Comput Optim Appl (2008) 41: 337–348
DOI 10.1007/s10589-007-9103-3

A recursive algorithm for constrained two-dimensional cutting problems

Yuanyan Chen

Received: 30 July 2006 / Revised: 18 January 2007 / Published online: 26 October 2007
© Springer Science+Business Media, LLC 2007

Abstract This paper presents a recursive algorithm for constrained two-dimensional guillotine cutting problems of rectangular items. The algorithm divides a stock plate into a sequence of small rectangular blocks. For the current block considered, it selects an item, puts it at the left-bottom corner of the block, and determines the direction of the dividing cut that divides the unoccupied region of the block into two smaller blocks for further consideration. The dividing cut is either along the upper edge or along the right edge of the selected item. The upper bound obtained from the unconstrained solution is used to shorten the searching space. The computational results on benchmark problems indicate that the algorithm can improve the solutions, and is faster than other algorithms.

Keywords Recursive algorithm · Constrained two-dimensional cutting · Guillotine cuts

1 Introduction

The Two-Dimensional Cutting (TDC) problem consists of cutting a given set of small rectangular items from a large stock rectangular plate, so as to maximize the total value of the items cut. This problem is formally referred to as the two-dimensional, rectangular, Single Large Object Placement Problem (SLOPP) [1]. It appears in several industrial areas, such as the cutting of wood plates to make furniture, the cutting of cardboard to make boxes, and the cutting of glass plates and metal plates to make related products.

This paper deals with one of the most interesting TDC problems, the Constrained Two-Dimensional Cutting (CTDC) problem. The problem is characterized as follows:

Y. Chen (✉)
Department of Computer Science, Guangxi Normal University, Guilin 541004, China
e-mail: ychen@mailbox.gxnu.edu.cn

The dimension of the stock plate R is $L \times W$ (length L and width W); the ith item has dimension $l_i \times w_i$, value v_i and an upper demand bound b_i, $i = 1, \ldots, m$. The task is to cut the plate into x_i pieces of item type i, such that $0 \leq x_i \leq b_i$, $i = 1, \ldots, m$, and the total utility $\sum_{i=1}^{m} v_i x_i$ is maximized. In this paper, it is assumed that:

(1) All items have fixed orientation, that is, pieces of dimensions (a, b) and (b, a) are not the same;
(2) All applied cuts are of *guillotine* type, i.e., cuts that start from one edge and run parallel to the other two edges;
(3) Parameters L, W, l_i, w_i, and b_i, $i = 1, \ldots, m$, are nonnegative integers.

We say that the n-dimensional vector $(x_1, \ldots, x_n)$ of integer and nonnegative numbers corresponds to a *cutting pattern*, if it is possible to produce x_i pieces of type i, $i = 1, \ldots, m$, in the large stock plate R without overlapping. Some industrial cutting processes limit the way of producing a *guillotine* cutting pattern. At first stage the cuts are performed parallel to one side of the plate and then, at the next stage, orthogonal to the preceding cuts, and so on. If there is a limit on the number of stages, say k, the cut is called a *k-staged cuts*; otherwise, it is called *non-staged* (note that a non-staged cuts is equivalent to define k large enough). If the number of stages tends to be a positive infinitude, the cutting pattern is called *general guillotine pattern*. Therefore, the *guillotine* patterns considered in this paper are the general ones.

For the CTDC problem, it can be distinguished:

(1) The unweighted CTDC: The value v_i of each item i is equal to its area $s_i = l_i w_i$. The objective is to maximize the total occupied area, which is equivalent to minimizing the waste.
(2) The weighted CTDC: The values of the items are independent from their areas. The objective is to maximize the total value of the items cut.

For CTDC problem, there are two approaches: the top–down approach [2–4] and the bottom-up one [5–11, 17]. The top–down approach uses a tree-search procedure. All possible cuts that can be made on the stock sheet are enumerated by means of a tree in which branching corresponds to guillotine cuts and nodes represent subrectangles obtained through the guillotine cut. The bottom–up approach is based on the observation that any pattern satisfying the guillotine constraint can be obtained through horizontal and vertical builds of rectangles. All possible combinations of smaller rectangles are generated to obtain larger rectangles until no more guillotine patterns can be obtained. Both approaches can use upper and lower bounds to discard some non-promising branches.

Many authors have investigated the CTDC problem. For small instances some exact algorithms have been proposed by applying a general tree search based upon a depth-first search method [2, 7], and also by the use of a branch-and-bound procedure based upon a best-first search method [8–10].

For problems with relatively larger scale, many heuristic algorithms have been developed. Morabito and Arenales [12] presented an algorithm based on depth-first search and hill-climbing strategies. Other researchers have also developed some heuristic approaches to the CTDC problem [13, 14]. Recently, Alvarez-Valdes

et al. [15] developed several heuristic algorithms for guillotine (un)constrained two-dimensional cutting problems, especially the tabu search algorithm obtained high quality results in moderate computing times. More recently, Hifi [16] presented a hybrid algorithm for constrained two-dimensional cutting problems, in which a depth-first search using hill-climbing strategies and dynamic programming techniques are combined. The algorithm can produce good solutions for the large problems instances. Cui [17, 18] presented two exact algorithms for the CTDC: one is an exact algorithm for generating homogenous T-shape patterns to simplify the cutting process [17]; the other is an exact algorithm for generating homogenous three-staged cutting patterns based on branch-and-bound procedure combined with dynamic programming techniques [18]. These two algorithms are exact if only patterns with specified features (such as homogenous T-shape or three-staged patterns) were concerned. They can be also used as heuristics for generating general CTDC patterns.

If b_i $(i = 1, \ldots, m)$ is set to be infinite, the CTDC problem becomes the Unconstrained Two-Dimensional Cutting (UTDC) problem. Algorithms for UTDC problems have been reported in the literature. For example, the general guillotine patterns were discussed in [19, 20]; staged patterns in [21, 22]; T-shape patterns in [23]; and simple block patterns in [24]. The solution value of the related UTDC problem can be used as an upper bound, if branch-and-bound techniques are used in solving the CTDC problem.

This paper presents a recursive heuristic algorithm for the CTDC. The computational results indicate that the algorithm can produce good solutions in short computing time for problems with various scales. The next section describes the scheme of the recursive algorithm. Section 3 tests the algorithm with benchmark problems, and compares the computational results with those obtained from other algorithms. Section 4 terminates the paper with conclusions.

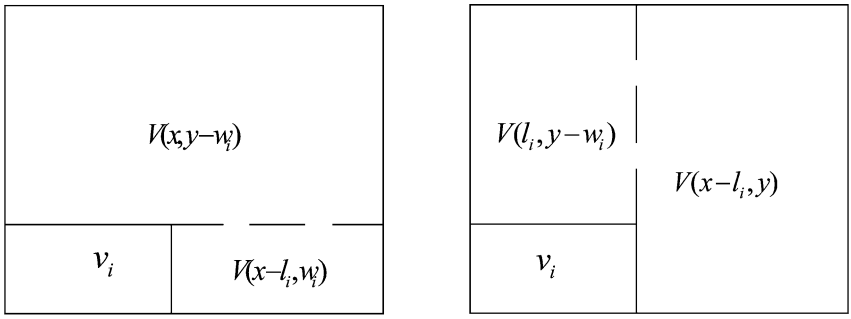
2 The approach

2.1 The scheme of the algorithm

Assume that $L_{\min}$ and $W_{\min}$ are respectively the minimum length and width of all items, $V(x, y)$ is the maximum value of the items included in the current block $x \times y$, and P is the father (or superior) pattern of block $x \times y$. Assume that $\mathbf{B} = \{B_1, \ldots, B_m\}$, in which B_i $(i = 1, \ldots, m)$ is the number of item type i that has been arranged in P. Place item $l_i \times w_i$ at the left-bottom corner of the block. Divide the unoccupied region of the block into two smaller blocks either horizontally along the upper edge (Fig. 1a) or vertically along the right edge (Fig. 1b) of the item.

Assume that $V_{xi}(x, y)$ denotes the value of block $x \times y$ when the dividing cut is horizontal, and $V_{yi}(x, y)$ denotes that when the dividing cut is vertical. Then

$$V_{xi}(x, y) = v_i + V(x - l_i, w_i) + V(x, y - w_i),$$
$$V_{yi}(x, y) = v_i + V(l_i, y - w_i) + V(x - l_i, y).$$



(a) Dividing horizontally (b) Dividing vertically

Fig. 1 Two methods to divide the unoccupied region of the block

Considering that we can select the item from m types to place at the left-bottom corner of the block, we have the following recursive formulation:

$$
\begin{aligned}
&V(x, y) = 0, \quad \text{if } x < L_{\min} \text{ or } y < W_{\min};\\
&V_{xi}(x, y) = V_{yi}(x, y) = 0, \quad \text{if } x < l_i \text{ or } y < w_i \text{ or } b_i - B_i < 1;\\
&V(x, y) = \max_i\{V_{xi}(x, y), V_{yi}(x, y)\}, \quad \text{if } x \geq l_i \text{ and } y \geq w_i \text{ and } B_i < b_i;\\
&\quad 0 \leq x \leq L,\ 0 \leq y \leq W,\ i = 1, \ldots, m.
\end{aligned}
$$

Although the recursive algorithm above is feasible, the computation time may be too long to afford, especially for large-scale problems. So we must make use of some heuristic rules to produce a feasible solution at some reasonable time limit. The following rules are helpful for shortening the computation time.

(1) Using upper and lower bounds to improve the performance of the recursive function.
(2) Arranging the items in descending order of their values. This is helpful for obtaining tighter lower bound early, for items with higher unit values are considered early. Tighter lower bound is useful for pruning the searching tree efficiently.
(3) Constraining the maximum computation time for the current trial block. The searching process will be terminated as soon as possible if the computation time t for the current trial block has reached a given running time $t_{\max}$.

2.2 The recursive procedure for the unconstrained solution

Cui [24] presented a recursive algorithm for the UTDC problem. The algorithm can be used to estimate the upper bound, because it generates cutting patterns exactly in the same way as that in this paper, except that the demanded upper bound is not considered. Assume that recursive function URF(x, y) returns the maximum value of the unconstrained solution to block $x \times y$. Let U_{xy} be the maximum value of the unconstrained solution to the block, initially it is set to -1. The steps of recursive function URF(x, y) are:

Step 1. Return U_{xy} if $U_{xy} \geq 0$; Return 0 if $x < L_{\min}$ or $y < W_{\min}$; otherwise let $U_{xy} = 0$ and $i = 0$;
Step 2. Let $i = i + 1$. Go to Step 5 if $i > m$.
Step 3. Go to Step 2 if $x < l_i$ or $y < w_i$; otherwise let $U_i = \max(U_{xi}, U_{yi})$, in which

$$U_{xi} = v_i + \text{URF}(x - l_i, w_i) + \text{URF}(l_i, y - w_i),$$
$$U_{yi} = v_i + \text{URF}(l_i, y - w_i) + \text{URF}(x - l_i, y).$$

Step 4. Let $U_{xy} = U_i$ if $U_i > U_{xy}$. Go to Step 2.
Step 5. Return U_{xy}.

2.3 The recursive function with some heuristic rules

Assume that recursive function RecFun($x, y, \mathbf{B}, \mathbf{C}, V_f, R_f$) returns the maximum value of block $x \times y$, in which $\mathbf{B}$ is the vector of the items already placed in the father pattern, $\mathbf{C} = \{C_1, \ldots, C_m\}$, with C_i denoting the number of item type i included in the block, V_f is the total value of the items already placed on the plate before the current block $x \times y$ is considered. It is also referred to as the value of the father pattern. R_f is a logical variable with value 1 or 0, where 1 stands for true and 0 for false. $\mathbf{B}$ is determined before the recursive function is called. The elements of $\mathbf{C}$ are initialized to zero, determined within the recursive function, and returned to the father pattern. Assume that S is the region in the plate that is complementary to block $x \times y$, i.e., the plate consists of region S and block $x \times y$. If all blocks in S have been considered, i.e., no items can be further placed in S, let $R_f = 1$; otherwise let $R_f = 0$. Let V_{xy} be the current best value of the block. Let $\mathbf{C}' = \{C'_1, \ldots, C'_m\}$, $\mathbf{C}'' = \{C''_1, \ldots, C''_m\}$ be two vectors to record respectively the numbers of items placed in the two smaller child blocks (see Fig. 1), and k be the index of the item selected to place at the left-bottom corner of block $x \times y$. The steps of function RecFun($x, y, \mathbf{B}, \mathbf{C}, V_f, R_f$) are:

Step 1. Return 0 if $t > t_{\max}$ or $x < L_{\min}$ or $y < W_{\min}$; Let $V_{xy} = 0$, $i = 0$ and $k = 0$.
Step 2. Let $i = i + 1$. Go to Step 7 if $i > m$.
Step 3. Go to Step 2 if $x < l_i$ or $y < w_i$ or $b_i - B_i < 1$.
Step 4.
  Step 4.1. Let $U_{xy} = v_i + \text{URF}(x - l_i, w_i) + \text{URF}(x, y - w_i)$;
    Go to Step 5 if $U_{xy} \leq V_{xy}$ or ($R_f = 1$ and $V_f + U_{xy} \leq V^*$);
    Let $B_i = B_i + 1$ and $\mathbf{C}' = \mathbf{0}$;
    Let $V_{xR} = \text{RecFun}(x - l_i, w_i, \mathbf{B}, \mathbf{C}', v_i + V_f, 0)$.
  Step 4.2. Let $U_{xy} = v_i + V_{xR} + \text{URF}(x, y - w_i)$;
    Go to Step 4.6 if $U_{xy} \leq V_{xy}$ or ($R_f = 1$ and $V_f + U_{xy} \leq V^*$);
    Let $\mathbf{B} = \mathbf{B} + \mathbf{C}'$ and $\mathbf{C}'' = \mathbf{0}$;
    Let $V_{xT} = \text{RecFun}(x, y - w_i, B, C'', v_i + V_{xR} + V_f, R_f)$.
  Step 4.3. Let $V_{xi} = v_i + V_{xR} + V_{xT}$. Go to Step 4.5 if $V_{xi} \leq V_{xy}$.
  Step 4.4. Let $k = i$, $V_{xy} = V_{xi}$, $\mathbf{C} = \mathbf{C}' + \mathbf{C}''$ and $V^* = \max(V^*, V_f + V_{xy})$.
  Step 4.5. Let $\mathbf{B} = \mathbf{B} - \mathbf{C}'$.
  Step 4.6. Let $B_i = B_i - 1$.

Step 5.
  Step 5.1. Let $U_{xy} = v_i + \text{URF}(l_i, y - w_i) + \text{URF}(x - l_i, y)$;
    Go to Step 6 if $U_{xy} \le V_{xy}$ or ($R_f = 1$ and $V_f + U_{xy} \le V^*$);
    Let $B_i = B_i + 1$ and $\mathbf{C}' = \mathbf{0}$;
    Let $V_{yT} = \text{RecFun}(l_i, y - w_i, \mathbf{B}, \mathbf{C}', v_i + V_f, 0)$.
  Step 5.2. Let $U_{xy} = v_i + V_{yT} + \text{URF}(x - l_i, y)$;
    Go to Step 5.6 if $U_{xy} \le V_{xy}$ or ($R_f = 1$ and $V_f + U_{xy} \le V^*$);
    Let $\mathbf{B} = \mathbf{B} + \mathbf{C}'$ and $\mathbf{C}'' = \mathbf{0}$;
    Let $V_{yR} = \text{RecFun}(x - l_i, y, B, C'', v_i + V_{yT} + V_f, R_f)$.
  Step 5.3. Let $V_{yi} = v_i + V_{yT} + V_{yR}$. Go to Step 5.5 if $V_{yi} \le V_{xy}$.
  Step 5.4. Let $k = i$, $V_{xy} = V_{yi}$, $\mathbf{C} = \mathbf{C}' + \mathbf{C}''$ and $V^* = \max(V^*, V_f + V_{xy})$.
  Step 5.5. Let $\mathbf{B} = \mathbf{B} - \mathbf{C}'$.
  Step 5.6. Let $B_i = B_i - 1$.
Step 6. Go to Step 2.
Step 7. Let $C_k = C_k + 1$ if $k > 0$.
Step 8. Return V_{xy}.

In which Steps 4 and 5 correspond respectively to dividing the unoccupied region horizontally (Fig. 1a) and vertically (Fig. 1b). $\mathbf{C}'$ is the vector of the numbers of items included in block $(x - l_i) \times w_i$; $\mathbf{C}''$ is that in block $x \times (y - w_i)$. Initially both $\mathbf{C}'$ and $\mathbf{C}''$ are set to $\mathbf{0}$; One piece of item type i is selected and placed at the left-bottom corner of the block, and this is indicated by $B_i = B_i + 1$.

In Step 4.1, firstly, the upper bound of block $x \times y$ is determined as: $U_{xy} = v_i + \text{URF}(x - l_i, w_i) + \text{URF}(x, y - w_i)$; Secondly, the current branch can be discarded if $U_{xy} \le V_{xy}$ or ($R_f = 1$ and $V_f + U_{xy} \le V^*$), where the upper bound U_{xy} and the lower bound V_{xy} are local bounds that are only valid for the current block under consideration, and V^* may be referred to as the global bound, it is the current best value of the global pattern, $V_f + U_{xy}$ is the value of current searching branch if $R_f = 1$, but it is an incorrect value if $R_f = 0$; Thirdly, the current branch can be considered, so B is updated to: $B_i = B_i + 1$ and the elements of C' are initialized to zero; Finally, the value of block $(x - l_i) \times w_i$ is determined as: $V_{xR} = \text{RecFun}(x - l_i, w_i, \mathbf{B}, \mathbf{C}', v_i + V_f, 0)$, in which $R_f = 0$. The reason is that the complementary region S now includes the unoccupied block $x \times (y - w_i)$, in which some items may be placed later.

In Step 4.2, V_{xR} has been determined in Step 4.1, so the upper bound of block $x \times y$ is modified to: $U_{xy} = v_i + V_{xR} + \text{URF}(x, y - w_i)$; Similarly, the current branch can be discarded if $U_{xy} \le V_{xy}$ or ($R_f = 1$ and $V_f + U_{xy} \le V^*$); The unoccupied block $x \times y - w_i$ in block $x \times y$ will be considered at next step, so B is updated to $\mathbf{B} = \mathbf{B} + \mathbf{C}'$ and the elements of C'' are initialized to zero; The value of block $x \times (y - w_i)$ is $V_{xT} = \text{RecFun}(x, y - w_i, \mathbf{B}, \mathbf{C}'', V, R_f)$, in which R_f retains its original value.

The current value V_{xi} of block $x \times y$ is obtained in Step 4.3. If it is larger than the current best value V_{xy}, the current best solution of block $x \times y$, the current best value of the global pattern, and the numbers of items placed in current block $x \times y$ are renewed (Step 4.4). Step 4.5 restores vector $\mathbf{B}$ to its original value before Step 4.2, so

as to consider other item types. Step 4.6 restores vector B_i to its original value before Step 4.1.

Step 5 can be interpreted similarly.

2.4 The algorithm

The algorithm of this paper consists of the following steps:

Step 1. Obtain the unconstrained solution U_{xy} from the procedure described in Sect. 2.2, $0 \leq x \leq L$ and $0 \leq y \leq W$.
Step 2. Let $\mathbf{B} = \mathbf{C} = \mathbf{0}$.
Step 3. Let $V^* = \text{RecFun}(L, W, \mathbf{B}, \mathbf{C}, 0, 1)$.
Step 4. Output the optimal solution.

3 The computational results

This section presents the computational results of 62 small/medium scale problem instances and 20 large scale problem instances, which are taken from Hifi (2004) [16] and available on the Internet at http://www.laria.u-picardie.fr/hifi/OR-Benchmark/. About the problem details, please refer to Hifi (2004) [16].

This section compares the algorithm of this paper (referred to as the REC) with two best algorithms for the CTDC problem: the TS500 algorithm [15] for generating general guillotine patterns based on Tabu Search; and the TDH2 algorithm [16] for generating general guillotine patterns based on Top-Down approach using Hill-climbing strategies. The notations for the algorithms are used here for convenience.

The computational results are shown in Table 1 for the weighted problems and in Table 2 for the unweighted ones, where ID is the problem index; Column Opt/upper represents the optimal value and in some cases it represents the upper bounds (without proof of optimality), which is obtained from Tables 1 and 2 of [16]; Column TS500 displays the value obtained by TS500 (see Tables 4, 8 and 11 of [15]); Column TDH2 displays the value obtained by TDH2 (see Tables 1 and 2 of [16]); Column REC displays the value obtained by REC, which is presented in this paper. The computers used to perform the computations are as follows: TS500 on a computer with a Pentium II processor at 350 MHz; TDH2 on a UltraSparc10 with clock rate 250 MHz and main memory 128 MB; REC on a Personal Computer with clock rate 300 MHz (AMD K6-2 CPU) and main memory 128 MB. In order to permit a relatively fair comparison of run times, we chose to use this older PC for our computational experiments.

It should be noted that different problem indexes have been used for the problems. Problem 2, 3, 2s, 3s above are denoted respectively as P2, P3, P2s, P3s in [15].

The ratio to optimality indicates how the solution is close to optimal. It is determined as $A(I)/Opt(I)$, where I is an instance of the problem, $A(I)$ denotes the approximate solution value obtained by each algorithm and $Opt(I)$ is the optimal (or the upper bound) solution value. Average Ratio to optimality is averaged on a group of problem instances.

Table 1 The computational results on the weighted problem instances. The symbol * means that the reported value denotes the upper bound of the treated instance, without proof of optimality

ID	Opt/upper*	TS500	TDH2	REC	ID	Opt/upper*	TS500	TDH2	REC
26 Small and medium problem instances									
CHW1	2892	2892	2892	2892	2	2892	2892	2892	2892
CHW2	1860	1860	1860	1860	3	1860	1860	1860	1860
CW1	6402	6402	6402	6402	A1	2020	2020	2020	2020
CW2	5354	5354	5354	5354	A2	2505	2455	2505	2505
CW3	5689	5689	5689	5689	STS2	4620	4620	4620	4620
CW4	6175	6170	6175	6175	STS4	9700	9700	9700	9700
CW5	11659	11644	11659	11659	CHL1	8671	8660	8660	8660
CW6	12923	12923	12923	12923	CHL2	2326	2326	2326	2326
CW7	9898	9898	9898	9898	CHL3	5283	5283	5283	5283
CW8	4605	4605	4605	4605	CHL4	8998	8998	8998	8998
CW9	10748	10748	10748	10748	Hch11	11303	11089	11255	11235
CW10	6515	6515	6515	6515	Hch12	9954	9918	9953	9954
CW11	6321	6321	6321	6321	Hch19	5240	5240	5240	5240
10 Large-scale problem instances									
ATP40	67654*	66362	67154	66656	ATP45	75888*	74691	74691	75808
ATP41	215699*	206542	206542	206542	ATP46	151813*	149911	149911	149911
ATP42	34098*	3343	33503	34015	ATP47	153747*	148764	150234	150043
ATP43	222570*	214651	214651	214840	ATP48	170914*	166927	167660	167535
ATP44	74887*	73410	73868	73868	ATP49	226346*	215728	218388	217683

The computational results of Table 1 and Table 2 are summarized in Table 3. It includes the ratio to optimality of each algorithm: R-TS500 for the TS500 algorithm, R-TDH2 for the TDH2 algorithm, R-REC for the REC algorithm, and computation time of each algorithm: T-TDH2 for the TDH2 algorithm, T-REC for the REC algorithm. The detailed computation times of TS500 algorithm are not found from Alvarez-Valdes et al. (2002) [15].

The first two blocks of Table 3 summarize the results given by the different algorithms for the small/medium and large *weighted* problem instances. The next two blocks represent the results for the *unweighted version* of the problem. Each block is represented by a line Min (resp. Max) which represents the minimum (resp. maximum) ratio to optimality (or computational time) produced (or consumed) by each algorithm, and another line (Av.), which represents the average ratio to optimality (or computational time) over all considered instances. The last line of Table 3 summarizes the global average ratios to optimality and the computational time for both weighted and unweighted versions of the CTDC problem.

From Table 3 it is seen that the average ratio to optimality of the patterns generated by the REC is higher than that of the patterns generated by the TS500 for all four groups, and is a little higher than that of the patterns generated by the TDH2 for three groups. The global average ratio (0.9972) to optimality is the maximum for the algorithm of this paper, and its average computation time (34.388 seconds) is shorter than

Table 2 The computational results on the unweighted problem instances. The symbol * means that the reported value denotes the upper bound of the treated instance, without proof of optimality

ID	Opt/upper*	TS500	TDH2	REC	ID	Opt/upper*	TS500	TDH2	REC
36 Small and medium problem instances									
OF1	2737	2713	2737	2737	STS2s	4653	4653	4653	4653
OF2	2690	2586	2690	2690	STS4s	9770	9770	9770	9770
W	2721	2721	2721	2721	CHL1s	13099	13099	13099	13099
CU1	12330	12330	12330	12330	CHL2s	3279	3279	3279	3279
CU2	26100	26100	26100	26100	CHL3s	7402	7402	7402	7402
CU3	16723	16679	16723	16723	CHL4s	13932	13932	13932	13932
CU4	99495	99366	99495	99495	CHL5	390	390	390	390
CU5	173364	173364	173364	173364	CHL6	16869	16869	16869	16869
CU6	158572	158572	158572	158572	CHL7	16881	16838	16881	16840
CU7	247150	247150	247150	247150	Hch13s	12215	12208	12214	12214
CU8	433331	432714	433331	433331	Hch14s	11994	11967	11993	12002
CU9	657055	657055	657055	657055	Hch15s	45361	45223	45361	45313
CU10	773772	773485	773772	773485	Hch16s	61040	61002	61002	61040
CU11	924696	922161	924696	924311	Hch17s	63112	62802	63029	63102
2s	2778	2778	2778	2778	Hch18s	911	904	904	904
3s	2721	2721	2721	2721	A3	5451	5436	5436	5451
A1s	2950	2950	2950	2950	A4	6179	6179	6179	6179
A2s	3535	3535	3535	3535	A5	12985	12929	12976	12976
10 Large-scale problem instances									
ATP30	140904	140144	140904	140904	ATP35	622644*	620700	621021	621021
ATP31	824931*	814081	823976	823674	ATP36	130744	130338	130744	130744
ATP32	38068	38030	38068	38068	ATP37	387276	381966	387118	387118
ATP33	236818*	234920	236611	236611	ATP38	261698*	259380	261395	261395
ATP34	362520*	360084	361167	361197	ATP39	268750	267168	268750	268355

algorithm TDH2 (75.56 seconds). Here, we assume that the computational speeds of two computers (UltraSparc10 with clock rate 250 MHz and PC with clock rate 300 MHz) are basically same, so the conclusion about computation time is credible.

It should be indicated that for Problems Hch14s, the optimal value Opt (= 11994) reported in the literature is wrong, because it cannot be smaller than REC (= 12002). To support this statement, the pattern generated by the REC is given in Fig. 2.

4 Conclusions

From the results presented, we may make the following conclusions:

(1) The algorithm can generate patterns of high quality for both small- and large-scale instances.

Table 3 Summary of the computational results of small/medium and large-scale problem instances

	Ratio to optimality			Computation time(s)	
	R-TS500	R-TDH2	R-REC	T-TDH2	T-REC
Group 1: 26 Small and medium weighted problem instances					
Min	0.9800	0.9958	0.9940	0.52	0.100
Max	1	1	1	71.05	94.772
Av.	0.9982	0.9998	0.9997	16.96	8.135
Group 2: 10 Large-scale weighted problem instances					
Min	0.9531	0.9575	0.9575	121.57	7.420
Max	0.9875	0.9926	0.9989	445.67	1000.08
Av.	0.9724	0.9762	0.9796	263.23	130.180
Average	0.9913	0.9937	0.9941	85.37	42.036
Group 3: 36 Small and medium unweighted problem instances					
Min	0.9923	0.9923	0.9976	0.11	0.100
Max	1	1	1.0007	56.95	70.690
Av.	0.9987	0.9995	0.9999	11.68	5.117
Group 4: 10 Large-scale unweighted problem instances					
Min	0.9863	0.9963	0.9964	121.50	3.214
Max	0.9990	1	1	412.30	364.585
Av.	0.9929	0.9989	0.9988	260.38	112.227
Average	0.9972	0.9994	0.9997	65.75	28.402
All treated problems instances					
Global Av.	0.9942	0.9965	0.9972	75.56	34.388

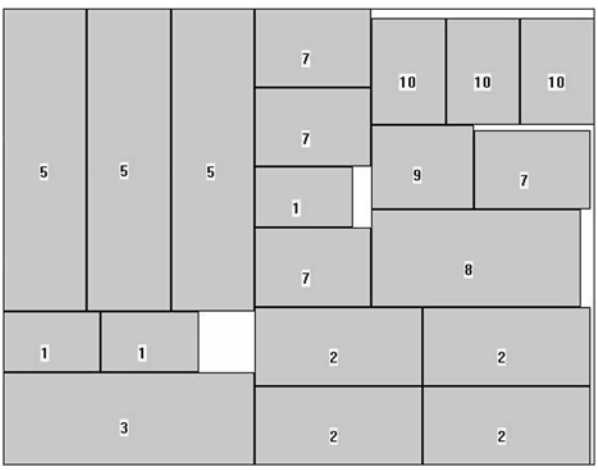


Fig. 2 The cutting pattern for Problem Hch14s (computation time 2.030 s, value 12 002)

(2) Some heuristic ideas, such as upper and lower bounding rules, are very helpful for improving the efficiency of the algorithm. They make the algorithm run significantly more quickly than algorithm TDH2.

It is well-known that, for relatively large instances, computing times can be excessive if exact algorithms are used for generating general guillotine patterns. Since the algorithm of this paper can generate high quality patterns in an acceptable amount of time, it may be an attractive alternative for large-scale instances.

Acknowledgements This research is supported in part by the China National Natural Science Foundation Project (60763011). The author would like to thank Prof. Yaodong Cui of Guangxi Normal University, China, and also the anonymous referees, for their helpful suggestions which contributed to the improvement of this paper.

References

1. Wascher, G., Haussner, H., Schumann, H.: An improved typology of cutting and packing problems. Eur. J. Oper. Res. **183**, 1109–1130 (2007)
2. Christofides, N., Whitlock, C.: An algorithm for two-dimensional cutting problems. Oper. Res. **25**, 30–34 (1977)
3. Christofides, N., Hadjiconstantinou, E.: An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. Eur. J. Oper. Res. **83**, 21–38 (1995)
4. Hifi, M., Zissimopoulos, V.: Constrained two-dimensional cutting: an improvement of Christofides and Whitlock's exact algorithm. J. Oper. Res. Soc. **5**, 8–18 (1997)
5. Wang, P.Y.: Two algorithms for constrained two-dimensional cutting stock problems. Oper. Res. **32**(3), 573–586 (1983)
6. Vasko, F.J.: A computational improvement to Wang's two-dimensional cutting stock algorithm. Comput. Ind. Eng. **16**, 109–115 (1989)
7. Hifi, M.: An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. Comput. Oper. Res. **24**, 727–736 (1997)
8. Viswanathan, K.V., Bagchi, A.: Best-first search methods for constrained two-dimensional cutting stock problems. Oper. Res. **41**, 768–776 (1993)
9. Daza, V.P., Alvarenga, A.G., Diego, J.: Exact solutions for constrained two-dimensional cutting problems. Eur. J. Oper. Res. **84**, 633–644 (1995)
10. Cung, V.D., Hifi, M., Le Cun, B.: Constrained two-dimensional cutting stock problems. A best-first branch-and-bound algorithm. Int. Trans. Oper. Res. **7**, 185–210 (2000)
11. Amaral, A.R.S., Wright, M.: Efficient algorithm for the constrained two-dimensional cutting stock problem. Int. Trans. Oper. Res. **8**, 3–13 (2001)
12. Morabito, R.N., Arenales, M.N.: Staged and constrained two-dimensional guillotine cutting problems: an AND/OR graph approach. Eur. J. Oper. Res. **94**, 548–560 (1996)
13. Zissimopoulos, V.: Heuristic methods for solving (un)constrained two-dimensional cutting stock problems. Method. Oper. Res. **49**, 345–357 (1984)
14. Fayard, D., Hifi, M., Zissimopoulos, V.: An efficient approach for large-scale two-dimensional guillotine cutting stock problems. J. Oper. Res. Soc. **49**, 1270–1277 (1998)
15. Alvarez-Valdes, R., Parajon, A., Tamarit, J.M.: A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. Comput. Oper. Res. **29**, 925–947 (2002)
16. Hifi, M.: Dynamic programming and hill-climbing techniques for the constrained two-dimensional cutting stock problem. J. Comb. Optim. **8**, 65–84 (2004)
17. Cui, Y.: An exact algorithm for generating homogenous T-shape cutting patterns. Comput. Oper. Res. **34**(4), 1107–1120 (2007)
18. Cui, Y.: Heuristic and exact algorithms for generating homogenous constrained three-staged cutting patterns. Comput. Oper. Res. **35**, 212–225 (2008)
19. Beasley, J.E.: Algorithms for unconstrained two-dimensional guillotine cutting. J. Oper. Res. Soc. **36**, 297–306 (1985)

20. Young-Gun, G., Seong, Y.J., Kang, M.K.: A best-first branch and bound algorithm for unconstrained two-dimensional cutting problems. Oper. Res. Lett. **31**, 301–307 (2003)
21. Cui, Y., Wang, Z., Li, J.: Exact and heuristic algorithms for staged cutting problems. Proc. Inst. Mech. Eng. Part B: J. Eng. Manuf. **219**(B2), 201–208 (2005)
22. Hifi, M.: Exact algorithms for large-scale unconstrained two and three staged cutting problems. Comput. Optim. Appl. **18**, 63–88 (2001)
23. Cui, Y.: Generating optimal T-shape cutting patterns for rectangular blanks. Proc. Inst. Mech. Eng. Part B: J. Eng. Manuf. **218**(B8), 857–866 (2004)
24. Cui, Y.: Simple block patterns for the two-dimensional cutting problem. Math. Comput. Model. **45**(7–8), 943–953 (2007)