms in the horizontal and vertical directions, denoted p and q, respectively, are parameters that satisfy $pq \geq n$ (see Figure 4 (a) with $p = q = 6$). It introduces one of the four orthogonal relations (left-of, right-of, above and below) uniquely

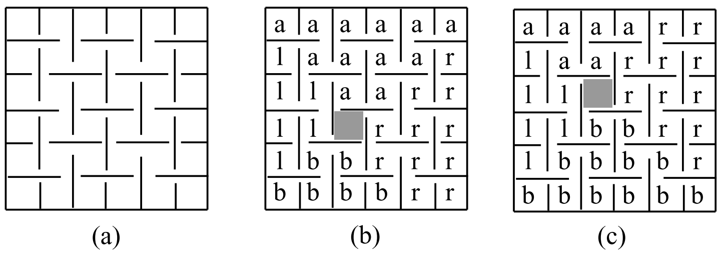

Figure 4: Rooms of bounded sliceline grid representation and relative locations

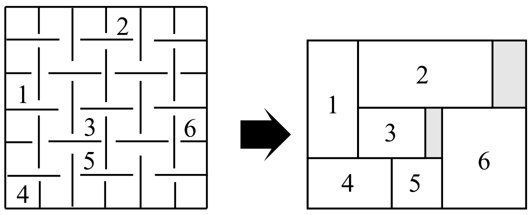

Figure 5: A bounded sliceline grid representation α and a solution $\pi \in \Pi_\alpha$

for each pair of rooms (see Figures 4 (b) and (c)). In these figures, a room with label l (resp., r, a, b) is left-of (resp., right-of, above, below) the shaded room. A solution is represented by an assignment of the items to rooms, where at most one item can be assigned to each room. The assigned items inherit the relations defined on the rooms. Figure 5 shows an example of an assignment of items to rooms and a placement that satisfy all specified constraints of relative locations. For example, in the figure, item 3 is placed to the right of item 1, item 3 is placed below item 2, item 3 is placed above item 4 and so forth. For a given assignment α, let Π_α be the set of all placements that satisfy the constraints given by α. The placement corresponding to a coded solution α is one of the best placements in Π_α. A decoding algorithm proposed by Nakatake et al. [14] runs in linear time with respect to the number of small rooms pq, and can find one of the best placements $\pi \in \Pi_\alpha$ for a given coded solution α. As for the existence of a coded solution that corresponds to an optimal placement, it is known that an assignment α such as $\pi \in \Pi_\alpha$ always exists for any placement π if and only if $p \geq n$ and $q \geq n$ hold. The bounded sliceline grid coding scheme with parameters $p \geq n$ and $q \geq n$ satisfies all of the four desirable properties of a coding scheme.

There are many other coding schemes which describe the relative locations for each pair of items. Guo et al. [15] proposed a tree representation called O-tree: Two ordered trees for the horizontal and vertical directions are used to represent a coded solution. This coding scheme can represent non-slicing structures and the number of all possible coded solutions is $O(n!2^{2n-2}/n^{1.5})$; this is smaller than the number of all coded solutions by the binary tree representation for slicing structures. There exists a coded solution corresponding to any placement π that satisfy the bottom left stability; that is, in the resulting placement, all items cannot be moved any further to the bottom or to the left. Chang et al. [16] extended the result by Guo et al. [15]. They proposed another tree representation called B*-tree; it is easy to implement this data structure and a decoding algorithm for B*-tree runs in linear time with respect to the number of items. Sakanushi et al. [17] proposed another coding scheme called quarter-state sequence: They utilized a string of items and labels to represent a solution and their decoding algorithm runs in linear time of the number of items.

4 Heuristics for Rectangle Packing

In this section, we describe heuristic algorithms for rectangle packing problems. We first explain some heuristic algorithms based on the permutation coding scheme. Those algorithms consist of two phases: (1) construct a permutation and (2) place the items one by one according to the permutation.

For the first phase, a standard strategy to construct a permutation is "a larger item has higher priority than a smaller one". To realize this, the items are sorted by some criteria, e.g., decreasing height, decreasing width or decreasing area. It is difficult to decide a priori which criterion is the best for numerous instances that arise in practice. Hence, many algorithms generate several permutations with different criteria, and apply a decoding algorithm to all such permutations.

Let's consider the second phase, i.e., decoding algorithm for permutations. We first explain *level algorithms* in which the placement is obtained by placing items from left to right in rows forming levels (see Figure 6 for an example). The first level is the bottom of the object, and each subsequent level is along the horizontal line coinciding with the top of the tallest item packed on the level below.

The most popular level algorithms are the *next fit*, *first fit* and *best fit* strategies, which are extended from the algorithms for the (one-dimensional) bin packing problem. Let i $(i = 1, 2, \ldots, n)$ denote the current item to be placed, and s be the level created most recently, where the bottom of the object is level 1 created at the beginning of an algorithm.

- *Next fit* strategy: Item i is packed on level s left justified (i.e., place it at the left-most feasible position) if it fits. Otherwise, a new level $(s := s + 1)$ is created and i is packed on it left justified.

- *First fit* strategy: We check whether or not item i fits from level 1 to

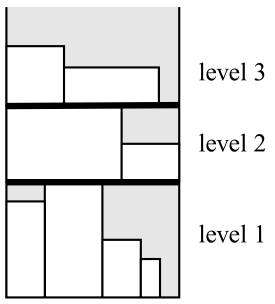

Figure 6: An example of level packing

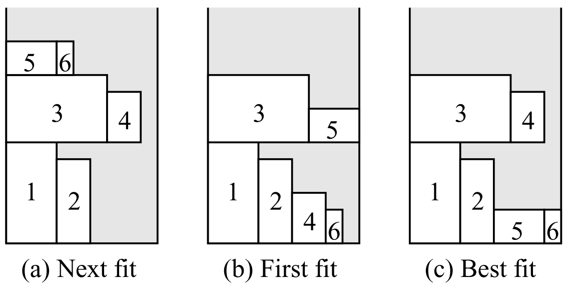

Figure 7: Three level algorithms for the strip packing problem

level s, and pack it left justified on the first level where it fits. If no level can accommodate i, it is placed on a new level as in the next fit strategy.

- *Best fit* strategy: Item i is packed left justified on the level that minimizes the unused horizontal space among those where it fits. If no level can accommodate i, it is placed on a new level as in the next fit strategy.

Computation time of these algorithms is $O(n)$, $O(n \log n)$ and $O(n \log n)$, respectively, if appropriately implemented. The above strategies are illustrated through the example in Figure 7 (in this figure, items are sorted by decreasing height and are numbered accordingly). The resulting placements of these algorithms always satisfy the guillotine cut constraint. More precisely, they are so-called two-stage guillotine placements in that they can be cut out in two stages: the first stage for horizontal cuts and the second stage for vertical cuts.

A different classical approach, and the most documented one, is the *bottom left* approach. The first algorithm of this type was proposed by Baker

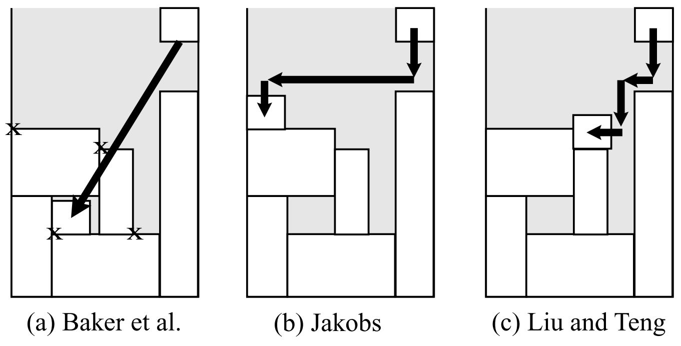

Figure 8: Three bottom left algorithms for the strip packing problem

et al. [18] in 1980, and some variants of this method have been proposed the last couple of decades. A common characteristic of this type of algorithms is to place items one by one at the bottom left stable positions; that is, in the resulting placement, all items cannot be moved any further to the bottom or to the left.

Baker et al. [18] used a bottom-left rule that places each item at the leftmost point among the lowest possible positions. This approach is called *bottom left fill* (in short, BLF) strategy in [19, 20], which is illustrated through the example in Figure 8 (a). The 'x' marks in the figure show the bottom left stable positions. There are natural algorithms that require $O(n^3)$ time in the worst case for this strategy, and Hopper and Turton implemented one of them in their article [20]. Chazelle [21] devised an efficient algorithm that requires $O(n^2)$ time and $O(n)$ space in the worst case.

Jakobs [22] utilized another bottom-left method: For each item, first place it at the top right location of the object and make successive sliding moves down and to the left alternately as long as possible (see an example in Figure 8 (b)). This strategy is called *bottom left* (in short, BL) in [19, 20] and it runs in $O(n^2)$ time, if appropriately implemented. We will see a comparison of BL and BLF algorithms through our computational experiments later on. Liu and Teng [23] developed another bottom-left heuristics similar to Jakobs's algorithm. In their strategy, the downward movement has priority such that items slide leftwards only if no downward movement is possible (see Figure 8 (c)). This algorithm also runs in $O(n^2)$ time.

There are more algorithms which utilize the permutation to represent a solution. For example, Lodi et al. [3] proposed several decoding algorithms such as *floor ceiling, alternate directions* and *touching perimeter*, and experimentally compared these algorithms with other decoding algorithms in the literature. Wu et al. [4] proposed complicated decoding algorithms, which runs in $O(n^4 \log n)$ or $O(n^5 \log n)$ time and has achieved certain computational success.

We discuss a different type of heuristic algorithm proposed by Burke

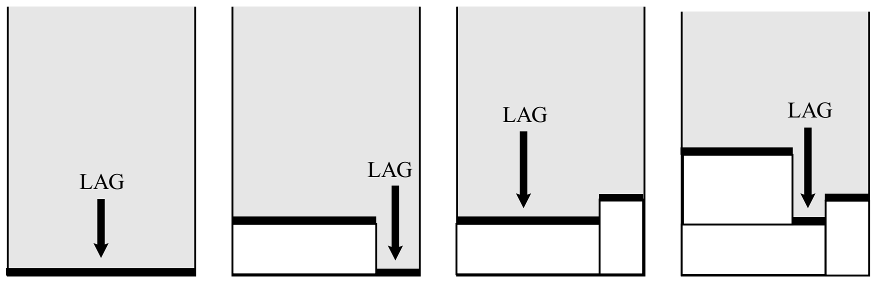

Figure 9: A heuristic algorithm for the strip packing by Burke et al. [19]

et al. [19] in 2004. This method does not have a permutation of items to place, but dynamically decide the next item to place during the packing stage. More precisely, it finds the lowest available gap (LAG) within the large object and then places the item that best fits there (see Figure 9 for an example). This enables the algorithm to make informed decisions about which item should be packed next and where it should be placed. A natural implementation of this strategy runs in $O(n^2)$ time. We will also present experimental results for this algorithm.

At the end of this section, we compare typical heuristic algorithms through computational experiments. Test instances given by Hopper and Turton [20] were used for the experiments. There are seven different categories C1, C2, ..., C7 with the number of items ranging from 17 to 197, with each category having three instances. The optimal solution for all instances are known; these instances have placements without any dead space (in other words, these instances have perfect packings). The results of algorithms BL-R, BL-DW, BL-DH, BLF-R, BLF-DW and BLF-DH are taken from [20], where BL (resp., BLF) means that item are placed with BL (resp., BLF) strategy into the object, and R, DW and DH signify the types of permutations: R means random permutation, DW (resp., DH) means that items are sorted by decreasing width (resp., decreasing height). The results by Burke et al. reported in [19] algorithm (denoted BKW) are also shown.

Computational results are shown in Table 1. Each row corresponds to a heuristic algorithm and each column corresponds to a category of instances, where n is the number of items for each instance. In this table, the relative distances in % between the optimal and resulting solutions are reported. The computation time for each instance is within one second on a PC with an 850 MHz CPU (for BKW) or a 200 MHz CPU (for others). From Table 1, we can observe that BLF outperformed BL by up to 25% and that pre-ordering the items by decreasing width or decreasing height for BL and BLF algorithms increased the packing quality by up to 10% compared to random

Table 1: Solution quality of heuristic algorithms for the strip packing problem

	C1	C2	C3	C4	C5	C6	C7
n	16	25	28	49	72	97	196
BL	25	39	33	33	31	34	41
BL-DH	17	68	27	21	18	19	31
BL-DW	18	31	24	18	22	21	29
BLF	14	20	17	15	11	12	10
BLF-DH	11	42	12	6	5	5	4
BLF-DW	11	12	12	5	5	5	5
BKW	12	7	10	4	3	2	2

permutations. Moreover, the algorithm by Burke et al. outperformed other algorithms, especially for large instances.

5 Metaheuristics for Rectangle Packing

In the last decade, many local search and metaheuristic algorithms for rectangle packing problems have been proposed. Dowsland [24] was one of the early researchers who implemented metaheuristics for rectangle packing problems. Her simulated annealing (in short, SA) algorithm explores both feasible and infeasible (i.e., some items overlap) solutions. During the search, the objective is to reduce the overlapping area. Computational results for small problem instances are reported in [24].

Let's explain some metaheuristic algorithms based on the permutation coding scheme. Those algorithms consist of two phases; (1) find a good permutation using metaheuristics and (2) the decoding algorithm places the items one by one following the permutation order. In [22], Jakobs proposed a metaheuristic algorithm for the strip packing problem. In this algorithm, he uses a genetic algorithm (in short, GA) to find a good permutation, and places items with using the bottom left strategy explained in the previous section. He treats not only rectangle packing problems but also irregular packing problems, and reports several computational results. Liu and Teng [23] also proposed a GA algorithm using their BL type decoding algorithm.

In [20], Hopper and Turton compare the performance of various metaheuristics (multistart local search (MLS), SA, GA and so on) with two decoding rules on small and large test instances. The computational results reported in [20] are shown in Table 2. The first decoding rule is the BL

Table 2: Solution quality of metaheutistic algorithms for the strip packing problem

	C1	C2	C3	C4	C5	C6	C7
GA+BL	6	10	8	9	11	15	21
SA+BL	4	7	7	6	6	7	13
MLS+BL	9	18	11	14	14	20	25
GA+BLF	4	7	5	3	4	4	5
SA+BLF	4	6	5	3	3	3	4
MLS+BLF	7	10	7	7	6	7	7

heuristic proposed by Jakobs [22], and the second one is the BLF strategy proposed by Baker et al. [18]. The stopping criterion for each algorithm is a fixed number of iterations, and the computation time for each instance is about 50,000 times as large as the simple heuristic algorithms (BL and BLF). The representation of this table is similar to Table 1. From this table, we observe that the performance of the hybrid algorithms is strongly dependent on the decoding rule and the instance size. Moreover, it is also reported that certain computation time is needed to attain better solutions with metaheuristics than the solutions obtained by well-designed heuristics such as BLF-DW and BLF-DH.

We now discuss metaheuristic algorithms based on other coding schemes. For the sequence pair representation, Murata et al. [10] proposed an SA algorithm and Imahori et al. [13] proposed an iterated local search (in short, ILS) algorithm. They used metaheuristics to find a good coded solution where each coded solution is evaluated with their own decoding algorithms. Chang et al. [16] and Nakatake et al. [14] proposed SA algorithms using B*-trees and BSG, respectively. One of the advantages of the above algorithms is generality: Imahori et al. [13] incorporated "spatial cost functions" into their algorithms which was used to handle various types of rectangle packing problems and scheduling problems. Chang et al. [16] and Nakatake et al. [14] designed algorithms which can treat the rectangle packing problem with additional constraints such as pre-placed items, soft modules and etc.

Recently, more effective algorithms for rectangle packing problems have been proposed. Lesh et al. [25] proposed a stochastic search variation of the bottom left heuristics for the strip packing problem. Their algorithm outperforms other heuristic and metaheuristic algorithms based on the bottom left strategy reported in the literature. Furthermore, they incorporated their algorithm in an interactive system that combines the advantages of computer speed and human reasoning. Using the interactive system, they succeeded in producing significantly better solutions than their original algorithm quickly.

Imahori et al. [26] proposed an improved metaheuristic algorithm based on sequence pair representation. Metaheuristic algorithms generate numerous number of coded solutions and evaluate all of them. Hence, the efficiency of metaheuristic algorithms strongly depends on the time complexity of decoding algorithms. Imahori et al. proposed new decoding algorithms to evaluate all coded solutions in various neighborhoods efficiently. As a result, they attained an amortized constant time to evaluate one coded solution in basic neighborhoods.

Bortfeldt [27] proposed a GA for the strip packing problem that works without any encoding of solutions. Instead of using a coding scheme, fully defined layouts are directly manipulated by specific genetic operators. He conducted thorough computational experiments using existing benchmark instances with up to 5000 rectangles, and compared his algorithm with eleven competing methods which were proposed in 1993 to 2004. He reported that his GA performed best among them.

For more information of the metaheuristic algorithms applied to rectangle packing problems, we refer the reader to articles by Hopper and Turton [20] and Bortfeldt [27].

6 Irregular Packing Problem

In this section, we consider the two-dimensional *irregular packing problem*, which has been actively studied in the last decade. The irregular packing problem has many practical applications, e.g., the garment, shoe and shipbuilding industries, and many variants of this problem have been considered in the literature. Among the numerous variants of this problem, the irregular strip packing problem has been studied extensively.

Irregular strip packing problem: We are given n items of arbitrary shapes, and one object (called a strip) with constant width W but variable height H. The objective is to minimize the height H of the strip such that all the items can be packed into the strip.

In this section, we mainly focus on fixed orientation packing. The problem in which items can be rotated (freely or by some fixed degrees) has also been studied in the literature. One of the main differences between rectangle and irregular packing problems is that the intersection test between irregular items is considerably more complex than the case with rectangular items. To overcome this difficulty, some approximation techniques and geometric algorithms have been incorporated into the packing algorithms.

One popular idea for speeding up the intersection test is to represent the items (irregular shapes) approximately. Oliveira and Ferreira [28] proposed two approaches to the irregular strip packing problem, and one of them uses this type of approach. Their approach is based on a raster representation (in other words, bitmap representation) of the irregular shapes to be placed. This approximation allows a quick test overlapping, but suffers from inaccuracy, caused by the approximation inherent in the raster representation. Their another approach uses a polygon-based representation which does not use any approximation technique. Both methods allow overlap in the solutions, and the extent of overlap is penalized by an evaluation function. They try to find a good solution via SA, where the algorithms aim to reduce the overlap to zero.

Okano [29] proposed a heuristic algorithm for the irregular two-dimensional bin packing problem using a scanline representation. He approximates a two-dimensional item with a set of parallel line segments. He also uses a clustering technique; some items are gathered and packed tightly, and then these items are treated as one new item. Okano designed his algorithm for the irregular two-dimensional bin packing problem; however, his technique is also useful for treating the irregular strip packing problem. He conducted computational experiments with real instances from a ship building company, and reported that the quality of the resulting layouts was sufficiently high for practical use.

Jakobs [22] used another type of approximation scheme; for all irregular shapes, the minimum bounding rectangles of the given items is calculated and his algorithm treats these rectangles instead of the original shapes. For these shapes, a good placement is computed by his BL decoding algorithm and GA techniques. After finding a good placement for the rectangles, the algorithm replaces rectangles with the original irregular shapes, and computes a better placement of the irregular items. He reported computational results for this algorithm. Jakobs also discussed an idea of clustering several shapes, and finding the minimum bounding rectangle of several items.

Dighe and Jakiela [30] proposed an algorithm for the irregular strip packing problem, which is based on a clustering method with a tree structure. Their algorithm uses a tree structure to represent a solution: The leaves of a tree correspond to items, and clustering operations are applied to items from the leaves to the root of the tree. To find a good coded solution (i.e., tree), they utilized GA. Dighe and Jakiela created new test instances and conducted computational experiments on these instances.

One of the most popular geometric techniques used for the intersection test is *no-fit polygon*. The concept of no-fit polygon was introduced by Art [31] in 1966, who used the term “shape envelope” to describe the positions where two items can be placed without intersection. Albano and Sapuppo [32] proposed an algorithm to solve the irregular strip packing problem with this geometric technique. This was the first paper that used

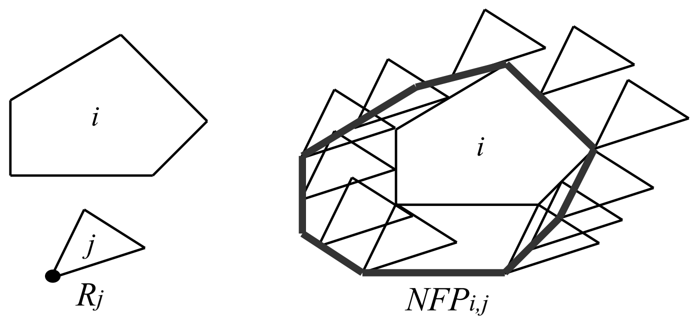

Figure 10: An example of the no-fit polygon $NFP_{i,j}$ of convex items i and j

the term "no-fit polygon". This concept is also known as Minkowski sums, and is utilized in various fields such as motion planning for polygonal robots.

The no-fit polygon of item j relative to item i ($NFP_{i,j}$) is the set of all loci of the reference point of item j (denoted R_j) such that items i and j have a common point when the position of item i is fixed (each item is considered as the set of points on the boundary and inside it). If both items i and j are convex, the boundary of $NFP_{i,j}$ is the trace of R_j when item j slides along the boundary of item i. See Figure 10 for an example of the no-fit polygon $NFP_{i,j}$ of convex polygons i and j. From the definition of no-fit polygons, it follows that:

- if the reference point R_j of item j is placed in the interior of $NFP_{i,j}$, then item j overlaps item i;
- if R_j is placed on the boundary of $NFP_{i,j}$, then item j touches item i;
- if R_j is placed in the exterior of $NFP_{i,j}$, then item j neither overlaps nor touches item i.

The problem of finding the relative position of two polygons is transformed into a simpler problem of finding the relative position of one point and one polygon. To achieve a non-overlapping compact layout, each item should have its reference point on the boundary of at least one no-fit polygon and in the exterior of all the other no-fit polygons.

Albano and Sapuppo [32] proposed an algorithm to solve the irregular strip packing problem using no-fit polygons. They approached the problem using a bottom left algorithm which utilized the no-fit polygon to reduce the geometric complexity of the packing process. Their algorithm places each item one by one at the right frontier called the leading edge of the current layout only, i.e., without hole filling capabilities. See Figure 11 for an example of the leading edge and holes in a layout. Blazewicz et al. [33] presented an extension of the work performed by Albano and Sapuppo [32].

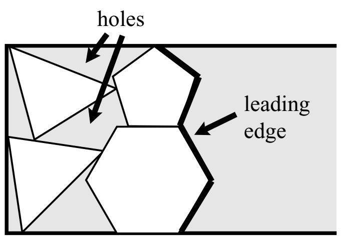

Figure 11: The leading edge and holes in a layout of irregular shapes

Their method is an extension of the bottom left fill algorithm; that is, their approach attempts to fill holes in the existing layout before attempting to place an item on the leading edge. Their algorithm utilizes the tabu search technique to produce moves from one solution to another.

Oliveira et al. [34] also tackled the irregular strip packing problem using no-fit polygons. Their algorithm places all small items one by one at a nonoverlapping position touching at least one item already placed. Several criteria to choose the next item to place and its orientation were proposed (in this article, they treated a problem such that each item can be rotated by some fixed degrees). Different evaluation functions were also proposed to evaluate partial solutions and to decide the position of each item. A total of 126 variants of the algorithm, generated by the complete set of combinations of criteria and evaluation functions, were computationally compared. In their computational experiments, they solved several types of test instances; test instances from a fabric cutting company, and a test instance generated by Blazewicz et al. [33]. Oliveira et al. compared their results against an implementation of Albano and Sapuppo's algorithm [32], and against results from Blazewicz et al. [33]. In some cases, their new algorithm generated better solutions than the best known solutions in the literature; more precisely, their algorithm generated solutions that ranged from 6.2% better to 4% worse than the best known results.

Gomes and Oliveira [35] developed shape ordering heuristics for an extended irregular packing algorithm similar to that given by Oliveira et al. [34]. The algorithm is improved by the introduction of the inner-fit rectangle, which is derived from the concept of no-fit polygon and represents the feasible set of points for placing a new polygon inside the object. In addition to this extension of geometric techniques, the paper introduces a 2-exchange heuristic for manipulating a permutation that specifies the order of placing items one by one. They generated some initial permutations with various criteria, e.g., random, decreasing order of area, decreasing order of the longest length of items, and improved them using the 2-exchange heuristic over a number of iterations. In [35], they conducted thorough computational experiments and compared the proposed algorithm with existing algorithms. They improved almost all best known solutions for well-known benchmark instances (except for an instance called SHAPE0).

Gomes and Oliveira [36] also developed a hybrid algorithm of simulated annealing with linear programming (LP) technique to solve the irregular strip packing problem. In this algorithm, a neighborhood structure based on the exchange of items on the layout was used for SA. For a given layout (which is neither feasible nor tight), they solved LP to locally optimize the layout. Computational tests were conducted using 15 benchmark instances that are commonly used in the literature, and the best results published so far were improved for all instances by their new algorithm.

Burke et al. [37] proposed a heuristic algorithm for the irregular strip packing problem with new shape overlap resolution techniques applied to the given shapes directly (i.e., without reference to no-fit polygons). In this article, they treated not only polygons (i.e., shapes with line representation) but also shapes that incorporate circular arcs and holes, and proposed overlap resolution techniques for line & line, line & arc, and arc & arc. Items are placed one by one according to a permutation (coded solution) with the bottom left fill strategy, and tabu search is used in order to find a good permutation. They conducted computational experiments on 26 existing benchmark instances and 10 new test instances with items having circular arcs and holes. Their technique produced 25 new best solutions for the 26 existing benchmark instances; most of them were found within 5 minutes on a PC with a 2 GHz CPU.

7 Conclusions

In this chapter, we surveyed practical algorithms for the two-dimensional rectangle packing problem and the irregular packing problem, both of which have many industrial applications. For the rectangle packing problem, we first introduced some coding schemes (in other words, how to represent a solution) such as permutation, binary tree, sequence pair and bounded sliceline grid. We then explained various heuristic and metaheuristic algorithms, most of which are based on these coding schemes. For some representative algorithms, we reported computational results on benchmark instances and compared them. The irregular packing problem has also been studied extensively in the last decade. The main difference between rectangle and irregular packing problems is that the intersection test between irregular items is considerably more complex. To overcome this difficulty, some different types of methodologies have been proposed; no-fit polygon is one of the representative and effective ones. We explained some heuristic and metaheuristic algorithms based on these techniques for the irregular packing problem.

The survey in this chapter is by no means comprehensive, but we hope this article gives valuable information to the readers who are interested in devising practical algorithms for cutting and packing problems. Fortunately, there have been many survey papers (thirty or more in these twenty years) on cutting and packing problems; e.g., Dychhoff [1] and Wäscher et al. [2] presented typologies of cutting and packing problems and categorized existing literature, and Hopper and Turton [20] investigated heuristic and metaheuristic algorithms for the rectangle packing problem.

References

[1] Dyckhoff, H., A typology of cutting and packing problems, *Eur. J. of Oper. Res.,* 44, 145, 1990.

[2] Wäscher, G., Haußner, H., and Schumann, H., An improved typology of cutting and packing problems, *Working Paper 24, Faculty of Economics and Management, Guericke University Magdeburg*, 2004.

[3] Lodi, A., Martello, S., and Vigo, D., Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, *INFORMS J. on Comput.,* 11, 345, 1999.

[4] Wu, Y.L., Huang, W., Lau, S., Wong, C.K., and Young, G.H., An effective quasi-human based heuristic for solving the rectangle packing problem, *Eur. J. of Oper. Res.,* 141, 341, 2002.

[5] Gilmore, P.C. and Gomory, R.E., Multistage cutting stock problems of two and more dimensions, *Oper. Res.,* 13, 94, 1965.

[6] Valdés, R.A., Parajón, A., and Tamarit, J.M., A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems, *Computers and Oper. Res.,* 29, 925, 2002.

[7] Vanderbeck, F., A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem, *Management Sci.,* 47, 864, 2001.

[8] Morabito, R. and Morales, S., A simple and effective recursive procedure for the manufacturer's pallet loading problem, *J. of the Oper. Res. Society,* 49, 819, 1998.

[9] Preas, B.T. and van Cleemput, W.M., Placement algorithms for arbitrarily shaped blocks, in *Proc. of the DAC,* 1979, 474.

[10] Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y., VLSI module placement based on rectangle-packing by the sequence-pair, *IEEE Trans. on CAD,* 15, 1518, 1996.

[11] Takahashi, T., An algorithm for finding a maximum-weight decreasing sequence in a permutation, motivated by rectangle packing problem, Technical Report of the IEICE, VLD96, 1996, 31.

[12] Tang, X., Tian, R., and Wong, D.F., Fast evaluation of sequence pair in block placement by longest common subsequence computation, *IEEE Trans. on CAD,* 20, 1406, 2001.

[13] Imahori, S., Yagiura, M., and Ibaraki, T., Local search algorithms for the rectangle packing problem with general spatial costs, *Math. Prog.,* 97, 543, 2003.

[14] Nakatake, S., Fujiyoshi, K., Murata, H., and Kajitani, Y., Module packing based on the BSG-structure and IC layout applications, *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems,* 17, 519, 1998.

[15] Guo, P.N., Takahashi, T., Cheng, C.K., and Yoshimura, T., Floorplanning using a tree representation, *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems,* 20, 281, 2001.

[16] Chang, Y.C., Chang, Y.W., Wu, G.M., and Wu. S.W., B*-trees: a new representation for non-slicing floorplans, in *Proc. of the DAC,* 2000, 458.

[17] Sakanushi, K., Kajitani, Y., and Mehta, D.P., The quarter-state-sequence floorplan representation, *IEEE Trans. on Circuits and Sys.,* 50, 376, 2003.

[18] Baker, B.S., Coffman Jr., E.G., and Rivest, R.L., Orthogonal packing in two dimensions, *SIAM J. on Comput.,* 9, 846, 1980.

[19] Burke, E.K., Kendall, G., and Whitwell, G., A new placement heuristic for the orthogonal stock-cutting problem, *Oper. Res.,* 52, 655, 2004.

[20] Hopper, E. and Turton, B.C.H., An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, *Eur. J. of Oper. Res.,* 128, 34, 2001.

[21] Chazelle, B., The bottom-left bin-packing heuristic: an efficient implementation, *IEEE Trans. on Computers,* 32, 697, 1983.

[22] Jakobs, S., On genetic algorithms for the packing of polygons, *Eur. J. of Oper. Res.,* 88, 165, 1996.

[23] Liu, D. and Teng, H., An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, *Eur. J. of Oper. Res.,* 112, 413, 1999.

[24] Dowsland, K., Some experiments with simulated annealing techniques for packing problems, *Eur. J. of Oper. Res.,* 68, 389, 1993.

[25] Lesh, N., Marks, J., McMahon, A., and Mitzenmacher, M., New heuristic and interactive approaches to 2D rectangular strip packing, *ACM J. of Experimental Algorithmics,* 10(1-2), 1, 2005.

[26] Imahori, S., Yagiura, M., and Ibaraki, T., Improved local search algorithms for the rectangle packing problem with general spatial costs, *Eur. J. of Oper. Res.,* 167, 48, 2005.

[27] Bortfeldt, A., A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces, *Eur. J. of Oper. Res.,* (to appear).

[28] Oliveira, J.F. and Ferreira, J.S., Algorithms for nesting problems, in *Applied Simulated Annealing,* Vidal, R.V. (ed.), 1993, 255.

[29] Okano, H., A scanline-based algorithm for the 2D free-form bin packing problem, *J. of the Oper. Res. Soc. Japan,* 45, 145, 2002.

[30] Dighe, R. and Jakiela, M.J., Solving pattern nesting problems with genetic algorithms employing task decomposition and contact detection, *Evolutionary Computation,* 3, 239, 1996.

[31] Art, R.C., An approach to the two dimensional irregular cutting stock problem, Technical Report, 36-Y08, IBM Cambridge Scientific Center Report, 1966.

[32] Albano, A. and Sapuppo, G., Optimal allocation of two-dimensional irregular shapes using heuristic search methods, *IEEE Trans. on Systems, Man and Cybernetics,* 10, 242, 1980.

[33] Blazewicz, J., Hawryluk, P., and Walkowiak, R., Using a tabu search for solving the two-dimensional irregular cutting problem, *Annals of Oper. Res.,* 41, 313, 1993.

[34] Oliveira, J.F., Gomes, A.M., and Ferreira, J.S., TOPOS – A new constructive algorithm for nesting problems, *OR Spektrum,* 22, 263, 2000.

[35] Gomes, A.M. and Oliveira, J.F., A 2-exchange heuristic for nesting problems, *Eur. J. of Oper. Res.,* 141, 359, 2002.

[36] Gomes, A.M. and Oliveira, J.F., Solving irregular strip packing problems by hybridising simulated annealing and linear programming, *Eur. J. of Oper. Res.,* (to appear).

[37] Burke, E.K., Hellier, R., Kendall, G., and Whitwell, G., A new bottom-left-fill heuristic algorithm for the 2D irregular packing problem, *Oper. Res.,* (to appear).