ELSEVIER

European Journal of Operational Research 146 (2003) 388–402

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

One-dimensional cutting stock problem to minimize the number of different patterns

Shunji Umetani [*], Mutsunori Yagiura, Toshihide Ibaraki

Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8301, Japan

Received 18 April 2001; accepted 2 November 2002

Abstract

As the cost associated with the change of cutting patterns become more important in recent industry, we consider 1D-CSP in which the number of different cutting patterns is constrained within a given bound. The proposed approach is based on metaheuristics, and incorporates an adaptive pattern generation technique. According to our computational experiments, it is observed that the proposed algorithm provides comparable solutions to other existing heuristic approaches for 1D-CSP.

Keywords: Cutting stock problem; Pattern minimization; Local search; Metaheuristics; Pattern generation

1. Introduction

One-dimensional cutting stock problem (1D-CSP) is one of the representative combinatorial optimization problems, which has many applications in, e.g., steel, paper and fiber industries. To define an instance of 1D-CSP, we are given a sufficient number of stock rolls which have the same length L, and m types of products with given lengths $(l_1, l_2, \dots, l_m)$ and their demands $(d_1, d_2, \dots, d_m)$. A cutting pattern is a combination of products cut from a stock roll. A solution to 1D-CSP consists of a set of cutting patterns and the corresponding frequencies, i.e., the number of times each pattern is applied.

One of the most important costs for 1D-CSP is the amount of residual pieces of processed stock rolls, called *trim loss*, which are usually treated as waste product. Hence, the problem of minimizing the total trim loss (or the number of processed stock rolls) has been intensively studied. This problem is often formulated as an integer programming (IP) problem, and its linear programming (LP) relaxation is exploited in many heuristic algorithms; e.g., first solve the LP and then modify the LP solution to an integer solution heuristically. In this approach, however, it is impractical to consider all feasible cutting patterns, which correspond to the columns in LP formulation, in

[*] Corresponding author. Tel.: +81-75-753-5514; fax: +81-75-753-4920.

E-mail addresses: umetani@amp.i.kyoto-u.ac.jp (S. Umetani), yagiura@amp.i.kyoto-u.ac.jp (M. Yagiura), ibaraki@amp.i.kyoto-u.ac.jp (T. Ibaraki).

particular when l_i are much smaller than L. Gilmore and Gomory [4,5] proposed an ingenious procedure to find cutting patterns necessary to improve the LP solution by solving the associated knapsack problem.

In recent years, however, the costs of other factors than trim loss have become more important. Among them is the cost associated with pattern changes. In the solutions of LP based approaches, the number of different patterns tends to become close to the number of products, which is often too large.

Three types of algorithms have been proposed to minimize or reduce the number of pattern changes in 1D-CSP. The first is a greedy type heuristic algorithm called the sequential heuristic procedure (SHP) proposed by Haessler [8,9]. SHP sequentially adds new cutting patterns to the current solution until all demands are satisfied. In each step, it first generates candidates of cutting patterns which satisfy some portion of remaining demands, and then heuristically selects a cutting pattern from the candidate list, whose trim loss is small and frequency is high. Vahrenkamp [17] proposed a variant of SHP, in which a new pattern is generated by a simple randomized algorithm. Gradisar et al. [7] applied SHP to a variant of 1D-CSP in which stock rolls may have different lengths. Sweeney and Hassler [16] proposed a hybrid algorithm based on SHP and the LP based approach. Basic ideas of these algorithms are summarized in [10].

The second is a heuristic method called the pattern combination, which is proposed by Johnston [12] and Goulimis [6]. The algorithm starts from a solution obtained by an LP based approach, and reduce the number of different patterns by combining two patterns into one pattern; i.e., it selects two patterns in the solution and replaces them with a new pattern such that the amount of each product covered by the new pattern is equivalent to that covered by the removed patterns. Recently Foerster and Wäscher [2] proposed an algorithm of this type, called KOMBI, which uses many types of combinations. For example, three patterns are replaced with two new patterns, four patterns are replaced with three new patterns, etc.

The third is an exact algorithm for the pattern minimization problem (PMP) proposed by Vanderbeck [18], where PMP minimizes the number of different cutting patterns while using given number of stock rolls or less. PMP is first formulated as a quadratic integer programming problem, which is then decomposed into multiple bounded integer knapsack problems which have strong LP relaxations. Then a branch-and-bound algorithm procedure is applied while utilizing the column generation technique. According to computational experiments, it was observed this algorithm could solve many small problems exactly, but failed to obtain optimal solutions for several instances of moderate sizes in two hours.

In this paper, we propose a new heuristic algorithm for reducing the number of different cutting patterns. As it is not easy to minimize the number of different cutting patterns directly, we fix it to n (a program parameter), and search a solution whose quadratic deviation of the cut products from the demands is sufficiently small. For this problem, we propose an iterated local search algorithm with adaptive pattern generation, and we call the algorithm ILS-APG. The minimization of n may then be attained by iteratively applying ILS-APG with different values of n, e.g., by using binary search. For simplicity, in this paper, we explain our algorithm for a fixed n. Note that we allow both surplus and shortage from the demands. To our knowledge, handling the demand constraint in this manner is new; however, there are some real applications in which surplus and shortage are equally penalized, e.g., a problem found in a chemical fiber company as mentioned in Section 7. In general, it becomes easier to find a solution with a small deviation from the demands as n becomes larger. In this sense, there is a trade-off between the number of different cutting patterns and the deviation from the demands.

The iterated local search (ILS) is based on local search (LS). LS starts from an initial solution and repeats replacing it with a better solution in its neighborhood until no better solution is found in the neighborhood, where the neighborhood is the set of solutions obtainable from the current solution by slight perturbations. The resulting solution, for which no better solution exists in its neighborhood, is called locally optimal. ILS repeats LS from the initial solutions obtained by randomly perturbing good locally optimal solutions found in the past search. In spite of its simplicity, ILS is known to be quite effective [11].

Our ILS-APG tries to find a set of n cutting patterns yielding small deviations from the demands. The neighborhood used is the set of solutions obtainable by deleting one cutting pattern in the current solution and adding a different one. To evaluate the objective value of a solution in the neighborhood, it is necessary to compute the frequencies of cutting patterns. For this purpose, we adopt a simple heuristic algorithm based on the nonlinear Gauss–Seidel method, which is explained in Section 3. Basic components of LS are explained in Section 4.

In defining the neighborhood, it is not realistic to consider all possible cutting patterns to add, since the size of neighborhood may become exponential in m. To overcome this, we restrict the candidate cutting patterns to those generated by an adaptive pattern generation heuristic, which is proposed in Section 5. Finally the whole framework of the ILS-APG is described in Section 6.

Computational experiments are conducted in Section 7 for random problem instances and real problem instances. The random problem instances are generated by CUTGEN1 [3] and the real problem instances are taken from an application in a chemical fiber company. ILS-APG is compared with SHP, KOMBI and a heuristic algorithm used in the chemical fiber company. According to the computational results, ILS-APG finds good solutions with smaller number of different cutting patterns for most of the tested instances, compared to other algorithms. Possible improvements of ILS-APG are discussed in Section 8.

2. Formulation of 1D-CSP

In this section, we define the one-dimensional cutting stock problem to minimize the quadratic deviation from demands using a fixed number n of different cutting patterns. We are given a sufficient number of stock rolls of length L, and m types of products $M = \{1, 2, \ldots, m\}$ which have given lengths $(l_1, l_2, \ldots, l_m)$ and demands $(d_1, d_2, \ldots, d_m)$. A cutting pattern 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 one stock roll. We say a cutting pattern p_j satisfying

$$\sum_{i \in M} a_{ij} l_i \leqslant L \tag{1}$$

feasible. It is often necessary in practice to consider additional constraints on cutting patterns. One of the common constraints is that the trim loss of a cutting pattern should be smaller than the smallest product, i.e.,

$$L - \sum_{i \in M} a_{ij} l_i < \min_{i \in M} l_i. \tag{2}$$

A cutting pattern satisfying (2) is called a *complete-cut* cutting pattern [15]. Another common constraint is that the number of products in a stock roll is restricted to a given range, because the number of knives attached to the cutting machine is usually restricted. Our algorithm can deal with all these constraints. Let S denote the set of all feasible cutting patterns that satisfy all such constraints as well as (1).

A solution of 1D-CSP consists of a set of cutting patterns $\Pi \subseteq S$, and the frequencies of cutting patterns $\boldsymbol{x} = (x_1, x_2, \ldots, x_{|\Pi|}) \in \boldsymbol{Z}_+^{|\Pi|}$, where x_j represents the frequency of patterns $p_j \in \Pi$. Now 1D-CSP to minimize the number of different patterns is formulated as follows:

MIN-PAT

$$\begin{aligned} \text{minimize} \quad & |\Pi| \\ \text{subject to} \quad & \sum_{i \in M} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2 \leqslant D, \\ & \Pi \subseteq S, \\ & x_j \in \boldsymbol{Z}_+ \ \text{for } p_j \in \Pi, \end{aligned} \tag{3}$$

where D is a given bound for the deviation from the demands.

If $D = 0$, MIN-PAT is same as the pattern minimization problem (PMP) presented by Vanderbeck [18]. As PMP reduces to the bin packing problem which is known to be strongly NP-complete, MIN-PAT is a hard problem. McDiarmid [13] considered the special case of PMP where any two products fit on a stock roll ($l_i + l_j \leqslant L \forall i, j$) but no three do ($l_i + l_j + l_k > L \forall i, j, k$). For this special case of PMP, he showed that PMP is strongly NP-hard even though the minimum number of processed stock rolls is trivial (as exactly $\lceil \sum_{i \in M} d_i/2 \rceil$ stock rolls are required).

As already mentioned in Section 1, we consider in this paper the 1D-CSP in which the number of different patterns is constrained to be constant n and the deviation from the demands is minimized:

MIN-DEV

$$\begin{aligned} \text{minimize} \quad & f(\Pi, \boldsymbol{x}) = \sum_{i \in M} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2 \\ \text{subject to} \quad & |\Pi| = n, \\ & \Pi \subseteq S, \\ & x_j \in \boldsymbol{Z}_+ \quad \text{for } p_j \in \Pi. \end{aligned} \tag{4}$$

The above two formulations ignore the trim loss; but, if necessary, we can control the quantity of trim loss to some extent by adding appropriate constraints on cutting patterns; e.g., restricting S to be the set of complete-cut patterns. Our algorithm ILS-APG indirectly reduces the trim loss by restricting candidate cutting patterns to those having small trim losses, in the adaptive pattern generation procedure (Section 5).

3. Calculating the frequencies of cutting patterns

To evaluate the objective values $f(\Pi, \boldsymbol{x})$ of (4), it is necessary to compute the frequencies $\boldsymbol{x}$ for a given set of cutting patterns Π. In this section, we explain our algorithm for this. If Π is specified, the problem of finding an optimal $\boldsymbol{x}$ becomes the following integer quadratic programming problem:

$(Q(\Pi))$

$$\begin{aligned} \text{minimize} \quad & f(\Pi, \boldsymbol{x}) = \sum_{i \in M} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2 \\ \text{subject to} \quad & x_j \in \boldsymbol{Z}_+ \text{ for } p_j \in \Pi. \end{aligned} \tag{5}$$

Since it is hard to solve $Q(\Pi)$ exactly, we first solve the relaxation $\bar{Q}(\Pi)$ of $Q(\Pi)$, in which the integer constraints $x_j \in \boldsymbol{Z}_+$ are replaced with $x_j \geqslant 0$. After calculating an optimal solution $\bar{\boldsymbol{x}}$ of $\bar{Q}(\Pi)$, we round $\bar{\boldsymbol{x}}$ to its nearest integer solution $\hat{\boldsymbol{x}}$,

$$\begin{aligned} & \hat{x}_j \leftarrow \lceil \bar{x}_j \rceil \text{ if } \bar{x}_j - \lfloor \bar{x}_j \rfloor \geqslant 0.5 \\ & \text{and} \\ & \hat{x}_j \leftarrow \lfloor \bar{x}_j \rfloor \text{ otherwise.} \end{aligned} \tag{6}$$

For convenience, we abbreviate $f(\Pi, \hat{\boldsymbol{x}})$ to $\hat{f}(\Pi)$ in the following discussion. The solution $\hat{\boldsymbol{x}}$ obtained from Π is denoted by $\hat{\boldsymbol{x}}(\Pi)$ if we want to specify Π.

There may be various algorithms to solve $\bar{Q}(\Pi)$. We use the nonlinear Gauss–Seidel method [1] NGS(Π), because it is easy to implement and is usually efficient as the computational experiment in Section 7 explains.

In each iteration of NGS(Π), one variable x_l in the current solution $\boldsymbol{x}^{(k)}$ is updated as follows. Let $\tilde{x}_l$ satisfy equation $\partial f / \partial x_l |_{x_j = x_j^{(k)}, j \neq l} = 0$. Such $\tilde{x}_l$ can be calculated by

$$\tilde{x}_l \leftarrow x_l^{(k)} - \frac{\left. \frac{\partial f}{\partial x_l} \right|_{\boldsymbol{x} = \boldsymbol{x}^{(k)}}}{2 \sum_{i \in M} a_{il}^2}, \tag{7}$$

where

$$\begin{aligned} & \left. \frac{\partial f}{\partial x_l} \right|_{\boldsymbol{x} = \boldsymbol{x}^{(k)}} = -2 \sum_{i=1}^{m} \lambda_i a_{il}, \\ & \lambda_i = d_i - \sum_{p_j \in \Pi} a_{ij} x_j^{(k)}. \end{aligned} \tag{8}$$

Then the x_l is updated by $x_l \leftarrow \max\{\tilde{x}_l, 0\}$; other variables in $x^{(k)}$ are unchanged.

Iterations are conducted so that all variables x_l are scanned in a prespecified order and every variable x_l with $p_l \in \Pi$ is checked in n iterations. The entire algorithm is described as follows, where ε is a sufficiently small positive constant.

Nonlinear Gauss–Seidel method: NGS(Π)

Step 1. Set $\lambda_i \leftarrow d_i$ for all $i \in M$, and

$$x_j^{(0)} \leftarrow 0, \ \left. \frac{\partial f}{\partial x_j} \right|_{\boldsymbol{x} = \boldsymbol{x}^{(0)}} \leftarrow -2 \sum_{i \in M} \lambda_i a_{ij}$$

for all $p_j \in \Pi$. Let $k \leftarrow 0$.

Step 2. If either of the conditions

$$(1) \ \left| \left. \frac{\partial f}{\partial x_j} \right|_{\boldsymbol{x} = \boldsymbol{x}^{(k)}} \right| < \varepsilon,$$

or

$$(2) \ \left. \frac{\partial f}{\partial x_j} \right|_{\boldsymbol{x} = \boldsymbol{x}^{(k)}} > 0$$

and $x_j = 0$, is satisfied for every $p_j \in \Pi$, then output $\boldsymbol{x}$ and halt.

Step 3. Choose a variable x_l that satisfies neither (1) nor (2) in Step 2, and set

$$\Delta x_l \leftarrow -\frac{1}{2\sum_{i\in M} a_{il}^2} \left.\frac{\partial f}{\partial x_l}\right|_{\boldsymbol{x}=\boldsymbol{x}^{(k)}}$$

and $x_l^{(k+1)} \leftarrow \max\{0, x_l^{(k)} + \Delta x_l\}$. Let $x_j^{(k+1)} \leftarrow x_j^{(k)}$ for $j \neq l$.

Step 4. Update $\lambda_i \leftarrow \lambda_i - a_{il}\Delta x_l$ for all $i \in M$ and

$$\left.\frac{\partial f}{\partial x_j}\right|_{\boldsymbol{x}=\boldsymbol{x}^{(k+1)}} \leftarrow -2\sum_{i\in M} \lambda_i a_{ij}$$

for all $p_j \in \Pi$. Let $k \leftarrow k+1$ and return to Step 2.

4. Local search

The local search (LS) procedure is used to find a set of cutting patterns Π with a small cost $\hat{f}(\Pi) = f(\Pi, \hat{\boldsymbol{x}}(\Pi))$ for the $\hat{\boldsymbol{x}}(\Pi)$ computed as in Section 3. A natural definition for the neighborhood used in LS would be

$$N^{\text{all}}(\Pi) = \bigcup_{p_j\in\Pi} N_j^{\text{all}}(\Pi), \tag{9}$$

where

$$N_j^{\text{all}}(\Pi) = \{\Pi \cup \{p_{j'}\} \setminus \{p_j\} \mid p_{j'} \in S\}, \quad p_j \in \Pi, \tag{10}$$

i.e., $N^{\text{all}}(\Pi)$ is the set of all solutions obtainable by removing one cutting pattern from Π and adding another cutting pattern in S. However, in general, the size $|S|$ is roughly estimated as $\mathrm{O}(m^q)$, where q represents the average number of products in a cutting pattern, and can be very large. Hence, it is necessary to introduce much smaller neighborhood to make LS practical. For this purpose, we define a small subset $S'(j) \subset S$ for each $p_j \in \Pi$, and use

$$N_j^{\text{red}}(\Pi) = \{\Pi \cup \{p_{j'}\} \setminus \{p_j\} \mid p_{j'} \in S'(j)\}, \quad p_j \in \Pi, \tag{11}$$

instead of (10). Then the neighborhood of Π is defined by

$$N^{\text{red}}(\Pi) = \bigcup_{p_j\in\Pi} N_j^{\text{red}}(\Pi). \tag{12}$$

Here $S'(j)$ is the set of feasible cutting patterns generated by the adaptive pattern generation algorithm to be described in Section 5.

5. A heuristic algorithm to generate cutting patterns

In this section, we explain how to generate a subset $S'(j) \subset S$ of cutting patterns, which have prospect of improving the current solution Π. It is based on the residual demands $\boldsymbol{r} = (r_1, r_2, \ldots, r_m)$, whose definition will be given later. Let $p = (a_1, a_2, \ldots, a_m)$ be a cutting pattern to be generated, and let x be the frequency of p. Then problem $P_1(\boldsymbol{r})$ of finding a new pattern p and its frequency x is defined as follows:

$$(P_1(\boldsymbol{r}))$$
$$\begin{aligned} \text{minimize} \quad & \sum_{i\in M} (a_i x - r_i)^2 \\ \text{subject to} \quad & \sum_{i\in M} a_i l_i \leqslant L, \\ & x \in \boldsymbol{Z}_+, \\ & a_i \in \boldsymbol{Z}_+, \quad i \in M. \end{aligned} \tag{13}$$

Problem $P_1(\boldsymbol{r})$ is a nonlinear integer programming problem, and is difficult to solve exactly. Thus, we use an approximate algorithm which is based on the relaxation $\bar{P}_1(\boldsymbol{r})$ obtained by replacing the integer constraints $x \in \boldsymbol{Z}_+$ and $\boldsymbol{a} \in \boldsymbol{Z}_+^m$ by $x \geqslant 0$ and $\boldsymbol{a} \geqslant 0$, respectively. The relaxation $\bar{P}_1(\boldsymbol{r})$ may have many optimal solutions, among which we use the following solution that has no trim loss:

$$\begin{aligned} \bar{a}_i(\boldsymbol{r}) &= \left(\frac{L}{\sum_{i\in M} r_i l_i}\right) r_i, \quad i \in M, \\ \bar{x}(\boldsymbol{r}) &= \frac{\sum_{i\in M} r_i l_i}{L}. \end{aligned} \tag{14}$$

Then, we consider how to round $\bar{a}_i(\boldsymbol{r})$ to integer values a_i. This problem is described as follows:

$$(P_2(\boldsymbol{r}))$$
$$\begin{aligned} \text{minimize} \quad & \sum_{i\in M} (a_i - \bar{a}_i(\boldsymbol{r}))^2 \\ \text{subject to} \quad & \sum_{i\in M} a_i l_i \leqslant L, \\ & a_i \in \{\lceil \bar{a}_i(\boldsymbol{r}) \rceil, \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor\}, \quad i \in M. \end{aligned} \tag{15}$$

Since each decision variable a_i can take only two values $\lceil \bar{a}_i(\boldsymbol{r}) \rceil$ and $\lfloor \bar{a}_i(\boldsymbol{r}) \rfloor$, the problem $P_2(\boldsymbol{r})$ is equivalent to the following 0–1 knapsack problem:

$$(P_2'(\boldsymbol{r}))$$

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in M} \Big(1 - 2(\bar{a}_i(\boldsymbol{r}) - \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor)\Big) y_i \\ \text{subject to} \quad & \sum_{i \in M} y_i l_i \leqslant L - \sum_{i \in M} \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor l_i, \\ & y_i \in \{0,1\}, \quad i \in M. \end{aligned} \tag{16}$$

We can get an optimal solution for problem $P_2(\boldsymbol{r})$ from an optimal solution $\boldsymbol{y}^* = (y_1^*, y_2^*, \ldots, y_m^*)$ for problem $P_2'(\boldsymbol{r})$ by setting $a_i \leftarrow \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor$ if $y_i^* = 0$ and $a_i \leftarrow \lceil \bar{a}_i(\boldsymbol{r}) \rceil$ otherwise. Taking into account that an optimal solution for problem $P_2'(\boldsymbol{r})$ may not always give a useful cutting pattern (since $P_2'(\boldsymbol{r})$ is an approximation to $P_1(\boldsymbol{r})$), we try to generate many good solutions for problem $P_2'(\boldsymbol{r})$ heuristically by using the following adaptive pattern generation algorithm APG($\boldsymbol{r}$).

Algorithm APG($\boldsymbol{r}$) is based on Sahni's heuristic algorithm [14] for the 0–1 knapsack problem. It first applies a simple greedy method and outputs a solution $(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_m)$. The greedy method starts from $\boldsymbol{y} = 0$, and repeats the following steps: in each step, choose an i with the smallest $(1 - 2(\bar{a}_i(\boldsymbol{r}) - \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor))/l_i$ among those satisfying

$$y_i = 0 \quad \text{and} \quad \sum_{k \in M} y_k l_k + l_i \leqslant L - \sum_{k \in M} \lfloor \bar{a}_k(\boldsymbol{r}) \rfloor l_k, \tag{17}$$

and then set $y_i \leftarrow 1$. The algorithm stops if no i satisfies condition (17) and outputs the resulting $\boldsymbol{y}$ as $\tilde{\boldsymbol{y}}$. After this, the greedy method is repeatedly applied to m problem instances, each of which is obtained by fixing one variable y_q to $1 - \tilde{y}_q$, $q \in M$. Thus we generate $m + 1$ candidate cutting patterns. The whole procedure is described as follows.

Algorithm APG($\boldsymbol{r}$)

```
Step 1. Let $(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_m) \leftarrow$ GREEDY$(\boldsymbol{r}, 0, 0)$ and
    output the pattern obtained from $(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_m)$.
Step 2. For each $q \in M$, let $(y_1, y_2, \ldots, y_m) \leftarrow$
    GREEDY$(\boldsymbol{r}, q, y_q)$ for $y_q \leftarrow 1 - \tilde{y}_q$, and output
    the pattern obtained from $(y_1, y_2, \ldots, y_m)$.
```

Here GREEDY$(\boldsymbol{r}, q, v)$ is a subroutine that applies the greedy method to instance (16) in which the variable y_q is fixed to v. For convenience, we used GREEDY$(\boldsymbol{r}, 0, 0)$ to denote the greedy method applied to the original instance (16).

Algorithm GREEDY $(\boldsymbol{r}, q, y_q)$

```
Step 1. Set $y_i \leftarrow 0$ for all $i \in M \setminus \{q\}$.
Step 2. Sort all products $i \in M \setminus \{q\}$ in the ascending order of
    $(1 - 2(\bar{a}_i(\boldsymbol{r}) - \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor))/l_i$, and let $\sigma(k)$ denote the $k$th product in this order.
Step 3. For $k = 1, 2, \ldots, m - 1$, let $y_{\sigma(k)} \leftarrow 1$ if
    $\sum_{i \in M, i \neq k} y_{\sigma(i)} l_{\sigma(i)} + l_{\sigma(k)} \leqslant L - \sum_{i \in M} \lfloor \bar{a}_i(\boldsymbol{r}) \rfloor l_i$, and
    $y_{\sigma(k)} \leftarrow 0$ otherwise.
Step 4. Output $(y_1, y_2, \ldots, y_m)$ and halt.
```

Finally let $\Pi' \subseteq S$ and $\boldsymbol{x}$ denote the frequencies of patterns in Π'. We define the residual demand r_i of each product i when a cutting pattern p_j is removed from Π' as follows:

$$r_i(\Pi', \boldsymbol{x}) = \max\left(0, d_i - \sum_{p_j \in \Pi'} a_{ij} x_j\right), \quad i \in M, \tag{18}$$

where if $\Pi' = \emptyset$ we define that $r_i(\Pi', \boldsymbol{x}) = d_i$. Then the residual demand of the current pattern set Π with frequencies $\hat{\boldsymbol{x}}(\Pi)$ is defined by $r_i(\Pi \setminus \{p_j\}, \hat{\boldsymbol{x}}(\Pi))$. For convenience, we abbreviate $r_i(\Pi \setminus \{p_j\}, \hat{\boldsymbol{x}}(\Pi))$ to $\hat{r}_i(j)$, when the current solution Π is obvious. Let $\hat{\boldsymbol{r}} = (\hat{r}_1(j), \hat{r}_2(j), \ldots, \hat{r}_m(j))$.

To see how many of the cutting patterns generated by APG($\hat{\boldsymbol{r}}(j)$) are useful, we conducted computational experiment. We took instances from real applications provided by a chemical fiber company. The details of these instances are explained in Section 7, but their sizes are small enough to enumerate all the feasible cutting patterns. In this experiment, we chose a solution Π and a pattern $p_j \in \Pi$. Fig. 1 represents the differences in the objective values $\Delta\hat{f}(\Pi') = \hat{f}(\Pi') - \hat{f}(\Pi)$ (vertical axis) between the current solution Π and all $\Pi' \in N_j^{\text{all}}(\Pi)$ against the deviation from the continuous pattern $\sum_{i \in M} (a_i - \bar{a}_i(\hat{\boldsymbol{r}}(j)))^2$ (horizontal axis). If $\Delta\hat{f}(\Pi') < 0$, the solution Π' is better than the current solution Π. From Fig. 1, we can observe that the number of cutting patterns Π' with

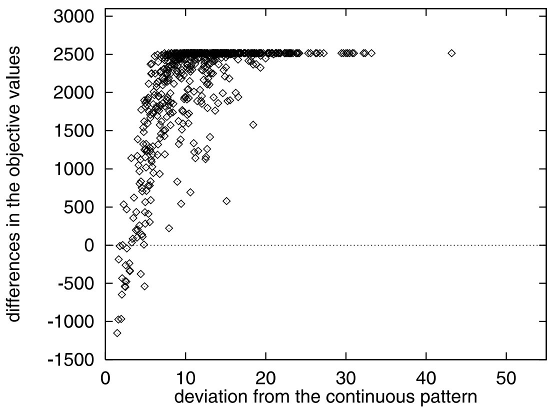

Fig. 1. All patterns in $N_j^{\text{all}}(\Pi)$ for the current solution Π.

$\Delta\hat{f}(\Pi') < 0$ is quite small among all the feasible cutting patterns, and that there is a strong correlation between the horizontal and vertical axes in Fig. 1, i.e., cutting patterns with small deviations from $\bar{\boldsymbol{a}}(\hat{\boldsymbol{r}}(j))$ are likely to improve the current solution.

Next, in Fig. 2, we show the objective values of solutions in $N_j^{\text{red}}(\Pi)$, where the horizontal and vertical axes are the same as in Fig. 1, and there are $m+1$ solutions in $N_j^{\text{red}}(\Pi)$, which is generated by APG($\boldsymbol{r}$) of this section. Fig. 2 tells that all cutting patterns in $N_j^{\text{red}}(\Pi)$ are close to $\bar{\boldsymbol{a}}(\hat{\boldsymbol{r}}(j))$, and in this case, all cutting patterns Π' in $N_j^{\text{red}}(\Pi)$ satisfy $\Delta\hat{f}(\Pi') < 0$. Thus APG($\boldsymbol{r}$) appears to be very effective to generate good solutions Π' from the current Π.

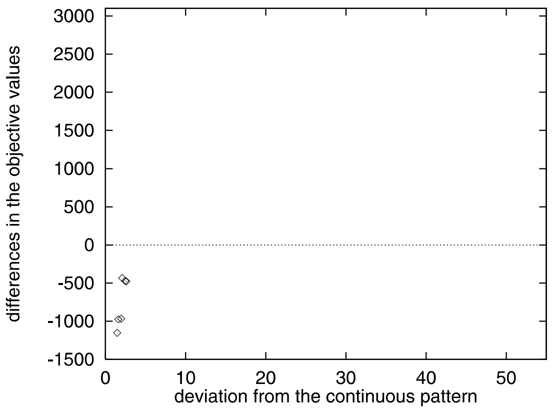

Fig. 2. Patterns in $N_j^{\text{red}}(\Pi)$ for the current solution Π.

6. Searching cutting patterns by ILS-APG

In this section, we explain our iterated local search algorithm ILS-APG; first a framework of ILS-APG, and then its subroutines INIT and LS(Π). ILS-APG first applies local search LS(Π) to the initial solution Π constructed by INIT, and then repeats local search LS(Π), each time generating an initial solution Π^{init} by randomly perturbing the best solution obtained by then. Here, *trial* denotes the current number of iterations of local search, and MAXTRIALS (program parameter) gives the upper bound of *trial*. Π^* denotes the best solution obtained by then.

Algorithm ILS-APG

```
Step 1. Set $trial \leftarrow 1$. Set $\Pi^{\text{init}}$ by executing INIT, and then set $\Pi^* \leftarrow \Pi^{\text{init}}$ and $\Pi \leftarrow \Pi^{\text{init}}$ ($\Pi$ denotes the current solution).
Step 2. Let $\Pi' \leftarrow \text{LS}(\Pi)$, and let $\Pi^* \leftarrow \Pi'$ if $\hat{f}(\Pi') \leqslant \hat{f}(\Pi^*)$.
Step 3. If $trial \geqslant$ MAXTRIALS, then output $\Pi^*$ and halt. Otherwise choose a $\Pi \in N^{\text{red}}(\Pi^*)$ randomly, let $trial \leftarrow trial + 1$, and return to Step 2.
```

We now explain how to construct an initial solution Π^{init} by INIT. We use the adaptive pattern generation method APG($\boldsymbol{r}$) in Section 5 also for this purpose. The algorithm starts from $\Pi = \emptyset$ and repeat adding the cutting pattern p with the minimum $\hat{f}(\Pi \cup \{p\})$ among the candidate patterns generated by APG($\boldsymbol{r}$). The residual demand $\boldsymbol{r}$ here is given by $\boldsymbol{r} = \boldsymbol{r}(\Pi, \hat{\boldsymbol{x}}(\Pi))$ of (18), where $\hat{\boldsymbol{x}}(\Pi)$ is obtained by (5) and (6) (for convenience we set $r_i = d_i$ for all i if $\Pi = \emptyset$). Algorithm INIT is formally described as follows. Recall that n is the number of different cutting patterns.

Algorithm INIT

```
Step 1. Set $\Pi^{\text{init}} \leftarrow \emptyset$ and $k \leftarrow 0$.
Step 2. Let $S'$ be the set of patterns generated by APG($\boldsymbol{r}(\Pi^{\text{init}}, \hat{\boldsymbol{x}}(\Pi^{\text{init}}))$).
Step 3. Choose a pattern $p \in S'$ with the minimum $\hat{f}(\Pi^{\text{init}} \cup \{p\})$, and set $\Pi^{\text{init}} \leftarrow \Pi^{\text{init}} \cup \{p\}$ and $k \leftarrow k + 1$.
Step 4. If $k = n$, then output solution $\Pi^{\text{init}}$ and halt; otherwise return to Step 2.
```

We now explain the details of local search LS(Π). As explained in Section 4, LS(Π) is based on the neighborhood $N^{\text{red}}(\Pi)$ constructed by calling APG($\hat{\boldsymbol{r}}(j)$) of Section 5. Furthermore, to reduce the number of candidate patterns $|S'(j)|$, we select only γ candidate patterns by preferring smaller $\sum_{i\in M}(a_i - \bar{a}_i(\hat{\boldsymbol{r}}(j)))^2$, where γ is a program parameter.

LS(Π) uses the first admissible move strategy, implemented by using a queue that maintains the cutting patterns in Π. For the cutting pattern p_j on top of the queue, we find the pattern $p_{j'}$ that minimizes $\hat{f}(\Pi \cup \{p_{j'}\} \setminus \{p_j\})$ among those in $S'(j)$. If $\Pi' = \Pi \cup \{p_{j'}\} \setminus \{p_j\}$ is better than Π (i.e., $\hat{f}(\Pi') < \hat{f}(\Pi)$), then we move to Π' immediately and put $p_{j'}$ to the tail of the queue; otherwise we remove p_j from the queue and add it to its tail.

Algorithm LS(Π) is formally described as follows. Recall that γ is a parameter which sets the number of candidate patterns. In the algorithm, Q denotes the queue that maintains the cutting patterns in Π. Procedure ENQUEUE(Q, p_k) adds a pattern p_k at the end of Q, and TOP(Q) returns the top pattern of Q, and DEQUEUE(Q) deletes TOP(Q) from Q.

Algorithm LS(Π)

```
Step 1. Set $Q$ be an empty queue, and then ENQUEUE($Q, p_j$) for all $p_j \in \Pi$ in an arbitrary order. Set $k \leftarrow 0$.
Step 2. Set $p_j \leftarrow$ TOP($Q$), and then DEQUEUE($Q$).
Step 3. Let $S'(j)$ be the set of patterns which $\gamma$ smallest deviations $\sum_{i\in M}(a_i - \bar{a}_i(\hat{\boldsymbol{r}}(j)))^2$.
Step 4. Choose $p_{j'} \in S'(j)$ with the minimum $\hat{f}(\Pi \cup \{p_{j'}\} \setminus \{p_j\})$, and set $\Pi' \leftarrow \Pi \cup \{p_{j'}\} \setminus \{p_j\}$.
Step 5. If $\hat{f}(\Pi') < \hat{f}(\Pi)$, then set $\Pi \leftarrow \Pi'$ and $k \leftarrow 0$, ENQUEUE($Q, p_{j'}$) and return to Step 2; otherwise ENQUEUE($Q, p_j$).
Step 6. Set $k \leftarrow k + 1$. If $k < |\Pi|$ then return to Step 2; otherwise output $\Pi$ and halt.
```

7. Computational experiments

We conducted computational experiment for random instances generated by CUTGEN1 [3] and for some instances taken from real applications in a chemical fiber company. We compared ILS-APG with the following three algorithms: SHP [8,9], KOMBI [2] and a heuristic algorithm called the generation and test method (GT), which is used in the chemical fiber company in Japan. GT sequentially generates a set of candidate patterns by repeatedly adding a new cutting pattern to the current set, and is similar to SHP. We coded ILS-APG and SHP in C language and executed on an IBM-compatible personal computer (PentiumII 450 MHz, 128 MB memory). The results of GT were provided by the chemical fiber company, where GT was run on an IBM-compatible personal computer (Pentium 133 MHz, 32 MB memory). The results of KOMBI were taken from [2], as we could not get the source code of KOMBI. KOMBI was run on an IBM-compatible 486/66 personal computer using MODULA-2 as the programming language under MS-DOS 6.0. SHP has a program parameter MAXTL to control the quantity of trim loss of the output solution, which is set to 0.03 in our experiment. The program parameter MAXTRIAL of ILS-APG is set to 100, and γ is set to $\lceil (m+1)/10 \rceil$.

Before presenting computational results, it is necessary to emphasize that the problem solved by SHP and KOMBI is different from that solved by ILS-APG and GT, because demand constraints are treated differently. ILS-APG and GT allow the shortage and/or overproduction of the products, where ILS-APG minimizes their total squared deviation and GT reduces the deviation heuristically. SHP and KOMBI do not allow shortage, and SHP allows neither shortage nor overproduction. Therefore precise comparison of algorithm performance is not possible. But we may be able to capture their general tendency from the computational results.

To evaluate the quality of squared deviation $f(\Pi, \boldsymbol{x})$, it may be convenient to introduce a simple criterion of goodness. Let us consider f be *acceptable* if $f \leqslant b_{\text{acpt}}$ holds, where

$$b_{\text{acpt}} = \sum_{i\in M} \max\{(0.01 d_i)^2, 1\}, \tag{19}$$

i.e., 1% of the demand or a single deviation for each product i. In the following results, the number of acceptable instances n_{acpt} is always given for ILS-APG.

We first conducted computational experiment for the random instances generated by CUTGEN1 [3], and compared ILS-APG with SHP and KOMBI [2]. We generated 18 classes of random instances, which are defined by combining different values of the parameters L, m, v_1, v_2, $\bar{d}$. The lengths l_i were treated as random variables taken from interval $[v_1 L, v_2 L]$. $\bar{d}$ is the average of demands $(d_1, d_2, \ldots, d_m)$. In our experiments, L was set to 1000, m was set to 10, 20 and 40, (v_1, v_2) was set to (0.01, 0.2), (0.01, 0.8) and (0.2, 0.8), and $\bar{d}$ was set to 10 and 100. The parameter *seed* was set to 1994. For each class, 10 problem instances were generated and solved. These classes of problem instances were also solved by KOMBI in [2], where 100 instances are tested for each class.

Table 1 shows the results of SHP, KOMBI and ILS-APG, where $\bar{b}_{\mathrm{acpt}}$ is the average of b_{acpt} defined in (19) and *tloss* is the ratio (percentage) of the total trim loss to the length of stock rolls:

$$tloss = \frac{100 \sum_{p_j \in \Pi} \left(L - \sum_{i \in M} a_{ij} l_i \right)}{L \sum_{p_j \in \Pi} x_j}. \qquad (20)$$

Note that $|\Pi|$ and *tloss* (trim loss) are averages of 10 instances for SHP and ILS-APG, where they are averages of 100 instances for KOMBI (the data are taken from [2]). KOMBI obtained solutions with smaller average sizes $|\Pi|$ than those of SHP for 14 classes out of 18. This suggests that KOMBI performs better than SHP for random instances. Note that the number of cutting patterns $n = |\Pi|$ is a parameter which can be chosen by the user of ILS-APG, while it is the output of SHP, KOMBI and GT. As the primal purpose of this experiment was to test the performance of ILS-APG with small n, we ran ILS-APG for five cases $n = \alpha$, $\alpha - 1, \alpha - 2$ and $n = \beta, \beta - 1$, where parameters α and β are defined as follows:

$$\alpha = |\Pi_{\mathrm{SHP}}|,$$
$$\beta = \min\left\{ |\Pi_{\mathrm{SHP}}|, |\Pi_{\mathrm{SHP}}| - \left\lceil \overline{|\Pi_{\mathrm{SHP}}|} - \overline{|\Pi_{\mathrm{KOMBI}}|} \right\rceil \right\}. \qquad (21)$$

Here $|\Pi_{\mathrm{SHP}}|$ denotes the size of $|\Pi|$ obtained by SHP, $\overline{|\Pi_{\mathrm{SHP}}|}$ denotes the average size of $|\Pi_{\mathrm{SHP}}|$ for the corresponding type of instances and $\overline{|\Pi_{\mathrm{KOMBI}}|}$ denotes the average size of $|\Pi|$ obtained by KOMBI. β is intended to represent the smaller of $|\Pi_{\mathrm{SHP}}|$ and $|\Pi_{\mathrm{KOMBI}}|$ approximately (recall that $|\Pi_{\mathrm{KOMBI}}|$ for individual instance is not given in [2]).

From Table 1, we observe that ILS-APG obtains acceptable solutions in many cases, while using smaller numbers of different patterns than SHP and KOMBI. This may indicate that the primal goal of ILS-APG is achieved. Table 1 also shows that *tloss* of ILS-APG is smaller than that of SHP in almost all instances. Although the trim loss of KOMBI is considered to be very close to the optimal value, it is larger than that of ILS-APG for some instances. (KOMBI is based on the solutions of Stadtler's algorithm [15] which is able to minimize the trim loss for almost all instances.) As noted in the beginning of this section, this is partially because ILS-APG has weaker constraint than KOMBI, as it allows shortage of demands.

Table 2 shows the CPU time of SHP, KOMBI and ILS-APG (with $|\Pi| = \alpha$), respectively, for random instances in Table 1. For these classes of problem instances, SHP and KOMBI are faster than ILS-APG except for some instances (recall that KOMBI was run on a slower personal computer). In summary, for randomized instances, we may conclude that, ILS-APG tends to produce solutions of better quality in the sense of smaller $|\Pi|$ and smaller trim loss, at the cost of consuming more computation time.

We next conducted computational experiments for real-world problem instances provided by a chemical fiber company. The data of these problems are available at our WWW site.[1] There are 40 instances with m ranging from 6 to 29, $L = 9080, 5180$, d_i ranging from 2 to 264, and l_i ranging from 500 to 2000.

Table 3 (resp., Table 4) shows the results of SHP, GT and ILS-APG for the problem instances with $L = 9080$ (resp., 5180). The results of ILS-APG are shown for three cases $n = \alpha, \alpha - 1, \alpha - 2$, where

$$\alpha = \min\{|\Pi_{\mathrm{SHP}}|, |\Pi_{\mathrm{GT}}|\}. \qquad (22)$$

[1] http://www-or.amp.i.kyoto-u.ac.jp/members/umetani/problems/benchmark.html.

Table 1
Computational results of SHP, KOMBI234 and ILS-APG for the random instances generated by CUTGEN1

Class	v_1	v_2	m	$\bar{d}$	$\bar{b}_{\text{acpt}}$	SHP		KOMBI		ILS-APG									
						$\lvert\Pi\rvert$	*tloss*	$\lvert\Pi\rvert$	*tloss*	$\lvert\Pi\rvert = \alpha$		$\lvert\Pi\rvert = \alpha - 1$		$\lvert\Pi\rvert = \alpha - 2$		$\lvert\Pi\rvert = \beta$		$\lvert\Pi\rvert = \beta - 1$	
										n_{acpt}	*tloss*	n_{acpt}	*tloss*	n_{acpt}	*tloss*	n_{acpt}	*tloss*	n_{acpt}	*tloss*
1	0.01	0.2	10	10	9.70	3.9	5.09	3.40	4.47	10	2.79	9	2.81	4	2.74	9	2.81	4	2.74
2	0.01	0.2	10	100	16.93	5.5	1.83	7.81	0.47	9	2.67	7	2.90	2	2.73	9	2.67	7	2.90
3	0.01	0.2	20	10	19.20	6.1	3.42	5.89	2.52	10	1.24	9	0.96	9	0.96	9	0.96	9	0.96
4	0.01	0.2	20	100	34.02	8.4	1.20	14.26	0.25	7	1.37	5	1.17	2	1.14	7	1.37	5	1.17
5	0.01	0.2	40	10	36.80	9.3	3.04	10.75	1.10	10	0.71	10	0.61	10	0.67	10	0.71	10	0.61
6	0.01	0.2	40	100	69.88	13.1	1.57	25.44	0.12	6	0.72	6	0.70	4	0.69	6	0.72	6	0.70
Average	–	–	–	–	31.09	7.72	2.69	11.26	1.49	8.7	1.58	7.7	1.53	5.2	1.49	8.3	1.70	6.8	1.51
7	0.01	0.8	10	10	10.00	10.3	16.78	7.90	15.41	10	16.97	10	16.35	10	16.86	9	15.01	7	14.06
8	0.01	0.8	10	100	16.97	11.9	16.58	9.96	15.00	10	16.01	10	15.70	9	16.09	9	16.09	8	17.11
9	0.01	0.8	20	10	19.90	18.9	15.12	15.03	11.00	9	12.15	10	13.24	9	12.81	9	11.45	9	12.06
10	0.01	0.8	20	100	33.54	21.7	15.61	19.28	10.72	10	11.99	10	11.78	9	11.46	10	11.86	9	11.72
11	0.01	0.8	40	10	39.5	37.6	11.93	28.74	7.33	8	5.27	8	5.49	8	5.10	8	4.97	8	4.88
12	0.01	0.8	40	100	64.51	41.2	11.15	37.31	7.29	8	6.37	10	6.89	8	6.21	7	6.37	7	6.00
Average	–	–	–	–	30.74	23.60	14.53	19.70	11.13	9.2	11.46	9.7	11.58	8.8	11.42	8.7	10.96	8.0	10.97
13	0.2	0.8	10	10	10.00	10.8	18.66	8.97	19.17	10	19.27	10	17.86	9	18.23	9	18.23	8	17.08
14	0.2	0.8	10	100	16.97	11.2	18.48	10.32	18.55	10	19.11	9	18.16	6	18.05	9	18.16	6	18.05
15	0.2	0.8	20	10	19.90	19.5	17.48	16.88	14.76	10	17.09	10	17.23	10	17.24	10	16.69	10	16.74
16	0.2	0.8	20	100	33.54	21.3	18.00	19.91	14.67	10	17.08	10	17.37	9	17.88	9	17.88	9	16.93
17	0.2	0.8	40	10	39.6	37.7	14.33	31.46	10.30	10	10.18	9	10.42	10	10.07	10	10.10	10	10.00
18	0.2	0.8	40	100	64.39	40.7	14.45	38.28	10.22	10	10.61	10	10.67	10	10.60	10	10.32	10	10.76
Average	–	–	–	–	30.73	23.50	16.90	20.97	14.61	10.0	15.56	9.7	15.29	9.0	15.35	9.5	15.23	8.8	14.93

Table 2
The CPU time in seconds for the random instances generated by CUTGEN1

Class	v_1	v_2	m	$\bar{d}$	SHP	KOMBI	ILS-APG
1	0.01	0.2	10	10	0.09	0.14	0.11
2	0.01	0.2	10	100	0.11	1.14	0.57
3	0.01	0.2	20	10	2.28	1.74	0.62
4	0.01	0.2	20	100	2.71	16.00	2.89
5	0.01	0.2	40	10	180.10	38.03	3.25
6	0.01	0.2	40	100	256.58	379.17	20.85
Average	–	–	–	–	73.65	72.70	4.72
7	0.01	0.8	10	10	0.01	0.07	0.36
8	0.01	0.8	10	100	0.02	0.20	1.38
9	0.01	0.8	20	10	0.04	1.34	3.37
10	0.01	0.8	20	100	0.06	3.25	15.43
11	0.01	0.8	40	10	0.22	36.27	68.23
12	0.01	0.8	40	100	0.32	76.31	412.81
Average	–	–	–	–	0.11	19.57	83.60
13	0.2	0.8	10	10	0.01	0.08	0.20
14	0.2	0.8	10	100	0.02	0.13	0.35
15	0.2	0.8	20	10	0.03	1.81	1.15
16	0.2	0.8	20	100	0.04	2.60	2.72
17	0.2	0.8	40	10	0.16	50.93	7.16
18	0.2	0.8	40	100	0.24	70.94	43.81
Average	–	–	–	–	0.08	21.08	9.23

Table 3 (resp., Table 4) tells that 20/20 (resp., 19/20) are acceptable for $n = \alpha$, 16/20 (resp., 17/20) are acceptable for $n = \alpha - 1$, and 10/20 (resp., 15/20) are acceptable for $n = \alpha - 2$. Note that GT also gives acceptable solutions in 19/20 instances (it failed to obtain a solution in one instance). SHP is designed so that a solution with $f = 0$ is output (i.e., always acceptable). But SHP and GT achieve this performance at the cost of using larger $n = |\Pi|$ in many cases, as observed in Tables 3 and 4. If we compare SHP and GT from the view point of the obtained size $|\Pi|$, SHP outperforms GT in Table 3, but the relation is reversed in Table 4. This suggests that SHP performs well for problem instances in which the ratio of product lengths l_i to the length of stock rolls L is relatively small, but not so if the ratio is relatively large. If we evaluate the quality of solutions from *tloss* (trim loss), SHP has the smallest *tloss*, than ILS-APG with $n = \alpha$, and GT. The performance of SHP is remarkable in this respect, but ILS-APG also performs reasonably well (considering that the minimization of the trim loss is not a primal target of ILS-APG). It is worth mentioning that ILS-APG achieve almost the same *tloss* even if smaller $n = \alpha - 1$ and $\alpha - 2$ are used; its performance is robust in the sense of *tloss*.

Table 5 (resp., Table 6) shows the CPU time of SHP, GT and ILS-APG, respectively, for the problem instances in Table 3 (resp., Table 4), where we show the CPU time of ILS-APG with $|\Pi| = \alpha$. The CPU time of SHP becomes extremely large for some instances, because SHP generates many candidate patterns for such instances. Similar tendency is observed for GT. The CPU time of ILS-APG is comparable to other two algorithms, and appears to be more stable.

8. Reduction of CPU time

To understand the rapid growth of CPU time of ILS-APG with m in Table 2, we conducted additional experiment. That is, for the problem instances generated by CUTGEN1 with $\bar{d} = 100$,

Table 3
The quadratic deviation f and the trim loss *tloss* by three algorithms (for instances with $L = 9080$)

m	b_{acpt}	SHP			GT			ILS-APG					
		$\|\Pi\|$	f	*tloss*	$\|\Pi\|$	f	*tloss*	$n = \alpha$		$n = \alpha - 1$		$n = \alpha - 2$	
								f	*tloss*	f	*tloss*	f	*tloss*
6	6.00	5	0	2.95	3	1	3.61	1	3.67	13	3.27	236	10.62
7	7.00	4	0	5.62	3	9	1.58	1	0.99	1	0.99	15	1.00
8	13.97	4	0	1.02	6	1	0.43	4	4.71	7	4.73	34	4.73
9	9.96	5	0	2.78	6	10	0.76	2	6.75	2	6.86	21	6.39
10	10.82	5	0	1.73	6	4	1.17	4	2.18	23	2.15	97	2.19
11	11.69	5	0	2.33	7	4	1.75	8	2.24	23	3.00	10	6.31
13	13.00	6	0	2.52	7	2	1.87	2	7.52	4	6.97	4	6.17
13	13.00	4	0	2.99	6	0	2.99	2	4.14	3	3.95	30	4.14
14	14.00	5	0	5.33	5	13	3.01	5	0.63	17	1.03	105	0.83
15	16.07	5	0	1.29	7	19	2.12	7	1.81	11	0.95	40	1.78
16	16.84	6	0	2.87	8	3	7.02	2	2.88	7	2.69	19	2.44
17	17.12	12	0	1.02	8	5	6.98	0	3.08	6	3.31	3	2.94
18	18.00	6	0	2.29	10	5	10.77	8	2.73	12	2.65	43	2.14
19	21.59	8	0	2.33	10	9	4.04	3	2.52	3	2.76	17	2.69
20	20.00	9	0	4.69	–	–	–	2	3.35	3	3.44	4	1.72
23	25.92	8	0	2.58	11	0	6.02	18	3.14	17	3.91	15	3.64
26	38.75	9	0	1.91	13	0	14.39	23	2.78	15	2.82	16	1.73
28	28.00	8	0	3.42	14	18	6.31	8	3.52	7	3.78	25	2.77
28	29.46	12	0	2.36	14	5	7.79	1	2.15	2	2.04	1	2.15
29	29.00	13	0	2.95	10	9	11.39	2	2.38	3	2.02	4	1.75
Average		6.95	0.00	2.75	8.11[a]	6.16[a]	4.95[a]	5.15	3.16	8.95	3.17	36.7	3.41

[a] Excluding the instance with $m = 20$.

$(v_1, v_2) = (0.01, 0.8)$ and $m = 10, 15, 20, 30, 40$, we applied the local search LS(Π) from the initial solutions Π generated by INIT. Local search was applied to 10 problem instances for each m, in which two cases of $\gamma = m + 1$ and $\gamma = 5$ were tested, where $n = |\Pi|$ is set to $\alpha - 2$ (recall that $\alpha = |\Pi_{\text{SHP}}|$). Table 7 gives the results of this experiment, where

- $\bar{n}$ the average size of Π ($|\Pi| = \alpha - 2$, where $\alpha = |\Pi_{\text{SHP}}|$).
- $\#\hat{f}$: the average number of evaluations of $\hat{f}(\Pi)$ in one local search (i.e., the number of calls to NGS(Π), the nonlinear Gauss–Seidel method).
- #loops: the average number of iterations of the loop (Steps 2–4) in one execution of NGS(Π).
- #moves: the average number of moves in one local search.
- CPU time: the average CPU time of one local search in seconds.

From Table 7, we see that $\bar{n}$, $\#\hat{f}$, #loops and #moves are approximately proportional to $m^{0.99}$, $m^{2.24}$, $m^{0.50}$ and $m^{1.56}$, respectively. As the time to execute one loop of NGS(Π) can be estimated as O(mn), this tells that the average CPU time of one local search is roughly given by

$$\#\hat{f} \cdot \#\text{loops} \cdot \mathrm{O}(m\bar{n}) = \mathrm{O}(m^{4.74}), \tag{23}$$

which may be justified by the column of CPU time (i.e., proportional to $m^{4.54}$).

These observations suggest that, in order to prevent the rapid growth of the CPU time with m, it is important

(i) to reduce the size of neighborhood without sacrificing the power of local search,
(ii) to improve the Gauss–Seidel method (or to use other methods) so that #loops and the time for one loop can be reduced.

Although point (ii) still remains to be a topic of future research, we tried point (i) by controlling the parameter γ in Step 3 of LS(Π) (recall that γ restricts the number of candidate patterns). For

Table 4
The quadratic deviation f and the trim loss of *tloss* by three algorithms (for instances with $L = 5180$)

m	b_{acpt}	SHP			GT			ILS-APG					
		$\lvert\Pi\rvert$	f	*tloss*	$\lvert\Pi\rvert$	f	*tloss*	$n = \alpha$		$n = \alpha - 1$		$n = \alpha - 2$	
								f	*tloss*	f	*tloss*	f	*tloss*
6	6.00	7	0	4.93	4	2	5.01	1	5.22	3	4.65	103	4.35
7	7.00	10	0	7.55	6	4	3.69	0	7.55	3	7.16	3	7.17
8	13.97	10	0	3.16	4	1	3.43	16	7.72	138	3.57	179	3.82
9	9.96	9	0	6.75	6	3	8.37	1	8.78	3	8.66	6	8.53
10	10.82	14	0	4.03	8	1	3.71	3	3.91	1	7.71	3	7.79
11	11.69	12	0	2.89	8	1	4.10	1	4.61	2	5.57	4	5.12
13	13.00	14	0	2.36	6	10	2.07	9	2.05	25	3.83	106	2.36
13	13.00	9	0	2.83	7	1	3.29	1	3.20	1	3.20	1	3.20
14	14.00	11	0	3.20	6	1	3.60	1	5.56	6	3.72	37	5.54
15	16.07	16	0	2.87	7	2	4.37	3	4.83	2	5.04	2	4.37
16	16.84	10	0	2.71	8	1	2.42	3	1.92	19	4.15	38	3.05
17	17.12	9	0	2.92	9	9	3.34	2	3.27	2	3.36	2	3.27
18	18.00	11	0	1.88	11	14	2.51	5	4.07	1	5.61	17	4.06
19	21.59	25	0	4.75	12	5	5.05	3	5.34	1	5.59	2	5.02
20	20.00	8	0	3.81	8	2	2.63	1	3.52	4	2.33	9	5.74
23	25.92	15	0	1.39	11	9	4.72	2	5.09	5	5.15	14	4.86
26	38.75	29	0	2.26	16	1	3.23	0	4.77	4	4.62	1	4.40
28	28.00	11	0	1.25	13	10	1.73	5	3.81	12	3.89	14	3.36
28	29.46	15	0	1.37	12	5	2.58	7	4.51	5	4.74	16	4.72
29	29.00	13	0	1.22	13	4	8.20	2	5.33	2	5.48	2	5.23
Average		12.90	0.00	3.21	8.75	4.30	3.90	3.30	4.75	11.95	4.90	27.95	4.80

Table 5
The CPU time in seconds (for instances with $L = 9080$)

m	SHP	GT	ILS-APG
6	0.02	0.22	0.03
7	0.01	0.82	0.28
8	0.02	1.32	0.10
9	0.02	0.93	0.41
10	0.08	1.54	0.35
11	0.05	2.42	0.41
13	0.17	3.62	0.68
13	0.30	2.85	0.23
14	0.16	1.54	0.32
15	0.13	2.47	0.73
16	0.17	9.11	0.56
17	46.62	3.57	1.52
18	0.22	26.97	0.48
19	2.33	3.24	2.06
20	4.09	–	0.68
23	0.40	36.03	2.64
26	1.27	7.80	5.06
28	4.80	596.99	2.62
28	197.74	25.70	8.09
29	597.02	78.27	2.37
Average	42.78	42.39[a]	1.48

[a] Excluding the instance with $m = 20$.

Table 6
The CPU time in seconds (for instances with $L = 5180$)

m	SHP	GT	ILS-APG
6	0.01	0.11	0.09
7	0.09	0.27	0.23
8	0.07	0.22	0.20
9	0.10	1.65	0.43
10	0.20	0.44	0.59
11	0.30	0.99	1.00
13	0.20	2.41	0.45
13	0.06	0.88	0.32
14	0.07	1.92	0.31
15	1.54	1.04	0.46
16	0.04	6.86	0.98
17	0.06	3.85	1.46
18	0.05	2.85	1.93
19	2.37	5.55	3.27
20	0.03	10.87	0.46
23	0.17	12.80	2.70
26	2.26	15.92	14.52
28	0.27	20.87	3.88
28	0.49	9.17	5.08
29	0.23	6.15	4.42
Average	0.43	5.24	2.14

Table 7
Performance of local search for instances generated by CUTGEN1

γ	m	$\bar{n}$	$\#\hat{f}$	#Loops	#Moves	CPU time	n_{acpt}	*tloss*
$m+1$	10	9.9	179.5	32.57	2.0	0.0953	8	16.13
	15	14.7	352.3	38.96	2.4	0.468	7	11.24
	20	19.7	974.8	34.65	7.7	1.91	6	11.17
	30	28.7	2098.0	49.33	10.4	11.4	6	5.54
	40	39.2	4014.9	65.54	17.4	51.4	7	5.47
5	10	9.9	92.6	29.27	2.3	0.0455	8	15.82
	15	14.7	175.6	23.95	4.8	0.145	9	11.72
	20	19.7	265.8	31.12	7.3	0.459	6	11.12
	30	28.7	397.3	35.04	5.7	1.50	6	5.25
	40	39.2	553.8	61.41	11.4	6.65	6	5.19

example, in all the experiments of Section 7, we used $\gamma = \lceil (m+1)/10 \rceil$ instead of $\gamma = m+1$. This reduced the CPU time to about 1/10 of that of Table 7, almost without sacrificing the power of local search. Table 7 also contains the results with $\gamma = 5$ (constant). In this case, $\#\hat{f}$ decreases to $m^{1.29}$ from $m^{2.24}$, the average CPU time of one local search decreases to $m^{3.60}$ from $m^{4.54}$. This still does not seem to sacrifice the power of local search much.

9. Conclusion

The cost associated with the change of cutting patterns become more important in recent cutting stock industry. Based on this observation, we considered a formulation of 1D-CSP in which the number of different cutting patterns is constrained, and proposed a metaheuristic algorithm ILS-APG, which incorporated an adaptive pattern generation technique. The ILS-APG searches a solution with small deviations from the given demands while using the fixed number of different cutting patterns. We conducted computational experiments for random problem instances and real problem instances, and observed that performance of ILS-APG is comparable to two other existing algorithms SHP and KOMBI (which solve a slightly different formulation of 1D-CSP).

References

[1] D.P. Bertsekas, Nonlinear Programming, Athena Scientific, Belmont, MA, 1995.

[2] H. Foerster, G. Wäscher, Pattern Reduction in One-dimensional Cutting Stock Problems, The 15th Triennial Conference of the International Federation of Operational Research Societies, 1999.

[3] T. Gau, G. Wäscher, CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem, European Journal of Operational Research 84 (1995) 572–579.

[4] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting-stock problem, Operations Research 9 (6) (1961) 849–859.

[5] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting-stock problem – Part II, Operations Research 11 (6) (1963) 863–888.

[6] C. Goulimis, Optimal solutions for the cutting stock problem, European Journal of Operational Research 44 (1990) 197–208.

[7] M. Gradisar, M. Kljajić, G. Resinovic, J. Jesenko, A sequential heuristic procedure for one-dimensional cutting, European Journal of Operational Research 114 (1999) 557–568.

[8] R.W. Haessler, A heuristic programming solution to a nonlinear cutting stock problem, Management Science 17 (12) (1971) 793–802.

[9] R.W. Haessler, Controlling cutting pattern changes in one-dimensional trim problems, Operations Research 23 (3) (1975) 483–493.

[10] R.W. Haessler, Cutting stock problems and solutions procedures, European Journal of Operational Research 54 (1991) 141–150.

[11] D.S. Johnson, Local optimization and the traveling salesman problem, in: Proceedings of 17th Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, 443, 1990, pp. 446–461.

[12] R.E. Johnston, Rounding algorithm for cutting stock problems, Journal of Asian-Pacific Operations Research Societies 3 (1986) 166–171.

[13] C. McDiarmid, Pattern minimisation in cutting stock problems, Discrete Applied Mathematics 98 (1999) 121–130.

[14] S. Sahni, Approximates for the 0/1 knapsack problem, Journal of the Association for Computing Machinery 22 (1) (1975) 115–124.

[15] H. Stadtler, A one-dimensional cutting stock problem in the aluminium industry and its solution, European Journal of Operational Research 44 (1990) 209–223.

[16] E. Sweeney, R.W. Haessler, One-dimensional cutting stock decisions for rolls with multiple quality grades, European Journal of Operational Research 44 (1990) 224–231.

[17] R. Vahrenkamp, Random search in the one-dimensional cutting stock problem, European Journal of Operational Research 95 (1996) 191–200.

[18] F. Vanderbeck, Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem, Operations Research 48 (2000) 915–926.