数理解析研究所講究録 1349 巻 2004 年 36-45

Local Search Algorithms for the Two-Dimensional Cutting Stock Problem with a Given Number of Different Patterns

京都大学・情報学研究科　今堀 慎治 (Shinji Imahori)
柳浦 睦憲 (Mutsunori Yagiura)
足達 信也 (Shinya Adachi)
茨木 俊秀 (Toshihide Ibaraki)
Graduate School of Informatics,
Kyoto University

豊田工業大学・工学研究科　梅谷 俊治 (Shunji Umetani)
Graduate School of Engineering,
Toyota Technological Institute

Abstract: We consider the two-dimensional cutting stock problem which arises in many applications in industries. In recent industrial applications, it is argued that the setup cost for changing patterns becomes more dominant and it is impractical to use many different cutting patterns. Therefore, we consider the pattern restricted two-dimensional cutting stock problem, in which the total number of applications of cutting patterns is minimized while the number of different cutting patterns is given as a parameter n. For this problem, we develop local search algorithms. As the size of the neighborhood plays a crucial role in determining the efficiency of local search, we propose to use linear programming techniques for the purpose of restricting the number of solutions in the neighborhood. In this process, to generate a cutting pattern, it is required to place all the given products (rectangles) in the stock sheet (two-dimensional area) without mutual overlap. For this purpose, we develop a heuristic algorithm using an existing rectangle packing algorithm with the sequence pair coding scheme. Finally, we generate random test instances of this problem and conduct computational experiments, to see the effectiveness of the proposed algorithms.

Keywords: Two-dimensional cutting stock problem, Linear programming, Rectangle packing, Neighborhood, Local search.

1 Introduction

We consider the two-dimensional cutting stock problem, which is one of the representative combinatorial optimization problems, and arises in many industries such as steel, paper, wood, glass and fiber. This problem is NP-hard, since this is a generalization of the two-dimensional bin packing problem and the one-dimensional cutting stock problem, which are already known to be NP-hard, and various heuristic algorithms have been proposed [1, 8]. In recent cutting industries, the setup cost for changing patterns becomes more significant and it is often impractical to use many different cutting patterns. We consider the two-dimensional cutting stock problem with a given number of patterns n (2DCSPn), in which the total number of applications of cutting

patterns is minimized while the number of different cutting patterns is given as a parameter [7]. We assume that each product can be rotated by 90°, and do not assume any constraints on products' placement such as "guillotine cut".

2DCSPn asks to determine a set of cutting patterns and their numbers of applications. The problem of deciding the number of applications for each pattern becomes an integer programming problem (IP), and we propose a heuristic algorithm based on its linear programming (LP) relaxation. We also propose a local search algorithm to find a good set of cutting patterns. As the size of the neighborhood plays a crucial role in determining the efficiency of local search, we propose to use linear programming techniques for the purpose of restricting the number of solutions in the neighborhood. To generate a feasible cutting pattern, we have to place all products in the two-dimensional area without mutual overlap. For this purpose, we use a rectangle packing algorithm with a coding scheme called sequence pair [2, 3, 5]. We conduct some computational experiments to evaluate our algorithm.

2 Formulation of 2DCSPn

We first define the two-dimensional cutting stock problem (2DCSP). We are given a sufficient number of stock sheets of the same length L and width W, and m types of products $M = \{1, 2, \ldots, m\}$, where each product i has length l_i, width w_i and demand d_i.

A cutting pattern p_j is described as $p_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$, where $a_{ij} \in \boldsymbol{Z}_+$ (the set of nonnegative integers) is the number of product i cut from pattern p_j. We call a pattern p_j feasible if its all products can be cut down from one stock sheet without overlap, and let S denotes the set of all feasible patterns. A solution of 2DCSP consists of a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_{|\Pi|}\} \subseteq S$, and their numbers of applications $X = (x_1, x_2, \ldots, x_{|\Pi|})$, where $x_j \in \boldsymbol{Z}_+$. A typical cost function is the total number of stock sheets used in the solution.

Now we consider a variant of 2DCSP. As noted before, it is impractical to use many different patterns in recent cutting industries. Hence, we consider 2DCSP with an input parameter n, where n is the number of different patterns $|\Pi|$. We call this problem the two-dimensional cutting stock problem with a given number of patterns n (2DCSPn), which is formally defined as follows:

$$
\begin{aligned}
\text{2DCSP}n\text{:} \quad \text{minimize} \quad & f(\Pi, X) = \sum_{p_j \in \Pi} x_j \\
\text{subject to} \quad & \sum_{p_j \in \Pi} a_{ij} x_j \geq d_i, \text{ for } i \in M, \\
& \Pi \subseteq S, \\
& |\Pi| = n, \\
& x_j \in \boldsymbol{Z}_+, \text{ for } p_j \in \Pi.
\end{aligned}
$$

2DCSPn asks to determine a set of cutting patterns Π, where a cutting pattern p_j is a combination of products cut from one stock sheet, and their numbers of applications X so that the total number of stock sheets used is minimized while satisfying the demands of all products. In the next section, we consider the problem of computing the numbers of applications

$X = (x_1, x_2, \ldots, x_n)$, where $x_j \in \boldsymbol{Z}_+$ for a given set of patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$. We propose a local search algorithm to find a good set of cutting patterns Π in Section 4.

3 Computing the number of applications for each pattern

In this section, we consider the problem of computing $X = (x_1, x_2, \ldots, x_n)$, where x_j denotes the number of applications for pattern p_j, for a given set of patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$. This problem is described as the following integer programming problem (IP):

$$\begin{aligned} \mathrm{IP}(\Pi): \quad & \text{minimize} && f(X) = \sum_{p_j \in \Pi} x_j \\ & \text{subject to} && \sum_{p_j \in \Pi} a_{ij} x_j \geq d_i, \text{ for } i \in M, \\ & && x_j \in \boldsymbol{Z}_+, \text{ for } p_j \in \Pi. \end{aligned}$$

This problem is already known to be strongly NP-hard since x_j must be integer, and hence we consider a heuristic algorithm for this problem.

Our heuristic algorithm first solves the LP relaxation LP(Π) of IP(Π), in which the integer constraints $x_j \in \boldsymbol{Z}_+$ are replaced with $x_j \geq 0$.

$$\begin{aligned} \mathrm{LP}(\Pi): \quad & \text{minimize} && f(X) = \sum_{p_j \in \Pi} x_j \\ & \text{subject to} && \sum_{p_j \in \Pi} a_{ij} x_j \geq d_i, \text{ for } i \in M, \\ & && x_j \geq 0, \text{ for } p_j \in \Pi. \end{aligned}$$

Let $\overline{X} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ denote an optimal solution of LP(Π). In order to obtain an integer solution, we sort all variables $\overline{x}_j$ in the descending order of $\overline{x}_j - \lfloor \overline{x}_j \rfloor$, and round up $\overline{x}_j$ to $\lceil \overline{x}_j \rceil$ in the resulting order of j until all demands are satisfied. We round down $\overline{x}_j$ to $\lfloor \overline{x}_j \rfloor$ for the remaining variables.

In our local search algorithm for finding a good set of patterns, which will be explained in the next section, we must solve many LP(Π). If LP is naively solved from scratch whenever we evaluate a new set of cutting patterns in the neighborhood, the computation time becomes very expensive. We therefore incorporate a sensitive analysis technique based on the criss-cross method to utilize the optimal LP solution for the current set of cutting patterns.

4 Local search algorithm to find a good set of patterns

As noted before, a solution of 2DCSPn consists of a set of n patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = (x_1, x_2, \ldots, x_n)$. Our local search (LS) generates a set of patterns in the neighborhood $N(\Pi)$ of the current set of patterns Π. The numbers of applications for the new set of patterns Π' are obtained by solving IP(Π') explained in the previous section.

The following ingredients must be specified in designing LS: Search space, move strategy, a function to evaluate solutions, an initial solution and neighborhood. In our local search algorithm, we define the search space as the set of all Π and we adopt first admissible move

strategy. A set of patterns Π is evaluated by the optimal value of LP(Π). Note that, we also compute an integer solution of IP(Π) heuristically, and update the best solution if its value is better than those obtained so far. We construct an initial solution heuristically based on the next fit algorithm for the (one dimensional) bin packing problem, since the problem to find a feasible solution is also NP-hard.

4.1 Neighborhood for local search

We use the neighborhood obtained by exchanging only one cutting pattern in the current set of patterns. However, the size of this neighborhood is still too large, and most of them may not lead to improvement. We propose a heuristic method to generate a smaller set of patterns to improve the efficiency of the local search algorithm.

The solutions in our basic neighborhood of the current solution (Π, X) are those generated by changing one pattern $p_j \in \Pi$ by the following basic operation: Remove t $(t = 0, 1, 2)$ products from the pattern p_j and add one product to the pattern. The size of this neighborhoods is $O(nm^{t+1})$ for each t. We use the information of overproduction $r_i = \sum_j a_{ij}x_j - d_i$ for each product i to decide which products to remove. We sort surplus products in the descending order of overproduction r_i, and remove products in this order in our neighborhood search. On the other hand, we use a dual optimal solution $\overline{Y} = (\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m)$ of LP(II) to determine the product to be added. Larger $\overline{y}_i$ tends to indicate that increasing a_{ij} in pattern p_j is more effective. We sort products such as $r_i = 0$ in the descending order of $\overline{y}_i$, and add a product in this order in our neighborhood search.

We introduce other operations to make the search more effective. For each pattern, surplus products are divided into two sets. One is the set of products which do not affect the current LP solution even if a product is removed from the pattern (i.e., the set of products i such that $a_{ij} \geq 1$ and $x_j \leq r_i$ for the pattern p_j). The other is the set of products which affect the LP solution if it is removed from the pattern (i.e., the set of products i such that $a_{ij} \geq 1$ and $x_j > r_i$ for the pattern p_j). It takes only $O(m)$ time to divide the products into these sets, and all products which do not affect the current LP solution are removed from the pattern. We call this operation as the redundancy reduction operation, and it is applied before the basic operation.

We explain another operation of adding products after changing a pattern by the basic operation. We divide all products into two sets according to whether overproduction r_i is 0 or positive. We first add products such that $r_i = 0$ to the pattern as much as possible. In this stage, we sort products in the descending order of $\overline{y}_i$, and add them in this order. Whenever a product is added to the pattern, we recompute an LP optimal solution and update $\overline{y}_i$. When it becomes impossible to add such products, then products with $r_i > 0$ are added to the pattern as much as possible. We sort these products in the ascending order of r_i, and add them in this order. Since surplus products do not affect the LP solution, we can not improve the current LP solution by this operation and it is not necessary to solve LP problem again. However, we may improve the current LP solution in the subsequent iteration with this operation. We call this operation as the filling-up operation, and it is applied after the basic operation. By these two operations, we can improve the quality of pattern p_j.

If all products in pattern p_j are removed by the redundancy reduction and basic operations,

we must reconstruct a new pattern from scratch by the basic and filling-up operations. This situation always occurs for pattern p_j with $x_j = 0$, and such reconstruction may not find a pattern with small trim loss. Therefore, we replace p_j with a new pattern in this case. For this purpose, we keep cn (c is a parameter and we use $c = 3$) good cutting patterns obtained by then in memory, and choose one from them, where we define good cutting pattern as those having small trim loss. We call this as the replacement operation.

Now, our new neighborhood is the set of solutions obtained from II by applying the operations proposed in this section (i.e., basic, redundancy reduction, filling-up and replacement). We call this neighborhood as the enhanced neighborhood. Two neighborhoods, basic and enhanced, will be computationally compared in Section 5.

4.2 Generating feasible cutting patterns

In order to generate feasible cutting patterns, we must solve the two-dimensional rectangle packing problem, which is also NP-hard. That is, to judge whether a combination of products is realized as a feasible cutting pattern or not, we need an algorithm to place a given set of products into the stock sheet without mutual overlap. For this purpose, we develop a heuristic algorithm using an existing rectangle packing algorithm [2, 3], which is based on local search with a coding scheme called sequence pair [5]. In their algorithm, they use a pair of permutations of all given rectangles to represent a solution, and compute positions of all rectangles with their decoding algorithm. We tailored their algorithm to our local search algorithm for 2DCSPn. We omit the details of this algorithm here, see [4] for more detailed explanation.

5 Computational experiments

We conducted computational experiments to evaluate the proposed algorithms. The algorithms were coded in the C language and run on a handmade PC (Intel Pentium IV 2.8GHz, 1GB memory).

5.1 Test instances

We generated random test instances of 2DCSP following the generation scheme described in [6, 8]. The instances are characterized by the following three parameters.

Number of product types: We have four classes 20, 30, 40 and 50 of the number of product types m (e.g., $m = 20$ in class 20).

Range of demands: Demand d_i of type S (S stands for small) is randomly taken from interval $[1, 25]$, type L (large) is taken from $[100, 200]$, and type V (variable) is taken from either intervals $[1, 25]$ or $[100, 200]$ with the equal probability for each product i.

Size of stock sheet: We have five classes $\alpha, \beta, \gamma, \delta$ and ε of the stock sheets. Class α is the smallest stock sheet which can contain six products on the average, while class ε is the largest containing about 50 products.

Hence, there are 60 types of instances and we generated one instance for each type. These instances are named like "20Sα", "20Sβ", ..., "20Sε", "20Lα", ..., "20Vε", "30Sα", ..., "50Vε". In our computational experiments, we apply our local search algorithms ten times to each instance with different initial solutions, and report the average results of ten trials. All test instances are electronically available from our web site (`http://www-or.amp.i.kyoto-u.ac.jp/~imahori/packing/`).

5.2 Comparison of basic and enhanced neighborhoods

First, basic and enhanced neighborhoods were computationally compared. For each instance, we applied our local search algorithm with each type of neighborhood ten times, and report the average quality of the obtained solutions and computational time, where local search halts only when a locally optimal solution is reached. For simplicity, we set the number of different cutting patterns to the number of product types (i.e., $n = m$). Results are shown in Table 1. Column

Table 1: Comparison two neighborhoods in solution quality and computational time

	basic		enhanced	
m	quality	time	quality	time
20	15.17	13.88	10.49	18.42
30	14.81	41.18	8.71	45.76
40	11.91	221.61	8.76	144.93
50	10.94	955.64	8.18	638.86

"m" shows the number of product types. For each m, we have 15 instances with different ranges of demands and different sizes of stock sheet; e.g., we have instances 20Sα, 20Sβ, ..., 20Vε for $m = 20$. Column "quality" shows the average of the following ratio,

$$\text{quality} = 100 \cdot (f - f_{LB})/f_{LB},$$

where f is the number of stock sheets used in the solution, and f_{LB} is a lower bound of the number of required stock sheets. The smaller quality means the better performance of the algorithm. Column "time" shows the average CPU time in seconds of one local search. These notations are also used in Table 2.

From Table 1, we can observe that the enhanced neighborhood gives smaller quality value than the basic neighborhood in all cases, while using similar computational time. It indicates that the redundancy reduction, filling-up and replacement operations proposed in Section 4.1 make the search more powerful and efficient. Based on this, we will use the enhanced neighborhood in the following experiments.

5.3 Effect of the number of patterns n

Next, we conducted computational experiments for different number of patterns n, i.e., n was set to $m, 0.8m, 0.6m$ and $0.4m$. Results are given in Table 2. The leftmost column shows the

Table 2: Quality and time with various number of different patterns on various classes

	$n = m$		$n = 0.8m$		$n = 0.6m$		$n = 0.4m$	
	quality	time	quality	time	quality	time	quality	time
class 20	10.491	18.424	12.247	6.308	14.432	3.753	20.810	1.954
class 30	8.707	45.758	9.899	18.533	12.175	6.424	16.758	3.098
class 40	8.758	144.932	10.360	45.330	12.218	15.630	16.891	7.091
class 50	8.177	636.861	9.940	143.135	11.504	37.292	15.315	12.117
class S	12.531	149.200	14.555	47.335	16.377	15.373	20.769	6.348
class L	6.185	262.195	7.390	57.178	9.100	16.865	12.403	5.369
class V	8.384	223.087	9.890	55.467	12.269	15.086	19.158	6.478
class α	10.516	46.944	12.436	10.001	16.323	1.843	28.203	0.264
class β	8.048	71.712	10.027	15.579	12.269	3.388	19.436	0.780
class γ	7.924	141.175	9.661	28.295	11.530	6.258	14.912	1.627
class δ	7.331	311.113	8.483	60.768	9.397	15.827	10.795	6.567
class ε	11.348	486.525	12.451	151.9901	13.391	51.559	13.871	21.087
average	9.033	211.494	10.612	53.327	12.582	15.775	17.443	6.065

instance classes. For example, "class 20" represents 15 instances with $m = 20$. Each figure in this row is the average of 150 trials (that is, 10 trials with different initial solutions for each instance, and there are 15 instances for class 20). "class S" represents 20 instances whose demand is taken from interval $[1, 25]$, and each figure is the average of 200 trials. Other rows can be similarly interpreted. Now from the rows for classes 20, 30, 40 and 50 in Table 2, we can observe that as m becomes larger (i.e., from class 20 to 50), computational time increases and solution quality becomes slightly better. As n becomes smaller, the size of neighborhood becomes smaller and local search algorithm converges to locally optimal solutions rather quickly, making the quality of obtained solution poorer.

From the rows for different ranges of demands (i.e., S, L and V), we can observe that the solution quality for class S is the worst even though all classes use similar computational time. This is due to the influence of rounding and overproduction. Namely, we compute the numbers of applications x_j by rounding from the LP solution, and it introduces a little overproduction for several product types. As the total demands is smaller for class S, the effect of one unit of overproduction to the quality is larger.

From the rows for different sizes of stock sheet (i.e., class $\alpha, \beta, \gamma, \delta$ and ε), we can observe that the solution qualities for classes α and ε are worse than others. The reason for class ε is similar to the previous one. For many test instances of class ε, we could find good solutions if the numbers of applications x_j can be fractional. However, these solutions degrade after obtaining integer solutions. On the other hand, as the size of stock sheet becomes smaller, it becomes harder to find a placement of products with small unused area, since there are not enough small products to fill up the stock sheet.

5.4 Trade-off curve between n and solution quality

Finally, we conducted more detailed experiment to obtain the trade-off curve between n and the quality of the obtained solutions. We used two instances 40Vα and 40Vδ. The area of the stock sheet of 40Vδ is four times as large as that of 40Vα. Results are shown in Figures 1 and 2. In these figures, horizontal axis is n, and vertical axis shows the solution quality and CPU time in seconds. 40Vα-LP (resp., 40Vδ-LP) shows the average quality of obtained LP solution (i.e., the numbers of applications can be fractional) for 40Vα (resp., 40Vδ), and 40Vα-IP (resp., 40Vδ-IP) shows the average quality of obtained IP solution (i.e., the numbers of applications must be integer) for 40Vα (resp., 40Vδ). 40Vα-time and 40Vδ-time show the average CPU times in seconds for ten trials. When n is very small (i.e., $n \leq 11$ for 40Vα and $n \leq 2$ for 40Vδ), we could not find initial feasible solutions. From Figures 1 and 2, we observe that the computational time tends to increase and the solution quality improves as n increases. For larger n, the improvement in quality becomes tiny instead of the computational time is increasing steadily. Note that, if the numbers of applications can be fractional, it is known that an optimal solution for 2DCSP uses at most m different patterns. Nevertheless, our obtained LP solutions for these instances with $n = 40$ are slightly worse than those with larger n. From these observations, there is still room for improvement in our neighborhood search. We also observe that the gap between LP and IP solutions for 40Vδ are more significant than the gap for 40Vα.

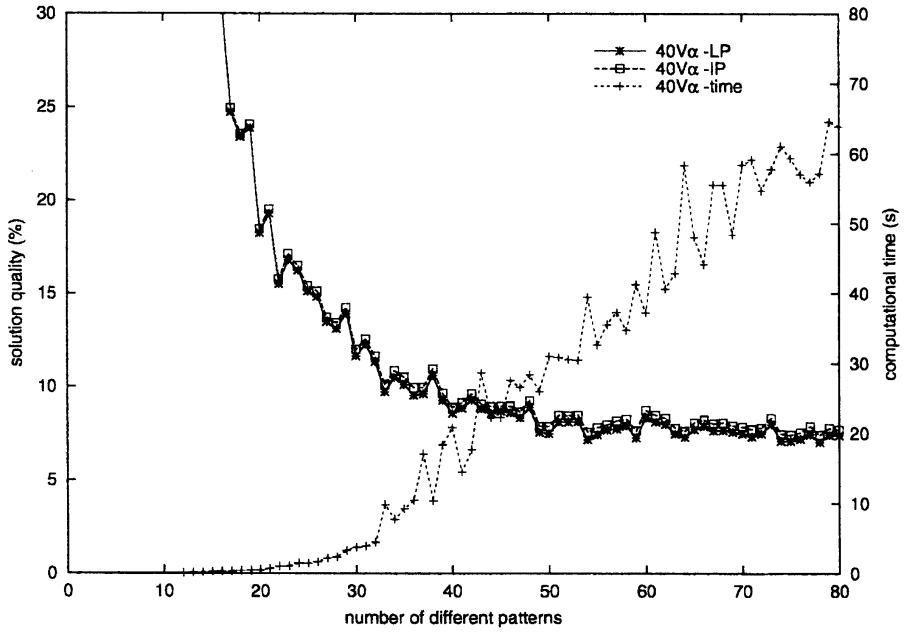

Figure 1: Trade-off between n and solution quality for 40Vα

6 Conclusion

In this paper, we considered the two-dimensional cutting stock problem with a given number of cutting patterns. We proposed a local search algorithm using linear programming techniques. In

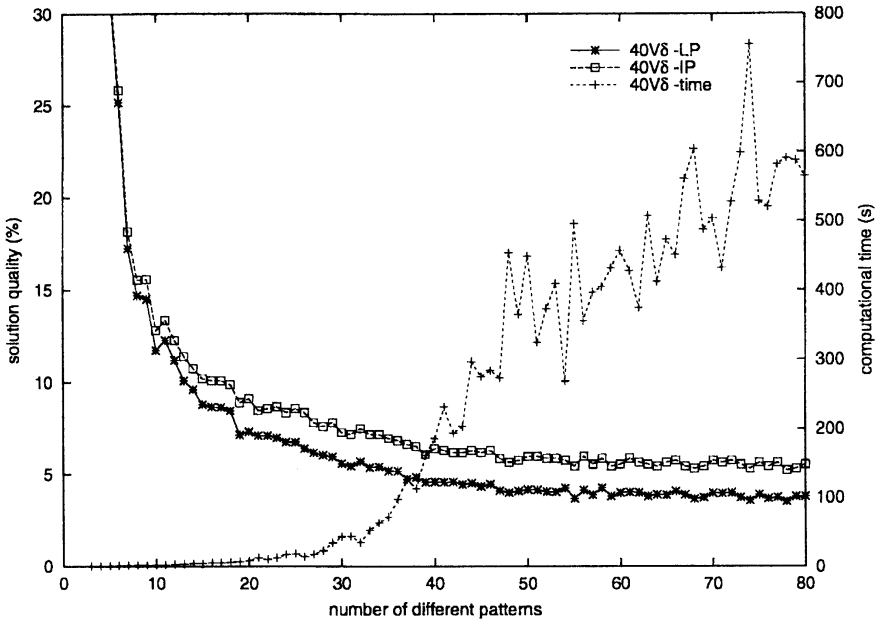

Figure 2: Trade-off between n and solution quality for $40V\delta$

our local search algorithm, we heuristically compute the number of applications for each cutting pattern from the solution of the associated LP relaxation. To realize an efficient search, we restrict the size of the neighborhood by considering only those solutions obtained by exchanging one pattern in the current set of patterns. First, we proposed the basic neighborhood defined by removing t products from one pattern and add one product to the pattern. Then, we improved this basic neighborhood using two ideas: (1) We remove all products which do not affect the current LP solution and we add products as much as possible to the pattern, (2) We keep good patterns during the search and use them when the current pattern is found to be useless. To check the feasibility of each pattern, we proposed a rectangle packing algorithm which is based on an effective algorithm proposed by Imahori et al. [2, 3].

We reported computational results of the local search algorithms using two different neighborhoods, and compared their performance. We confirmed the effectiveness of various ideas to generate a new pattern. We also pointed out that our approach can provide reasonable trade-off curves between the number of different cutting patterns and the quality. As a future work, we are planning to improve the solution quality by introducing more efficient neighborhood search and by incorporating advanced metaheuristic algorithms.

References

[1] P.C. Gilmore and R.E. Gomory, Multistage cutting stock problems of two and more dimensions, Operations Research, 13, pp. 94–119, 1965.

[2] S. Imahori, M. Yagiura and T. Ibaraki, Local search algorithms for the rectangle packing problem with general spatial costs, Mathematical Programming Series B (to appear).

[3] S. Imahori, M. Yagiura and T. Ibaraki, Improved local search algorithms for the rectangle packing problem with general spatial costs, submitted for publication (available at http://www-or.amp.i.kyoto-u.ac.jp/~imahori/packing).

[4] S. Imahori, M. Yagiura, S. Umetani, S. Adachi and T. Ibaraki, Local search algorithms for the two-dimensional cutting stock problem with a given number of different patterns, submitted for publication (available at http://www-or.amp.i.kyoto-u.ac.jp/~imahori/packing).

[5] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, VLSI module placement based on rectangle-packing by the sequence-pair, IEEE Transactions on Computer Aided Design, 15, pp.1518–1524, 1996.

[6] J. Riehme, G. Scheithauer and J. Terno, The solution of two-stage guillotine cutting stock problems having extremely varying order demands, European Journal of Operational Research, 91, pp. 543–552, 1996.

[7] S. Umetani, M. Yagiura and T. Ibaraki, An LP-based local search to the one dimensional cutting stock problem using a given number of cutting patterns, IEICE Transactions on Fundamentals, E86-A, pp. 1093–1102, 2003.

[8] R.A. Valdes, A. Parajon, and J.M. Tamarit, A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems, OR Spectrum, 24, pp.179-192, 2002.