European Journal of Operational Research 202 (2010) 368–378

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A worst case analysis of a dynamic programming-based heuristic algorithm for 2D unconstrained guillotine cutting

X. Song [a,*], C.B. Chu [b], R. Lewis [a], Y.Y. Nie [c], J. Thompson [a]

[a] School of Mathematics, Cardiff University, Senghennydd Road, Cardiff, Wales CF24 4AG, UK
[b] Laboratoire Genie Industriel, Ecole Centrale Paris, Grande voie des Vignes, 92295 Chatenay-Malabry Cedex, France
[c] Shenyang Institute of Automation, Chinese Academy of Sciences, No. 114, Nanta Street, Shenyang 110016, PR China

ARTICLE INFO

Article history:
Received 10 October 2006
Accepted 29 May 2009
Available online 24 June 2009

Keywords:
Heuristic
Dynamic programming
2D cutting

ABSTRACT

In this paper, a dynamic programming-based recursive method is proposed for solving an unconstrained 2D rectangular cutting problem. The algorithm is an incomplete method, in which some intricate cutting patterns may not be obtained. The worst case performance of the algorithm is evaluated and some theoretical analyses for the algorithm are performed. Compared to traditional dynamic programming, this algorithm gives a high percentage of optimal solutions (94.84%, 86.67% and 77.83% for small, medium and large sized unweighted instances, 99.67%, 99.50% and 97.00% for small, medium and large sized weighted instances) but features a far lower computational complexity. Computational results are also presented for some known benchmarks.

1. Introduction

Cutting and packing problems arise in a large number of industrial applications, such as in steel, glass and paper manufacturing, clothing and shoe manufacturing, and also in shipbuilding and car production. These problems are stated and treated in different disciplines such as Management Science, Engineering, Information and Computer Science, Mathematics, and Operational Research. The motivation for this shared interest is that cutting utilization plays an important role in industry when applied to production operations starting with incoming raw materials. The reduction of scrap not only affects the cost of materials used but may also reduce the costs of handling, storing, disposal of scrap, labor and machine time.

In this paper, we study the 2D cutting problem (2DCP), which is defined as follows: There are m types of rectangular sheets requested by the customer, which are called *items*. Positive integers l_i, w_i and p_i, $i = 1, 2, \ldots, m$, are the length, width and profit of the ith type of the item, respectively. The problem then consists of cutting a number of items ordered by the customer from the standard stock rectangular sheet with fixed length L and width W so that the total profit of the items cut out is maximized. Throughout this paper, the term *cutting pattern* refers to a cutting of a subset of the items from the stock rectangular sheet (L, W).

There is no bound on the number of times that item i can be used on the cutting pattern, thus we call this problem an unconstrained 2D cutting problem (U_2DCP). Let $l_{min} = \min_{i=1,2,\ldots,m}\{l_i\}$ and $w_{min} = \min_{i=1,2,\ldots,m}\{w_i\}$ Now assume $\mathbf{Z}_+^m$ is the set of m dimensional non-negative integer vectors. The objective function of this problem is as follows,

$$\max z = \sum_{i=l}^{m} p_i \cdot s_i, \tag{1}$$

where s_i, $i = 1, 2, \ldots, m$, denotes the number of times item i appears in the pattern, $\mathbf{s} = (s_1, s_2, \ldots, s_m)$ represents a feasible cutting pattern and $\mathbf{s} \in \mathbf{Z}_+^m$.

In this paper we consider the following restrictions on the cutting pattern:

- Only guillotine cuts are allowed. That is, a rectangle can only be cut from one edge of the rectangle to the opposite edge and the cutting line is parallel to the two remaining edges.
- The orientation of the items is fixed, i.e. an item of length l and width w is different from an item of length w and width l when $l \neq w$.
- The number of cutting stages is unlimited. That is, there is no bound on the number of guillotine cuts used to cut out the desired item.
- All input parameters (l_i, w_i, p_i, L, W) are integers therefore there is an optimal solution where the cuts on the rectangles are to be made in integer steps.

According to the typology proposed by Wascher et al. (2006), this problem is a Single Large (rectangular) Object Placement Problem (SLOPP). Since two-dimensional cutting problems are NP-hard as shown in (Garey and Johnson, 1979), most of the algorithms in

* Corresponding author.
E-mail addresses: songx6@cardiff.ac.uk, songlily2000@hotmail.com (X. Song).

0377-2217/$ - see front matter Crown Copyright © 2009 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2009.05.047

the literature for medium and large size problems are heuristic algorithms. For a detailed survey, see (Dyckhoff, 1990; Haessler and Sweeney, 1991; Sweeney and Paternoster, 1992; Dowsland et al., 1992; Lodi et al., 2002; Wascher et al., 2006). The detailed descriptions of the algorithms found in the literature to solve this rectangular SLOPP are given in Section 1.1. We also briefly address our contributions and the outline of the whole paper in Sections 1.2 and 1.3.

1.1. Literature review

Within the restrictions mentioned above, the requirement of using guillotine cuts is often considered in industry. consequently, much research has also focussed on this type of problem. To the best of our knowledge, there are a total of six exact algorithms for solving such problems. Gilmore and Gomory (1966) first developed a dynamic programming procedure for generating patterns both for staged and general guillotine cuts. In their method, the maximum profit obtained from cutting a rectangle of length l and width w is produced in one of three ways as follows: First if there are items of lengths and widths less than or equal to l and w, respectively, then the optimum value may simply be the maximum profit of those items. Second, the rectangle could be divided either horizontally or, third, vertically into two sub-rectangles with the best solution being found by adding up the already known optimum value of the two sub-rectangles. Then the overall maximum provides the optimum value. This process is then applied to rectangles of increasing dimensions until the solution for the required rectangle is reached.

This basic method has since been improved in a variety of ways. For example, Herz (1972) suggests the use of *canonical dissections* (called normalized cuts in our paper), which restrict the sub-problems to rectangles whose dimensions are an integral combination of item lengths and widths. More recently, Beasley (1985a) has also presented some improvements to the Gilmore–Gomory procedure and has demonstrated that their procedure is capable of solving problems with as many as 50 sizes in 2 seconds on a CDC 7600. The computational complexity of Beasley's method is $O(n \cdot L \cdot W \cdot (L+W))$, where n is the maximum number of stages that an item needs in order to be cut from the stock. Furthermore, Beasley also uses canonical dissections to reduce the computational complexity to $O(n \cdot |\overline{\mathbf{L}}| \cdot |\overline{\mathbf{W}}| \cdot (|\overline{\mathbf{L}}| + |\overline{\mathbf{W}}|))$, where $|\overline{\mathbf{L}}|$ and $|\overline{\mathbf{W}}|$ are the numbers of available canonical dissections within L and W.

In addition to the three exact methods mentioned above, Hifi (2001) has also proposed two different exact algorithms for solving unconstrained (un)weighted 2DCPs in which there is a pre-specified upper bound on the number of cutting stages. In this case, problems where a maximum of two cutting stages are allowed are solved by applying the version of the dynamic programming procedure originally developed by Gilmore and Gomory (1966), while problems with a maximum of three cutting stages are solved using a top down approach combined with a dynamic programming procedure. Meanwhile, Young-Gun et al. (2003) have also proposed a best-first branch-and-bound algorithm based upon a bottom-up approach. Most recently, Cintra et al. (2008) have also solved various kinds of cutting stock problems by using dynamic programming and column generation techniques. In solving the U_2DCP, the authors have further improved the dynamic programming approach proposed by Beasley (1985a) by reducing the size of sets of canonical dissections within L and W and by using more efficient computer programming techniques such as binary search. In making such modifications the computational complexity of the algorithm has been slightly improved.

It is notable that all of the algorithms mentioned above have been used mostly for solving small and medium sized problems. For larger problems, some researchers have chosen to develop heuristic algorithms, which are only able to find approximate solutions, but which will usually feature much shorter run times than exact approaches. For example Hinxman (1976) has formulated and tackled unconstrained problems by using a problem reduction methodology. Morabito et al. (1992), on the other hand, have used a *Depth-first search and Hill climbing* strategy, producing algorithm DH, which we consider in our experiments later. Fayard and Zissimopoulos (1995), meanwhile, have designed an algorithm based on solving a series of one-dimensional Knapsack problems using dynamic programming techniques (referred to as algorithm KD in later sections). Later, Hifi (1997) proposed a combined DH/KD algorithm, which was reported to be better than both the KD and DH methods individually. Following this, Scheithauer and Sommerweiss (1998) developed new heuristics for the U_2DCP, based on the G4-heuristics for the pallet loading problem. In this case, the authors' aim was to generate optimal packing patterns with a 4-block structure, which often turns out to be useful in practice.

In the past decade, new techniques have also continued to be developed. Alvarez-Valdes et al. (2002), for example, show how tabu search can be used for solving large-scale U_2DCP problems. In their experimental report, however, they admit that the performance of their approach is slightly inferior to that of algorithm DH/KD, which is currently considered to be the best heuristic algorithm for unconstrained problems. Lodi and Monaci (2003) proposed two Integer Linear Programming (ILP) models for 2DCP involving a polynomial number of variables and constraints, which can be easily adapted to all the variants of 2DCP including the U_2DCP. However, the problems solved in their work are confined to 2-stage 2DCP (that is, the maximum number of cuts allowed to obtain each item is fixed to 2). Most recently Cui (2007) has proposed the *simple block* (SB) pattern. The SB pattern is defined recursively and each cut on the stock produces just one simple block. A horizontal cut produces a horizontal block with width equal to that of the leftmost item in the block, and a vertical cut produces a vertical block with length equal to that of the bottom-most item in the block. The algorithm generates the optimal SB pattern, and selects optimally the first piece in each block. It then uses upper bounds to prune some unpromising branches from the search trees. In a follow-on paper Cui and Zhang (2007) have also suggested that the two-stage general block pattern in U_2DCP can be cut into general blocks in two stages, with two or more stages being required to cut the blocks into items. A dynamic programming recursion is used to determine the strip layout on each block. Both of the algorithms proposed by Cui (2007) and Cui and Zhang (2007) aim at finding the optimal solution for some specified cutting pattern in order to reduce run times without loss of the cutting efficiency.

Note that the number of times a particular item i can appear in a feasible cutting pattern has a natural upper bound of $\left\lfloor \frac{L}{l_i} \right\rfloor \cdot \left\lfloor \frac{W}{w_i} \right\rfloor$. If the upper bounds are set to be less than this, then we obtain the *constrained* 2D cutting problem (C_2DCP), which is usually treated differently in the literature. Several different exact algorithms have been suggested for this problem, including those described in Christofides and Whitlock (1977), Wang (1983), Viswanathan and Bagchi (1988), Vasko (1989), Dowsland et al. (1992), Christofides and Hadjiconstantinou (1995) and Hifi and M'Hallah (2005). Approximate algorithms for this problem have also been proposed by Morabito and Arenales (1996), Cung et al. (2000), Belov and Scheithauer (2006), Hifi and M'Hallah (2006), Gongalves (2007) and Cui (2008). Finally, we note that there are also a variety of exact and heuristic methods for solving non-guillotine problems, including Beasley (1985b), Baldacci and Boschetti (2007) and Alvarez-Valdes et al. (2007).

1.2. Contributions

In this paper, we propose a dynamic programming-based heuristic algorithm for finding approximate solutions to the U_2DCP. The basic idea of the algorithm is that the maximum profit obtained from cutting a rectangle of length l and width w is produced in one of two ways. The rectangle is either cut horizontally and then vertically or vertically and then horizontally to produce an item. The resultant value for each type of cut is then calculated and stored for each available item. The overall maximum then provides the maximum profit. This process is applied to rectangles of increasing dimensions until a solution of the stock rectangular sheet is reached. The approach thus generates a subset of all possible cutting patterns, which contains an optimal cutting pattern in many instances. The algorithm works well because the optimal patterns that are not investigated tend to have duplicated parts or symmetrical parts to those that are investigated.

We also provide a worst case analysis to specify a tight lower bound of our algorithm. Indeed, the algorithm returns optimal solutions in a high percentage of cases (94.84%, 86.67% and 77.83% for small, medium and large sized unweighted instances, 99.67%, 99.50% and 97.00% for small, medium and large sized weighted instances) but its computer running times and computational complexity are considerably lower than traditional exact methods.

1.3. Outline

The remainder of the paper is as follows: In Section 2, our algorithm is described in detail. In Section 3, the worst case analysis of this algorithm is then presented. Section 4 contains the computational study and the discussion of the results. Finally, in Section 5, we present our conclusions.

2. A dynamic programming-based heuristic algorithm

In essence, the algorithm we propose, called the dynamic programming-based heuristic (DPH) algorithm, is an incomplete method, in which certain cutting patterns may not be obtainable. The DPH algorithm generates a subset of all possible cutting patterns which, we find, happens to contain an optimal cutting pattern in many instances. We first introduce the related concepts in Section 2.1, then in Section 2.2, the heuristic dynamic program recursion is introduced. Finally in Section 2.3, we give the main steps of the heuristic algorithm.

2.1. Twice guillotine cut

We now define the cutting types which are used to determine cutting patterns. In all the definitions, all cuts made are guillotine cuts.

Definition 1 (*Normalized cut and normalized cutting pattern*). A cutting pattern is normalized if both the left-hand edge and the bottom edge of all items in the pattern are adjacent to a cut (see Fig. 1b). A cut is normalized if it generates a normalized cutting pattern. Any pattern can be normalized by some movement of the items in the pattern, i.e., if there is an item which is not adjacent to a cut on the left-hand edge, simply move the item to the left until it touches a cut. Similarly, we can move the item downwards until the bottom edge touches a cut. For example, in Fig. 1a, we can move item 2 downwards and move item 3 and 4 to the left until we obtain Fig. 1b. The pattern in Fig. 1a is a *non-normalized cutting pattern*.

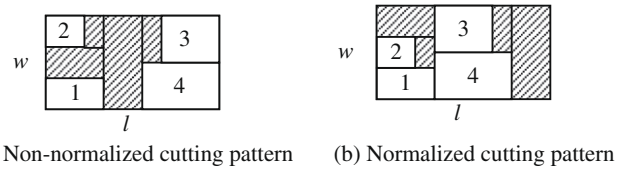
All cuts mentioned in this paper are normalized cuts.

Definition 2 (*Waste rectangle and trimming cut*). A newly generated rectangle, which is too small to contain any item, is called a *waste rectangle* (see Fig. 2, rectangle A). If one of the two new rectangles generated by a guillotine cut is a waste rectangle, we call this guillotine cut a *trimming cut* (see Fig. 2, cut line a).

Definition 3 (*Twice guillotine cut and twice cutting type*). A cut is a *twice guillotine cut* if it produces an item i at the bottom left of the stock sheet with either a horizontal cut followed by a vertical cut or a vertical cut followed by a horizontal cut (see Fig. 3). A cutting pattern is of *twice cutting type* if it is obtained by successive twice guillotining cuts. For example, in Fig. 4, item 3 is obtained by a twice guillotine cut (vertical cut first, horizontal cut second). Then in the newly generated rectangle of size $(l - l_3, w)$, item 5 can be cut out with a twice guillotine cut (horizontal cut first, vertical cut second). In the newly generated rectangle of size $(l_3, w - w_3)$, item 1 can be cut out with a 0-cut first and a vertical cut second. All the other items can be cut out with a twice guillotine cut on the newly generated rectangles from the previous cuts in a similar way (we do not list them all here).

For convenience, we say that all the trimming cuts are twice guillotine cuts. It is understandable if we assume there exists an extra *0-cut*, which is perpendicular to the original trimming cut (see Fig. 2, cut line b).

We see that only two ways of cutting can be obtained with a twice guillotine cut for a rectangular stock sheet of size (l, w). The first way of cutting (Fig. 3a) is obtained firstly by a horizontal



(a) Non-normalized cutting pattern (b) Normalized cutting pattern

Fig. 1. Normalized cutting pattern (shaded area is the waste area).

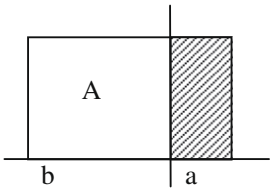
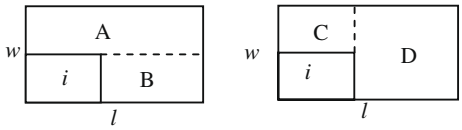


Fig. 2. Trimming cut and 0-cut (shaded area is the waste).



(a) Horizontal cut first (b) Vertical cut first

Fig. 3. Two ways of cutting the rectangular item i.

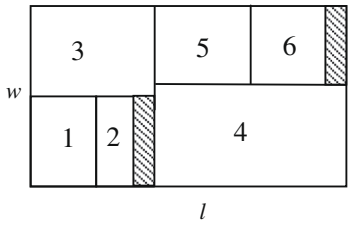


Fig. 4. Twice cutting type (shaded area is the waste).

guillotine cut, and then with a vertical cut. The second (Fig. 3b) is obtained firstly by a vertical guillotine cut and then a horizontal cut. We found in a large number of experiments that most of the optimal cutting patterns are of twice cutting type.

Using the above methods, the DPH algorithm was developed for finding an optimal twice guillotine cutting pattern. Compared to the exact algorithms, this algorithm has much lower computational complexity and gives a high percentage of optimal solutions. Compared to the approximate algorithms, this algorithm is easy to implement and the simple structure of the algorithm makes it possible to find a tight lower bound on the approximation ratio of the algorithm, which is rare in comparison with other approximate algorithms reported in the literature.

2.2. Dynamic programming recursion

Let $F(l,w)$ represent the optimal value for the cutting patterns of twice cutting type in a sub-rectangle of size (l,w). The following recursive procedure has been established in the literature (Cui, 2007):

$$F(l,w)=\min_{i=1,2,\ldots,m,l\geqslant l_i,w\geqslant w_i}\{p_i+F(l_iw-w_i)+F(l-l_i,w),p_i+F(l-l_i,w)+F(l,w-w_i)\},\quad l\in\overline{\mathbf{L}},\ w\in\overline{\mathbf{W}} \tag{2}$$

where $\overline{\mathbf{L}}=\{l|l=\sum_{i=1}^{m}l_ia_i,\quad 1\leqslant l\leqslant L-l_{min},\ a_i\in Z^+,\quad i=1,2,\ldots,m\}\bigcup\{L\}$, $\overline{\mathbf{W}}=\{w|w=\sum_{i=1}^{m}w_ib_i1\leqslant w\leqslant W-w_{min},\ b_i\in Z^+,\ i=1,2,\ldots,m\}\bigcup\{W\}$

Note that the values of l and w belong to the set $\overline{\mathbf{L}}$ and $\overline{\mathbf{W}}$, respectively, which is analyzed in Beasley (1985a). Since for each given (l,w), where $l\in\overline{\mathbf{L}},\ w\in\overline{\mathbf{W}}$, recursion (2) needs $O(m)$ operation time. Thus the total computation time of the above mentioned recursion is $O(m\cdot|\overline{\mathbf{L}}|\cdot|\overline{\mathbf{W}}|)$.

Theorem 1. *Let* $\rho_l=\left\lfloor\frac{L}{l_{min}}\right\rfloor$, $\rho_w=\left\lfloor\frac{W}{w_{min}}\right\rfloor$ *represent the largest integer less than or equal to* $\frac{L}{l_{min}}$, $\frac{W}{w_{min}}$, *respectively. We have* $|\overline{\mathbf{L}}|\leqslant\sum_{j=1}^{\rho_l}C_{j+m-1}^{j}$, *and* $|\overline{\mathbf{W}}|\leqslant\sum_{j=1}^{\rho_w}C_{j+m-1}^{j}$ *where* $C_{j+m-1}^{j}=\frac{(j+m-1)!}{j!(m-1)!}$.

Proof. The problem of computing $|\overline{\mathbf{L}}|$ has the same structure as the problem of computing the number of terms in the polynomial $\sum_{j=1}^{\rho_l}+(l_1+l_2+\cdots+l_m)^j$. In fact, each term in the polynomial $(l_1+l_2+\cdots+l_m)^j$ represents a possible combination of lengths of j items. For example, the item $l_1\cdot l_2^{j-1}$ corresponds to the combination $l_1+(j-1)\cdot l_2$. To obtain all possible combinations of $l_1,l_2,\ldots,l_m$ that satisfy $\sum_{i=1}^{m}a_i\cdot l_i\leqslant L$, j should be taken from 1 to ρ_l in the polynomial $(l_1+l_2+\cdots+l_m)^j$ correspondingly. The condition that $j>\rho_l$ will not be considered, since under this condition we have $\sum_{i=1}^{m}a_i\cdot l_i>L$ according to the definition of ρ_l. When $\left\lfloor\frac{L}{l_{max}}\right\rfloor=\left\lfloor\frac{L}{l_{min}}\right\rfloor$, where $l_{max}=\max_{1,2,\ldots,m}\{l_i\}$, each term in the polynomial mentioned above corresponds to a feasible solution of the U_2DCP. This is not true otherwise.

For the given positive integer j and m there are in total C_{j+m-1}^{j} terms in the polynomial $(l_1+l_2+\cdots+l_m)^j$ (Hohn, 1941). Thus the total number of terms of the polynomial $\sum_{j=1}^{\rho_l}(l_1+l_2+\cdots+l_m)^j$ is $\sum_{j=1}^{\rho_l}C_{j+m-1}^{j}$. We obtain $|\overline{\mathbf{L}}|\leqslant\sum_{j=1}^{\rho_l}C_{j+m-1}^{j}$, where $|\overline{\mathbf{L}}|=\sum_{j=1}^{\rho_l}C_{j+m-1}^{j}$ is valid only if $\left\lfloor\frac{L}{l_{max}}\right\rfloor=\left\lfloor\frac{L}{l_{min}}\right\rfloor$. The proof is the same for $|\overline{\mathbf{W}}|$. □

2.3. Main steps of the heuristic algorithm

In this section, we describe how to use the developed dynamic programming recursion (2) to generate optimal cutting patterns of twice guillotine cutting type. The basic idea of the algorithm has been given in Section 1.2. We now give in more detail the main steps of the algorithm.

In the following, let $G(l,w)=i^*$ represent the item cut out at the bottom left of the sub-rectangle in the optimal cutting patterns of twice cutting type in a sub-rectangle of length l and width w. Also let $Q(l,w)\in\{V,H\}$ be the method of cutting as shown in Fig. 3a and b (where V represents the way of cutting the sub-rectangle first vertically, then horizontally and H represents the way of cutting the rectangle first horizontally, then vertically). Finally, let *span_l(l)* be the function, which returns the span between the current value l ($l\in\overline{\mathbf{L}}$) and the next one within the set $\overline{\mathbf{L}}$. Let *span_w(w)* be the function, which returns the span between the current value w ($w\in\overline{\mathbf{W}}$) and the next one within the set $\overline{\mathbf{W}}$. The input parameters of the algorithm are therefore (L,W) and (l_i,w_i,p_i), $i=1,2,\ldots,m$, whilst the outputs are $F(l,w)$, $Q(l,w)$ and $G(l,w)$ for all l ($l\in\overline{\mathbf{L}}$) and w ($w\in\overline{\mathbf{W}}$). The main steps of the algorithm are as follows:

Step 1 Initialization: Set $F(l,w)=0$ and $G(l,w)=0$ for all $l\leqslant L$ and $w\leqslant W$. For sub-rectangles with length $l\geqslant l_{min}$ and width $w\geqslant w_{min}$, set $i=1$, $l=l_{min}-1$, $w=w_{min}$. Let $\psi=0$ to store values.

Step 2 Set $l=l+span\_l(l)$, $w=w$. Now, for each i, $i=1,2,\ldots,m$, compute

$$\psi=\begin{cases}\max\limits_{l_i\leqslant l,w_i\leqslant w}\left\{\begin{matrix}p_i+F(l_i,w-w_i)+F(l-l_i,w)\\ p_i+F(l-l_i,w_i)+F(l,w-w_i)\end{matrix}\right\}, & l\geqslant l_{min}\text{ and }w\geqslant w_{min}\\ 0, & \text{otherwise.}\end{cases}$$

If $\psi>F(l,w)$, set $F(l,w)=\psi$, $G(l,w)=i$ and $Q(l,w)=\begin{cases}V & \text{if }\psi=p_i+F(l_i,w-w_i)+F(l-l_i,w)\\ H & \text{if }\psi=p_i+F(l-l_i,w_i)+F(l,w-w_i)\end{cases}$.

Step 3 If $l<L$, go to step 2, else go to step 4.

Step 4 Set $w=w+span\_w(w)$. Initialize l by setting $l=l_{min}-1$. If $w<W$, set $F(l,w)=0$ and go to step 3, else stop.

The DPH algorithm finds the optimal value $F(L,W)$ in an orderly manner by increasing the dimension of the sub-rectangles. Note that to find the corresponding cutting pattern, we must use the dynamic programming recursive method again in reverse. That is, for the given optimal value $F(l,w)$, we have the corresponding $G(l,w)$ and $Q(l,w)$. If $G(l,w)=i$ and $Q(l,w)=V$, then the rectangle (l,w) could be decomposed into one item i and two sub-rectangles $(l_i,w-w_i)$ and $(l-l_i,w)$. If $G(l,w)=i$ and $Q(l,w)=H$, then the rectangle (l,w) could be decomposed into one item i and two sub-tangles $(l-l_i,w_i)$ and $(l,w-w_i)$. In this way, we could obtain the optimal pattern by deciding which item is to be cut next and in which way to cut for a sub-rectangle with decreasing dimensions. Originally, $l=L$ and $w=W$.

Because the procedure to find the optimal solution needs to be executed on each sub-rectangle of length l and width w until it is too small to contain any item, the computational complexity of this procedure is $O(\xi)$, where $\xi=\max_{i=1,2,\ldots,m}\left\{\left\lfloor\frac{L\cdot W}{l_i\cdot w_i}\right\rfloor\right\}$ denotes the maximum number of items that can appear in the cutting pattern. Thus the general computational complexity of the algorithm is $O(\max\{\xi,|\overline{\mathbf{L}}|\cdot|\overline{\mathbf{W}}|\cdot m\})$.

3. Worst case analysis of DPH

Although the DPH algorithm finds optimal solutions for the cutting patterns that are of twice guillotine type, the same cannot be said for the general cutting pattern. The reason is that some of the combinatorial conditions for a stock rectangular sheet of size (l,w) may not be obtainable with a twice guillotine cut. As is shown in the example in Fig. 5, no item (marked in sequential numbers) can be cut out with two guillotine cuts.

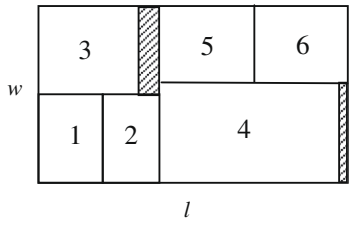


Fig. 5. The combinatorial condition not obtained (shaded area is the waste).

In this section, we study performance bounds for the DPH algorithm which deals with U_2DCP. First, in Section 3.1, we analyze the structure of the cutting patterns that cannot be obtained by successive twice guillotine cuts. Then in Section 3.2, we give the worst case analysis of our algorithm.

3.1. Intricate structure

To aid the following analysis, we use the following definitions.

Definition 4 (*intricate cutting pattern*). For a given cutting pattern, smaller rectangular sheets are generated successively with guillotine cuts. If in one of the smaller rectangular sheets, which is not a waste rectangle, an item can only be cut out using three or more guillotine cuts (see Fig. 6), we say that this cutting pattern is an *intricate cutting pattern*, which cannot be obtained with the DPH algorithm. With this definition, we know that an intricate cutting pattern must be an n-staged pattern, where $n \geqslant 3$. However, the opposite may not be true. Fig. 4 gives an example of a 3-staged pattern that is of a twice cutting type, whereas Fig. 5 gives an example of a 3-staged pattern that is an intricate cutting pattern. Obviously all of the 2-staged cutting patterns are of twice cutting type.

Definition 5 (*Simplest intricate cutting pattern*). If in an intricate cutting pattern, there are in total six items to be cut out, this intricate cutting pattern is called a *simplest intricate cutting pattern*. Fig. 5 gives an example of a simplest intricate cutting pattern. It is easy to deduce that a cutting pattern with less than six items is of twice cutting type, which will not be an intricate cutting pattern.

Definition 6 (*intricate structure*). For a given rectangular sheet of size (l, w), suppose the first guillotine cut is vertical. Then there exist a series of guillotine vertical cuts dividing the stock rectangular sheet (l, w) into several strips with the same width w. Within each strip, there is a series of horizontal guillotine cuts dividing the strips into a series of rectangles. Thus we obtain rectangles $(A_1, A_2, \ldots, A_p), (B_1, B_2, \ldots, B_q), \ldots$, as shown in Fig. 6. An intricate structure contains rectangles $(A_1, A_2, \ldots, A_p), (B_1, B_2, \ldots, B_q), \ldots$ and within each rectangle A_i, $i = 1, 2, \ldots, p$, and B_j, $j = 1, 2, \ldots, q$, there is at least one vertical guillotine cut.

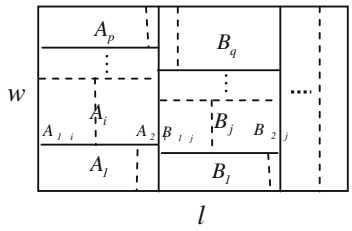


Fig. 6. Intricate structure.

To judge if a cutting pattern is an intricate cutting pattern, we give the theorem as follows:

Theorem 2. *Only the cutting patterns that contain the intricate structure are intricate cutting patterns.*

Proof. For any given rectangular sheet of size (l, w) in a given cutting pattern, we only discuss the case that the first guillotine cut is vertical, the discussion is analogous for the case that the first guillotine cut is horizontal.

Suppose there exists a series of guillotine cuts parallel to the y-axis (see Fig. 7, trimming cuts are not included) that divide the rectangular stock sheet (l, w) into several strips $A, B, \ldots$, with the same width w. There are several cases for the following cuts:

Case 1: If there exists a strip A, within which no horizontal cut is available or only a trimming cut is available (Fig. 8), then the strip A contains (or is) a single item, which can be cut out with a twice guillotine cut.

Case 2: If within each strip there is at least one horizontal cut which is not a trimming cut, then there must be series of rectangles $(A_1, A_2 \ldots, A_p), (B_1, B_2, \ldots, B_q), \ldots$, that are cut out and each rectangle contains at least one item (see Fig. 9).

Case 2.1: If within the series of rectangles, there is one, say rectangle A_1, in which no vertical guillotine cut is available (Fig. 10), rectangle A_1 must be an item, which can be cut out with a twice guillotine cut.

Case 2.2: In the condition that the cutting pattern contains the series of rectangles $(A_1, A_2, \ldots, A_p), (B_1, B_2, \ldots, B_q), \ldots$, and within each rectangle there is at least one vertical guillotine cut, we see there is no item that can be cut out with a twice guillotine cut. Only in this case can we say that the given cutting pattern is an intricate cutting pattern.

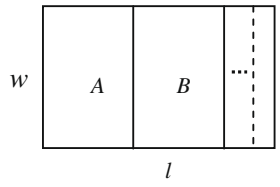


Fig. 7. A series of guillotine cuts parallel to the y-axis.

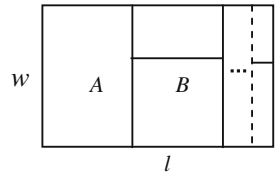


Fig. 8. Case 1: Within trip A, there is no horizontal cut.

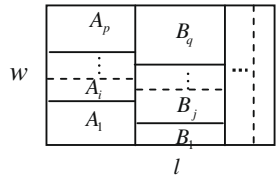


Fig. 9. Case 2: Within each trip, there is at least one horizontal cut.

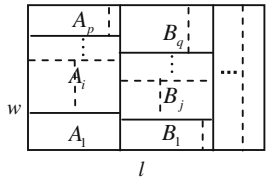


Fig. 10. Case 2.1: Within one of the rectangles, there is no vertical cut.

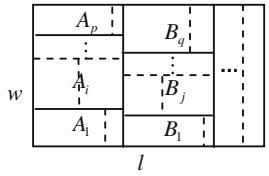


Fig. 11. Case 2.2: Within each of the rectangle, there is at least one vertical cut.

Figs. 5 and 11 give two examples. It is obvious that within strip A, there must be a rectangle A_i, which contains at least two stock rectangular sheets A_{i_1} and A_{i_2}, which are not waste rectangles, according to the definition of normal cuts. The condition is the same for other strips.

Thus, only the cutting patterns that contain the intricate structure are intricate cutting patterns. □

3.2. Approximation ratio of DPH

We now prove the approximation ratio of the DPH algorithm as follows.

Let z denote the value of the globally optimal solution of the problem, defined in Section 1, and let $\bar{z}$ denote the value of the approximate solution provided by DPH.

Theorem 3. *If $\rho = \max\{\rho_l, \rho_w\}$ is equal to 1, 2 or 3, we have $z = \bar{z}$.*

Proof. Suppose we are given four items with length l_a, l_b, l_c, l_d. Also, without loss of generality, let l_a be the minimum length of the four lengths, it is obvious that $l_a + l_b + l_c + l_d \geqslant 4l_a$. For the case $\rho < 4$, we have $4l_a > L$. Thus $l_a + l_b + l_c + l_d > L$ is obtained. In this case, the rectangle A_{i_1}, A_{i_2}, B_{j_1} and B_{j_2} cannot exist together in one stock rectangular sheet. That is, the intricate structure cannot be contained in any cutting pattern. So the optimal cutting pattern must be of twice cutting type. The optimal solution found with DPH algorithm is the optimal solution. Thus, $z = \bar{z}$. □

Theorem 4. $\frac{\bar{z}}{z} \geqslant \frac{6}{7}$.

Proof. This proof is given in two steps. First, we prove that if the optimal pattern is a simplest intricate cutting pattern (Fig. 5), then $\frac{\bar{z}}{z} \geqslant \frac{6}{7}$. We then prove that if the optimal pattern contains the intricate structure (Fig. 6), then $\frac{\bar{z}}{z} \geqslant \frac{6}{7}$. That is, Theorem 4 holds in the general case.

If the optimal pattern is a simplest intricate cutting pattern, it is easy to deduce that by taking away any of the items from the cutting pattern, the remaining structure is of a twice cutting type, which can be obtained by DPH (Fig. 5). Since DPH finds the optimal solution for the cutting pattern of twice cutting type, the remaining structure could be enumerated with DPH even when the item with smallest profit is taken away. Suppose the profits of the 6 items in the simplest intricate cutting pattern are $p_1, p_2, \ldots, p_6$ and the $p_{min} = \min_{i=1,2,\ldots,6} p_i$. Obviously, we have $p_{min} \leqslant \sum_{i=1}^{6}\{pi\}$. Thus $\frac{\bar{z}}{z} \geqslant \frac{\sum_{i=1}^{6}\{pi\} - p_{min}}{\sum_{i=1}^{6}\{pi\}} \geqslant \frac{5}{6}$.

However, $\frac{5}{6}$ is not a tight lower bound. It can be easily proved that by taking away any item k, DPH can find a better cutting pattern by filling into the empty area generated by the removal of item k. For example, if in Fig. 5, $l_1 + l_2 \leqslant l_5 + l_6$ and $w_3 \leqslant w_4$, by taking away item 4, DPH can find a cutting pattern with item 3 filling into the empty area. Thus DPH can find a cutting pattern with the solution $\sum_{i=1}^{6}\{pi\} - p_k + p_j$.

Obviously, $p_k > p_j \geqslant p_{min}$, otherwise $\bar{z} = z$, which is not the worst case we are discussing. Besides, $p_k \leqslant 2p_j$, otherwise $\sum_{i=1}^{6}\{pi\} - p_k + p_j < \sum_{i=1}^{6}\{pi\} - 2p_j + p_j = \sum_{i=1}^{6}\{pi\} - p_j$ thus taking away item j from the optimal solution would form a better solution, which is not the worst case that we are discussing. In the worst case, we have $p_k = 2p_j$.

Since in this case, $z = p_k + p_j + \sum_{i=1}^{6}\{pi \cdot y_i\} = 3p_j + \sum_{i=1}^{6}\{pi \cdot y_i\}$, where $y_i = \begin{cases} 0 & i = j \text{ or } k \\ 1 & \text{otherwise} \end{cases}$, we have

$$\begin{cases} p_j \leqslant \frac{1}{7}\sum_{i=1}^{6}\{pi\} & \frac{\bar{z}}{z} = \frac{\sum_{i=1}^{6}\{pi\} - p_k + p_j}{\sum_{i=1}^{6}\{pi\}} \geqslant \frac{\sum_{i=1}^{6}\{pi\} - p_j}{\sum_{i=1}^{6}\{pi\}} \geqslant \frac{6}{7}, \\ p_{min} \leqslant \frac{1}{7}\cdot\sum_{i=1}^{6}\{pi\} & \frac{\bar{z}}{z} = \frac{\sum_{i=1}^{6}\{pi\} - p_{min}}{\sum_{i=1}^{6}\{pi\}} \geqslant \frac{6}{7}. \end{cases}$$

That is, by either taking away the item with minimum profit p_{min} or by taking away item k and filling in the empty area with item j, the approximation ratio is the same $\frac{\bar{z}}{z} \geqslant \frac{6}{7}$.

In general, $\frac{\bar{z}}{z} \geqslant \frac{6}{7}$ holds for instance of the simplest intricate cutting pattern.

Furthermore, any intricate structure could be decomposed into several simplest intricate cutting patterns by considering collections of small adjacent items as single large items when necessary (Fig. 12 gives an example, where an intricate structure is decomposed into four parts of the simplest intricate cutting pattern). Suppose that the general intricate structure is decomposed into s parts of the structure as Fig. 5. The optimal solution of each part is z_i, $i = 1, 2, \ldots, s$ and the solution found with DPH is $\bar{z}_i$, $i = 1, 2, \ldots, s$. From the first step of the proof, we have $\frac{\bar{z}_i}{z_i} \geqslant \frac{6}{7}$. Then $\frac{\bar{z}}{z} = \frac{\sum_{i=1}^{s}\bar{z}_i}{\sum_{i=1}^{s}z_i} \geqslant \frac{6}{7}$.

Thus we conclude that $\frac{\bar{z}}{z} \geqslant \frac{6}{7}$ in the general case. □

As an example, Fig. 13 gives a worst case scenario, where the approximation ratio is close to $\frac{6}{7}$. In that example, the length and width of the stock sheet are both 1000. The size of the items are as follows: (1) 427×334; (2) 214×666; (3) 281×749; (4) 191×749; (5) 571×251. The total area of the items packed with the DPH algorithm is 858,616 and the total area of the items packed with the DP algorithm is 999,415, which is also the optimal value. So the approximate ratio of this example is $\frac{858,616}{999,415}$, which is close to $\frac{6}{7}$. A more accurate approximate ratio can be obtained by enlarging the size of the stock sheets and changing the size of

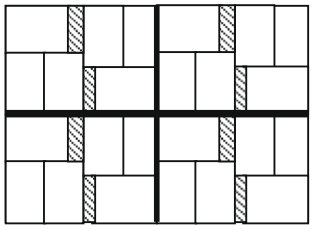
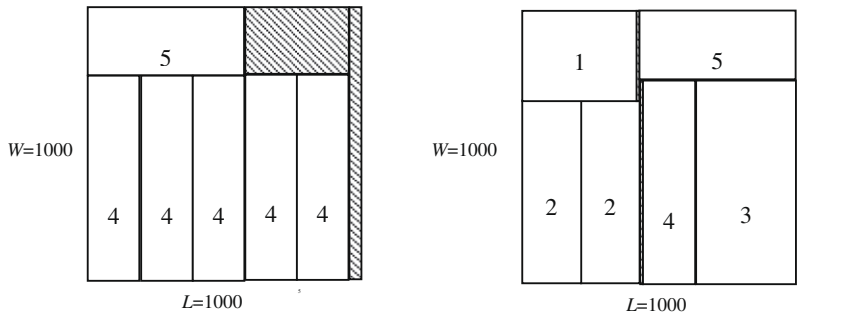


Fig. 12. An example to decompose an intricate structure to four simplest intricate cutting patterns (shaded area is the waste area).



(a) Cutting pattern obtained with HDP algorithm total area=858,616

(b) Cutting pattern obtained with DP algorithm total area=999,415

Fig. 13. An example to show the worst condition (shaded area is the waste).

the items correspondingly (note the area of each item is $\approx \frac{1}{7}$ that of the stock sheet, except for item 3, whose area is $\approx \frac{2}{7}$ that of the stock sheet).

4. Computational results

In this section, we now report the empirical results for the DPH algorithm. The purpose of our experiments is twofold. First, in Section 4.1, we evaluate the quality of the results from the DPH algorithm by comparing them to known optimal solutions. Second, in Section 4.2, we evaluate the influence of the input parameter (L, W, m, ρ) on the performance of the DPH algorithm by generating random instances according to different combinations of input parameters and comparing the results with those of the exact dynamic programming (DP) algorithm proposed by Beasley (1985a). For the latter experiments, both algorithms were coded in Visual

Table 1
Computational results: Benchmark problems (unweighted).

ID	Input parameters						HDP		Literature opt.			Dev. (%)
	(L, W)	(m, ρ)	l_{min}	w_{min}	l_{max}	w_{max}	$\bar{z}$	Time	Author	z	Time	
gcut1	(250,250)	(10,4)	66	86	167	184	Δ	0	Cintra et al. (2008)	56,460	0	0
gcut2	(250,250)	(20,4)	66	68	168	186	Δ	0	Cintra et al. (2008)	60,536	0	0
gcut3	(250,250)	(30,4)	63	71	176	186	Δ	0	Cintra et al. (2008)	61,036	0	0
gcut4	(250,250)	(50,5)	132	145	364	356	Δ	0.03	Cintra et al. (2008)	61,698	0	0
gcut5	(500,500)	(10,4)	132	134	355	372	Δ	0	Cintra et al. (2008)	246,000	0	0
gcut6	(500,500)	(20,4)	129	147	365	374	Δ	0	Cintra et al. (2008)	238,998	0	0
gcut7	(500,500)	(30,4)	131	127	362	374	Δ	0.01	Cintra et al. (2008)	242,567	0	0
gcut8	(500,500)	(50,4)	131	127	362	374	Δ	0.03	Cintra et al. (2008)	246,633	0	0
gcut9	(1000,1000)	(10,4)	292	341	673	688	Δ	0	Cintra et al. (2008)	971,100	0	0
gcut10	(1000,1000)	(20,4)	269	260	730	742	Δ	0	Cintra et al. (2008)	982,025	0	0
gcut11	(1000,1000)	(30,130)	20	81	1039	621	Δ	0.01	Cintra et al. (2008)	980,096	0	0
gcut12	(1000,1000)	(50,4)	254	269	746	723	Δ	0.07	Cintra et al. (2008)	979,986	0	0
gcut13	(3000,3000)	(32,26)	365	116	970	1890	Δ	1.5	Cintra et al. (2008)	8,997,780	23	0
U1	(4500,5000)	(10,20)	232	347	1110	1490	22,363,541	0.81	Cui and Zhang (2007)	22,370,130	–	0.03
U2	(5050,4070)	(10,9)	562	732	1791	1830	20,232,223	0.14	Cui and Zhang (2007)	20,232,224	–	0
U3	(7350,6579)	(20,15)	727	467	2828	2647	48,142,836	3.65	Cui and Zhang (2007)	48,142,840	–	0
M1	(100,156)	(10,7)	16	32	59	87	Δ	0	Young-Gun et al. (2003)	15,024	<0.01	0
M2	(253,294)	(10,7)	38	52	130	187	Δ	0	Young-Gun et al. (2003)	73,176	0.01	0
M3	(318,473)	(10,7)	76	75	204	238	Δ	0.01	Young-Gun et al. (2003)	142,817	0.01	0
M4	(501,556)	(10,7)	81	113	296	310	Δ	0	Young-Gun et al. (2003)	265,768	0.02	0
M5	(750,806)	(10,7)	121	163	445	449	Δ	0	Young-Gun et al. (2003)	577,882	0.02	0
UU1	(550,550)	(25,5)	119	101	319	320	Δ	0.01	Young-Gun et al. (2003)	242,919	0.02	0
UU2	(750,800)	(30,5)	153	174	466	501	Δ	0.03	Young-Gun et al. (2003)	595,288	0.05	0
UU3	(1100,1000)	(50,5)	227	222	670	644	Δ	0.03	Young-Gun et al. (2003)	1,072,764	0.09	0
UU4	(1000,1200)	(38,5)	216	245	635	724	Δ	0.09	Young-Gun et al. (2003)	1,179,050	0.15	0
UU5	(1450,1300)	(50,5)	299	262	940	841	Δ	0.21	Young-Gun et al. (2003)	1,868,999	0.17	0
UU6	(2050,1457)	(38,5)	424	342	1,325	939	Δ	0.12	Young-Gun et al. (2003)	2,950,760	0.18	0
UU7	(1465,2024)	(50,5)	301	411	926	1,280	Δ	0.35	Young-Gun et al. (2003)	2,930,654	0.37	0
UU8	(2000,2000)	(55,5)	406	466	1297	1280	Δ	0.42	Young-Gun et al. (2003)	3,959,352	0.32	0
UU9	(2500,2460)	(60,5)	551	502	1610	1597	Δ	0.68	Young-Gun et al. (2003)	6,100,692	0.41	0
UU10	(3500,3450)	(55,5)	712	701	2254	2226	Δ	1.21	Young-Gun et al. (2003)	11,955,852	1.05	0
UU11	(3500,3765)	(25,95)	37	108	1894	2009	13,153,234	2.09	Young-Gun et al. (2003)	13,157,811	802.71	0.03
APT10	(2097,1713)	(51,20)	105	91	832	676	3,589,585	1.43	Young-Gun et al. (2003)	3,589,703	275.20	0
APT11	(2600,1612)	(58,130)	20	81	1039	621	Δ	2.12	Young-Gun et al. (2003)	4,188,937	207.29	0
APT12	(2662,1941)	(35,20)	141	99	1061	743	5,155,344	1.07	Young-Gun et al. (2003)	5,156,065	151.32	0.01
APT13	(1674,2090)	(54,20)	101	105	660	816	Δ	1.5	Young-Gun et al. (2003)	3,498,302	7.03	0
APT14	(2090,2138)	(42,19)	110	144	817	849	Δ	1.31	Young-Gun et al. (2003)	4,463,550	44.82	0
APT15	(2222,2726)	(49,19)	141	145	871	1073	Δ	2.20	Young-Gun et al. (2003)	6,047,188	2417.04	0
APT16	(2899,2614)	(53,18)	190	149	1,117	1001	7,561,200	3.06	Young-Gun et al. (2003)	7,566,719	3915.61	0.07
APT17	(2313,1962)	(59,21)	129	98	915	774	4,535,262	2.11	Young-Gun et al. (2003)	4,535,302	439.14	0
APT18	(2775,2105)	(31,20)	156	108	1073	820	5,820,472	1.12	Young-Gun et al. (2003)	5,825,956	213.32	0.09
APT19	(2284,2994)	(35,20)	130	155	909	1192	6,825,504	1.62	Young-Gun et al. (2003)	6,826,674	110.06	0.02

–, Results not published; Δ, optimal solution value.

Studio C++ and were run on a PC using an Intel(R) Core(TM)2 2.83 GHz processor using 3.23 GB of RAM under Microsoft Windows XP.

4.1. Test on benchmark problems

Our computational study was conducted on 66 problem instances of different sizes, for (weighted) unconstrained cutting stock. All test problems have been used in various other papers (Beasley, 1985a; Cintra et al., 2008; Hifi, 1997; Morabito et al., 1992; Fayard et al., 1998; Alvarez-Valdes et al., 2002; Cui, 2007; Cui and Zhang, 2007; Young-Gun et al., 2003). They can be downloaded from

http://www.laria.u-picardie.fr/hifi/OR-Benchmark/2Dcutting/.
http://www.ime.usp.br/glauber/gcut/.

These instances have been divided into two different problem sets.

The first problem set (unweighted unconstrained case) includes 42 problem instances. Instances gcut1–gcut13 were taken from Beasley (1985a), which were tested later on by Cintra et al. (2008). U1–U3 were from Hifi (1997). M1–M5 were from Morabito et al. (1992). Both DH (Morabito et al., 1992) and DH/KD (Hifi, 1997) algorithms have been used to solve these problems. UU1–UU11 represent 11 large randomly generated problem instances generated by Fayard et al. (1998). APT10–APT19 were randomly generated by Alvarez-Valdes et al. (2002).

The second problem set (weighted unconstrained case) has 24 problem instances. W1–W3 were from Hifi (1997). UW1–UW11 were from Fayard et al. (1998) and APT20-APT29 were from Alvarez-Valdes et al. (2002).

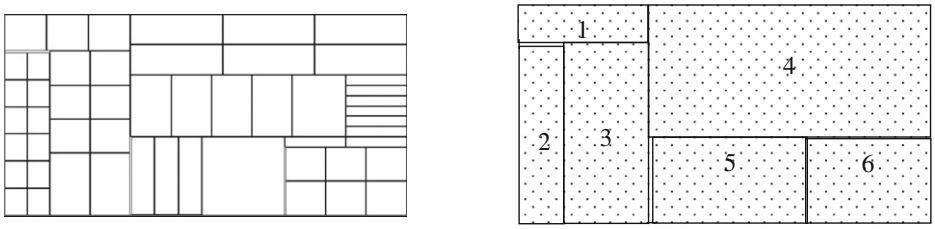
The results of our experiments with the 66 problem instances are detailed in Tables 1 and 2. For each instance we give the input parameters $(L, W, m, \rho, l_{min}, w_{min}, l_{max}, w_{max})$. In Table 2, we also give the extra input parameters p_{min} and p_{max} for weighted instances (where p_{min} and p_{max} represent the minimum and maximum profits of the small items, respectively). Next in both tables, we give the solution value produced by the DPH algorithm together with the CPU time. Finally, we give details of the optimal solutions for each instance: The authors who report the information, the optimal solution value and reported CPU time. The deviation of the DPH algorithm solution from the optimal is then given.

In Table 1, the value columns give the total area of the small items packed onto the large rectangular stock sheet. In Table 2, the value columns give the total profit of the small items packed onto the large rectangular stock sheet. In both tables, we use 'Δ' to represent the optimal value. The column time shows the CPU

Table 2
Computational results: Benchmark problems (weighted).

ID	Input parameters								HDP		Literature opt.			Dev. (%)
	(L, W)	(m, ρ)	l_{min}	w_{min}	l_{max}	w_{max}	ρ_{max}	ρ_{max}	$\bar{z}$	Time	Author	z	Time	
W1	(5000,5000)	(20,12)	437	562	2110	3721	671	3876	Δ	2.01	Cui and Zhang (2007)	162,867	–	0
W2	(3427,2769)	(20,9)	451	316	2007	2532	398	4351	Δ	0.21	Cui and Zhang (2007)	35,159	–	0
W3	(7500,7381)	(40,13)	5845	845	5751	3787	845	3608	Δ	8.90	Cui and Zhang (2007)	234,108	–	0
UW1	(500,500)	(25,5)	111	117	307	286	155	738	Δ	0.01	Young-Gun et al. (2003)	6036	0.01	0
UW2	(560,750)	(35,5)	113	156	360	478	165	732	Δ	0.03	Young-Gun et al. (2003)	8468	0.04	0
UW3	(700,650)	(35,5)	141	131	449	419	154	696	6226	0.03	Young-Gun et al. (2003)	6302	0.02	0.01
UW4	(1245,1015)	(45,5)	260	204	800	633	168	725	Δ	0.12	Young-Gun et al. (2003)	8326	0.01	0
UW5	(1100,1450)	(35,5)	229	309	704	937	166	717	Δ	0.06	Young-Gun et al. (2003)	7780	0.04	0
UW6	(1750,1542)	(47,5)	358	325	1136	982	19	748	Δ	0.21	Young-Gun et al. (2003)	6615	0.04	0
UW7	(2250,1875)	(50,5)	486	377	1432	1163	182	737	Δ	0.32	Young-Gun et al. (2003)	10,464	0.09	0
UW8	(2645,2763)	(55,5)	539	566	1708	1769	161	740	Δ	0.70	Young-Gun et al. (2003)	7692	0.14	0
UW9	(3000,3250)	(45,5)	627	709	1942	2095	151	708	Δ	0.51	Young-Gun et al. (2003)	7038	0.23	0
UW10	(3500,3650)	(60,5)	706	751	2245	2354	178	747	Δ	1.40	Young-Gun et al. (2003)	7507	0.30	0
UW11	(555,632)	(25,7)	96	983	354	335	110	700	Δ	0.03	Young-Gun et al. (2003)	15,747	0.60	0
APT20	(2840,2858)	(45,18)	167	167	1133	1143	25,400	492,229	Δ	2.40	Young-Gun et al. (2003)	5,545,818	1134.04	0
APT21	(2866,1784)	(48,18)	240	100	1140	701	10,945	470,884	Δ	1.43	Young-Gun et al. (2003)	3,484,406	4.79	0
APT22	(2711,2110)	(52,19)	167	112	1083	826	9242	463,175	4,140,487	1.95	Young-Gun et al. (2003)	4,145,317	260.78	0
APT23	(1856,2636)	(53,20)	96	147	733	1053	10,302	466,604	3,545,888	1.61	Young-Gun et al. (2003)	3,546,535	19.56	0
APT24	(2070,2729)	(21,44)	131	136	822	1054	18,646	456,930	Δ	1.70	Young-Gun et al. (2003)	3,948,037	286.42	0
APT25	(2885,1715)	(19,36)	159	93	1142	679	11,013	270,736	Δ	1.03	Young-Gun et al. (2003)	3,507,615	71.45	0
APT26	(2359,1656)	(48,21)	129	82	938	616	9701	311,393	Δ	1.12	Young-Gun et al. (2003)	2,683,689	2.75	0
APT27	(1793,1875)	(27,20)	102	98	709	733	17,492	269,201	2,435,046	0.95	Young-Gun et al. (2003)	2,438,174	1.17	0
APT28	(2020,2796)	(54,19)	123	148	806	1098	7963	576,621	Δ	2.15	Young-Gun et al. (2003)	4,065,011	23.32	0
APT29	(1839,2829)	(59,19)	99	149	733	1108	10,423	429,339	Δ	2.07	Young-Gun et al. (2003)	3,652,858	11.86	0

–, Results not published; Δ, Optimal solution value.



(a) Optimal cutting pattern

(b) Intricate cutting pattern obtained by merging adjacent items at the corresponding position of the optimal cutting pattern a)

Fig. 14. Optimal pattern not obtained by HDP algorithm (Instance ATP18).

time in seconds. Since for some instances, the running time is too small to be accurately obtained we represent the running times smaller than 0.01 as 0.

Note that for the computer running time of the optimal solutions, we treated different instances in different ways. For the instances gcut1–gcut13, both Beasley (1985a) and Cintra et al. (2008) solved them to optimality. However, the work from Cintra et al. (2008) was reported to have slightly less computational complexity than Beasley's (1985a). Thus we use Cintra et al.'s (2008) computer running time as a comparison. Cui and Zhang (2007) have given the optimal solutions to the cases U1–U3, W1–W3, UU1–UU11 and UW1–UW11, however, they did not give the exact CPU times. We thus leave the time column empty for instances U1–U3 and W1–W3. For instances UU1–UU11, UW1–UW11, we found that Young-Gun et al. (2003) reported the exact computer running time for solving the problems to optimality. Later on, in Cui (2007), we found the computer running time of APT10–APT29 by Young-Gun et al. (2003), which were reported to be optimal. Thus we took Young-Gun et al.'s (2003) computer running time as a comparison for instances UU1–UU11, UW1–UW11 and APT10–APT29.

The computer setting of Cintra et al. (2008) is a personal computer equipped with Intel Pentium IV, 1.8 GHz processor and 512 MB RAM. The computer setting of Young-Gun et al. (2003) is a Pentium IV 1.6 GHz PC with 256 MB of memory.

The deviation is calculated as $\frac{z-\bar{z}}{z}$ for a literature instance, where z represents the literature optimal solution value and $\bar{z}$ represents the solution value given by DPH algorithm.

Examining Tables 1 and 2, we see that our DPH algorithm solves most of the benchmark problems very quickly. Most of the instances were solved in less than 1 second. It took about 8 seconds to solve instance 'W3' because this instance is a large-scale instance with stock sheet length = 7500, width = 7381, $m = 40$ and $\rho = 13$.

Compared with the optimal results, the DPH algorithm generates solution values with maximum 0.09% deviation from the optimal solution values and 0.006% on average for the unweighted instances. For the weighted instances, there is only one instance which has 0.01% deviation from the optimal solution values. All the other instances are solved to optimality or with less than 0.01% deviation from the optimal solution values.

The sum of the running time of gcut1–gcut13 is 23 seconds from the work of Cintra et al. (2008) and the sum of the running time of all the other instances except for UU1–UU3 and W1–W3 is 10,404.07 seconds from the work of Young-Gun et al. (2003). In contrast, the running time of our DPH algorithm are 1.65 seconds and 42.62 seconds, correspondingly. However, we used different computers to run these instances. To be fair, we found a laptop with Intel Pentium 4, 1.8 GHz, 256 MB RAM under Windows XP 2002 which is comparable to the computers of Cintra et al. (2008) and Young-Gun et al. (2003). We designed a programme to run on both our current computer and the old laptop. It took 3 seconds and 15 seconds on each computer, respectively. So we can say that our current computer is five time faster than computers of Cintra et al. (2008) and Young-Gun et al. (2003) approximately. Taking the differences of the computers into consideration, we see that our DPH algorithm is approximately three times faster than that of the work from Cintra et al. (2008) and 50 times faster than that of the work from Young-Gun et al. (2003). The possible reason that the time improvement of our DPH algorithm is less significant compared to Cintra et al.'s (2008) is that the instances solved by Cintra et al. (2008) are mainly small and medium sized instances. It is noticeable that the time improvement of our DPH algorithm is more significant as the size of the instances increases.

An example of a problem (instances ATP 18) where the optimal cutting pattern is not obtained by the DPH algorithm is given in Fig. 14. We see that by merging some small items, the optimal cutting pattern is an intricate cutting pattern. This is the reason that the DPH algorithm cannot identify the optimal solution here. Similar reasons hold for all the other instances that the DPH algorithm did not find the optimal solution. Such features support Theorem 3.

4.2. Test on randomly generated problems

From the computational complexity analysis of DPH, we know that there are three key factors that affect the running time of the algorithm: $|\overline{\mathbf{L}}|$, $|\overline{\mathbf{W}}|$ and m, where from Theorem 1, $|\overline{\mathbf{L}}|$ and $|\overline{\mathbf{W}}|$ are related to the parameter ρ. We therefore extensively tested our approach using randomly generated data sets under different combinations of parameters m in $\{5,10,20\}$ and ρ in $\{10,20\}$. For each combination of m and ρ, 100 instances were generated randomly using a uniform distribution on some ranges given below. For a given length $L = 200$ and width $W = 100$ of stock rectangular sheet, the length l_i and width w_i of item i were in $\left[\left\lceil\frac{L}{\rho}\right\rceil, \left\lceil\frac{L}{2}\right\rceil\right]$ and $\left[\left\lceil\frac{W}{\rho}\right\rceil, \left\lceil\frac{W}{\rho}\right\rceil\right]$.

The results of our experiments are given in Tables 3 and 4 where the instances are denoted as *small sized instances*. Under the same conditions, we also tested the algorithm with stock sheet size $L = 2000$ and width $W = 1000$. The experimental results are given in Tables 5 and 6 (instances are denoted as *medium sized instances*). To test the efficiency of DPH algorithm for solving large scaled instances, we enlarge m from $\{5,10,20\}$ to $\{50,100,200\}$ and keep ρ in $\{10,20\}$. The experimental results are given in Tables 7 and 8, and these instances are denoted as *large sized instances*. Note

Table 3
The performance of the HDP algorithm ($L = 200, W = 100$).

	Unweighted instances							Weighted instances						
(m,ρ):	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.
Opt. Sol.	98	95	89	95	95	97	94.83	100	100	100	99	99	100	99.67
Av. Appr. ratio	0.9966	0.9969	0.9863	0.9979	0.9992	0.9994	0.9960	1	1	1	0.9978	0.9744	1	0.9954
Min. Appr. ratio	0.9949	0.9959	0.9958	0.9950.	0.9985	0.9992	–	1	1	1	0.9978	0.9744	1	–

Table 4
The comparison of running time of the HDP algorithm with DP ($L = 200, W = 100$).

	Unweighted instances							Weighted instances						
(m,ρ):	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.
HDP	0.0003	0.0005	0.001	0.0014	0.0027	0.0031	0.0015	0.0039	0.0039	0.0058	0.0059	0.0097	0.0098	0.0065
DP	0.0211	0.0375	0.0675	0.0880	0.0986	0.1202	0.0721	0.0228	0.0373	0.0692	0.0927	0.1024	0.1263	0.0751
Time ratio	0.0142	0.0133	0.0163	0.0159	0.0274	0.0258	0.0208	0.1711	0.1046	0.0838	0.0636	0.0947	0.0776	0.0992

Table 5
The performance of the HDP algorithm ($L = 2000, W = 1000$).

	Unweighted instances							Weighted instances						
(m,ρ):	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.
Opt. Sol.	100	96	91	76	88	69	86.67	100	100	99	100	99	99	99.50
Av. Appr. ratio	1	0.9974	0.9975	0.9979	0.9985	0.9990	0.9984	1	1	0.9982	1	0.9960	0.9994	0.9989
Min. Appr. ratio	1	0.9962	0.9909	0.9923	0.9951	0.9975	–	1	1	0.9982	1	0.9960	0.9994	–

Table 6
The comparison of running time of the HDP algorithm with DP.

	Unweighted instances							Weighted instances						
(m,ρ):	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.	(5,10)	(5,20)	(10,10)	(10,20)	(20,10)	(20,20)	Av.
HDP	0.0008	0.0020	0.0134	0.0339	0.1116	0.1806	0.0570	0.0013	0.0025	0.0144	0.0298	0.0952	0.1434	0.0478
DP	0.4803	1.8281	10.1558	27.8511	50.6750	83.7414	29.1219	0.6055	1.5177	9.6116	25.6802	47.8948	76.0928	26.9004
Time ratio	0.0017	0.0011	0.0013	0.0012	0.0022	0.0022	0.0020	0.0021	0.0016	0.0015	0.0012	0.0020	0.0019	0.0017

Table 7
The performance of the HDP algorithm ($L = 2000, W = 1000$).

	Unweighted instances							Weighted instances						
(m,ρ):	(50,10)	(50,20)	(100,10)	(100,20)	(200,10)	(200,20)	Av.	(50,10)	(50,20)	(100,10)	(100,20)	(200,10)	(200,20)	Av.
Opt. Sol.	80	67	79	62	86	93	77.83	100	98	97	91	97	99	97
Av. Appr. ratio	0.9995	0.9996	0.9997	0.9999	0.9999	0.9999	0.9997	1	0.9992	0.9985	0.9987	0.9990	0.9990	0.9991
Min. Appr. ratio	0.9985	0.9987	0.9992	0.9994	0.9997	0.9999	–	1	0.9990	0.9959	0.9973	0.9987	0.9990	–

Table 8
The comparison of running time of the HDP algorithm with DP ($L = 2000, W = 1000$).

	Unweighted instances							Weighted instances						
(m,ρ):	(50,10)	(50,20)	(100,10)	(100,20)	(200,10)	(200,20)	Av.	(50,10)	(50,20)	(100,10)	(100,20)	(200,10)	(200,20)	Av.
HDP	0.4870	0.6402	1.3470	1.6477	3.4550	4.0944	1.9452	0.4203	0.5325	1.1400	1.3870	2.8711	3.3953	1.6244
DP	92.8194	125.0173	114.1669	143.0995	120.1378	148.5889	123.9716	90.6753	120.1931	110.5736	138.9869	122.6917	149.1772	122.0496
Time ratio	0.0052	0.0051	0.0118	0.0115	0.0288	0.0276	0.0157	0.0046	0.004	0.0103	0.0100	0.0234	0.0228	0.0133

that in all experiments, the required quantities of the items are not limited for the problem studied is an unconstrained 2D cutting problem. For unweighted instances, the profit $p_i = l_i \cdot w_i$, whereas for weighted instances, p_i is randomly generated using a uniform distribution in $[0, L \cdot W]$.

As a comparison to our results, we coded the DP algorithm proposed by Beasley (1985a) and ran their exact algorithm on our computer to solve the problems exactly. In Tables 3, 5 and 7, we show the performance of the DPH algorithm by referring to the number of optimal solutions (Opt. Sol.), the average approximation ratio (Av. Appr. ratio) and the minimum ratio (Min. Appr. ratio). The average approximation ratio is calculated without considering the cases where the solution is optimal. In Tables 4, 6 and 8, we show the run time comparison with average run time of the 100 instances using DPH algorithm (DPH) and DP algorithm (DP), respectively, and the time ratio of the average run time of DPH algorithm over that of DP algorithm (Time Ratio). Since there are no outliers of the run times for these instances, we do not report the maximum run times here.

Considering the unweighted instances, from Table 3, we see that for the stock sheets with length $L = 200$ and width $W = 100$, 94.83% of the instances were optimally solved, whereas the totality of the instances was solved very close to the optimum. The experimental minimum approximation ratio was found to be 0.9949 and the average was 0.9960. Similar results can also be witnessed in Tables 5 and 7 for the stock sheets with length $L = 2000$ and width $W = 1000$. It is interesting to note that the percentage of the optimal solutions found with the DPH algorithm decreases as the size of the problem is increased. From Tables 3–5 and 7, this percentage decreases from 94.83% to 86.67% and then to 77.83%. In contrast, however, the average approximation ratio increases as the size of the problem is increased, from 0.9960 to 0.9984, then further to 0.9997. The probable reason for this is that with more types of items, more cutting patterns could be formed. Thus the probability that the optimal cutting pattern is an intricate cutting pattern is higher. Also it is more likely that a cutting pattern of twice cutting type generates a closer solution value to that of the optimal solution which could not be found by DPH algorithm.

Considering Tables 4, 6 and 8, we see that the running time ratios of our method to those of the traditional DP method on average are similar in Tables 4 and 8, whereas in Table 6, the ratio is 0.0020, which is 7–10 times lower than those in Tables 4 and 8. That means, the instances in Table 6 can be solved must faster with DPH algorithm than with DP algorithm. The possible reason is that in Table 6, the parameters m and ρ are relatively small against the length L and width W of the stock sheet. Thus parameters $|\overline{\mathbf{L}}|$ and $|\overline{\mathbf{W}}|$ are relatively large according to the analysis in Theorem 1. Thus the running time of the DP algorithm is much longer than that of the DPH algorithm, which can be verified theoretically by comparing their computational complexities. Besides, we find that the larger the parameters m and ρ, the longer the running times. This phenomenon is verified by Theorem 1. That is, the larger the

parameters m and ρ, the larger the parameters $|\overline{\mathbf{L}}|$ or $|\overline{\mathbf{W}}|$ will be, which adds up to the computing time. So this algorithm works faster on solving instances with smaller m and ρ.

The analyses of the weighted instances are similar to those of the unweighted instances. However, it is notable that both the percentages of the optimal solutions and the approximation ratio of the weighted instances are much higher than those of the unweighted instances. From small instances to large instances, the average percentages of the optimal solutions are 99.67%, 99.50% and 97% and the average approximation ratios are 0.9954, 0.9989 and 0.9991, respectively. The possible reason is that for weighted instances, it is possible that a small sized item carries high profit which ends up with the possibility that the optimal solutions consists more small sized items than bigger ones. With small items, more cutting patterns could be formed. Thus the probability that the optimal cutting pattern is an intricate cutting pattern is higher. Also it is more likely that a cutting pattern of twice cutting type generates closer solution value to that of the optimal solution which could not be found by the DPH algorithm.

5. Conclusion

In this paper, a heuristic dynamic-programming recursive algorithm has been presented for approximately solving the unconstrained 2D cutting stock problem. We have analyzed the parameters which can influence the computer running time and we have also proved that in the worst case this algorithm gives the approximation ratio $\frac{6}{7}$. Computational results for a large number of benchmark and randomly generated problems were carried out and we see that although the computational complexity is lowered significantly with this algorithm (compared to an exact algorithm), a high percentage of optimal solutions are still produced. In the rare case that all parameters L, W, m and ρ are large, we envisage that the computer running time could be controlled within an acceptable scope by merging extremely small items in advance in order to reduce the size of $|\overline{\mathbf{L}}|$ and $|\overline{\mathbf{W}}|$.

References

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

Alvarez-Valdes, R., Parajon, A., Tamarit, J.M., 2007. A tabu search algorithm for a two-dimensional non-guillotine cutting problem. European Journal of Operational Research 183 (3), 1167–1182.

Baldacci, R., Boschetti, M.A., 2007. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. European Journal of Operational Research 183 (3), 1136–1149.

Beasley, J.E., 1985a. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 36 (4), 297–306.

Beasley, J.E., 1985b. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33, 49–64.

Belov, G., Scheithauer, G., 2006. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. European Journal of Operational Research 171, 85–106.

Christofides, N., Whitlock, C., 1977. An algorithm for two-dimensional cutting problems. Operations Research 25, 30–44.

Christofides, N., Hadjiconstantinou, E., 1995. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. European Journal of Operational Research 83, 21–38.

Cintra, G., Miyazawa, F.K., Wakabayashi, Y., Xavier, E.C., 2008. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. European Journal of Operational Research 191 (1), 61–85.

Cui, Y.D., 2007. Simple block patterns for the two-dimensional cutting problem. Mathematical and Computer Modeling 45 (7-8), 943–953.

Cui, Y.D., Zhang, X.Q., 2007. Two-stage general block patterns for the two-dimensional cutting problem. Computers and Operations Research 34, 2882–2893.

Cui, Y., 2008. Heuristic and exact algorithms for generating homogenous constrained three-staged cutting patterns. Computers and Operations Research 35, 212–225.

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

Dowsland, K.A., Dowsland, W.B., 1992. Packing problems. European Journal of Operational Research 56, 2–14.

Dyckhoff, H., 1990. A typology of cutting and packing problems. European Journal of Operational Research 44, 145–159.

Fayard, D., Zissimopoulos, V., 1995. An approximation algorithm for solving unconstrained two-dimensional Knapsack problems. European Journal of Operational Research 84, 618–632.

Fayard, D., Hifi, M., Zissimopoulos, V., 1998. An efficient approach for large-scale two-dimensional guillotine cutting stock problems. The Journal of the Operational Research Society 49 (12), 1270–1277.

Garey, M., Johnson, D., 1979. Computer and intractability. A Guide to the Theory of NP-Completeness. Freeman, New York.

Gilmore, P.C., Gomory, R.E., 1966. The theory and computation of Knapsack functions. Operations Research 14, 1045–1074.

Gongalves, J.F., 2007. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. European Journal of Operational Research 183 (3), 1212–1229.

Haessler, R., Sweeney, P., 1991. Cutting stock problem and solution procedures. European Journal of Operational Research 54, 141–150.

Herz, J.C., 1972. A recursive computing procedure for two-dimensional stock cutting. IBM Journal of Research and Development 16, 462–469.

Hifi, M., 1997. The DH/KD algorithm: A hybrid approach for unconstrained two-dimensional cutting problems. European Journal of Operational Research 97, 41–52.

Hifi, M., 2001. Exact algorithms for large-scale unconstrained two and three staged cutting problems. Computational Optimization and Applications 18, 63–88.

Hifi, M., M'Hallah, R., 2005. An exact algorithm for constrained two-dimensional two-staged cutting problems. Operations Research 53, 140–150.

Hifi, M., M'Hallah, R., 2006. Strip generation algorithms for constrained two-dimensional two-staged cutting problems. European Journal of Operational Research 172, 515–527.

Hinxman, A.I., 1976. Problem reduction and the two-dimensional trim-loss problem. In: Artificial Intelligence and Simulation Summer Conference, Univ. of Edinburgh, pp. 158–165.

Hohn, F.E., 1941. The number of terms in a polynomial. The American Mathematical Monthly 48 (10), 686–687.

Lodi, A., Martello, S., Monaci, M., 2002. The two-dimensional packing problems: A survey. European Journal of Operational Research 141, 3–13.

Lodi, A., Monaci, M., 2003. Integer programming models for 2-staged two-dimensional knapsack problems. Mathematical Programming 94, 257–278.

Morabito, R.N., Arenales, M.N., Arcaro, V.F., 1992. An AND–OR-graph approach for two-dimensional cutting problems. European Journal of Operational Research 58, 263–271.

Morabito, R.N., Arenales, M.N., 1996. Staged and constrained two-dimensional guillotine cutting problems: An AND/OR-graph approach. European Journal of Operational Research 94, 548–560.

Scheithauer, G., Sommerweiss, U., 1998. 4-Block heuristic for the rectangle packing problem. European Journal of Operational Research 108, 509–526.

Sweeney, P., Paternoster, E., 1992. Cutting and packing problems: A categorized, application-oriented research bibliography. Journal of the Operational Research Society 43, 691–706.

Vasko, F.J., 1989. A computational improvement to Wang's two-dimensional cutting stock algorithm. Computers and Industrial Engineering 16, 109–115.

Viswanathan, K.V., Bagchi, A., 1988. An exact best-first search procedure for the constrained rectangular guillotine knapsack problem. Proceedings of the American Association for Artificial Intelligence, 145–149.

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

Wascher, G., Haußner, H., Schumann, H., 2006. An improved typology of cutting and packing problems. European Journal of Operational Research 183 (3), 1109–1130.

Young-Gun, G., Seong, Y.J., Kang, M.K., 2003. A best-first branch and bound algorithm for unconstrained two-dimensional cutting problems. Operational Research Letters 31, 301–307.