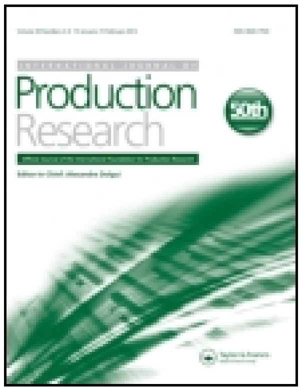

International Journal of Production Research
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

An improved best-first branch-and-bound algorithm for constrained two-dimensional guillotine cutting problems

K. Yoon [a], S. Ahn [a] & M. Kang [a]
[a] Department of Industrial and Management Engineering, Hanyang University ERICA Campus, Ansan, South Korea

Published online: 22 Jun 2012.

To cite this article: K. Yoon , S. Ahn & M. Kang (2013) An improved best-first branch-and-bound algorithm for constrained two-dimensional guillotine cutting problems, International Journal of Production Research, 51:6, 1680-1693, DOI: 10.1080/00207543.2012.693965

To link to this article: http://dx.doi.org/10.1080/00207543.2012.693965

International Journal of Production Research
Vol. 51, No. 6, 15 March 2013, 1680–1693

An improved best-first branch-and-bound algorithm for constrained two-dimensional guillotine cutting problems

K. Yoon, S. Ahn* and M. Kang

Department of Industrial and Management Engineering, Hanyang University ERICA Campus, Ansan, South Korea

(*Received 21 July 2011; final version received 9 April 2012*)

The constrained two-dimensional cutting problem involves maximising the sum of the profits obtained from small rectangular pieces cut from a large rectangular plate where the number of each type of cut piece cannot exceed a prescribed quantity. This paper proposes a best-first branch-and-bound algorithm to find the optimal solution to the problem. The proposed algorithm uses an efficient method to remove the duplicate patterns, and it improves the existing upper bounds. It also prevents the construction of dominated patterns and introduces a new bounding strategy that can prune more than one node at a time. Computational results are compared with a well-known exact algorithm to demonstrate the efficiency of the proposed algorithm. The proposed algorithm is as fast as or faster than the existing algorithm and reduces the average computing time by up to 99% for benchmark problems. For some problems, it can also find optimal solutions that existing algorithms are not able to find.

Keywords: cutting problem; optimisation; branch and bound; constrained two-dimensional cutting; guillotine cutting pattern

1. Introduction

The two-dimensional cutting problem (TDC) involves determining a cutting pattern that minimises waste (or maximises total profit) of pieces cut from a large rectangular plate. Each small rectangular piece has its own size and profit. A cutting pattern is the combination of pieces to be cut. The constrained TDC problem (CTDC) limits the number of each type of piece that is produced; in contrast, if the number of each type of piece is unlimited, then the problem is the unconstrained TDC (UTDC). The TDC problem can also be classified by the profit of each piece. If each piece has its own profit, regardless of its area, this case is called a weighted problem. However, if the profit of each piece is equal to its area, then it is an unweighted problem. Another important criterion to classify the problem is the number of cuts (or stages). The TDC problem is a non-staged or general problem if there is no constraint on the number of stages; otherwise it is a staged (generally two-staged or three-staged) problem. They have been researched separately. The less number of stages, the better production efficiency is achieved because the number of machine's setup decreases and patterns become simpler. On the other hand, the material utilisation is higher in the non-staged problem. In this paper, we investigate both weighted and unweighted CTDCs, assuming that all pieces have a fixed orientation, all cuts are of the guillotine type, and there is no constraint on the number of stages.

The TDC problem can be used as an auxiliary problem when solving the cutting stock problem (CSP). The CSP describes the procedure of cutting the required number of small rectangular pieces of different sizes from a set of large rectangular plates at a minimum plate cost. To solve the CSP, the TDC problem must be solved many times as a sub-problem because a solution to the CSP is a set of solutions to the TDC problem. Thus, a fast algorithm for the TDC is needed to solve the CSP.

Many studies have considered algorithms for the TDC. Recently, many studies have focused on special classes of cutting patterns, such as the p-group, T-shaped, or p-staged, due to the impractical computational time and particular characteristics of certain cutting machines (Yanasse and Morabito 2008, Cui and Yang 2011, Wy and Kim 2011). However, far fewer studies have considered exact algorithms for non-staged CTDC because non-staged CTDC is more difficult to solve than any other type of TDC (i.e., it is more difficult than UTDC or staged TDC). Two algorithmic approaches exist for non-staged CTDC: top-down and bottom-up approaches. Christofides and Whitlock (1977) originally proposed the top-down approach, which generates all feasible cutting patterns by

*Corresponding author. Email: sunahn@hanyang.ac.kr

ISSN 0020–7543 print/ISSN 1366–588X online
© 2013 Taylor & Francis
http://dx.doi.org/10.1080/00207543.2012.693965
http://www.tandfonline.com

repeatedly cutting each sub-plate into new sub-plates. Hifi and Zissimopoulos (1997) proposed an improved algorithm that used more effective lower and upper bounds. They applied the depth-first searching strategy, which requires only a small amount of memory, although at the expense of more computational time and difficulties in the presence of additional constraints (Cung *et al.* 2000). In contrast, the bottom-up approach generates all feasible cutting patterns using horizontal or vertical layouts of two patterns that are themselves constructed from two other patterns. Wang (1983) first proposed this concept, and Viswanathan and Bagchi (1993) developed a best-first branch-and-bound algorithm based on this concept. Hifi (1997) improved this algorithm by applying more effective lower and upper bounds, and Cung *et al.* (2000) improved its efficiency by adding some branching strategies and storing 'pattern codes' in the searching nodes to remove duplicate patterns, which have the same size and the same combination of pieces. This algorithm requires more memory as the number of searched nodes increases because it uses a best-first search strategy. However, this is offset by the small amount of computational time required and the ease with which various additional constraints can be handled (Cung *et al.* 2000). G *et al.* (2003) and Kang and Yoon (2011) also used this approach to solve UTDC and dramatically reduced the amount of memory required by using efficient branching strategies and upper bounds. The algorithm proposed here also uses this approach.

The purpose of this study was to develop an efficient exact algorithm for CTDC to solve large-scale practical problems in a reasonable time. Existing exact algorithms are impractical for large problems due to their computational inefficiency. This paper presents an improved best-first branch-and-bound algorithm for CTDC. The proposed algorithm uses a more efficient method than that of Cung *et al.* (2000) to remove duplicate patterns. Additionally, more efficient upper bounds and a bounding strategy are proposed. We also propose a method of preventing the formation of unpromising patterns, which can be dominated by other patterns, based on the work of Kang and Yoon (2011).

Computational results indicate that the algorithm is very efficient compared with that of Cung *et al.* (2000). This work shows the possibility of using the proposed algorithm to solve large-scale practical problems that would otherwise be intractable in a reasonable time.

This paper is organised as follows. Section 2 explains the bottom-up approach used in the algorithm. Section 3 proposes an improved best-first branch-and-bound algorithm for CTDC. This section first describes effective branching strategies that remove unnecessary duplicate and dominated patterns and then presents two improved upper bounds and a bounding strategy that can prune several nodes at a time. Section 4 demonstrates the efficiency of the proposed algorithm using computational experiments and includes an analysis of the experimental results. Section 5 presents the conclusions.

2. Best-first branch-and-bound using bottom-up approach

This paper proposes a best-first branch-and-bound bottom-up approach. We first explain the key concepts underlying our approach: the bottom-up approach and the branch-and-bound algorithm.

In the bottom-up approach, a cutting pattern is built using a horizontal or vertical construction of two sub-patterns, as shown in Figure 1. Let A and B be two cutting patterns (l_A, w_A) and (l_B, w_B), where (l, w) denotes the length and width of the pattern, respectively. A new cutting pattern of $(l_A + l_B, \max\{w_A, w_B\})$ can be generated using

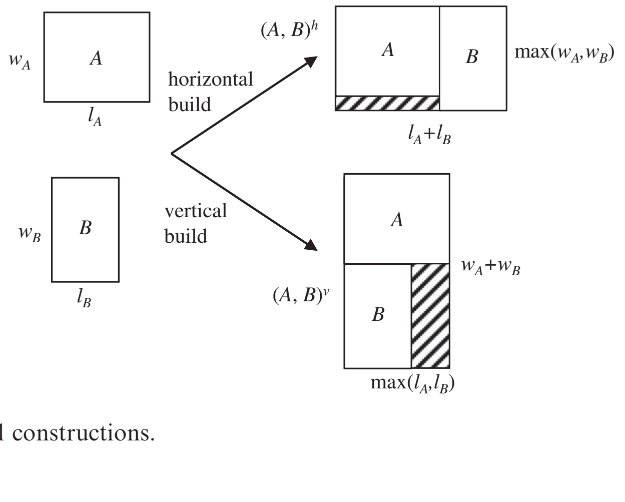

Figure 1. Horizontal and vertical constructions.

the horizontal construction of A and B. Similarly, the vertical construction of A and B generates a different cutting pattern of $(\max\{l_A, l_B\}, w_A + w_B)$ (see Figure 1).

Viswanathan and Bagchi (1993) proposed a best-first branch-and-bound algorithm using the bottom-up approach to generate all feasible cutting patterns. Each iteration of the best-first branch-and-bound algorithm consists of the following steps. First, select the leaf node with the highest upper bound and move it from the leaf node set (referred to as the open set **O**) into the branched node set (referred to as the closed set **C**). Then, branch the child nodes from the selected node and add the child nodes with upper bounds higher than the current best solution into the leaf node set. Each node represents a cutting pattern in Viswanathan and Bagchi's algorithm. The child nodes (cutting patterns) of a selected leaf node are generated by a horizontal or vertical construction between the selected node and all elements in the set of the branched nodes. The current solution is updated whenever a child node with a higher profit is constructed. If no leaf node with higher upper bounds than the current best solution exists, then the algorithm is complete. Using a good lower bound as the initial best solution reduces the total number of nodes generated.

Consider a CTDC problem with a plate (L, W) and n types of pieces such that the size, profit, and maximum number of the ith piece are (l_i, w_i), p_i, and b_i, respectively. Let (l_R, w_R) and $g(R)$ be the size and profit of the cutting pattern of node R. Because $g(R)$ is given, we need to calculate the upper bound of area P in Figure 2 to obtain the upper bound of node R, $U(R)$. Then, $U(R) = g(R) + u(P)$, where $u(P)$ is the upper bound of area P.

Because the UTDC problem is a relaxed CTDC problem, the optimal solution of the UTDC problem can be used as an upper bound for the CTDC problem. Gilmore and Gomory (1966) solved the following recursive function $F(x, y)$ using dynamic programming and produced the optimal solution to the UTDC problem with a plate (x, y).

$$F(x,y) = \max\{F_0(x,y), F(x_1,y) + F(x_2,y), F(x,y_1) + F(x,y_2)\}, \tag{1}$$

where

$$F_0(x,y) = \max\{0, p_i : l_i \le x, w_i \le y, \text{ for } i = 1,2,\ldots,n\} \tag{2}$$

and

$$x_1 + x_2 \le x, 1 \le x_1 \le x_2; y_1 + y_2 \le y, 1 \le y_1 \le y_2. \tag{3}$$

To obtain an upper bound of area P from Gilmore and Gomory's function, Viswanathan and Bagchi (1993) solved the following recursive function $u_1(x, y)$ for $x = L$ down to 1 and $y = W$ down to 1.

$$u_1(x,y) = \max\{h(x,y), v(x,y)\}, \tag{4}$$

where

$$h(x,y) = \max\{u_1(x+t,y), F(t,y) : 1 \le t \le L - x\}, \tag{5}$$

$$v(x,y) = \max\{u_1(x,y+t), F(x,t) : 1 \le t \le W - y\} \tag{6}$$

and

$$u_1(L,W) = 0 \tag{7}$$

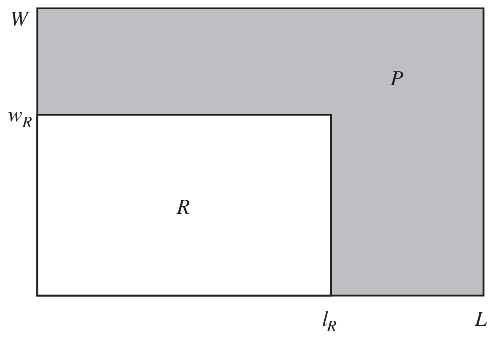

Figure 2. Size of cutting pattern R and the plate.

From the above results, $u_1(l_R, w_R)$ gives an upper bound for P. Hifi (1997) added two complementary upper bounds and finally used $u'(P)$ to estimate $u(P)$.

$$u'(P) = \min\{u_1(x_R, y_R), F(L, W) - g(R), V(area(P))\}, \tag{8}$$

where

$$V(area(P)) = \max\left\{\sum_{i=1}^{n} p_i x_i : \sum_{i=1}^{n} (l_i w_i) x_i \leq area(P), x_i \leq b_i \text{ for } i = 1, 2, \ldots, n\right\} \tag{9}$$

and *area*(P) denotes the area of the region P.

The function $V(area(P))$ represents a bounded knapsack problem that finds a combination of pieces to give the maximum profit sum with an area sum not greater than *area*(P). Hifi's original function is formulated only for pieces that can be placed in the area, not for all pieces, such as in the algorithm described above. To avoid calculating the function for each node, he proposed solving $V(LW)$ at the beginning of the algorithm using DP, which results in a formulation for all pieces (Martello and Toth 1990).

Cung *et al.* (2000) used the following estimate of $u''(P)$ by adding two complementary upper bounds. Let

$$S_P = \{i : (l_i \leq L - l_R \text{ or } w_i \leq W - w_R) \textit{ and } b_i(R) < b_i \text{ for } i = 1, \ldots, n\}, \tag{10}$$

where $b_i(R)$ is the number of ith pieces in R. Then

$$\begin{aligned} u''(P) &= \min\{u'(P), V(LW) - g(R), u_2(P)\} \\ &= \min\{u_1(x_R, y_R), F(L, W) - g(R), V(area(P)), V(LW) - g(R), u_2(P)\} \end{aligned} \tag{11}$$

where

$$u_2(P) = \sum_{i \in S_P} p_i(b_i - b_i(R)). \tag{12}$$

The best-first branch-and-bound algorithm used by Cung *et al.* (2000) is as follows:

```
Step 1 [initialise]: Let the set of leaf nodes O = {$R_1, R_2, \ldots, R_n$} where $R_i$ is the pattern with one $i$th piece.
    Let the set of branched nodes C = Ø.
    best ← lower bound found by a heuristic algorithm.
Step 2 [branch and bound]: Find pattern $R$ having the highest $U(R)$ in O.
    If not found or if $U(R) \leq best$, then go to Step 3.
    Move $R$ from O to C.
    Construct all feasible patterns $Q$ such that
    $Q$ is constructed horizontally or vertically with $R$ and the pattern in C,
    $Q \leq (L, W)$ and $b_i(Q) \leq b_i$ for $i = 1, 2, \ldots, n$.
    For each such $Q$,
    If $g(Q) > best$, then $best \leftarrow g(Q)$;
    If $U(Q) > best$, then $\mathbf{O} = \mathbf{O} \cup \{Q\}$.
    Remove all elements $R$ that contain $U(R) \leq best$ from O.
    Go to step 2.
Step 3 [finish]: Exit with best.
```

3. Proposed algorithm

3.1 *Dominated pattern*

Dominated patterns and duplicated patterns represent unpromising cutting patterns. Thus, many research approaches have included rules to detect or avoid the exploration of dominated patterns and reduce the search space.

To avoid the construction of a dominated pattern, Kang and Yoon (2011) proposed three methods for UTDC by setting a limit to the width or height of a pattern combining with R in **O**. All of these cannot be applied to CTDC because CTDC is bound by a constraint on the number of pieces. We can use the third method for CTDC with no modification. However, we should consider the constraint to use the second one. The following describes the method of determining the upper and lower width limits based on Kang and Yoon's (2011) method when constructing a new pattern by horizontal builds.

(1) *Lower width limit*

A horizontal build between $A \in \mathbf{O}$ and $B \in \mathbf{C}_{l=x}$ is dominated if $w_A \geq w_B$ and the newly generated unoccupied area on B is large enough to contain a piece. Unlike with UTDC, pieces should be checked to see whether they are available in CTDC. Horizontal builds between A and the nodes with widths less than that shown in Equation (13) are unnecessary.

$$\underline{w_1} = w_A - min\{w_i | l_i \leq x, w_i \leq w_A - w_B, b_i > b_i(R), i = 1, 2, \ldots, n\}. \tag{13}$$

Let (L, W), z, and $w(A)$ be the dimension of the stock plate, the current best solution value, and the waste area of pattern A, respectively. The unoccupied area of B is generated by a horizontal build between A and B if $w_A \geq w_B$. If the sum of the wasted area of A and the newly generated unoccupied area on B is greater than $LW - z$, then the maximum profit of a pattern that includes $(A, B)^h$ cannot be greater than z for an unweighted problem. Thus, Inequality (14) must be satisfied to construct the horizontal build between A and B. This is the same as the method of Kang and Yoon (2011).

$$w(A) + x \times (w_A - w_B) < LW - z. \tag{14}$$

For a weighted problem, the maximum profit per unit area,

$$\max\left\{\left(\frac{p_i}{l_i \times w_i}\right) | i = 1, 2, \ldots, n\right\},$$

must be multiplied by the whole term of the left-hand side, and LW by the right-hand side of Inequality (14). From Inequality (14), the horizontal builds between A and the nodes with widths less than $\underline{w}$ in Equation (15) are unnecessary.

$$\underline{w_2} = \lfloor w_A - ((LW - z) - w(A))/x \rfloor \tag{15}$$

Building on Equations (13) and (15), the maximum value of $\underline{w_1}$ and $\underline{w_2}$ in Equation (16) is the lower width limit for the nodes combining horizontally with A.

$$\underline{w} = \max\{\underline{w_1}, \underline{w_2}\} \tag{16}$$

(2) *Upper width limit*

A horizontal build between $A \in \mathbf{O}$ and $B \in \mathbf{C}_{l=x}$ is dominated if $w_B \geq w_A$ and the newly generated unoccupied area of A is large enough to contain a piece that is available. A constraint on the number of pieces should be added compared with the method of Kang and Yoon (2011).

$$\overline{w_1} = w_A + \min\{w_i | l_i \leq l_A,\ w_i \leq w_B - w_A,\ b_i > b_i(R),\ i = 1, 2, \ldots, n\}. \tag{17}$$

The unoccupied area of A is generated by a horizontal build between A and B if $w_B \geq w_A$. If the sum of wasted area of A and the newly generated unoccupied area on A is greater than $LW - z$, then the maximum profit of a pattern that includes $(A, B)^h$ cannot be greater than z for an unweighted problem. Therefore, Inequality (18) must be satisfied to construct a horizontal build between A and B. This is same as the method of Kang and Yoon (2011).

$$w(A) + l_A \times (w_B - w_A) < LW - z \tag{18}$$

Inequality (18) shows that the nodes with widths greater than $\overline{w_2}$ in Equation (19) cannot be combined with A.

$$\overline{w_2} = \lceil w_A + ((LW - z) - w(A))/l_A \rceil \tag{19}$$

From Equations (17) and (19), in the following Equation (20), the minimum value of $\overline{w_1}$ and $\overline{w_2}$ is the lower width limit for the nodes combining horizontally with A.

$$\overline{w} = \min\{\overline{w_1}, \overline{w_2}\} \tag{20}$$

From Equations (16) and (20), the nodes with widths greater than $\underline{w}$ and less than $\overline{w}$ can be combined horizontally with A. The lower and upper limits of length for the vertical builds can be obtained in a similar manner.

3.2 *Duplicate pattern*

Cutting patterns with the same size and combination of pieces are referred to as duplicate patterns. As is the case with dominated patterns, duplicate patterns should be removed to enhance the performance of the algorithm.

Horizontal builds $(A, (B, B)^h)^h$, $(B, (A, B)^h)^h$, and $((A, B)^h, B)^h$ result in the same cutting patterns. Cung *et al.* (2000) defined this type of duplicate as type D3. Assume a cutting pattern $A = (A_1, A_2, \ldots, A_r)^h$, where A_i ($i = 1, 2, \ldots, r$) are patterns that cannot be divided horizontally (NH-pattern). We can generate r! duplicate patterns by reordering $A_1, A_2, \ldots, A_r$. All patterns that violate Inequality (21) can be pruned in the search procedure.

$$l_{A_1} \leq l_{A_2} \leq \ldots \leq l_{A_r} \tag{21}$$

However, we still have $r - 1$ other duplicates obtained from the horizontal construction of A_1 and $(A_2, \ldots, A_r)^h$, $(A_1, A_2)^h$ and $(A_3, \ldots, A_r)^h, \ldots,$ $(A_1, \ldots, A_{r-1})^h$ and A_r. We can avoid this type of duplication by permitting horizontal construction such that the second of the two sub-patterns is the NH-pattern (i.e. A_r in the example). However, these concepts cannot completely remove D3 duplicates. Consider the case $l_{A_i} = l_{A_j}$ and $A_i \neq A_j$. Using the above strategy, duplicates $(\ldots, A_i, A_j, \ldots)$ and $(\ldots, A_j, A_i, \ldots)$ can be built. To remove this kind of duplication, we revise Inequality (21) to (22), where $I(A_i)$ represents the identifier of A_i.

$$l_{A_1} \leq l_{A_2} \leq \ldots \leq l_{A_r},\ I(A_i) \leq I(A_{i+1}) \text{ if } l_{A_i} = l_{A_{i+1}},\ i = 1, \ldots, r \tag{22}$$

We can use the memory address of each pattern as its own identifier so that the proposed branching strategy requires no additional memory space. This idea can be generalised in the same manner for vertical builds.

3.3. Upper bounds

This paper proposes two upper bounds for CTDC. The first is a revised version of Kang and Yoon's (2011) approach, created by adding a constraint on the number of pieces. Then, we improve Cung *et al.*'s (2000) upper bound, $u_2(P)$, in Equation (12) above.

Kang and Yoon (2011) proposed an upper bound for region P using horizontal strips by

$$u_h'(P) = \lfloor S_w(W - w_R) \times k_h'(L) \rfloor + \lfloor (W - S_w(W - w_R) \times k_h'(L - l_R) \rfloor, \tag{22}$$

where $S_w(W)$ is the maximum value of the linear combination of the widths not exceeding W, and k_h' (a) is the maximum value of a horizontal strip with length a. The $S_w(W)$ and k_h' (a) are integer knapsack problems and can be calculated by

$$S_w(W) = \max\left\{\sum_{i=1}^{n} w_i y_i : \ \sum_{i=1}^{n} w_i y_i \leq W,\ \forall y_i \in \mathrm{N}\right\} \tag{23}$$

and

$$k_h'(a) = \max\left\{\sum \left(\frac{p_i}{w_i}\right) x_i : \sum l_i x_i \leq a, \forall x_i \in \mathrm{N}\right\}. \tag{24}$$

However, because the upper bound $u'_h(P)$ is for UTDC, the integer knapsack problem in Equation (24) does not consider the constraint on the number of pieces. Thus, we revise Equation (24) as (25) by adding the constraint and recalculate $u'_h(P)$ using $k'_h(a)$ in Equation (25).

$$k'_h(a) = \max\left\{\sum \left(\frac{p_i}{w_i}\right)x_i : \sum l_i x_i \le a,\ x_i \le b_i,\ \forall x_i \in \mathrm{N}\right\} \tag{25}$$

Cung *et al.* (2000) proposed $u_2(P)$ in Equation (12) as an upper bound for region P. Although $u_2(P)$ does consider the constraint on the number of pieces, it does not consider the constraint where the total area of pieces not included in R must not exceed the area of P. To include that constraint, we use $u'_2(\mathrm{P})$ in Equation (26) instead of $u_2(P)$ in Equation (12).

$$u'_2(P) = \left\lfloor \max\left\{\sum_{i\in S_P} p_i x_i : \sum_{i\in S_P} (l_i w_i) x_i \le area(P),\ x_i \le b_i - b_i(R)\right\}\right\rfloor \tag{26}$$

The $u'_2(\mathrm{P})$ can be obtained by relaxing the bounded knapsack problem, which finds the maximum total profit of pieces, subject to the constraint that the total area of the pieces not included in R cannot be greater than $area(P)$. Because the optimal value for area P is an integer not greater than the result of the bounded knapsack problem, $u'_2(\mathrm{P})$ gives an upper bound for area P. The upper bound can be calculated by completely filling $area(P)$ with the remaining pieces in descending order of profit per unit area $\left(\frac{p_i}{l_i \times w_i}\right)$. Thus, the pieces must be sorted in order of profit per unit area at the start of the algorithm to enable a quick calculation. Because $u'_2(\mathrm{P})$ is never greater than $u_2(P)$, $u'_2(\mathrm{P})$ can substitute for $u_2(P)$.

3.4 *Bounding strategy*

Kang and Yoon (2011) proposed two bounding strategies that can prune more than one node at a time. The first uses an upper bound to the set of patterns, whereas the second one uses the waste values of the patterns. These bounding strategies can be applied to CTDC. Furthermore, the bounding strategy using the upper bound of the set of patterns is improved to prune more nodes by adding a constraint on the number of pieces.

Let $\mathbf{C}_{l=x}$ be the branched node set with length x and $\mathbf{H}_{l=x}$ be the set of nodes constructed using the horizontal build between $R \in \mathbf{O}$ and all of the patterns in $\mathbf{C}_{l=x}$.

The upper bound of $\mathbf{H}_{l=x}$ can be obtained using the horizontal strip of $(x, 1)$ and $(L - l_R - x, 1)$. This is possible because all of the nodes in $\mathbf{H}_{l=x}$ will be constructed by combining nodes with length x. Thus, the upper bound of $\mathbf{H}_{l=x}$, $U(\mathbf{H}_{l=x})$, can be calculated by

$$U(\mathbf{H}_{l=x}) = g(R) + \left\lfloor Sw(W - w_R) \times k'_h(L) + (W - Sw(W - w_R)) \times (k'_h(x) + k'_h(L - l_R - x))\right\rfloor. \tag{27}$$

Kang and Yoon (2011) pruned the nodes in $\mathbf{H}_{l=x}$ before they were generated if $U(\mathbf{H}_{l=x})$ was not greater than the current best solution. This strategy can be used in CTDC. However, we can prune more nodes by considering the pieces used in R.

If piece i is consumed in R (i.e. $b_i(R)$ equals b_i) then i cannot be used further in P. Thus, when we solve the integer knapsack problem to calculate the value of the horizontal strips, we can get a tighter value of $U(\mathbf{H}_{l=x})$ by not considering the pieces already consumed in R. This concept is shown in Figure 3. However, because of the large number of combinations of pieces that can be used in P, too much computational effort is required to solve the integer knapsack to obtain the values of $k'_h(L)$, $k'_h(x)$ and $k'_h(L - l_R - x)$ for all cases whenever R is constructed. Thus, we consider only $n+1$ cases, that is, in n cases, we use $n-1$ pieces (excluding one piece in each case), and in one case we use all n pieces. The $k^{\bar{j}}_h(a)$ term in Equation (28) is the maximum value of a horizontal strip with length a not considering piece j.

$$k^{\bar{j}}_h(a) = \max\left\{\sum \left(\frac{p_i}{w_i}\right)x_i : \sum l_i x_i \le a,\ \forall x_i \in \mathrm{N}, i \ne j\right\} \tag{28}$$

Once $k'_h(L)$ and $k^{\bar{j}}_h(L)$ $(j = 1, 2, \ldots, n)$ are solved at the beginning of the algorithm using DP, $k'_h(a)$ and $k^{\bar{j}}_h(a)$ for all $a \le L$ can be calculated readily from the solutions. Thus, we do not need to solve these integer knapsack problems during the algorithm procedure. If there are m pieces that cannot be used in P, we should find a strip that has the

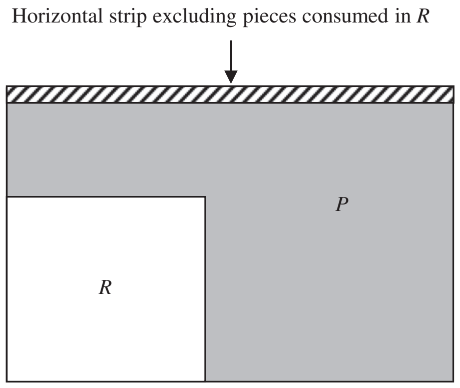

Figure 3. A new horizontal strip with tighter values.

minimum value among the m strips. If there is at least one piece that cannot be used in P, we can use $k''_h(a)$ in Equation (29) instead of $k'_h(a)$ in Equation (28) to obtain tighter $U(\mathbf{H}_{l=x})$ in Equation (26) to prune more nodes at a once.

$$k''_h(a) = \min\left\{k^{\bar{j}}_h(a) : b_j(R) = b_j \text{ for } j = 1, \ldots, n\right\}. \tag{29}$$

3.5 *Differences from previous work*

The main difference between this paper and Kang and Yoon's (2011) work is that this paper deals with the CTDC, whereas Kang and Yoon's (2011) report dealt with the UTDC. Some methods for UTDC can be applied to CTDC. For example, the upper bound and bounding strategy for the UTDC are applicable to the CTDC because the UTDC is a relaxed CTDC. The following are the differences from Kang and Yoon's (2011) work:

- A new efficient method is proposed to remove duplicate patterns, which is not used in Kang and Yoon (2011).
- An efficient bounding strategy is improved from a strategy of Kang and Yoon (2011).
- Two upper bounds are proposed. The first is improved from Cung *et al.* (2000), and the second is revised from Kang and Yoon (2011) by adding a constraint on the number of pieces.
- Two methods of preventing the formation of dominated patterns are proposed. The first is revised from Kang and Yoon's (2011) report to consider the constraint on the number of pieces; the second is adapted from Kang and Yoon (2011).

4. Computational results and analysis

This section compares the computational results of our proposed algorithm with that of Cung *et al.* (2000), which was previously the most efficient exact algorithm for the non-staged CTDC problem. A comparison with the solutions of the TDH algorithm (Hifi 2004) is included for large-scale problems. TDH is a heuristic algorithm for the generation of non-staged patterns based on a top-down approach with hill-climbing strategies. The proposed algorithm and Cung *et al.*'s (2000) algorithm were coded in C++ and tested on a 32-bit PC with a Pentium 4 Core2 Duo (E8400) 3-GHz CPU, 2 GB of memory, and the Windows XP operating system.

The 82 test problems from Hifi (2004) were divided into five groups. The problems are available at Hifi (2011), with detailed descriptions at Alvarez-Valdés *et al.* (2002). The five groups were:

- 23 weighted small and medium problems,
- 30 unweighted small and medium problems,
- 9 weighted and unweighted hard problems,

Table 1. Computational results for the weighted small and medium problem examples.

Problem ID	(*L*, *W*)	*n*	Optimum	Lower bound	Computing time (s) Cung *et al.* (2000)	Computing time (s) Proposed	Dev.[a] (%)
CHW1	(40, 70)	10	2892	2305	0.015	0.000	100.0
CHW2	(40, 70)	20	1860	1640	0.063	0.015	76.2
CW1	(125, 105)	25	6402	6402	0.031	0.015	51.6
CW2	(145, 165)	35	5354	5354	0.062	0.031	50.0
CW3	(267, 207)	40	5689	5026	0.093	0.093	0.0
CW4	(465, 387)	39	6175	6153	0.671	0.375	44.1
CW5	(524, 678)	35	11,659	10,683	2.562	1.125	56.1
CW6	(781, 657)	55	12,923	12,635	4.562	2.234	51.0
CW7	(276, 374)	45	9898	9484	0.296	0.203	31.4
CW8	(305, 287)	60	4605	4318	0.328	0.203	38.1
CW9	(405, 362)	50	10,748	10,254	0.515	0.343	33.4
CW10	(992, 970)	60	6515	5884	12.671	5.500	56.6
CW11	(982, 967)	60	6321	5747	12.109	5.234	56.8
2	(40, 70)	10	2892	2614	0.015	0.000	100.0
3	(40, 70)	20	1860	1740	0.062	0.015	75.8
A1	(50, 60)	20	2020	1860	0.047	0.015	68.1
A2	(60, 60)	20	2505	2395	0.531	0.031	94.2
STS2	(55, 85)	30	4620	4620	1.391	0.187	86.6
STS4	(99, 99)	20	9700	9505	1.000	0.140	86.0
CHL1	(132, 100)	30	8699	8222	2.421	0.218	91.0
CHL2	(62, 55)	10	2326	2240	0.000	0.000	–
CHL3	(157, 121)	15	5283	5283	0.031	0.031	0.0
CHL4	(207, 231)	15	8998	8998	0.078	0.078	0.0
Average computing time					39.554	16.086	–

Note: [a]Percentage gain versus Cung *et al.* (2000).

- 10 weighted large-scale problems, and
- 10 unweighted large-scale problems.

The computing time in the results excluded the elapsed time required to find a lower bound. We used the lower bounds in Cung *et al.* (2000) for those presented in that work and the lower bounds in Hifi (2004) for the others. The bold numbers in the table indicate that the proposed algorithm found a better solution than the best known solution.

The results for the problems in the first and second groups are shown in Tables 1 and 2. The proposed algorithm was as fast as or faster than the algorithm of Cung *et al.* (2000) for all test problems in these groups. On average, the computing time was reduced by 50%. However, the computing times were too short to fully demonstrate the efficiency of the proposed algorithm.

Table 3 shows the results for the problems in the third group that were introduced by Cung *et al.* (2000). The problems that include the letter 's' in their ID are unweighted. Marked improvements in computing time were obtained for this group. The proposed algorithm reduced the computing time by up to 99%. The time required for this group using the algorithm of Cung *et al.* (2000) was 25,950 s, but it was only 122 s with the proposed algorithm. For Hchs4s and Hchl5s, both algorithms yielded the same solutions, 12,006 for Hchl4s and 45,410 for Hchl5s, which differ from the results in Cung *et al.* (2000). It seems that there may be typing errors or implementation errors in Cung *et al.* (2000). Figures 4 and 5 show the optimal patterns for Hchl4s and Hchl5s.

Table 4 shows the result for the problems in the fourth group. Cung *et al.*'s (2000) algorithm could not solve ATP42, ATP43 and ATP49 because of a lack of memory, and the proposed algorithm could not solve ATP42 and ATP43. Although Cung *et al.*'s (2000) algorithm could not solve ATP49 because of a lack of memory, the proposed algorithm was successful; the optimal solution was previously unknown. The proposed algorithm reduced the computing time by up to 99% for the remainder. The solution of ATP41 and ATP48 required 8563 and 4510 s using the algorithm of Cung *et al.* (2000) but only 64 and 9 s with the proposed algorithm, respectively. Hifi's (2004) TDH solved all test problems in this group to optimality except for ATP49. Figure 6 shows the optimal pattern for ATP49.

Table 2. Computational results for the unweighted small and medium problem examples.

Problem ID	(L, W)	n	Optimum	Lower bound	Computing time (s)		Dev.[a] (%)
					Cung *et al.* (2000)	Proposed	
OF1	(70, 40)	10	2737	2713	0.00	0.00	–
OF2	(70, 40)	10	2690	2522	0.00	0.00	–
W	(70, 40)	20	2721	2599	0.02	0.02	0.0
CU1	(100, 125)	23	12,330	12,312	0.06	0.03	50.0
CU2	(150, 175)	34	26100	26,100	0.09	0.09	0.0
CU3	(134, 125)	43	16,679	16,608	0.67	0.38	44.1
CU4	(285, 354)	43	99,366	98,704	2.56	1.13	56.1
CU5	(456, 385)	47	173,364	171,651	4.56	2.23	51.0
CU6	(356, 447)	45	158,572	158,572	0.61	0.40	34.0
CU7	(563. 458)	25	247,150	245,570	1.14	0.53	53.4
CU8	(587, 756)	34	432,714	430,368	0.52	0.34	33.4
CU9	(856, 785)	25	657,055	657,055	6.11	2.36	61.4
CU10	(794, 985)	39	773,772	771,102	8.47	4.33	48.9
CU11	(977, 953)	47	924,696	916,730	10.94	6.14	43.9
2 s	(40, 70)	10	2778	2626	0.00	0.00	–
3 s	(40, 70)	20	2721	2623	0.00	0.00	–
A1s	(50, 60)	20	2950	2950	0.00	0.00	–
A2s	(60, 60)	30	3535	3445	0.00	0.00	–
STS2s	(55, 85)	30	4653	4625	0.00	0.00	–
STS4s	(99, 99)	20	9770	9481	0.02	0.02	6.3
CHL1s	(132, 100)	30	13,099	12,830	0.20	0.05	77.3
CHL2s	(62, 55)	10	3279	3244	0.02	0.02	0.0
CHL3s	(157, 121)	15	7402	7402	0.03	0.03	0.0
CHL4s	(207, 231)	15	13,932	13,932	0.08	0.08	0.0
CHL5	(20, 20)	10	390	361	0.00	0.00	–
CHL6	(130, 130)	30	16,869	16,492	0.03	0.03	0.0
CHL7	(130, 130)	35	16,881	16,572	0.09	0.03	66.7
A3	(70, 80)	20	5451	5348	0.05	0.00	100.0
A4	(90, 70)	20	6179	5951	0.11	0.03	71.6
A5	(132, 100)	20	12,985	12,710	0.22	0.02	93.1
Average computing time					36.58	18.27	–

Note: [a]Percentage gain versus Cung *et al.* (2000).

Table 3. Computational results for nine hard problem examples.

Problem ID	(L, W)	n	Optimum	Lower bound	Computing time (s)		Dev.[a] (%)
					Cung *et al.* (2000)	Proposed	
Hchl1	(130, 130)	30	11,303	10,708	15,325.20	98.63	99.4
Hchl2	(130, 130)	35	9954	9462	1307.73	14.88	98.9
Hchl3s	(127, 98)	10	12,215	11,800	4.19	0.16	96.3
Hchl4s	(127, 98)	10	**12,006**	11,532	727.41	5.30	99.3
Hchl5s	(205, 223)	25	**45,410**	44,118	25.53	1.06	95.8
Hchl6s	(253, 244)	22	61,040	59,687	0.16	0.09	40.4
Hchl7s	(263, 241)	40	63,112	61,837	0.50	0.14	72.0
Hchl8s	(49, 20)	10	911	825	6901.28	1.28	100.0
Hchl9	(65, 76)	35	5240	5030	1658.88	1.02	99.9
Average computing time					25,950.87	122.54	–

Note: [a]Percentage gain versus Cung *et al.* (2000).

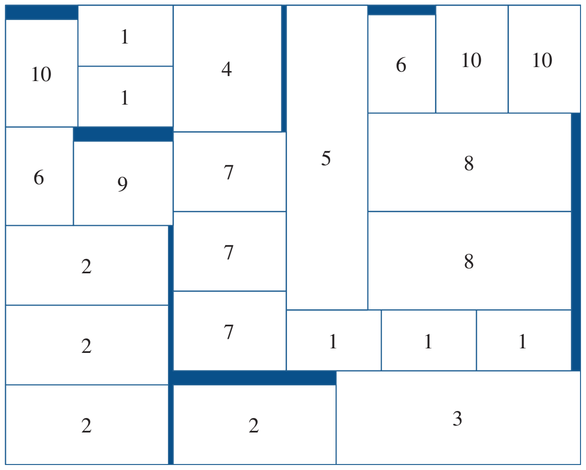

Figure 4. Optimal cutting pattern for Hchl4s.

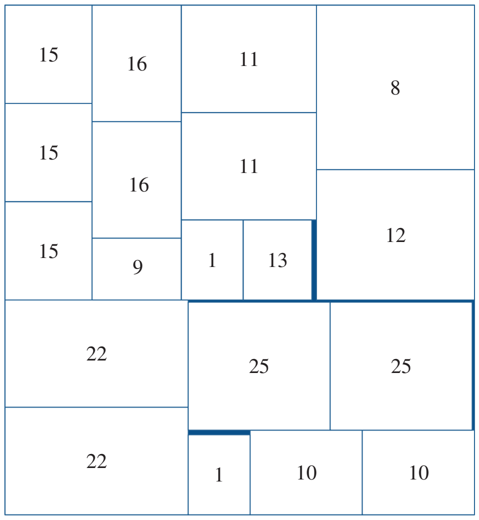

Figure 5. Optimal cutting pattern for Hchl5s.

Table 4. Computational results for the weighted large problem examples.

						Computing time (s)		
Problem ID	(L, W)	n	Optimum	Lower bound	TDH	Cung *et al.* (2000)	Proposed	Dev.[a] (%)
ATP40	(683, 138)	56	67,154	64,213	67,154	1.59	0.55	65.7
ATP41	(837, 367)	36	206,542	202,305	206,542	8563.01	64.80	99.2
ATP42	(167, 291)	59	–	33,012	33,503	–	–	–
ATP43	(362, 917)	49	–	212,062	214,651	–	–	–
ATP44	(223, 496)	39	73,868	70,465	73,868	2.88	0.64	77.7
ATP45	(188, 578)	33	74,691	74,205	74,691	2.05	0.50	75.6
ATP46	(416, 514)	42	149,911	147,021	149,911	1.16	1.06	8.1
ATP47	(393, 554)	43	150,234	142,935	150,234	104.83	3.77	96.4
ATP48	(931, 254)	34	167,660	165,428	167,660	4510.09	9.58	99.8
ATP49	(759, 449)	25	**219,354**	202,801	218,388	–	5913.50	–

Note: [a]Percentage gain versus Cung *et al.* (2000).

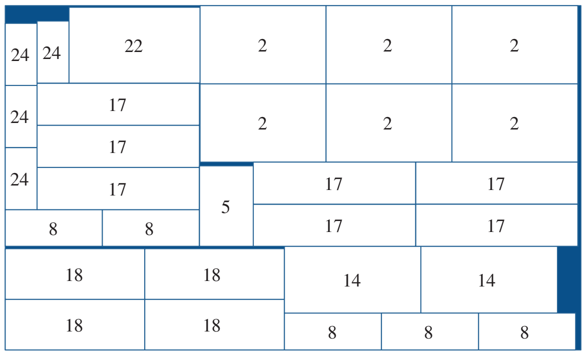

Figure 6. Optimal cutting pattern for ATP49.

Table 5 shows the result for the problems in the fifth group. The proposed algorithm solved ATP34 successfully, whereas Cung *et al.*'s (2000) algorithm could not solve it, because of a lack of memory. The optimal solution to the problem was previously unknown. TDH could not obtain the optimal solution for ATP34 and ATP37. Figure 7 shows the optimal pattern for ATP34.

Table 5. Computational results for the unweighted large problem examples.

						Computing time (s)		
Problem ID	(L, W)	n	Optimum	Lower bound	TDH	Cung *et al.* (2000)	Proposed	Dev.[a] (%)
ATP30	(927, 152)	38	140,904	140,168	140,904	1.33	0.66	50.6
ATP31	(856, 964)	51	823,976	821,073	823,976	27.11	9.23	65.9
ATP32	(307, 124)	56	38,068	37,886	38,068	10.17	0.32	96.8
ATP33	(241, 983)	44	236,611	235,580	236,611	2.61	1.61	38.3
ATP34	(795, 456)	27	**361,398**	356,159	361,167	–	52.34	–
ATP35	(960, 649)	29	621,021	613,656	621,021	305.97	7.86	97.4
ATP36	(537, 244)	28	130,744	129,190	130,744	0.50	0.05	90.8
ATP37	(440, 881)	43	**387,276**	385,811	387,118	3.23	2.88	11.1
ATP38	(731, 358)	40	261,395	259,070	261,395	59.53	2.45	95.9
ATP39	(538, 501)	33	268,750	266,378	268,750	1.53	1.19	22.5

Note: [a]Percentage gain versus Cung *et al.* (2000).

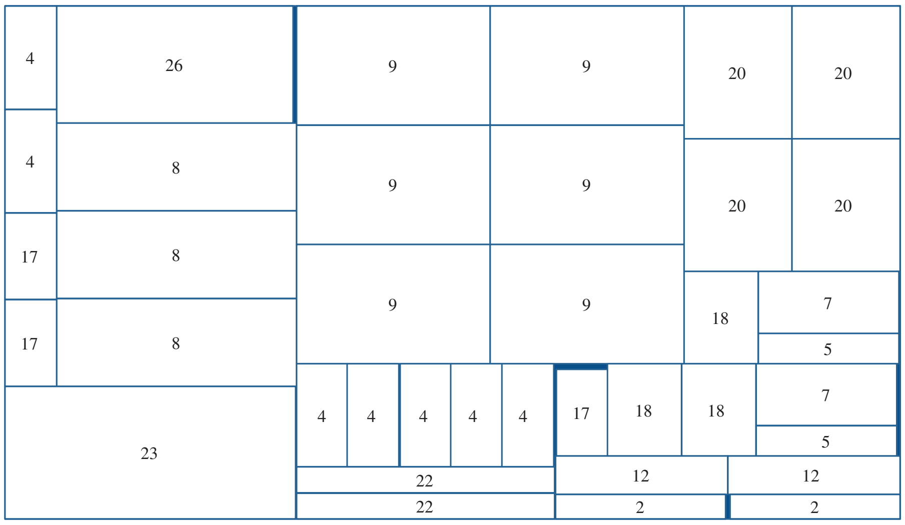

Figure 7. Optimal cutting pattern for ATP34.

The number of nodes searched can be used as a consistent measure of an algorithm's performance, because it is independent of the experimental environment. Table 6 gives the number of nodes generated with the computing time for 10 problems from groups 3–5 (large and hard problems). The proposed algorithm prunes many unpromising (dominated or duplicated) nodes. Pruning unpromising nodes not only removes the nodes but also removes a lot of nodes that would have branched from those nodes in the future. Thus, if some unpromising nodes are pruned at an early stage of the algorithm, we can achieve greater reduction in computing time and nodes searched.

In the proposed algorithm, unpromising nodes are pruned more, because efficient branching strategies and tighter upper bounds lead to a significant decrease in the number of branches. This is shown dramatically in Table 6. The proposed algorithm reduced the number of nodes generated from 99% to 61%. For example, 73% and 61% improvement in the number of generated nodes is shown for ATP46 and ATP37, respectively. However, there was no significant effect in computing time for these problems; they showed only 8% and 11% improvements, respectively. For ATP46 and ATP37, both the proposed algorithm and Cung *et al.*'s (2000) algorithm terminated at an early stage; the numbers of nodes generated were only 5609 and 20402. For the remainder, the numbers of generated nodes for the problems in Table 6 were about a million at minimum. Table 6 shows that the proposed algorithm reduced the number of nodes significantly. The larger and harder the problem is, the more marked is the improvement in computing time.

Table 6. Total number of generated nodes and computing times for 10 examples.

	Number of generated nodes			Computing time (s)		
Problem ID	Cung *et al.* (2000)	Proposed	Dev. (%)	Cung *et al.* (2000)	Proposed	Dev. (%)
Hchl1	128,816,7411	33,471,325	97.4	15,325.20	98.63	99.4
Hchl2	16,7788,375	6635,249	96.0	1307.73	14.88	98.9
Hchl4s	46,843,411	1479,352	96.8	727.41	5.30	99.3
Hchl8s	9643,389	52,092	99.5	6901.28	1.28	100.0
Hchl9	12,7034,062	273,929	99.8	1658.88	1.02	99.9
ATP35	11,849,874	705,931	94.0	305.97	7.86	97.4
ATP37	20,402	7917	61.2	3.23	2.88	11.1
ATP41	305,773,374	12,704,533	95.8	8563.01	64.80	99.2
ATP46	5609	1481	73.6	1.16	1.06	8.1
ATP48	101,228,787	2334,121	97.7	4510.09	9.58	99.8

5. Conclusions

In this paper, we propose a best-first branch-and-bound algorithm for the CTDC problem. The proposed algorithm uses efficient methods for removing duplicate patterns and preventing the construction of dominated patterns. It improves two existing upper bounds and also introduces two bounding strategies that can prune more than one node at a time, as proposed by Kang and Yoon (2011). Furthermore, one of these strategies that use the upper bound of a set of patterns was improved to prune more nodes by considering a constraint on the number of pieces.

The proposed algorithm was compared with the exact algorithm of Cung *et al.* (2000). A comparison with the heuristic algorithm of Hifi (2004) was also performed for large-scale problems. The proposed algorithm is as fast as or faster than Cung *et al.*'s (2000) algorithm. Additionally, the proposed algorithm was able to find the optimal solution for two hard problems (ATP34 and ATP49) that the two existing algorithms were unable to find.

The proposed algorithm achieved a significant reduction, of up to 99%, in computing time compared with the existing exact algorithm and has potential for application to practical large-scale problems.

Acknowledgements

This work was supported by the research fund of Hanyang University (HY-2011-P).

References

Alvarez-Valdés, R., Parajón, A., and Tamarit, J.M., 2002. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers & Operations Research*, 29 (7), 925–947.

Cristofides, N. and Whitlock, C., 1977. An algorithm for two-dimensional cutting problems. *Operations Research*, 25 (1), 30–44.

Cui, Y. and Yang, Y., 2011. A recursive branch-and-bound algorithm for constrained homogenous T-shape cutting patterns. *Mathematical and Computer Modelling*, 54 (5–6), 1320–1333.

Cung, V.D., Hifi, M., and Cun, B.L., 2000. Constrained two-dimensional cutting stock problems a best-first branch-and-bound algorithm. *International Transactions in Operational Research*, 7 (3), 185–210.

G., Y.G., Seong, Y.J., and Kang, M.K., 2003. A best-first branch and bound algorithm for unconstrained two-dimensional cutting problems. *Operations Research Letter*, 31 (4), 301–307.

Gilmore, P.C. and Gomory, R.E., 1966. The theory and computation of knapsack functions. *Operations Research*, 14 (6), 1045–1074.

Hifi, M. and Zissimopoulos, V., 1997. Constrained two-dimensional cutting: An improvement of Christofides and Whitlock's exact algorithm. *Journal of the Operational Research Society*, 48 (3), 324–331.

Hifi, M., 1997. An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. *Computers and Operations Research*, 24 (8), 41–52.

Hifi, M., 2004. Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems. *Journal of Combinatorial Optimisation*, 8 (1), 65–84.

Hifi, M., 2010. Library instance. Université de Picardie. Available from: http://www.laria.u-picardie.fr/hifi/OR-Benchmark/2Dcutting/2Dcutting.html [Accessed 1 February 2011].

Kang, M. and Yoon, K., 2011. An improved best-first branch and bound algorithm for unconstrained two-dimensional cutting problems. *International Journal of Production Research*, 49 (15), 4437–4455.

Martello, S. and Toth, P., 1990. *Knapsack problem: Algorithms and computer implementations*. Chichester: John Wiley & Sons.

Viswanathan, K.V. and Bagchi, A., 1993. Best-first search methods for constrained two-dimensional cutting stock problems. *Operations Research*, 41 (4), 768–776.

Wang, P.Y., 1983. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31 (3), 573–586.

Wy, J. and Kim, B., 2011. Two-staged guillotine cut, two-dimensional bin packing optimisation with flexible bin size for steel mother pate design. *International Journal of Production Research*, 48 (22), 6799–6820.

Yanasse, H.H. and Morabito, R., 2008. A note on linear models for two-group two-dimensional guillotine cutting problems. *International Journal of Production Research*, 46 (21), 6189–6206.