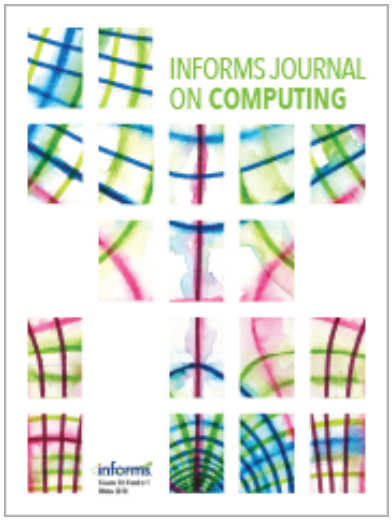


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

An Exact Algorithm for the Two-Dimensional Stage-Unrestricted Guillotine Cutting/Packing Decision Problem

Krzysztof Fleszar

To cite this article:
Krzysztof Fleszar (2016) An Exact Algorithm for the Two-Dimensional Stage-Unrestricted Guillotine Cutting/Packing Decision Problem. INFORMS Journal on Computing 28(4):703-720. https://doi.org/10.1287/ijoc.2016.0708

INFORMS Journal on Computing
Vol. 28, No. 4, Fall 2016, pp. 703–720
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2016.0708

An Exact Algorithm for the Two-Dimensional Stage-Unrestricted Guillotine Cutting/Packing Decision Problem

Krzysztof Fleszar
Suliman S. Olayan School of Business, American University of Beirut, Riad El-Solh, Beirut 1107 2020, Lebanon,
kfleszar@gmail.com

We propose a new exact algorithm for the two-dimensional stage-unrestricted guillotine cutting/packing decision problem, which asks if a set of rectangular items can be cut from a single stock rectangle using guillotine cuts only, with fixed item orientation or with 90-degree item rotation. Our algorithm constructs patterns of items by means of horizontal and vertical builds. To speed up the algorithm and reduce its memory requirement, patterns are constructed in the order of nondecreasing waste, the patterns that use the same subset of items are grouped together, and dominated patterns in each group are discarded. Moreover, a heuristic capable of completing a partial pattern is repeatedly used during the algorithm to quickly determine a feasible solution. Furthermore, the algorithm tries to prove infeasibility with a subset of items before considering all items. We test our algorithm on benchmark instances for the two-dimensional guillotine strip-cutting problem, which we solve by varying the strip height and testing with our algorithm whether a feasible solution exists. We show that our approach outperforms all previously proposed algorithms for the problem with fixed item orientation. Computational experiments for the problem with item rotation are also reported.

Keywords: exact algorithm; two-dimensional cutting/packing; guillotine cuts; rotation; strip cutting
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received March 2015; revised December 2015; accepted March 2016. Published online September 20, 2016.

1. Introduction

In the two-dimensional stage-unrestricted guillotine cutting/packing decision problem (2GC), we want to determine if a set of small rectangles (items) can be laid out on a single large rectangle (stock rectangle or bin), such that the items can be obtained from the stock rectangle using guillotine cuts only. A guillotine cut is defined as a straight-line horizontal or vertical cut that divides a rectangle into two smaller rectangles. The problem is stage-unrestricted, i.e., the cutting process is allowed to alternate between the stages of horizontal and vertical guillotine cuts any number of times.

2GC is the decision problem corresponding to the optimization problem of the two-dimensional guillotine strip cutting/packing problem (2GSC/2GSP), in which all items must be laid out on a single large rectangle (strip) with a fixed width and a variable height, such that the height of the strip is minimized.

The importance of 2GC stems from the fact that it occurs as a subproblem in exact algorithms for several two-dimensional guillotine cutting and packing (C&P) problems, including: the above-mentioned 2GSC/2GSP problem; the demand-unconstrained and demand-constrained two-dimensional guillotine knapsack problem (2GK), which asks to cut from a stock rectangle a subset of items with maximum total value; and the two-dimensional guillotine cutting stock/bin packing problem (2GCS/2GBP), which asks to cut all items from the minimum number of identical stock rectangles (see Wäscher et al. 2007 for a typology of C&P problems). Such problems occur in industry when rectangular pieces of glass, wood or other material must be cut from larger rectangular pieces using guillotine cuts only. Hence, a good algorithm for 2GC can help improve solution methods for all the above problems.

We propose an exact algorithm that can solve two variants of 2GC, one in which items must be laid out in the given orientation with respect to the stock rectangle (2GC|O), and the other in which a 90-degree item rotation is allowed (2GC|R).

The typology of Wäscher et al. (2007) divides all C&P problems into two categories with respect to the kind of assignment, i.e., one with input minimization objectives and the other with output maximization objectives. Because 2GC is a decision rather than optimization problem, it belongs to neither category and can be considered a problem variant in the above typology. 2GC is also a problem variant due to the guillotine cut restriction imposed on the item layout.

Many papers on two-dimensional C&P problems address problems without guillotine cut restriction. An exact algorithm for the two-dimensional packing decision problem, the nonguillotine counterpart of 2GC, is proposed by Clautiaux et al. (2008). Fekete et al. (2007) develop an exact algorithm for the two-dimensional knapsack problem. Martello et al. (2003), Alvarez-Valdes et al. (2009), and Boschetti and Montaletti (2010) propose exact algorithms for two-dimensional strip packing problem. For the two-dimensional bin packing problem, exact algorithms are proposed by Martello and Vigo (1998), Clautiaux et al. (2007), and Pisinger and Sigurd (2007). For the same problem with variable-size bins, exact algorithm is proposed by Pisinger and Sigurd (2005).

Christofides and Whitlock (1977) and Wang (1983) produced seminal works on guillotine cutting problems; both address 2GK. The former proposes a top-down method based on cutting the stock rectangle into smaller rectangles until items are obtained. The latter presents a bottom-up approach that starts from the items and combines them into larger rectangles using horizontal and vertical builds. Several works propose improvements to the above algorithms, e.g., (Vasko 1989, Oliveira and Ferreira 1990, Viswanathan and Bagchi 1993, Christofides and Hadjiconstantinou 1995, Morabito and Arenales 1996, Hifi 1997, Morabito and Pureza 2010). Some of those works also consider the stage-restricted variant of the problem. Beasley (1985a) examines demand-unconstrained 2GK with restricted and unrestricted number of cutting stages and presents a number of algorithms based on dynamic programming. An exact algorithm for 2GK is proposed by Dolatabadi et al. (2012). For 2GSP/2GSC, exact algorithms are proposed by Hifi (1998) and Bekrar et al. (2010). An exact solution of 2GBP is addressed by Pisinger and Sigurd (2007), who show how the guillotine constraint, as well as several other constraints, can be added to their exact algorithm for the two-dimensional bin packing problem. Also, Amossen and Pisinger (2010) propose an exact algorithm for the multidimensional guillotine bin packing problem, an extension of 2GBP to more than two dimensions. Finally, Furini et al. (2016) present a new framework for modeling guillotine cuts in mixed integer linear programming (MIP) models. They formulate MIP models for 2GK, 2GSP, and 2GCS, and propose an exact procedure for 2GK based on the MIP model.

In a recent paper, Clautiaux et al. (2013) propose a new graph-theoretical model for guillotine cutting problems and show how it can be used to solve 2GC|O. They test their algorithm on benchmark instances for 2GSC|O, which they solve by varying the height of the strip and testing with their algorithm whether a feasible solution exists. They compare the performance of their algorithm with the algorithms from Hifi (1998) and Bekrar et al. (2010) and show that their algorithm outperforms those from the preceding works. In this paper, we propose an exact algorithm that can solve 2GC|O much faster than the algorithm from Clautiaux et al. (2013). We also show how our algorithm can be adapted to solve 2GC|R.

2. Algorithm for 2GC|O

Let vector $[W, H]$ denote the *stock rectangle*, where W is the width, and H is the height. Let $N = \{1, \dots, n\}$ be the set of *item types*, $[w_i, h_i]$ the rectangle of item type $i \in N$, and $b = [b_1, \dots, b_n]$ the vector of item requirements, where b_i is the number of required items of type $i \in N$. We will assume that all widths and heights are positive integers and that $w_i \le W$ and $h_i \le H \ \forall i \in N$. We will also assume that all item types are unique, and that they are ordered by nonincreasing $w_i h_i$, breaking ties by nonincreasing $\min\{w_i, h_i\}$, and then by nonincreasing h_i.

2.1. Patterns

Let *pattern* p be defined by (1) a vector of item counts, $c(p) = [c_1(p), \dots, c_n(p)]$, where $c_i(p)$ is the number of items of type i included in the pattern, (2) a description of how the items are cut (described later), and (3) the smallest rectangle $[w(p), h(p)]$ from which the items can be cut in the manner described. Note that $c(p)$ defines a multiset of items included in the pattern, so we will sometimes refer to the multiset instead of the item counts.

Wang (1983) observed that all guillotine cuttable patterns can be constructed by starting from single-item patterns and repeatedly applying *horizontal and vertical builds*. We define *single-item pattern* p for an item of type $i \in N$ as the pattern with $c_i(p) = 1$, $c_j(p) = 0 \ \forall j \in N \backslash \{i\}$, and rectangle $[w(p), h(p)] = [w_i, h_i]$. A *horizontal* (resp. *vertical*) *build* of patterns p and q creates a *composite pattern* with item count vector $c(p) + c(q)$ and rectangle $[w(p) + w(q), \max\{h(p), h(q)\}]$ (resp. $[\max\{w(p), w(q)\}, h(p) + h(q)]$), from which patterns p and q can be obtained using a vertical (resp. horizontal) guillotine cut. Hence, each composite pattern can be recursively described by the two subpatterns from which it is constructed and the type of build.

Figure 1 depicts the two types of builds.

Pattern p is called *feasible* if it does not exceed item requirements and fits in the stock rectangle, i.e., if $c(p) \le b$ and $[w(p), h(p)] \le [W, H]$. Note that we do not need to require nonoverlapping, as the manner in which we construct patterns guarantees nonoverlapping of items in all patterns. A feasible pattern is called *partial* if it does not contain all items, i.e., if $c(p) \le b$, and *complete* if it contains all items, i.e., if $c(p) = b$. (For vectors $x, y \in \mathbb{R}^m$, $x \le y \Leftrightarrow \forall i \ x_i \le y_i$,

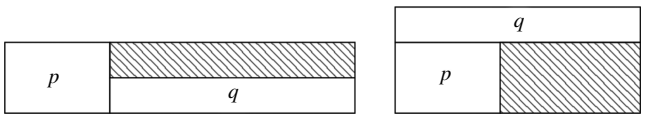


Figure 1 **Horizontal and Vertical Builds of Patterns p and q, and the Introduced Waste (Dashed)**

whereas $x \leq y \Leftrightarrow (\forall i,\ x_i \leq y_i) \wedge (\exists i,\ x_i \neq y_i)$. Inequalities $\geqq$ and $\geq$ are analogously defined.)

Let waste(p) be the unused area of pattern p, calculated as $\text{waste}(p) = w(p)h(p) - \sum_{i \in N} w_i h_i c_i(p)$. Single-item patterns have zero waste, but horizontal (resp. vertical) builds can introduce waste if heights (resp. widths) of subpatterns do not match, as shown in Figure 1. Because every composite pattern inherits the waste of its subpatterns, its waste is always greater than or equal to the waste of each of its subpatterns.

2.2. Pattern Construction Process

The basic idea of our exact algorithm for 2GC|O is to generate patterns through horizontal and vertical builds, as in the algorithm of Wang (1983) for 2GK. However, we generate patterns in a process more akin to that used by Viswanathan and Bagchi (1993) and Hifi (1997) for 2GK, and Hifi (1998) for 2GSC. We keep a *set of open patterns*, O, which contains all constructed patterns that have not taken part in any build so far. Initially, O contains one single-item pattern for each item type. In each iteration, a pattern p is removed from O and added to a *set of accepted patterns* A, which is initially empty. Then, new patterns are constructed through horizontal and vertical builds of p and q, $\forall q \in A$ (including $q = p$). All new feasible patterns are then added to O. If through this process at least one complete pattern is generated, the problem is feasible. If O becomes empty and no complete pattern has been generated, the problem is infeasible.

Consider an example problem instance with a stock rectangle of size $[5, 3]$ and three item types of sizes $[2, 2]$, $[1, 3]$, and $[3, 1]$, respectively. Two copies of the first item type and one copy of each of the remaining item types are required. Table 1 shows the evolution of the pattern construction process for this example. The first two columns show the contents of sets O and A, and the third shows the patterns discarded at each stage of the process. In the initial stage, set O contains three single-item patterns. Each subsequent stage is obtained by moving one pattern from set O to set A (highlighted) and constructing new patterns through horizontal and vertical builds combining the moved pattern with each pattern in set A. Constructed patterns are discarded if they are infeasible or if their waste exceeds $WH - \sum_{i \in N} w_i h_i b_i$, which equals 1 in this example (this reduction will be discussed in Section 2.6). The undiscarded patterns are added to set O. In the sixth stage, which is shown in the last row of Table 1, a complete pattern is added to set O. This pattern proves that the problem instance is feasible, and the pattern construction process is terminated.

2.3. Dominance

This process can generate a large number of patterns, even for small problem instances. To reduce this number, we restrict the search to only undominated patterns. Pattern p is *dominated* by pattern q if both include the same multiset of items and the rectangle of p is strictly larger than the rectangle of q, i.e., if $c(p) = c(q)$ and $[w(p), h(p)] \geq [w(q), h(q)]$. Additionally, if p and q have equal rectangles, we select one of the two patterns and consider it dominated.

Because undominated patterns "fit in the spaces" of dominated patterns, each dominated pattern can be replaced with an undominated one without using more space. Therefore, for every feasible instance of 2GC, it must be possible to construct at least one complete pattern using undominated patterns only. Conversely, if a complete pattern cannot be obtained using undominated patterns, it cannot be obtained when dominated patterns are also used.

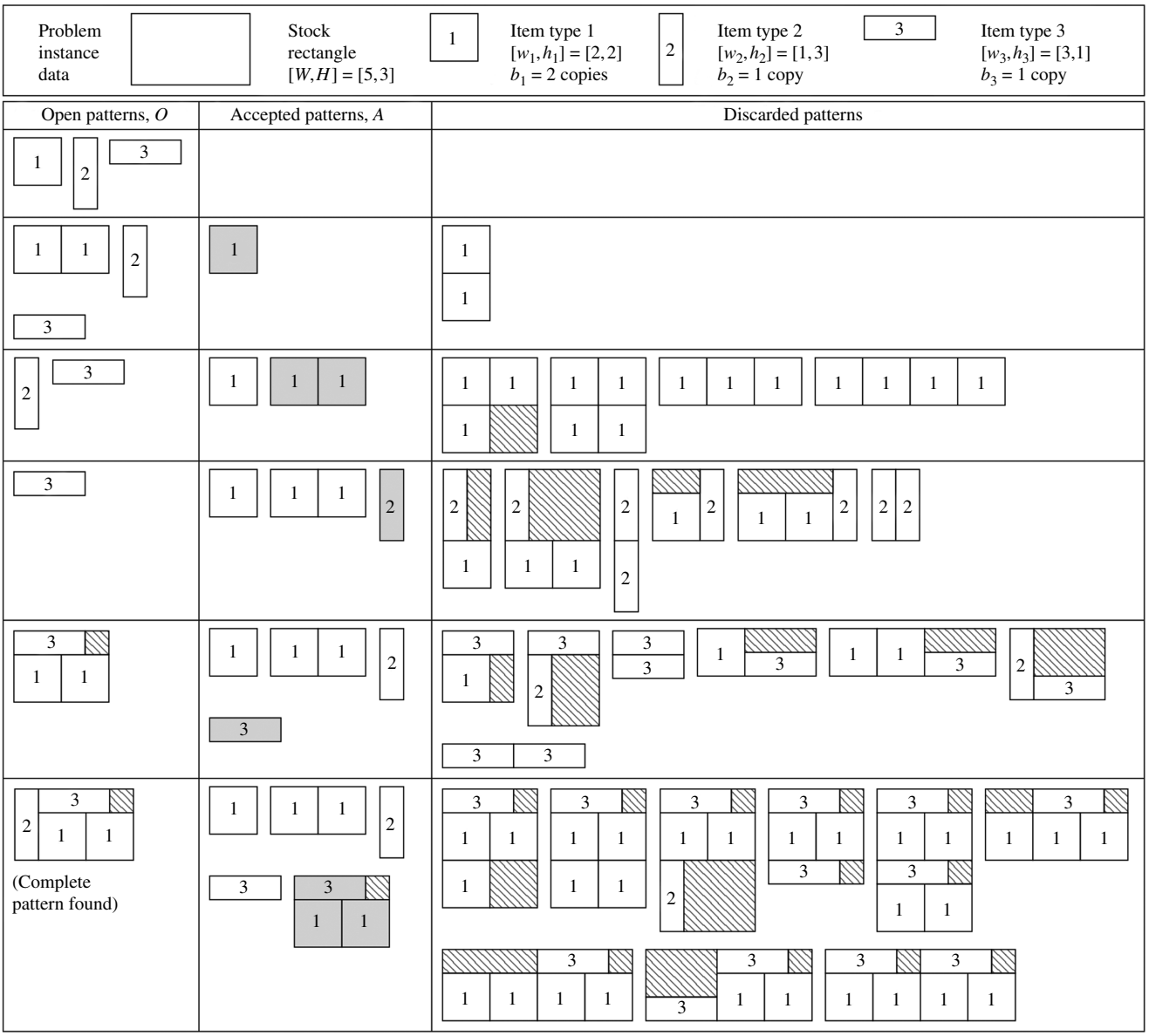
To facilitate testing for dominance, we group accepted patterns according to the multiset of items they include. Let G be the *set of pattern groups*, and let $g \in G$ be a *pattern group* that includes patterns $P(g) = \{p_1, p_2, \ldots, p_{n(g)}\}$ such that $c(g) = c(p_1) = c(p_2) = \cdots = c(p_{n(g)})$, where $c(g)$ will be called the multiset of group g. Pattern groups define a partition of the set of accepted patterns, A, where each group is defined for a different multiset of items, i.e., $A = \bigcup_{g \in G} P(g)$ and $\forall g_1, g_2 \in G\ g_1 \neq g_2 \Rightarrow c(g_1) \neq c(g_2)$.

2.4. Order of Patterns

To minimize the number of constructed patterns, a pattern p with the smallest waste is removed from set O in each iteration of the algorithm. Before p is accepted, its target pattern group $g \in G$ with $c(g) = c(p)$ is first identified. If p is dominated by any pattern in $P(g)$, then it is discarded, and no new patterns are constructed. Otherwise, p is added to $P(g)$, and new patterns involving p are constructed, as described in Section 2.2. Before those patterns are added to set O, they can also be tested for dominance against patterns in their respective target pattern groups; only currently undominated patterns need to be added to set O.

Note that if a pattern with the smallest waste is always removed from set O, then waste in the sequence of open patterns considered in the successive iterations of the algorithm is nondecreasing. To see this, suppose pattern p with x units of waste is removed from set O. Clearly, all patterns remaining in O must have at least x units of waste. Although

Table 1 Pattern Construction Process for an Example Problem Instance



new patterns might be added to O, all of them are constructed using p as one of the subpatterns. Hence, the new patterns will also have at least x units of waste.

A consequence of the above property is that every constructed pattern will have waste at least as large as each pattern in its target group; hence, accepted patterns in $P(g)$ never need to be discarded due to dominance. If pattern p considered to be accepted in pattern group $g \in G$ has waste$(p) \geq$ waste(q), $\forall\, q \in P(g)$, then the rectangle of p can never be strictly smaller than the rectangle of any $q \in P(g)$ (due to the same multiset of items). The rectangle of p can be equal to the rectangle of one $q \in P(g)$, if waste$(p) =$ waste(q); in this case, we can choose p to be the dominated pattern and keep set $P(g)$ intact.

The fact that accepted patterns stored in sets $P(g)$ are never dominated is a very important property of our algorithm. If dominated patterns were accepted, and their dominance were not detected for many iterations, they would take part in many builds, all of which would construct only more dominated patterns. This cascading effect could significantly increase the number of patterns constructed by the algorithm, which in turn would significantly increase the computation time as well as the amount of memory necessary to store open and accepted patterns.

2.5. Heuristic

Because patterns are constructed in order of nondecreasing waste, it might take a long time to determine a complete pattern for those problem instances

in which all complete patterns include relatively large waste.[1] In an attempt to generate complete patterns faster, a heuristic is used.

We use the insertion heuristic developed by Fleszar (2013) for 2GBP, which we simplify to the case of a single stock rectangle. For a given starting pattern, we consider all unpacked items in the order of item types, and attempt to insert each item into the partial pattern, as described by Fleszar (2013). If insertion of an item fails, the heuristic is stopped immediately, without attempting to insert the remaining items, as only complete patterns are of interest.

The heuristic is used at the beginning of the algorithm, with the single-item patterns used as starting patterns. It is also used later in the algorithm in an attempt to complete each partial pattern added to the set of open patterns.

2.6. Infeasibility Detection

To speed up the proof of infeasibility, we try to prove it with a subset of item types. To achieve this, we initialize set O with single-item patterns for item types from 1 to n^{open} only, where n^{open} is the number of open item types, $1 \le n^{\text{open}} \le n$. If $n^{\text{open}} < n$, the patterns generated will be limited to only those containing item types from 1 to n^{open}. If a feasible pattern with all those items cannot be found, then clearly the whole problem is infeasible.

When patterns with items of types 1 to n^{open} are generated, patterns with waste greater than $\text{wasteMax} = WH - \sum_{i=1}^{n^{\text{open}}} w_i h_i b_i$ are discarded, as those patterns would not lead to feasible patterns including all items of types 1 to n^{open}. The smaller wasteMax, the more patterns can be discarded. However, to decrease wasteMax, n^{open} must be increased, which quickly increases the number of possible patterns. To strike a balance, we set n^{open} based on the partial patterns constructed by the heuristic. Let n^{fail} be the smallest index of item type such that all heuristic attempts failed to find a feasible pattern for item types 1 to n^{fail}. Clearly, cutting all items of types 1 to $n^{\text{fail}} - 1$ is feasible, but the same is not certain for items of types 1 to n^{fail}. Hence, we can set $n^{\text{open}} = n^{\text{fail}}$. When this is done at the beginning of the algorithm, set O is initialized with single-item patterns for item types 1 to n^{open}. If n^{fail} is later increased due to additional heuristic attempts at solving the problem, additional single-item patterns for item types from $n^{\text{open}}+1$ to n^{fail} are added to set O, and n^{open} is set to n^{fail}. In this way, the algorithm always considers the smallest number of item types for which the proof of infeasibility may still be possible.

Note that the delayed opening of item types invalidates our previous claim of nondecreasing waste in the sequence of open patterns considered by the algorithm. Suppose item type i is opened after patterns for items 1 to $i-1$ have been constructed for some time, and the waste of the last pattern removed from O was $x > 0$. When item type i is opened, a single-item pattern for this item type with zero waste is added to O. Hence, the waste of the next pattern removed from O will be 0, which violates the nondecreasing order of waste. Despite that, patterns considered to be accepted in any pattern group will still arrive in the order of nondecreasing waste. The reason is that all open patterns with waste less than x that will now be considered, will include the newly opened item i, and hence they will be considered to be accepted in pattern groups including items of type i, which are all empty when item i is opened.

2.7. Additional Infeasibility Proofs

To further speed up the proof of infeasibility, we use the method from Fekete and Schepers (2004) developed for multidimensional cutting problems without guillotine cut requirement. They show that if dual-feasible functions are used to transform item dimensions and the total volume of transformed items exceeds the volume of the stock rectangle, then the original items cannot be obtained from the stock rectangle. In our algorithm, we use their functions $u^{(k)}\colon [0,1] \to [0,1]$ defined for $k \in \mathbb{N}$ as follows:

$$u^{(k)}(x) = \begin{cases} x & \text{if } x(k+1) \in \mathbb{Z}, \\ \lfloor (k+1)x \rfloor / k & \text{otherwise.} \end{cases}$$

Note that to use the functions all dimensions must be scaled such that $W = H = 1$, i.e., all widths must be divided by W and all heights divided by H. For convenience, we also define $u^{(0)}(x) = x$.

At the beginning of our algorithm, we attempt to prove infeasibility using $u^{(k_w)}$ to transform widths and $u^{(k_h)}$ to transform heights $\forall [k_w, k_h] \in \{0, \ldots, 10\}^2$. If for any pair $[k_w, k_h]$, the total area of transformed items exceeds one, the original problem is infeasible.

The method of Fekete and Schepers (2004) can also be used to prove that partial patterns generated in our algorithm cannot lead to complete patterns. It was previously observed by Viswanathan and Bagchi (1993) in the context of 2GCS that any one partial pattern included in a complete pattern can always be shifted to the left-bottom corner of the pattern rectangle, without changing the pattern rectangle. This is done by exchanging left and right parts of some horizontal builds as well as top and bottom parts of some vertical builds, such that the partial pattern of interest is shifted towards the left-bottom corner of the

[1] Note that if the problem is feasible, the overall waste will always be equal to $WH - \sum_{i \in N} w_i h_i b_i$, regardless of which complete pattern is selected, but this waste is composed of the waste of the pattern, which could be called the internal waste, and the space left between the edges of the rectangle of the pattern and the edges of the stock rectangle, which could be called the external waste.

complete pattern. Hence, the problem of completing a given partial pattern p can be thought of as the problem of cutting any remaining items from the unused space to the right of and above the rectangle of p. We divide this unused space into two partially overlapping rectangles: $R_{\text{right}}(p) = [W - w(p), H]$ and $R_{\text{above}}(p) = [W, H - h(p)]$. We then check if every item $i \in \{1, \ldots, n^{\text{open}}\}$ that is not included in p can be cut from $R_{\text{right}}(p)$ or $R_{\text{above}}(p)$. If an item cannot be cut from either, the partial pattern cannot be completed and, therefore, is discarded. Otherwise, two multisets of items are constructed: $c_{\text{right}}(p)$ includes those items that can only be cut from $R_{\text{right}}(p)$ and $c_{\text{above}}(p)$ includes those items that can only be cut from $R_{\text{above}}(p)$. Items that can be cut from both rectangles are ignored. Note that each of those multisets together with its respective rectangle forms a smaller subproblem of 2GC|O. Hence, we can try to prove infeasibility for each of them using dual-feasible functions. However, to save time, here we consider only $[k_w, k_h] \in \{0, 1, 2\}^2$. If any of the two subproblems is proven infeasible, then pattern p will never lead to a complete pattern and, therefore, is discarded. Additionally, if this happens for a single-item pattern and $n^{\text{open}} = n$, then the problem is proven infeasible.

2.8. Algorithm Pseudocode

Figure 2 presents a pseudocode of our algorithm for 2GC|O. In line 1, we try to prove infeasibility using dual-feasible functions, as described earlier. In line 2, the set of pattern groups, G, and the set of open patterns, O, are initialized to empty sets. Also, the smallest number of item types that all heuristic attempts failed to pack, n^{fail}, is initialized to 1, and the number of open item types, n^{open}, is initialized to 0. Finally, the maximum acceptable waste for patterns, wasteMax, is initialized to the stock rectangle area.

In lines 3–9, the heuristic is invoked n times, each time seeded with a different single-item pattern, in an attempt to construct a complete pattern. If a complete pattern is found, the algorithm is terminated. Otherwise, n^{fail} is appropriately updated. In the same loop, we also use dual-feasible functions to test if every single-item pattern can lead to a complete pattern (line 5). If not, then the problem is proven infeasible.

Lines 10–46 contain the main loop of our algorithm. First, new item types are opened in lines 11–19. In the first iteration of the main loop, item types 1 to n^{fail} are opened. In subsequent iterations, if n^{fail} has been increased, additional item types are opened. Opening is effected by adding single-item patterns into set O (line 13) and decreasing wasteMax (line 14). The number of open item types is saved in n^{open} in line 16. Line 17 ensures that all patterns with waste exceeding wasteMax are removed from the set of open patterns as well as from the pattern groups. We also remove patterns for which we can show through the use of dual-feasible functions that they cannot lead to complete patterns, as described earlier. Any pattern group that becomes empty is removed in line 18.

In lines 21–22, an open pattern, p, with the smallest waste is selected and removed from set O. Because many open patterns will often have the same waste, we break ties by preferring largest pattern rectangle area and, if these are also equal, largest shorter edge of the pattern rectangle. In our experiments, this tie-breaking helped reduce the peak number of open patterns.

After pattern p is removed from set O, its target pattern group g is found or, if it does not exist, created (line 23). Then, p is checked for dominance against all patterns in $P(g)$. If p is found to be dominated, it is discarded. Otherwise, it is added to $P(g)$, and then new patterns involving p are generated in lines 26–44.

To generate new patterns, the algorithm iterates over those pattern groups $g_2 \in G$ for which item requirements are not exceeded when p is combined with $q \in P(g_2)$ (line 26). For a selected g_2, we construct new patterns using horizontal and vertical builds between p and $q \in P(g_2)$, and save them in the set of candidate patterns, C (lines 27–33). Only patterns that fit in the stock rectangle, do not exceed wasteMax, are not dominated by another pattern in C, and are not discarded using dual-feasible functions, are saved in C (line 31). Note that all candidate patterns generated for a selected g_2 use the same multiset of items, $c(p) + c(g_2)$. If the pattern group, g_3, for this multiset exists, each candidate pattern in C is checked for dominance against patterns in $P(g_3)$ and discarded if found dominated (lines 35–36). In lines 37–41, the heuristic is invoked starting from each candidate pattern remaining in set C, in an attempt to construct a complete pattern, in a manner similar to that in lines 3–9. Finally, the candidate patterns remaining in C are added to the set of open patterns.

2.9. Data Structures

Appropriate data structures are necessary for the algorithm to perform efficiently. Open patterns in set O are organized in a heap, which allows for addition and removal of patterns in logarithmic time. For each open pattern, we store its rectangle, its waste, and the item index (for single-item patterns) or two pointers to subpatterns (for composite patterns). Note that we do not store the type of build, as it can easily be deduced from the rectangle of the pattern and the rectangles of its subpatterns. For an accepted pattern, we also store the pointer to the appropriate pattern group. This pointer is not stored for open patterns to delay or avert creation of some pattern groups.

The multiset of items for an accepted pattern can be obtained from its pattern group. For an open pattern, the multiset can be obtained by combining multisets

Input: Stock rectangle $[W, H]$, number of item types n, item rectangles $[w_i, h_i]$, and item requirements b_i, $\forall i \in N = \{1, \dots, n\}$
Output: Complete pattern if the problem is feasible; otherwise, infeasible status

1: **if** infeasibility can be proven using dual-feasible functions **then return** infeasible;
2: $G \leftarrow \varnothing$; $O \leftarrow \varnothing$; $n^{\text{fail}} \leftarrow 1$; $n^{\text{open}} \leftarrow 0$; wasteMax $\leftarrow WH$;
3: **foreach** $i \in N$ **do**
4: $\quad p \leftarrow$ single-item pattern for item i;
5: $\quad$ **if** p cannot lead to a complete pattern **then return** infeasible;
6: $\quad$ Construct pattern q heuristically starting from pattern p;
7: $\quad$ **if** q is a complete pattern **then return** q;
8: $\quad n^{\text{fail}} \leftarrow \max\{n^{\text{fail}}, \min\{j \in N\colon c_j(q) < b_j\}\}$;
9: **end**
10: **while** true **do**
11: $\quad$ **if** $n^{\text{open}} < n^{\text{fail}}$ **then**
12: $\quad\quad$ **foreach** $i \in \{n^{\text{open}} + 1, \dots, n^{\text{fail}}\}$ **do**
13: $\quad\quad\quad$ Add to O a single-item pattern for item i;
14: $\quad\quad\quad$ wasteMax $\leftarrow$ wasteMax $- w_i h_i b_i$;
15: $\quad\quad$ **end**
16: $\quad\quad n^{\text{open}} \leftarrow n^{\text{fail}}$;
17: $\quad\quad$ Remove from O and from $P(g)$ $\forall g \in G$ all patterns p such that waste$(p) >$ wasteMax or p cannot lead to a feasible pattern including all items of types 1 to n^{open};
18: $\quad\quad$ Remove from G all groups g for which $P(g) = \varnothing$;
19: $\quad$ **end**
20: $\quad$ **if** $O = \varnothing$ **then return** infeasible;
21: $\quad$ Identify pattern $p \in O$ with the smallest waste, breaking ties by the largest rectangle area and then by the largest shorter rectangle edge;
22: $\quad$ Remove p from O;
23: $\quad$ Find group $g \in G$ such that $c(g) = c(p)$; if not found, create such a group in G;
24: $\quad$ **if** p is not dominated by any pattern in $P(g)$ **then**
25: $\quad\quad P(g) \leftarrow P(g) \cup \{p\}$;
26: $\quad\quad$ **foreach** group $g_2 \in G$: $c(p) + c(g_2) \leqq b$ **do**
27: $\quad\quad\quad C \leftarrow \varnothing$;
28: $\quad\quad\quad$ **foreach** pattern $q \in P(g_2)$: waste(p) + waste$(q) \leq$ wasteMax **do**
29: $\quad\quad\quad\quad$ **foreach** buildType $\in$ {horizontal, vertical} **do**
30: $\quad\quad\quad\quad\quad$ Construct pattern r through buildType of p and q;
31: $\quad\quad\quad\quad\quad$ **if** $[w(r), h(r)] \leqq [W, H]$ **and** waste$(r) \leq$ wasteMax **and** r is not dominated by any pattern in C **and** p can lead to a feasible pattern including all items of types 1 to n^{open} **then** $C \leftarrow C \cup \{r\}$;
32: $\quad\quad\quad\quad$ **end**
33: $\quad\quad\quad$ **end**
34: $\quad\quad\quad$ **if** $C \neq \varnothing$ **then**
35: $\quad\quad\quad\quad$ Find group $g_3 \in G$ such that $c(g_3) = c(p) + c(g_2)$;
36: $\quad\quad\quad\quad$ **if** found remove from C all patterns dominated by any pattern in $P(g_3)$;
37: $\quad\quad\quad\quad$ **foreach** pattern $r \in C$ **do**
38: $\quad\quad\quad\quad\quad$ Construct pattern s heuristically starting from pattern r;
39: $\quad\quad\quad\quad\quad$ **if** s is a complete pattern **then return** s;
40: $\quad\quad\quad\quad\quad n^{\text{fail}} \leftarrow \max\{n^{\text{fail}}, \min\{j \in N\colon c_j(s) < b_j\}\}$;
41: $\quad\quad\quad\quad$ **end**
42: $\quad\quad\quad\quad O \leftarrow O \cup C$;
43: $\quad\quad\quad$ **end**
44: $\quad\quad$ **end**
45: $\quad$ **end**
46: **end**

Figure 2 Pseudocode of the Exact Algorithm for 2GC | O

of the groups of the two subpatterns, both of which must be among the accepted patterns, or by taking the single item for a single-item pattern.

Pattern groups are stored in a rooted tree, with each node apart from the root corresponding to a pattern group. For each group, we store in the corresponding node the list of patterns included in the group as well as the index and the count for one item type. To avoid redundancy, the index stored in a subnode is always greater than the index in its parent node. The multiset of items of a pattern group can be decoded by following the path from the node representing the group to the root. Such organization speeds up the search for a particular group in lines 23 and 35 as well as the enumeration of groups that do not conflict with a given multiset of items in line 26 of the algorithm. The disadvantage is that some pattern groups might be created even if no patterns for them exist.

Patterns in a group are ordered by increasing widths of rectangles. Because only undominated patterns are

stored in groups, widths of patterns in a group must be strictly increasing, i.e., no two widths can be equal, and heights must be strictly decreasing. If candidate patterns in set C are ordered in the same way, checking for dominance between candidate patterns and accepted patterns (line 36) can be performed in linear time.

2.10. Memory Requirements

Memory requirements of the algorithm can be estimated as follows: The number of pattern groups is limited by the number of possible multisets of items, which is $\prod_{i\in N}(b_i+1)$. If all items are unique, i.e., if $b_i = 1$, $\forall\, i \in N$, the formula simplifies to 2^n. Because all patterns in each group must have different widths and different heights, there can be at most $\min\{W, H\}$ patterns in each group. Hence, the total number of accepted patterns is at most $\min\{W, H\}\prod_{i\in N}(b_i+1)$ or, if all items are unique, $\min\{W, H\}2^n$. Because open patterns are generated through builds of two accepted patterns, the number of open patterns generated in the algorithm is at most $2(\min\{W, H\}\prod_{i\in N}(b_i+1))^2$ or, if all items are unique, $2(\min\{W, H\}2^n)^2$.

3. Algorithm for 2GC|R

The algorithm for 2GC|O presented in Section 2 can be adapted to the case of the problem with 90-degree item rotation through several modifications.

First, we assume that the rectangles of all items, the stock rectangle, and all patterns are *oriented horizontally*, i.e., the width is greater than or equal to the height. To achieve this, all patterns generated in the algorithm are oriented horizontally as soon as they are constructed. With such reorientation, checking if a pattern fits in the stock rectangle, or if it is dominated by another pattern, can be done for the problem with item rotation using the same conditions as in the problem with fixed item orientation. Note that the reorientation of a pattern is performed, if necessary, by simply swapping its width with its height. This makes decoding patterns a bit more involved than in the case without rotation, but the type of build of a pattern and any rotations performed can still be deduced from the rectangle of the pattern and the rectangles of its subpatterns.

Second, two new build types are considered in addition to the horizontal and vertical builds in line 29 of the algorithm, i.e., the horizontal build with the first pattern rotated by 90 degrees and the vertical build with the first pattern rotated by 90 degrees. Note that it is not necessary to consider any other types of builds, such as those with the second pattern or both patterns rotated by 90 degrees, as they will generate the same four composite patterns again, after all patterns are oriented horizontally.

Third, when a heuristic attempt is made to complete a partial pattern in lines 6 and 38, we add another attempt in which the partial pattern is rotated by 90 degrees. In the second attempt, the heuristic is executed only if the rotated partial pattern fits in the stock rectangle.

Finally, we remove the attempt to prove infeasibility using dual-feasible functions in line 1, which are only valid for the oriented case. Also, we do not use dual-feasible functions when we test if a partial pattern, p, can lead to a complete pattern in lines 5, 17, and 31. Instead, we orient rectangles $R_{\text{right}}(p)$ and $R_{\text{above}}(p)$ horizontally, and then check if every item $i \in \{1, \dots, n^{\text{open}}\}$ not included in p can be cut from $R_{\text{right}}(p)$ or $R_{\text{above}}(p)$. If an item cannot be cut from either, the partial pattern cannot be completed (but the pattern is not yet discarded). Otherwise, two multisets of items, $c_{\text{right}}(p)$ and $c_{\text{above}}(p)$, are constructed, as before. For each of the two resulting subproblems, we test feasibiility by comparing the total area of the multiset with the area of the rectangle. If any of the two subproblems is infeasible, then pattern p cannot be completed. When pattern p cannot be completed, we also need to check if pattern p after rotation also fits in the stock rectangle. If it does not, the pattern can be discarded. If it does, the same test is performed for the rotated p and the pattern is discarded only if it cannot be completed in both orientations.

Memory requirements of the algorithm for 2GC|R remain largely the same as for 2GC|O. The difference is that 2 in the formula estimating the number of open patterns is replaced by 4, due to the increased number of build types.

4. Computational Experiments

Our algorithm is coded in C++ and compiled using Microsoft Visual Studio 2013. Tests are run on a PC with Intel Core i7-3770 3.4 GHz processor and 32 GB RAM.

We test our algorithm for 2GC|O (resp. 2GC|R) in the same way as Clautiaux et al. (2013), i.e., by embedding it in a simple procedure for solving 2GSC|O (resp. 2GSC|R) that varies the height of the strip and tests if a feasible solution exists for each height. One of our algorithms, denoted A_{ub}, starts from an upper bound on the height and decreases it until the problem becomes infeasible. The other, denoted A_{lb}, starts from a lower bound on the height and increases it as long as the problem is infeasible.

Clautiaux et al. (2013) used upper bound $H_{\max} = \sum_{i\in N} h_i b_i$ and lower bound $H_{\min} = \lceil(\sum_{i\in N} w_i h_i b_i)/W\rceil$. We use a stronger upper bound obtained by running the finite-best-strip heuristic (FBS) of Berkey and Wang (1987) when solving 2GSC|O and its adaptation to the

Table 2 Summary of Results on Various Benchmark Data Sets

							A_{ub}			A_{lb}		
Problem	Set	$\sum b_i$	n	W	H_{opt}	No.	Opt	T_{avg}	T_{max}	Opt	T_{avg}	T_{max}
2GSC|O	ccm	10–23	10–19	20	20–25	42	42	0.02	0.14	42	0.02	0.13
	hifi	7–22	4–14	4–100	13–145	25	25	0.01	0.05	25	0.02	0.08
	ngcut	7–22	5–10	10–30	20–87	12	12	0.03	0.15	12	0.06	0.37
	beng	20–200	18–84	25–40	31–156	10	10	7.29	64.49	10	7.28	65.58
	cgcut	16–62	7–19	10–70	24–666	3	2	1,201.06	TL	2	1,201.23	TL
	gcut	10–50	10–50	250–3,000	1,016–13,394	13	6	2,150.27	TL	3	2,770.15	TL
	ht	16–197	14–169	20–160	15–244	21	9	2,089.99	TL	9	2,090.52	TL
	bkw	10–3,152	9–161	30–640	40–967	13	5	2,217.57	TL	5	2,218.02	TL
2GSC|R	ccm	10–23	9–18	20	17–21	42	42	0.01	0.03	42	0.01	0.03
	hifi	7–22	4–14	4–100	10–140	25	25	0.01	0.09	25	0.02	0.09
	ngcut	7–22	5–10	10–30	10–78	12	12	0.02	0.14	12	0.02	0.11
	beng	20–200	18–61	25–40	30–156	10	10	0.01	0.03	10	0.01	0.04
	cgcut	16–62	7–19	10–70	23–644	3	2	1,200.16	TL	2	1,200.14	TL
	gcut	10–50	10–50	250–3,000	706–15,675	13	6	2,061.50	TL	3	2,770.03	TL
	ht	16–197	11–153	20–160	15–243	21	14	1,322.48	TL	14	1,299.46	TL
	bkw	10–3,152	9–140	30–640	40–964	13	4	2,712.53	TL	4	2,710.39	TL

case with rotation proposed by Lodi et al. (1998) when solving 2GSC|R. This upper bound is denoted H_{FBS}. We also use a slightly stronger lower bound $H'_{min} = \max\{H_{min}, \max_{i\in N} h_i\}$. Note that to use this lower bound for 2GSC|R, each item must first be oriented horizontally, provided that its width after reorientation does not exceed the strip width.

4.1. Summary of the Results for 2GSC|O and 2GSC|R

Table 2 presents a summary of the results obtained by our algorithms A_{ub} and A_{lb} executed with a 1-hour time limit (TL), on the following benchmark data sets: ccm of Clautiaux et al. (2007), hifi of Hifi (1998), ngcut of Beasley (1985b), beng of Bengtsson (1982), cgcut of Christofides and Whitlock (1977), gcut of Beasley (1985a), ht of Hopper and Turton (2001), and bkw of Burke et al. (2004). The first column indicates the problem type and the second the benchmark data set. Then, for each data set, the minimum and maximum of the following measures are reported: the number of items $\sum b_i$; the number of item types n; the width of the strip W; and the optimal height of the strip H_{opt}. Also, the number of instances in each data set are reported in column "No." Finally, for each of our algorithms, we report the number of instances solved to optimality in the time limit, "Opt," as well as the average T_{avg} and maximum T_{max} processing times, in seconds.

Algorithms A_{ub} and A_{lb} solve the same number of instances to optimality in each data set except for set gcut, for which the former solves three more instances than the latter, both for 2GSC|O and 2GSC|R.

All instances of the first three data sets, ccm, hifi, and ngcut, are solved to optimality by A_{ub} and A_{lb} in less than one second each, for both 2GSC|O and 2GSC|R. These instances are easy for our algorithms because they have relatively few items (at most 23) and item types (at most 19), which limits the number of all possible multisets (less than 1E+6), and because the strip and item dimensions are relatively small.

To analyze the results of the remaining data sets, more detailed results are reported in Table 3 for 2GSC|O and in Table 4 for 2GSC|R. For each instance, we report: the number of items $\sum b_i$; the number of item types n; the width of the strip W; the lower bound H'_{min}, and upper bound H_{FBS} on the strip height; the optimal height H_{opt} (if optimality is not proven, then the proven lower and upper bounds are reported); and the number of all possible multisets of items, calculated as $\prod_{i\in N}(b_i + 1)$. The following four columns refer to algorithm A_{ub} and show the number of created pattern groups, the number of accepted patterns, the number of constructed open patterns, and the peak number of open patterns stored in set O. The remaining two columns report the processing times of A_{ub} and A_{lb} in seconds.

4.2. Analysis of Detailed Results for 2GSC|O

For 2GSC|O (Table 3), A_{ub} and A_{lb} solve all instances of set beng. These instances have more items (up to 200) and item types (up to 84) than the first three data sets. As a result, they also have much larger numbers of all possible multisets. However, for all but the smallest of these instances, $H_{opt} = H'_{min}$, and our heuristic is capable of finding feasible solutions for these heights in relatively short times.

The first two instances of set cgcut fall into the easy category, as they have few items and item types, and relatively small item and strip dimensions. The third instance has 62 items; although they are grouped in only 19 item types, neither of our algorithms can prove optimality of this instance in the 1-hour time limit. Nevertheless, A_{lb} can increase the lower bound

Table 3 Detailed Results on Harder Benchmark Data Sets for 2GSC | O

Instance	$\sum b_i$	n	W	$H'_{\min}$	H_{FBS}	H_{opt}	Multisets	Pattern groups	Accepted patterns	Open patterns		T (seconds)	
										Total	Peak	A_{ub}	A_{lb}
beng01	20	18	25	30	36	31	5.90E+05	20,141	17,514	65,644	32,635	1.21	1.19
beng02	40	31	25	57	62	57	6.52E+10	92,691	21,550	166,413	111,719	2.41	2.43
beng03	60	46	25	84	91	84	1.62E+16	34,117	8,480	75,745	62,157	1.46	1.46
beng04	80	56	25	107	113	107	5.98E+20	1,029,033	219,215	334,982	98,556	64.49	65.58
beng05	100	63	25	134	139	134	9.31E+24	360	238	3,653	2,581	0.13	0.10
beng06	40	31	40	36	42	36	6.52E+10	91	81	720	498	0.02	0.01
beng07	80	56	40	67	73	67	5.98E+20	33,028	8,684	67,921	42,390	1.96	1.98
beng08	120	68	40	101	108	101	4.71E+28	3,095	1,190	26,390	22,775	1.04	1.02
beng09	160	76	40	126	132	126	1.86E+35	70	90	843	306	0.08	0.03
beng10	200	84	40	156	160	156	7.95E+41	79	111	990	317	0.10	0.03
cgcut1	16	7	10	23	28	24	2.59E+03	206	276	609	315	0.03	0.02
cgcut2	23	10	70	63	83	65	9.83E+04	29,356	34,465	163,286	59,792	2.97	2.98
cgcut3	62	19	70	636	728	647–666	5.76E+11	529,735	202,512	314,938,129	208,412,961	TL	TL
gcut01	10	10	250	655	1,016	1,016	1.02E+03	254	254	2,907	1,265	0.01	1.17
gcut02	20	20	250	1,099	1,349	1,227	1.05E+06	700,487	1,056,846	43,400,274	33,509,804	683.68	TL
gcut03	30	30	250	1,631	1,810	1,755–1,803	1.07E+09	328,709	455,488	724,605,364	689,596,940	TL	TL
gcut04	50	50	250	2,926	3,216	2,932–3,069	1.13E+15	341,043	135,101	335,112,311	165,743,903	TL	TL
gcut05	10	10	500	1,091	1,572	1,297	1.02E+03	543	904	2,701	973	0.04	1.58
gcut06	20	20	500	2,465	3,139	2,650	1.05E+06	276,723	355,338	9,382,909	6,961,283	111.14	TL
gcut07	30	30	500	4,010	5,151	4,500–4,788	1.07E+09	219,033	186,886	601,388,290	598,953,534	TL	TL
gcut08	50	50	500	5,611	6,496	5,632–6,036	1.13E+15	398,425	152,631	340,290,435	75,070,146	TL	TL
gcut09	10	10	1,000	2,022	2,613	2,303	1.02E+03	334	419	1,062	322	0.03	2.33
gcut10	20	20	1,000	5,356	6,869	6,463	1.05E+06	681,255	1,345,067	228,239,522	167,358,385	1,951.09	TL
gcut11	30	30	1,000	6,537	8,356	7,374–7,736	1.07E+09	294,803	310,983	636,717,834	631,808,331	TL	TL
gcut12	50	50	1,000	12,522	14,231	12,785–13,394	1.13E+15	419,135	177,461	337,330,688	96,764,943	TL	TL
gcut13	32	32	3,000	4,772	5,973	4,772–5,036	4.29E+09	737,945	405,349	524,838,413	198,090,233	TL	TL
ht_lw161	16	16	20	20	25	20	6.55E+04	34	24	82	32	0.01	0.01
ht_lw163	16	14	20	20	28	20	3.69E+04	38	35	112	62	0.01	0.01
ht_lw172	17	15	20	20	29	21	7.37E+04	1,722	1,223	4,511	2,967	0.03	0.04
ht_lw251	25	22	40	15	17	15	1.42E+07	470	253	2,084	1,734	0.02	0.02
ht_lw252	25	21	40	15	19	16	1.06E+07	137,402	32,439	42,146	9,619	3.28	3.05
ht_lw253	25	19	40	15	16	15	5.97E+06	78	61	124	42	0.01	0.01
ht_lw281	28	26	60	30	39	30	1.51E+08	16,215	3,575	8,872	5,208	0.15	0.15
ht_lw283	28	28	60	30	39	30	2.68E+08	94	80	716	278	0.02	0.01
ht_lw292	29	28	60	30	36	31	4.03E+08	1,253,289	527,366	3,277,277	2,113,845	651.04	661.27
ht_lw491	49	47	60	60	78	60–61	3.17E+14	14,993,105	1,560,428	11,928,452	8,874,319	TL	TL

Table 3 (Continued)

Instance	$\sum b_i$	n	W	$H'_{\min}$	H_{FBS}	H_{opt}	Multisets	Pattern groups	Accepted patterns	Open patterns		T (seconds)	
										Total	Peak	A_{ub}	A_{lb}
ht_lw492	49	46	60	60	78	60–61	2.37E+14	10,301,210	1,071,917	20,283,811	19,059,660	TL	TL
ht_lw493	49	42	60	60	72	60–61	7.51E+13	14,256,324	1,754,857	3,659,293	1,844,705	TL	TL
ht_lw731	73	58	60	90	110	90–91	1.12E+20	15,010,411	1,982,883	20,805,740	18,485,323	TL	TL
ht_lw732	73	60	60	90	113	90–91	1.77E+20	14,435,304	1,550,359	25,978,368	16,858,425	TL	TL
ht_lw733	73	61	60	90	112	90–91	2.36E+20	13,761,165	1,460,942	14,731,468	11,043,234	TL	TL
ht_lw971	97	79	80	120	159	120–121	7.06E+26	14,634,733	1,288,144	26,281,960	16,084,901	TL	TL
ht_lw972	97	83	80	120	152	120–121	1.98E+27	13,438,315	913,014	14,370,640	10,876,647	TL	TL
ht_lw973	97	75	80	120	158	120–121	1.31E+26	8,649,557	715,724	21,496,623	18,158,005	TL	TL
ht_lw1961	196	167	160	240	284	240–244	8.85E+54	3,321,375	251,397	27,985,569	21,413,896	TL	TL
ht_lw1963	196	169	160	240	291	240–243	1.38E+55	5,224,127	357,203	30,629,638	26,672,111	TL	TL
ht_lw1972	197	161	160	240	305	240–242	1.63E+54	6,682,558	382,910	25,277,867	22,125,472	TL	TL
bkw_n1	10	9	40	40	46	40	7.68E+02	0	0	0	0	0.00	0.00
bkw_n2	20	18	30	50	65	50	5.90E+05	301	130	2,326	2,194	0.02	0.01
bkw_n3	30	23	30	50	68	50	1.01E+08	73,427	18,517	75,444	36,025	1.47	1.25
bkw_n4	40	28	80	80	126	80	9.51E+09	79,164	21,505	21,989	324	2.44	3.99
bkw_n5	50	39	100	100	119	100–102	3.13E+13	6,927,844	1,163,178	101,264,493	95,234,731	TL	TL
bkw_n6	60	33	50	100	109	100–101	3.08E+13	9,565,977	2,006,563	2,283,002	221,773	TL	TL
bkw_n7	70	43	80	100	160	100	3.90E+15	182,867	33,923	39,907	4,229	3.32	3.17
bkw_n8	80	55	100	80	108	80–81	7.39E+19	12,532,369	1,104,860	14,549,643	8,197,810	TL	TL
bkw_n9	100	48	50	150	177	150–151	7.07E+19	19,645,390	2,110,099	3,465,807	1,158,218	TL	TL
bkw_n10	200	70	70	150	191	150–151	5.19E+28	16,839,290	1,576,501	1,582,119	26,316	TL	TL
bkw_n11	300	97	70	150	169	150–152	4.38E+41	9,870,046	831,496	19,207,704	18,344, 296	TL	TL
bkw_n12	500	131	100	300	372	300–306	2.40E+57	1,551,402	124,445	11,510,188	9,453,753	TL	TL
bkw_n13	3,152	161	640	960	1,003	960–967	1.61E+207	5,264	4,146	538,939	394,885	TL	TL

Table 4 Detailed Results on Harder Benchmark Data Sets for 2GSC|R

Instance	$\sum b_i$	n	W	$H'_{\min}$	H_{FBS}	H_{opt}	Multisets	Pattern groups	Accepted patterns	Open patterns		T (seconds)	
										Total	Peak	A_{ub}	A_{lb}
beng01	20	18	25	30	36	30	5.90E+05	14	14	42	28	0.01	0.01
beng02	40	29	25	57	65	57	3.26E+10	18	21	76	45	0.01	0.01
beng03	60	39	25	84	87	84	1.43E+15	11	13	65	52	0.01	0.01
beng04	80	47	25	107	110	107	2.63E+19	498	309	497	182	0.03	0.04
beng05	100	53	25	134	139	134	1.82E+23	0	0	0	0	0.00	0.00
beng06	40	29	40	36	39	36	3.26E+10	0	0	0	0	0.00	0.00
beng07	80	47	40	67	72	67	2.63E+19	15	16	98	82	0.01	0.01
beng08	120	55	40	101	104	101	1.84E+26	0	0	0	0	0.00	0.00
beng09	160	57	40	126	129	126	1.86E+31	0	0	0	0	0.00	0.00
beng10	200	61	40	156	159	156	3.53E+36	0	0	0	0	0.00	0.00
cgcut1	16	7	10	23	26	23	2.59E+03	0	0	0	0	0.00	0.00
cgcut2	23	10	70	63	74	63	9.83E+04	999	510	6,055	3,046	0.11	0.06
cgcut3	62	19	70	636	703	636–644	5.76E+11	1,472,517	310,474	88,931,909	80,738,963	TL	TL
gcut01	10	10	250	655	852	706	1.02E+03	812	1,653	8,112	1,972	0.06	0.57
gcut02	20	20	250	1,099	1,510	1,136	1.05E+06	620,987	828,699	11,748,034	6,120,463	664.73	TL
gcut03	30	30	250	1,631	2,084	1,631–1,673	1.07E+09	838,996	387,306	366,694,995	253,227,240	TL	TL
gcut04	50	50	250	2,926	3,477	2,926–3,004	1.13E+15	597,476	187,808	285,862,276	83,124,209	TL	TL
gcut05	10	10	500	1,091	1,404	1,159	1.02E+03	894	1,299	5,829	1,244	0.08	0.75
gcut06	20	20	500	2,465	3,403	2,550	1.05E+06	677,822	632,537	9,428,747	4,478,542	374.33	TL
gcut07	30	30	500	4,010	4,974	4,010–4,190	1.07E+09	716,517	355,531	518,009,487	138,609,444	TL	TL
gcut08	50	50	500	5,611	6,787	5,611–5,789	1.13E+15	435,546	154,943	254,290,795	93,084,551	TL	TL
gcut09	10	10	1,000	2,022	2,426	2,215	1.02E+03	1,367	3,370	18,443	5,785	0.09	3.76
gcut10	20	20	1,000	5,356	6,877	5,571	1.05E+06	770,914	974,225	22,687,034	11,394,763	558.47	TL
gcut11	30	30	1,000	6,537	8,646	6,537–6,746	1.07E+09	879,777	446,999	378,095,214	188,728,395	TL	TL
gcut12	50	50	1,000	12,522	15,675	12,522–12,981	1.13E+15	368,558	150,570	309,410,614	69,834,665	TL	TL
gcut13	32	32	3,000	4,772	5,147	4,772–4,894	4.29E+09	699,934	264,946	270,551,622	102,886,390	TL	TL
ht_lw161	16	16	20	20	22	20	6.55E+04	28	23	59	36	0.01	0.01
ht_lw163	16	13	20	20	23	20	2.76E+04	130	86	106	21	0.01	0.01
ht_lw172	17	11	20	20	24	20	2.07E+04	255	142	473	328	0.01	0.01
ht_lw251	25	22	40	15	18	15	1.42E+07	65	51	182	131	0.01	0.01
ht_lw252	25	19	40	15	19	15	5.31E+06	9	9	26	19	0.01	0.01
ht_lw253	25	19	40	15	16	15	5.97E+06	0	0	0	0	0.00	0.00
ht_lw281	28	23	60	30	36	30	6.37E+07	6,360	2,201	4,594	2,427	0.07	0.07
ht_lw283	28	26	60	30	34	30	1.51E+08	2,238	725	2,271	1,039	0.03	0.04
ht_lw292	29	27	60	30	35	30	3.02E+08	212,356	45,145	74,460	25,029	3.49	3.03
ht_lw491	49	45	60	60	67	60–61	1.78E+14	34,202,993	7,028,696	7,292,650	222,761	TL	TL
ht_lw492	49	42	60	60	65	60	7.51E+13	210,239	39,930	127,285	86,428	4.59	3.68
ht_lw493	49	39	60	60	66	60	2.82E+13	10,628,395	2,204,394	2,208,896	22,524	1,199.19	918.31
ht_lw731	73	55	60	90	97	90	3.12E+19	7,317,710	1,505,591	2,118,924	307,336	437.87	345.99

Table 4 (Continued)

Instance	$\sum b_i$	n	W	$H'_{\min}$	H_{FBS}	H_{opt}	Multisets	Pattern groups	Accepted patterns	Open patterns		T (seconds)	
										Total	Peak	A_{ub}	A_{lb}
ht_lw732	73	52	60	90	96	90	8.18E+18	8,649	1,766	13,105	7,061	0.55	0.52
ht_lw733	73	55	60	90	97	90	2.19E+19	12,318,884	2,535,526	4,181,902	1,444,589	853.75	745.23
ht_lw971	97	75	80	120	128	120–121	1.57E+26	32,278,228	4,105,788	12,977,310	5,061,987	TL	TL
ht_lw972	97	74	80	120	128	120–121	7.84E+25	25,308,601	2,722,179	6,981,638	2,595,537	TL	TL
ht_lw973	97	67	80	120	129	120–121	5.11E+24	16,827,945	2,095,670	18,979,991	14,971,790	TL	TL
ht_lw1961	196	153	160	240	252	240–243	5.47E+52	2,049,896	214,183	14,179,824	10,865,592	TL	TL
ht_lw1963	196	153	160	240	253	240–242	8.07E+52	3,562,072	289,785	14,727,272	14,358,888	TL	TL
ht_lw1972	197	140	160	240	252	240–241	3.67E+50	3,490,678	307,656	12,770,787	10,503,116	TL	TL
bkw_n1	10	9	40	40	50	40	7.68E+02	0	0	0	0	0.00	0.00
bkw_n2	20	16	30	50	56	50	2.46E+05	1,204	541	2,468	795	0.55	0.01
bkw_n3	30	21	30	50	68	50	3.77E+07	30,002	9,128	11,078	1,131	0.50	0.24
bkw_n4	40	27	80	80	94	80–81	7.13E+09	11,150,512	4,069,444	4,079,714	11,601	TL	TL
bkw_n5	50	35	100	100	112	100	4.70E+12	12,320,852	2,113,789	20,368,078	17,258,975	2,823.18	2,780.61
bkw_n6	60	29	50	100	109	100–101	1.71E+12	13,065,905	3,934,759	3,955,592	14,647	TL	TL
bkw_n7	70	42	80	100	154	100–101	2.60E+15	13,822,105	1,686,086	1,688,926	6,004	TL	TL
bkw_n8	80	48	100	80	89	80–81	3.90E+18	18,853,441	2,787,530	12,733,254	9,928,603	TL	TL
bkw_n9	100	43	50	150	172	150–151	1.14E+18	17,021,967	2,406,256	2,491,755	73,064	TL	TL
bkw_n10	200	60	70	150	163	150–151	6.12E+25	13,714,831	1,877,158	1,897,537	110,600	TL	TL
bkw_n11	300	79	70	150	159	150–151	2.23E+36	18,472,798	2,312,198	4,447,002	2,131,161	TL	TL
bkw_n12	500	109	100	300	384	300–306	1.70E+50	614,651	65,020	5,299,658	3,946,698	TL	TL
bkw_n13	3,152	140	640	960	973	960–964	2.08E+185	2,949	2,312	202,077	162,061	TL	TL

for this instance from $H'_{\min} = 636$ to 647 and A_{ub} can decrease the upper bound from $H_{FBS} = 728$ to 666. Note that when solving this instance, our algorithms generate very large numbers of open patterns. When we tried running our algorithms for five hours, the number of open patterns stored in set O exceeded 1 billion, and our programs were stopped with the "out of memory" error message.

In set gcut, A_{ub} solves to optimality only six instances with 10 or 20 items each, and A_{lb} solves to optimality only three instances with 10 items each. The difficulty of set gcut comes from the fact that the strip and item dimensions are relatively large. This makes even the 20-item instances quite challenging for our algorithms.

For set ht, both our algorithms solve the first nine instances to optimality. The first six fall into the easy category (i.e., few items and small dimensions). The next two are solved quickly because $H_{opt} = H'_{\min}$ for both of them, and because our heuristic finds feasible patterns for these heights relatively easily. The last of the nine instances has $H_{opt} = H'_{\min} + 1$ and both our algorithms take more than 10 minutes to solve it. The remaining 12 instances of set ht have 49 and more items. Both our algorithms fail to solve these instances to optimality in the 1-hour time limit. However, the gaps obtained by A_{ub} are very narrow.

In set bkw, five instances are solved to optimality by A_{ub} and A_{lb}. For the remaining instances, the optimality gaps are quite narrow. Note that the largest instance in this set has 3,152 items and 161 item types. For this instance, A_{ub} can decrease the upper bound from $H_{FBS} = 1{,}003$ to 967, which is very close to the lower bound $H'_{\min} = 960$.

4.3. Analysis of Detailed Results for 2GSC|R

For 2GSC|R (Table 4), A_{ub} and A_{lb} solve to optimality all instances of set beng, each in a fraction of a second. For each instance, $H_{opt} = H'_{\min}$, and our heuristic quickly finds optimal solutions.

For sets cgcut and gcut, the results for 2GSC|R are very similar to those for 2GSC|O. The first two instances of cgcut are solved to optimality very quickly; the third is not solved to optimality, but the remaining gap is small. In set gcut, A_{ub} solves to optimality only six instances with 10 or 20 items each, and A_{lb} solves to optimality only three instances with 10 items each. As before, set gcut is difficult due to relatively large strip and item dimensions.

For set ht, our algorithms perform better for 2GSC|R than for 2GSC|O, solving to optimality 14 of 21 instances, including the nine instances solved to optimality for 2GSC|O. In all the remaining instances, the gaps are very small. Note that for all instances solved to optimality in set ht, in the case with rotation, $H_{opt} = H_{\min}$, whereas in the case without rotation, $H_{opt} = H_{\min}$ or $H_{\min} + 1$. It appears that, for this set, it is relatively easier to find solutions with $H_{opt} = H_{\min}$ when rotation is allowed. Hence, our algorithms can solve to optimality more instances in the case with rotation.

For set bkw, our algorithms perform slightly worse for 2GSC|R than for 2GSC|O, solving to optimality only four of 13 instances. However, the gaps of the unsolved instances are very small. Surprisingly, both our algorithms for 2GSC|R fail to prove feasibility of $H = 80$ for instance bkw_n4 and $H = 100$ for instance bkw_n7, which are proven optimal values for 2GSC|O. Because 2GSC|R is a relaxation of 2GSC|O, these values must be feasible for 2GSC|R. Because they also coincide with the lower bounds, they must also be optimal.

4.4. Comparison with Previous Methods for 2GSC|O

Table 5 compares performance of A_{ub} and A_{lb} on data set ccm with three algorithms of Clautiaux et al. (2013), denoted IGG_{ub}, IGG_{lb}, and $2SP{+}IGG_{lb}$. These algorithms solve 2GSC|O in the same way as A_{ub} and A_{lb}, i.e., by testing feasibility while varying strip height. They start from, respectively, an upper bound $H_{\max}$, a lower bound $H_{\min}$, and a lower bound obtained by solving exactly the nonguillotine relaxation of 2GSC|O.

In addition to columns previously defined, Table 5 also reports: the ordinal number of each instance; the lower bound $H_{\min}$ and upper bound $H_{\max}$, on the strip height; and the processing times of the algorithms of Clautiaux et al. (2013). Additionally, since $W = 20$ for all instances in this set, column W is omitted. Note that the results for the last instance in set ccm are not provided in Clautiaux et al. (2013).

Despite the fact that our tests are performed on a somewhat faster computer than that used by Clautiaux et al. (2013) (Intel T2600 2.16 GHz), we can confidently claim that our algorithms are, on average, much more efficient than IGG_{ub}, IGG_{lb}, and $2SP{+}IGG_{lb}$. The best of the previously proposed algorithms, $2SP{+}IGG_{lb}$, needs hundreds or thousands of seconds for several instances, and it does not solve instances 10 and 15 to optimality in the 1-hour time limit. By contrast, our algorithms solve all instances to optimality in much less than 1 second per instance. Algorithms A_{ub} and A_{lb} also solve the most difficult instance, 4, which has the largest number of item types, in only 0.14 and 0.13 seconds, respectively. Note that even if we start our algorithms from the same upper and lower bounds as Clautiaux et al. (2013), the average and maximum computation times remain the same.

Table 6 compares performance of our algorithms A_{lb} and A_{ub} with algorithms IGG_{lb} from Clautiaux et al. (2013), BKCS from Bekrar et al. (2010), as well

Table 5 Comparison with Previously Proposed Algorithms for 2GSC | O on Set ccm

No.	Instance	$\sum b_i$	n	H_{min}	H'_{min}	H_{FBS}	H_{max}	H_{opt}	Multisets	Pattern groups	Accepted patterns	Open patterns		T (seconds)				
												Total	Peak	A_{ub}	A_{lb}	IGG_{ub}	IGG_{lb}	$2SP+IGG_{lb}$
1	E00X23	23	13	20	20	27	106	21	294,912	2,392	2,491	18,823	7,674	0.08	0.09	449.25	202.23	30.52
2	E00N23	23	14	20	20	27	108	21	307,200	69	78	103	37	0.01	0.01	35.17	31.48	0.19
3	E02F22	22	17	20	20	31	122	20	737,280	811	499	7,475	6,626	0.05	0.03	1,633.92	539.03	471.79
4	E02F20	20	19	20	20	29	126	21	786,432	5,935	6,087	18,885	9,920	0.14	0.13	TL	1,065.42	967.24
5	E02N20	20	16	20	20	33	134	22	245,760	77	94	123	54	0.01	0.01	TL	437.68	0.82
6	E04F20	20	16	20	20	29	142	20	294,912	1,260	703	13,534	11,949	0.06	0.05	333.62	2.07	2.16
7	E05F20	20	15	19	19	29	115	20	221,184	40	37	156	66	0.02	0.01	342.68	231.51	209.25
8	E04F19	19	18	20	20	27	117	20	393,216	1,455	1,045	8,105	6,311	0.04	0.04	125.48	106.45	93.00
9	E03X18	18	16	20	20	29	114	22	147,456	101	101	148	44	0.01	0.02	1,636.48	1,796.83	1,570.47
10	E04N18	18	13	20	20	35	124	25	49,152	192	210	468	106	0.03	0.03	TL	TL	TL
11	E05F18	18	14	19	19	29	119	21	82,944	102	103	158	69	0.01	0.01	183.04	108.42	95.83
12	E02F17	17	16	20	20	34	112	21	98,304	213	184	1,481	998	0.03	0.01	138.37	104.98	94.65
13	E03N17	17	13	20	20	34	105	22	30,720	23	23	23	11	0.01	0.01	10.95	10.04	8.36
14	E04F17	17	16	20	20	30	97	21	98,304	417	368	1,719	789	0.03	0.01	44.92	42.43	37.63
15	E04N17	17	15	20	20	29	141	23	73,728	73	75	107	43	0.01	0.02	TL	TL	TL
16	E05N17	17	14	20	20	36	115	22	49,152	85	96	128	43	0.01	0.01	317.81	273.43	247.47
17	E03N16	16	14	20	20	31	118	22	32,768	21	24	24	16	0.01	0.01	2.06	1.00	2.63
18	E00N15	15	15	20	20	32	87	22	32,768	269	317	1,285	806	0.02	0.02	22.31	3.09	2.71
19	E03N15	15	15	20	20	32	116	21	32,768	792	1,094	5,925	2,935	0.04	0.03	15.57	4.76	4.24
20	E04N15	15	14	20	20	30	127	24	24,576	777	938	2,956	754	0.02	0.04	10.45	8.67	8.25
21	E04F15	15	13	20	20	25	82	22	18,432	315	335	611	276	0.01	0.01	64.70	68.20	60.75
22	E05F15	15	14	19	19	30	76	22	24,576	667	778	1,648	682	0.01	0.03	17.87	21.18	18.91
23	E05N15	15	13	19	19	27	142	23	18,432	808	864	3,510	1,480	0.03	0.04	400.40	506.51	453.55
24	E05X15	15	14	19	19	24	97	21	24,576	244	305	566	287	0.01	0.01	46.45	37.54	15.19
25	E07X15	15	13	19	19	30	91	22	18,432	29	29	29	18	0.01	0.02	222.95	153.04	101.93
26	E07F15	15	14	19	20	33	118	21	24,576	163	174	461	105	0.02	0.01	6.23	4.15	3.79
27	E07N15	15	14	19	20	31	147	21	24,576	38	37	110	69	0.01	0.01	3.03	0.12	0.29
28	E08N15	15	12	19	20	34	113	21	13,824	27	28	58	10	0.02	0.00	384.78	82.42	76.54
29	E08F15	15	14	19	19	30	115	21	24,576	34	41	47	19	0.01	0.01	2.37	1.01	0.82
30	E10N15	15	13	18	20	31	165	22	16,384	31	31	75	33	0.01	0.01	1.92	0.23	0.68
31	E10X15	15	14	18	18	31	112	22	24,576	121	119	335	89	0.03	0.03	3.34	0.40	0.38
32	E13X15	15	12	18	18	29	106	22	12,288	21	22	25	13	0.01	0.01	7.43	7.14	0.13
33	E13N15	15	13	18	20	35	85	22	18,432	53	50	83	18	0.02	0.02	2.53	1.03	1.08
34	E15N15	15	13	18	20	31	143	22	16,384	52	60	107	22	0.02	0.02	19.44	4.31	4.66
35	E20F15	15	14	16	16	23	77	21	24,576	105	128	176	78	0.01	0.03	749.56	767.93	713.36
36	E20X15	15	14	17	20	28	93	21	24,576	17	18	30	7	0.02	0.01	1.84	0.10	0.07
37	E00N10	10	10	20	20	27	88	23	1,024	81	108	163	82	0.01	0.02	0.34	0.10	0.19
38	E03N10	10	10	20	20	28	62	24	1,024	92	108	278	95	0.01	0.03	0.39	0.06	0.15
39	E07N10	10	10	19	19	30	88	23	1,024	72	71	134	49	0.01	0.03	0.40	0.09	0.11
40	E10N10	10	10	18	18	28	86	25	1,024	41	54	67	33	0.01	0.03	0.39	0.15	0.19
41	E13N10	10	10	18	18	26	69	22	1,024	119	110	159	40	0.01	0.03	0.45	0.12	0.24
42	E15N10	10	10	17	17	27	75	23	1,024	45	51	66	30	0.01	0.03	Missing	Missing	Missing

Notes. A_{ub} and A_{lb} were tested on Intel i7-3770 3.4 GHz. IGG_{ub}, IGG_{lb}, and $2SP+IGG_{lb}$ of Clautiaux et al. (2013) were tested on Intel T2600 2.16 GHz with TL = 3,600 seconds.

as BMVB and IMVB from Hifi (1998), on data set hifi. This benchmark set is easy for algorithms A_{ub}, A_{lb}, and IGG_{lb}. Note that A_{ub} and A_{lb} solve six of the 25 instances without reaching the main loop (those with zero open and accepted patterns as well as zero pattern groups). For those instances, for each strip height considered, the initial heuristic finds a feasible pattern, or infeasibility is proven with the method from Fekete and Schepers (2004) at the beginning of the algorithm.

Because different processors were used for testing, it is hard to compare the speed of the algorithms in Table 6. However, Clautiaux et al. (2013) show that IGG_{lb} outperforms BKCS, BMVB, and IMVB in terms of computational effort. Our algorithms are on average faster than IGG_{lb}, even after taking different processor speeds into account.

4.5. Impact of Algorithm Components

To evaluate the importance of various components of our algorithms, we test algorithm A_{ub} again on data set ccm with some components disabled. Table 7 presents the results in terms of the numbers of pattern groups, accepted patterns, and open patterns, as well as computation time in seconds. Columns

Table 6 Comparison with Previously Proposed Algorithms for 2GSC | O on Data Set hifi

Instance	$\sum b_i$	n	W	$H'_{\min}$	H_{FBS}	H_{opt}	Multisets	Pattern groups	Accepted patterns	Open patterns		T (seconds)					
										Total	Peak	A_{ub}	A_{lb}	IGG_{lb}	BKCS	BMVB	IMVB
SCP1	10	5	5	13	15	13	180	0	0	0	0	0.00	0.00	0.01	0.0	0.1	0.1
SCP2	11	5	4	40	43	40	270	0	0	0	0	0.00	0.00	0.03	0.0	1.2	3.4
SCP3	15	5	6	14	19	14	900	93	79	82	9	0.01	0.00	0.04	7.1	19.4	6.8
SCP4	11	6	6	19	23	20	360	61	66	91	23	0.01	0.01	0.20	2.5	4.1	78.6
SCP5	8	5	20	20	21	20	96	0	0	0	0	0.00	0.00	0.00	0.0	0.1	0.1
SCP6	7	7	30	32	43	38	128	1	1	3	3	0.00	0.00	0.00	0.0	3.4	54.6
SCP7	8	4	15	12	19	14	64	10	13	14	4	0.00	0.01	0.01	0.0	1.4	1.8
SCP8	12	4	15	17	20	17	240	11	13	39	14	0.01	0.00	0.00	1.6	2.3	0.4
SCP9	12	6	27	68	77	68	729	0	0	0	0	0.00	0.00	0.01	0.1	0.1	0.7
SCP10	8	4	50	78	100	80	72	0	0	0	0	0.00	0.00	0.00	0.2	0.5	0.1
SCP11	10	10	27	47	57	48	1,024	179	148	373	104	0.03	0.01	0.01	4.9	0.7	221.5
SCP12	18	6	81	34	38	34	4,096	80	94	250	94	0.02	0.00	0.07	TL	2.7	1.3
SCP13	7	7	70	42	57	50	128	0	0	0	0	0.00	0.00	0.03	0.4	16.8	39.5
SCP14	10	4	100	60	83	69	144	61	90	176	85	0.01	0.03	0.25	15.6	48.7	41.7
SCP15	14	10	45	34	46	34	4,608	169	164	707	497	0.01	0.01	0.06	TL	165.7	0.7
SCP16	14	7	6	32	40	33	1,296	288	280	495	189	0.01	0.01	0.46	606.1	181.4	654.8
SCP17	9	9	42	34	43	39	512	151	198	356	117	0.01	0.03	0.26	2.1	323.6	227.3
SCP18	10	5	70	89	104	101	216	42	58	83	43	0.01	0.06	0.75	9.4	327.8	321.5
SCP19	12	6	5	25	27	26	576	129	114	120	58	0.01	0.01	0.48	0.0	473.0	1,794.3
SCP20	10	9	15	19	25	21	768	123	144	215	109	0.01	0.01	0.40	1.7	673.9	874.3
SCP21	11	7	30	135	175	145	648	206	216	299	133	0.01	0.06	1.03	24.3	2,001.8	1,757.6
SCP22	22	10	90	34	43	34	65,536	275	289	1,942	633	0.05	0.01	0.32	TL	757.8	606.0
SCP23	12	5	15	28	39	35	432	210	244	891	401	0.01	0.01	0.67	38.0	1,031.9	691.9
SCP24	10	10	50	103	142	114	1,024	494	637	1,963	379	0.04	0.08	2.45	19.3	5,585.7	6,265.0
SCP25	15	14	25	35	43	36	24,576	680	407	1,349	776	0.03	0.02	2.15	TL	2,662.9	3,735.8
Avg.												0.01	0.02	0.39			

Notes. A_{ub} and A_{lb} were tested on Intel i7-3770 3.4 GHz. IGG_{lb} from Clautiaux et al. (2013) was tested on Intel T2600 2.16 GHz. BKCS from Bekrar et al. (2010) was tested on Intel Pentium Xeon 2.7 GHz with TL = 1,000 seconds. BMVB and IMVB from Hifi (1998) were tested on Sparc-Server 20/720.

Table 7 Impact of Components of the Algorithm on Set ccm

Variant of the algorithm	Pattern groups		Accepted patterns		Open patterns			T (seconds)	
	Avg.	Max.	Avg.	Max.	Avg.	Max.	Max. peak	Avg.	Max.
A_{ub} Base case	435	5,935	431	6,087	2,152	18,885	11,949	0.02	0.14
Without additional infeasibility proofs	1,882	33,738	2,009	32,224	9,474	125,004	70,023	0.06	0.98
Without limiting open item types	4,234	28,282	5,180	33,440	42,608	357,188	281,951	0.26	1.87
Without insertion heuristic	14,550	94,440	15,380	102,820	1,317,382	16,160,724	10,000,348	3.16	39.50
Without all of the above	45,760	347,623	48,629	220,678	2,508,581	27,446,430	11,303,182	6.25	64.44

with subheading "Avg." present values averaged over all instances, and columns with subheading "Max." present maximum values over all instances. Column "Max. peak" under "Open patterns" presents the largest peak number of open patterns over all instances.

Clearly, all three components contribute to the high performance of our algorithm. The biggest impact is due to the insertion heuristic, without which the average computation time increases two orders of magnitude. Note that even with all three components disabled, our algorithm is still capable of solving each instance of set ccm in less than 65 seconds.

5. Summary and Discussion

In this paper, we propose a new exact algorithm for solving the 2GC problem with and without rotation of items. The algorithm was tested by embedding it in a simple exact algorithm for 2GSC. Computational experiments showed that this algorithm is superior to the previously proposed exact algorithms for 2GSC.

Several features contribute to the good performance of our algorithm. The two most important features of our algorithm are generating patterns in the order of nonincreasing waste and discarding dominated patterns. These two features help our algorithm tremendously reduce the number of partial patterns generated by the algorithm. Appropriate data structures are necessary for these features to work efficiently.

Another important feature of our algorithm is the heuristic used in an attempt to quickly prove feasibility. When patterns are constructed through horizontal and vertical builds in the order of nondecreasing

waste, it may take a long time before complete patterns are constructed, especially if all such patterns have a considerable amount of waste. Using a heuristic that can quickly complete a partial pattern, such as the insertion heuristic from Fleszar (2013), can significantly speed up the proof of feasibility.

Finally, an important feature of our algorithm is its ability to limit the search to a subset of items and to prove infeasibility without considering all items. Even a small reduction in the number of items can considerably reduce the number of partial patterns generated by the algorithm and, hence, significantly reduce the time needed to prove infeasibility.

Our algorithm uses horizontal and vertical builds from Wang (1983) to solve the 2GC problem. Previously, horizontal and vertical builds were used in the algorithms for 2GK by Wang (1983), Oliveira and Ferreira (1990), Viswanathan and Bagchi (1993), and Hifi (1997); and in the algorithm for 2GSC by Hifi (1998). While those algorithms solve different problems, some comparisons with our algorithm with respect to the pattern generation process can be made.

Our algorithm generates patterns in the order of nondecreasing waste, groups them according to item multiset, and eliminates dominated patterns in each group. By contrast, the algorithms from Wang (1983) and Oliveira and Ferreira (1990) generate patterns in the order of increasing item cardinality, regardless of the waste they include. If the algorithm is run as a heuristic, waste is controlled by eliminating patterns with absolute or percentage waste exceeding a user specified threshold. All constructed patterns are kept in one list; only unnecessary equivalent patterns (having the same multiset of items and occupying the same rectangle) are eliminated.

The algorithms from Viswanathan and Bagchi (1993) and Hifi (1997) for 2GK, and that from Hifi (1998) for 2GSC, have more similarities to our algorithm. As in our algorithm, their pattern construction process uses the set of open patterns and the set of accepted patterns (called Open and Clist in the respective papers). However, the order of pattern construction in those algorithms is also different than ours. Because those algorithms solve optimization problems, they first attempt to construct the most promising patterns according to carefully designed lower or upper bound functions. These functions are also used to eliminate dominated patterns.

While the 2GC problem considered in this paper is much simpler than most C&P problems considered in the literature, this should not be seen as a limitation of our work. As indicated in the introduction, 2GC occurs as a subproblem in exact methods for many C&P problems. In our computational analysis, we show that when our algorithm is used to solve a subproblem of 2GSC, the resulting algorithm significantly outperforms the previously proposed algorithms for 2GSC. We hope that effective algorithms using our algorithm can be designed for other complex two-dimensional guillotine C&P problems.

The main limitation of our approach is its high memory requirement. For large problem instances, the algorithm may need to construct and store very large numbers of patterns. Hence, to solve such instances, our algorithm may require not only a very long time, but also a very large memory.

As is the case with most algorithms that construct patterns using horizontal and vertical builds from Wang (1983), our algorithm cannot be easily extended to the problem without guillotine cuts. However, our algorithm could be adapted to the case with a restricted number of guillotine cutting stages. To generate stage-restricted patterns, the pattern construction process and the heuristic would have to be modified. While other components of the algorithm need not be modified, some, for example the infeasibility tests, could be improved if stage-restricted patterns are required.

Acknowledgments

This work would not have been possible without the generous support of the American University of Beirut, which granted the author a one-year research leave during academic year 2014–2015.

References

Alvarez-Valdes R, Parreno F, Tamarit JM (2009) A branch and bound algorithm for the strip packing problem. *OR Spectrum* 31(2):431–459.

Amossen RR, Pisinger D (2010) Multi-dimensional bin packing problems with guillotine constraints. *Comput. Oper. Res.* 37(11):1999–2006.

Beasley J (1985a) Algorithms for unconstrained two-dimensional guillotine cutting. *J. Oper. Res. Soc.* 36(4):297–306.

Beasley JE (1985b) An exact two-dimensional non-guillotine cutting tree search procedure. *Oper. Res.* 33(1):49–64.

Bekrar A, Kacem I, Chu C, Sadfi C (2010) An improved heuristic and an exact algorithm for the 2d strip and bin packing problem. *Internat. J. Product Development* 10(1):217–240.

Bengtsson BE (1982) Packing rectangular pieces—A heuristic approach. *Comput. J.* 25(3):353–357.

Berkey JO, Wang PY (1987) Two-dimensional finite bin-packing algorithms. *J. Oper. Res. Soc.* 38(5):423–429.

Boschetti MA, Montaletti L (2010) An exact algorithm for the two-dimensional strip-packing problem. *Oper. Res.* 58(6):1774–1791.

Burke EK, Kendall G, Whitwell G (2004) A new placement heuristic for the orthogonal stock-cutting problem. *Oper. Res.* 52(4): 655–671.

Christofides N, Hadjiconstantinou E (1995) An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts. *Eur. J. Oper. Res.* 83(1):21–38.

Christofides N, Whitlock C (1977) An algorithm for two-dimensional cutting problems. *Oper. Res.* 25(1):30–44.

Clautiaux F, Carlier J, Moukrim A (2007) A new exact method for the two-dimensional orthogonal packing problem. *Eur. J. Oper. Res.* 183(3):1196–1211.

Clautiaux F, Jouglet A, Moukrim A (2013) A new graph-theoretical model for the guillotine-cutting problem. *INFORMS J. Comput.* 25(1):72–86.

Clautiaux F, Jouglet A, Carlier J, Moukrim A (2008) A new constraint programming approach for the orthogonal packing problem. *Comput. Oper. Res.* 35(3):944–959.

Dolatabadi M, Lodi A, Monaci M (2012) Exact algorithms for the two-dimensional guillotine knapsack. *Comput. Oper. Res.* 39(1):48–53.

Fekete SP, Schepers J (2004) A general framework for bounds for higher-dimensional orthogonal packing problems. *Math. Methods Oper. Res.* 60(2):311–329.

Fekete SP, Schepers J, van der Veen JC (2007) An exact algorithm for higher-dimensional orthogonal packing. *Oper. Res.* 55(3): 569–587.

Fleszar K (2013) Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Comput. Oper. Res.* 40(1):463–474.

Furini F, Malaguti E, Thomopulos D (2016) A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem. *INFORMS J. Comput.* Forthcoming.

Hifi M (1997) An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. *Comput. Oper. Res.* 24(8):727–736.

Hifi M (1998) Exact algorithms for the guillotine strip cutting/packing problem. *Comput. Oper. Res.* 25(11):925–940.

Hopper E, Turton B (2001) An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *Eur. J. Oper. Res.* 128(1):34–57.

Lodi A, Martello S, Vigo D (1998) Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. Voss S, Martello S, Osman I, Roucairol C, eds. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization* (Kluwer Academic Publishers, Boston), 125–139.

Martello S, Vigo D (1998) Exact solution of the two-dimensional finite bin packing problem. *Management Sci.* 44(3):388–399.

Martello S, Monaci M, Vigo D (2003) An exact approach to the strip-packing problem. *INFORMS J. Comput.* 15(3):310–319.

Morabito R, Arenales MN (1996) Staged and constrained two-dimensional guillotine cutting problems: An and/or-graph approach. *Eur. J. Oper. Res.* 94(3):548–560.

Morabito R, Pureza V (2010) A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem. *Ann. Oper. Res.* 179(1):297–315.

Oliveira J, Ferreira J (1990) An improved version of wang's algorithm for two-dimensional cutting problems. *Eur. J. Oper. Res.* 44(2):256–266.

Pisinger D, Sigurd M (2005) The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optim.* 2(2): 154–167.

Pisinger D, Sigurd M (2007) Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS J. Comput.* 19(1):36–51.

Vasko FJ (1989) A computational improvement to wang's two-dimensional cutting stock algorithm. *Comput. Indust. Engrg.* 16(1):109–115.

Viswanathan KV, Bagchi A (1993) Best-first search methods for constrained two-dimensional cutting stock problems. *Oper. Res.* 41(4):768–776.

Wang PY (1983) Two algorithms for constrained two-dimensional cutting stock problems. *Oper. Res.* 31(3):573–586.

Wäscher G, Haußner H, Schumann H (2007) An improved typology of cutting and packing problems. *Eur. J. Oper. Res.* 183(3): 1109–1130.