INFORMS Journal on Computing
Vol. 25, No. 1, Winter 2013, pp. 72–86
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1110.0478
© 2013 INFORMS

A New Graph-Theoretical Model for the Guillotine-Cutting Problem

François Clautiaux
LIFL, UMR CNRS 8022, Université de Lille 1, 59650 Villeneuve d'Ascq, France,
francois.clautiaux@univ-lille1.fr

Antoine Jouglet, Aziz Moukrim
HeuDiaSyC, UMR CNRS 6599, Université de Technologie de Compiègne, 60200 Compiègne, France
{antoine.jouglet@hds.utc.fr, aziz.moukrim@hds.utc.fr}

We consider the problem of determining whether a given set of rectangular items can be cut from a larger rectangle using so-called guillotine cuts only. We introduce a new class of arc-colored directed graphs called *guillotine graphs* and show that each guillotine graph can be associated with a specific class of pattern solutions that we call a *guillotine-cutting class*. The properties of guillotine graphs are examined, and some effective algorithms for dealing with guillotine graphs are proposed. As an application, we then describe a constraint programming method based on guillotine graphs, and we propose effective filtering techniques that use the graph model properties in order to reduce the search space efficiently. Computational experiments are reported on benchmarks from the literature: our algorithm outperforms previous methods when solving the most difficult instances exactly.

Key words: two-dimensional packing; graph model; constraint programming
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received October 2009; revised October 2010, April 2011, June 2011; accepted June 2011. Published online in *Articles in Advance* October 17, 2011.

1. Introduction

The two-dimensional orthogonal guillotine-cutting problem (2SP) is deciding whether a given set of rectangles (items) can be cut from a larger rectangle (bin) using guillotine cuts only. A *guillotine cut* is one that is parallel to one of the sides of the rectangle and goes from one edge all the way to the opposite edge of a currently available rectangle. This decision problem is interesting in itself and also as a subproblem of *open-dimension packing* problems (such as the *strip-packing* problem) or in *knapsack* problems (see Wäscher et al. 2007 for a typology of cutting and packing problems). The problem occurs in industry if pieces of steel, wood, or paper need to be cut out of larger pieces. It is sometimes referred to as the unconstrained two-dimensional guillotine-cutting problem. This problem is strongly NP-complete, as it generalizes the classical bin-packing problem 1BP (see Garey and Johnson 1979).

A guillotine-cutting instance D is a pair (I, B), where I is the set of n items i to be cut. An item i is of width w_i and height h_i ($w_i, h_i \in \mathbb{N}$). The bin B is of width W and height H. All items must be cut and may not be rotated, all sizes are discrete, and only guillotine cuts are allowed. A *cutting pattern* is a set of coordinates for the items to be cut. A pattern is considered *guillotine* (see Figure 1) if it can be obtained using guillotine cuts only. It can be checked in $O(n^2)$ time (Ben Messaoud et al. 2008).

The two-dimensional guillotine-cutting problem has been widely studied in the operations research literature. One approach to tackling this problem efficiently is to use restrictions on the cutting patterns, such as *staged cutting patterns* (see Gilmore and Gomory 1965, Beasley 1985, Belov and Scheithauer 2006) or *two-section cutting patterns* (see Cui et al. 2006). This paper addresses the standard guillotine-cutting problem. In the literature to date, we find two alternative methods for solving the guillotine problem (see Hifi 1998). The first approach, stemming from Christofides and Hadjiconstantinou (1995), consists of iteratively cutting the bin into two rectangles, using horizontal or vertical cuts, until all the required rectangles are obtained. The second approach (Viswanathan and Bagchi 1993) recursively merges items into larger rectangles, using so-called horizontal or vertical *builds* (see Wang 1983). To our knowledge, the most recent work on the subject is by Bekrar et al. (2010), and it provides an adaptation of the branch-and-bound method of Maretello et al. (2003).

Graph-theoretical approaches can be found in the literature to solve some packing problems. In particular, for the two-dimensional packing problem 2OPP,

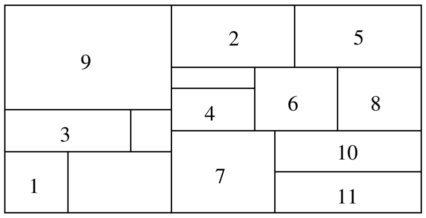

Figure 1 **A Guillotine Pattern**

Fekete and Schepers (2004) show that a pair of interval graphs (see Golumbic 1980) can be associated with distinct sets of patterns sharing a certain combinatorial structure, known as *packing classes*. Thus, two interval graphs G^W and G^H are associated with the width and the height, respectively. A vertex is associated with each item in both graphs. An edge is added in the graph G^W (respectively, G^H) between two vertices if the projections of the corresponding items on the horizontal (respectively, vertical) axis overlap. Fekete et al. (2007) propose a method for seeking a pair of interval graphs with the properties described above. Compared with the classical algorithms, their method avoids a large number of redundancies and is still competitive compared to more recent methods (Clautiaux et al. 2007, 2008).

In this paper we present an approach for the guillotine-cutting problem based on a graph-theoretical model called a *guillotine graph*. Unlike the model of Fekete and Schepers (2004), it relies on a particular recursive combinatorial structure specifically adapted to the guillotine case. This model is based on a single directed *arc-colored* graph, where circuits are related to horizontal or vertical builds. One important property of our model is that it can easily be extended to the k ($k \geq 3$)-dimensional case. We first show that each guillotine graph can be associated with a specific class of pattern solutions known as a *guillotine-cutting class*. Across this specific class of solutions and by the use of guillotine graphs, we can focus on dominant subsets of solutions to avoid redundancies in search methods. As an application, we then describe a constraint programming algorithm that relies heavily on the properties of guillotine graphs.

In §2, we introduce the concept of guillotine-cutting classes. Section 3 is devoted to definitions and properties of guillotine graphs. In §4, we focus on the techniques that enable guillotine graphs to be used: we propose algorithms to recognize them and to recover their associated guillotine patterns. In §5, we present an application of our model using a constraint programming method. In the same section, we compare our approach to the algorithms of Hifi (1998) and Bekrar et al. (2010) on randomly generated guillotine strip-packing instances from the literature. When applied to the difficult instances discussed in the literature, our method significantly outperforms the previous methods.

All proofs are reported in the online supplement (available at http://dx.doi.org/10.1287/ijoc.1110.0478).

2. Guillotine-Cutting Classes

To avoid equivalent patterns in the nonguillotine orthogonal-packing problem, Fekete and Schepers (2004) proposed the concept of a packing class. Packing classes are general and can model any pattern, guillotine or otherwise. When only guillotine patterns are sought, packing classes may not be suited to the problem because two different packing classes may give rise to patterns having the same combinatorial structure. We introduce the concept of a guillotine-cutting class that takes into account the fact that exchanging the positions of two rectangular blocks of items does not change the combinatorial structure of the solution. The definition uses the notion of so-called builds.

A build (Wang 1983) corresponds to the action of creating a new item by combining two other items (see Figure 2). The result of a horizontal build of two items i and j, denoted as $\{i, j\}^h$, is a *composite* item labeled $\min\{i, j\}$ of width $w_i + w_j$ and height $\max\{h_i, h_j\}$. It corresponds to any pattern in which items i and j are side by side in such a way that they can fit inside a rectangle of width $w_i + w_j$ and height $\max\{h_i, h_j\}$. In the same way, the result of a vertical build of two items i and j, denoted as $\{i, j\}^v$, is a composite item labeled $\min\{i, j\}$ of width $\max\{w_i, w_j\}$ and height $h_i + h_j$.

To create a guillotine pattern, builds have to be performed until there is only one final composite item. Thus, methods using builds search for a sequence of builds leading to a single composite item that fits inside the bin (see, for example, Viswanathan and Bagchi 1993). One drawback of this kind of representation using enumerative methods is the difficulty of managing equivalent sequences of builds efficiently. For example, it is easy to see that the composite item obtained from the sequence $(\{i_1, i_2\}^d, \{i_2, i_3\}^d)$ on items i_1, i_2, and i_3 with $d \in \{h, v\}$ is totally equivalent, patternwise, to any sequence of builds in set $\{(\{i_1, i_3\}^d, \{i_1, i_2\}^d), (\{i_2, i_3\}^d, \{i_1, i_2\}^d)\}$.

Going forward, instead of using sequences of builds, we shall use expressions that we call *recursive*

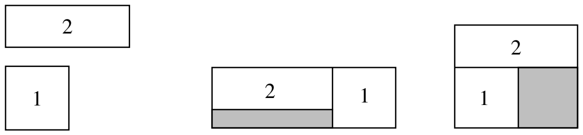

Figure 2 **Horizontal and Vertical Builds of Two Items 1 and 2**

multibuilds. These expressions allow us to eliminate this kind of redundancy by replacing successive sequences of builds in the same direction with only one operation. For example, a sequence of parallel builds $\{i_1, i_2\}^h, \{i_1, i_3\}^h, \ldots, \{i_1, i_k\}^h$ is replaced by the single operation $\{i_1, i_2, i_3, \ldots, i_k\}^h$. Thus a build operation is now performed on a set of items of any size instead of on just one pair of items. Note that using a set means that elements can neither be repeated nor ordered in any way. This is justified by the fact that the order in which the builds are performed does not have any impact on the structure of the pattern obtained. We now formally define the notion of recursive multibuild.

Definition 2.1. A recursive multibuild (RMB) is either a single item i or an expression $\{x_1, \ldots, x_k\}^d$, $k \geq 2$, where $d \in \{h, v\}$ is a dimension, and $x_1, \ldots, x_k$ are themselves recursive multibuilds, comprising either a single item or a recursive multibuild of dimension $d' \neq d$.

For example, the following recursive multibuild is related to Figure 1:

$$\big\{\{1,3,9\}^v, \{\{2,5\}^h, \{4,6,8\}^h, \{7, \{10,11\}^v\}^h\}^v\big\}^h. \quad (1)$$

In the following, the set of all RMBs that can be built out of all the items of a given set I is defined as $\mathcal{R}_I$.

Many different packing patterns can be obtained using the same RMB. Indeed, as we have already seen, an RMB represents a lot of different sequences of builds, each of the subsequences being interpretable as different placements. For example, $\{i, j\}^h$ can be interpreted as i to the left or to the right of j. Moreover, there are often several possible positions for the smallest items. However, all the obtained patterns share the same combinatorial structure and can be considered the same solution. We describe them as belonging to the same guillotine-cutting class.

Definition 2.2. Two solutions belong to the same guillotine-cutting class if they can be obtained from the same recursive multibuild.

Figure 3 shows some elements of the same guillotine-cutting class. Each of these patterns can be obtained from the RMB $\{\{1,2\}^v, \{3, \{4,5\}^h\}^v\}^h$. Other elements of this class can be obtained if the bottom left rule is not used. For example, sliding item 4 up can lead to another equivalent pattern.

Note that because any pattern can be obtained by the application of a sequence of builds (Wang 1983), it can be modeled by a corresponding RMB. Clearly, if one member of a guillotine-cutting class is feasible, then so are the other members. In this case we say that *the guillotine-cutting class is feasible*.

Note that there may still be redundancies. Two different RMBs may lead to a same pattern, when, for example, a horizontal or a vertical cut can be first applied indifferently (there is a cross pattern). This means that a given pattern may belong to several guillotine-cutting classes. It is important to note that these different guillotine-cutting classes are actually different from an operator point of view because they correspond to a different sequence of cuts.

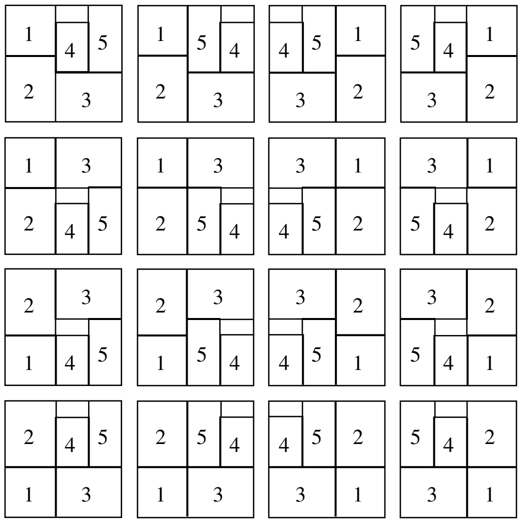

Figure 3 Some Elements of a Guillotine-Cutting Class

It is particularly important to be able to use this notion of RMB effectively in a search algorithm. Below we show that it is possible to represent an RMB as a graph to which efficient algorithms can be associated.

3. A New Graph Model for the Guillotine-Cutting Problem

The RMB concept is useful for representing guillotine-cutting classes. However, the direct use of RMBs in a computer implementation (by means of strings, for example) is far from straightforward. We therefore propose a way of representing RMBs via a special class of graphs, which we call guillotine graphs. We then show how guillotine graphs may be used effectively in a search method.

We first define the concept of guillotine graph and show that each guillotine graph can be associated with a guillotine-cutting class through the use of an RMB. The interest of such a concept is that, in practice (see §5), guillotine graphs can be used to manipulate and enumerate guillotine-cutting classes. Compared to a direct use of RMBs, the graph formalism allows us to reuse classical definitions and properties from graph theory.

Our graph model represents RMBs and thus the *recursive structure* of the problem. This is different from Fekete and Schepers (2004), where graphs are used to represent the relative coordinates of the items.

To avoid different graphs representing the same pattern, we introduce several refinements into our model—namely, *normal guillotine graphs* and *well-sorted normal guillotine graphs* (*WSNGs*). We show that there is a bijection between the set of WSNGs and the set of guillotine-cutting classes.

In this section, we focus on theoretical results (definitions and characterizations). A practical algorithm for dealing with these graphs will be presented in the next section.

3.1. Guillotine Graphs

We now define our new class of graphs. These graphs are a direct translation of recursive multibuilds into graph formalism.

We first recall some classical graph definitions. A *directed graph* G is a pair (V, A), where V is a set of vertices and A is a set of arcs between these vertices. The notation (u, v) is used for an arc between vertices u and v. The *neighborhood* $N(v)$ of any given vertex v is the set of vertices u such that (v, u) and/or (u, v) belong to A. We also note $N(v) = N^+(v) \cup N^-(v)$, where $N^+(v) = \{u \in X/(v, u) \in A\}$ and $N^-(v) = \{u \in X/(u, v) \in A\}$. A *Hamiltonian path* (respectively, circuit) μ is a path (respectively, circuit) that visits each vertex exactly once. A subgraph induced from a set $W \subseteq V$ is the graph $(W, A \cap (W \times W))$.

We also use the concept of *arc coloring* defined as follows. An arc coloring of a graph $G = (V, A)$ is a mapping χ from A to a set of k colors. We say that a circuit is *monochromatic* if all the arcs in the circuit have the same color.

To define the concept of a guillotine graph, we introduce the concept of *circuit contraction*, analogous to the classical concept of *arc contraction* used in graph theory. Contracting an arc $e = (u, v)$ in the graph $G = (V, A)$ consists of deleting u, v, and all arcs incident to u or v and introducing a new vertex v_e and new arcs, such that v_e is incident to all vertices that were incident to u or v. Contracting the circuit $\mu = [(6, 7), (7, 8), (8, 6)]$ as in Figure 4(a) yields Figure 4(b).

Definition 3.1. Let $G = (V, A)$ be a graph, and let $\mu = [v_1, v_2, \dots, v_k, v_1]$ be a circuit of G. Contracting μ is equivalent to iteratively contracting each arc of μ.

We are now ready to formally define *guillotine graphs*.

Definition 3.2. Let G be an arc-bicolored directed graph. G is a *guillotine graph* if it can be reduced to a single vertex x by iterative contractions of monochromatic circuits with the following properties:

1. There are no steps in which a vertex belongs to two different monochromatic circuits.

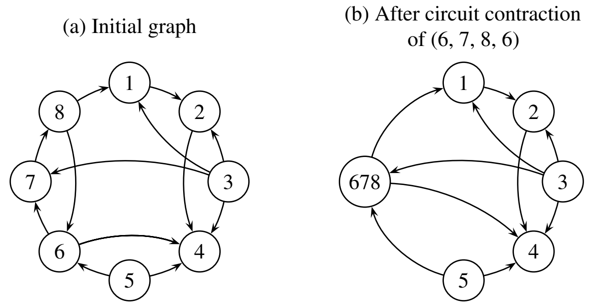

Figure 4 Circuit Contraction

2. When a circuit μ is contracted, either the current graph G is composed only of μ or exactly two vertices of μ are of a degree strictly greater than 2.

In a guillotine graph, the set of vertices is initially the set of items I of a given instance of guillotine-cutting problem. Put another way, each vertex is initially labeled by an RMB composed of a single item. At each step of the contraction process, each vertex corresponds to an RMB. Contracting a circuit of a guillotine graph is equivalent to building a new RMB that includes the different RMBs labeling the vertices of the circuit. Each contracted circuit has a given color in $\{h, v\}$, which is associated with the dimension of the corresponding RMB (horizontal or vertical).

Recall that $\mathcal{R}_I$ is the set of all RMBs that can be built out of all the items of a given set I. Let $\mathcal{G}_I$ be the set of guillotine graphs of vertex set $I = \{1, \dots, n\}$. The association between guillotine graphs and guillotine-cutting classes can be formally expressed with the application κ: $\mathcal{G}_I \to \mathcal{R}_I$. A possible implementation of κ is given in Algorithm 1, which associates an RMB $\kappa(G)$ with any guillotine graph G.

Algorithm 1 (κ: Associating a guillotine graph with a guillotine-cutting class)

```
  Data: A guillotine graph $G = (I, A)$
1 while $|I| > 1$ do
2   $U \leftarrow$ set of monochromatic circuits of $G$;
3   foreach $\mu = (u_1, \dots, u_k) \in U$ do
4     $d \leftarrow \chi(u_1, u_2)$ (the unique color of the circuit);
5     contract $\mu$ in $G$ and label the resulting
        vertex with $r = \{u_1, \dots, u_k\}^d$;
6 return the label of the unique vertex of G
```

In a graph G, contracting a circuit of color h (respectively, v) corresponds to a horizontal (respectively, vertical) RMB. When a circuit μ is contracted, the size associated with the residual vertex is the size of the composite item built, and its label is the corresponding RMB. The algorithm stops when only one vertex remains. The label of the last vertex is then the RMB corresponding to the related guillotine-cutting class. Note that Definition 3.2 can be directly generalized

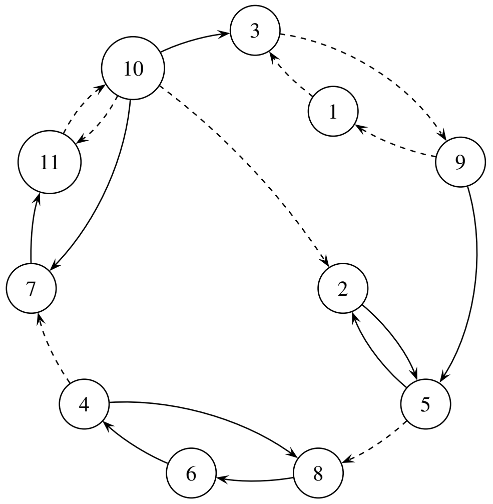

Figure 5 **Modeling the Guillotine-Cutting Class Associated with the Pattern of Figure 1 with a Guillotine Graph**

Note. The dotted arcs correspond to the vertical dimension, whereas the plain arcs correspond to the horizontal dimension.

for higher dimensions (just by considering k colors instead of two).

For example, applying Algorithm 1 to the graph of Figure 5 will result in the following operations. First, circuits $(3,1,9,3)$, $(2,5,2)$, $(8,6,4,8)$, and $(10,11,10)$ are contracted and replaced by single vertices labeled $\{1,3,9\}^v$, $\{2,5\}^h$, $\{4,6,8\}^h$, and $\{10,11\}^v$, respectively. Then, a new monochromatic circuit including vertex 7 and the newly created vertex labeled $\{10,11\}^v$ appears, and it is contracted again, giving a vertex labeled $\{7,\{10,11\}^v\}^h$. Then, a new vertex is created with the label $\{\{2,5\}^h,\{4,6,8\}^h,\{7,\{10,11\}^v\}^h\}^v$. Finally, the two remaining vertices are contracted to form a vertex labeled by the RMB of Equation (1).

Proposition 3.1. *If G is a guillotine graph (I,A), then Algorithm 1 applied to G terminates and returns an RMB related to I.*

Again, recall that $\mathcal{R}_I$ and $\mathcal{G}_I$ are, respectively, the set of all RMBs and the set of all guillotine graphs that can be built out of all the items of a given set I.

Proposition 3.2. *For all $r \in \mathcal{R}_I$, there exists a guillotine graph $G \in \mathcal{G}_I$ such that $\kappa(G) = r$.*

The following proposition states that the number of arcs in a guillotine graph is in $O(n)$. We use this result in the next section to prove that our algorithms run in $O(n)$ time and space.

Proposition 3.3. *Let G be a guillotine graph with at least two vertices. The number m of arcs in G is in $[n, 2n-2]$, and the bounds are tight.*

3.2. Normal Guillotine Graphs

A guillotine graph can be associated with a unique guillotine class. However, many equivalent graphs can be associated with a given guillotine-cutting class. We now introduce a dominant subset of these graphs, normal guillotine graphs, with a special structure. We show that only normal guillotine graphs need to be considered when enumerating all guillotine-cutting classes. In the next section, we make use of the special properties of normal guillotine graphs. We propose efficient algorithms for recognizing them and computing their related patterns.

In a normal guillotine graph, the two vertices x_i and x_j of a degree greater than 2 in a monochromatic circuit μ are such that if x_j follows x_i in μ, x_i cannot be the tail of any arc outside μ, and x_j cannot be the head of any arc outside μ.

Definition 3.3. Let G be a guillotine graph; G is a normal guillotine graph if at any step of the iterative contraction process, in each monochromatic circuit $\mu = (x_1, x_2, \ldots, x_{k-1}, x_k, x_1)$, all vertices are of degree 2, or there are two vertices x_i and x_j of a degree strictly greater than 2, such that $(x_i, x_j) \in \mu$, $|N^+(x_i)| = 1$, and $|N^-(x_j)| = 1$.

We show two equivalent guillotine graphs in Figure 6. The first (Figure 6(a)) is not normal because in the monochromatic circuit $[2,1,4,2)]$, $|N^+(1)| = |N^-(4)| = 2$. Figure 6(b) gives a simple example of what "normal" means. Roughly speaking, in a normal guillotine graph, in any monochromatic circuit, there is a single "entry" (vertex 2 in Figure 6(b)), which is the head of several arcs, and a single "exit" (vertex 4 in Figure 6(b)), which is the tail of several arcs. All the other vertices of a monochromatic circuit only have one predecessor and one successor.

Normal guillotine graphs have interesting properties that will be used below for creating efficient algorithms to handle them. The first property is that they contain a unique Hamiltonian circuit.

Lemma 3.1. *If G is a normal guillotine graph, it contains a unique Hamiltonian circuit.*

From the definition of normal guillotine graphs, together with the fact that they contain a Hamiltonian circuit, we can define an ordering σ on the vertices, which will be useful throughout the rest of this paper. Among the n possible orderings obtained by circular permutation, we only consider the *normal* ones.

Definition 3.4. Let $G = (I, A)$ be a normal guillotine graph, and let $\mu_G = (v_1, \ldots, v_n)$ be its

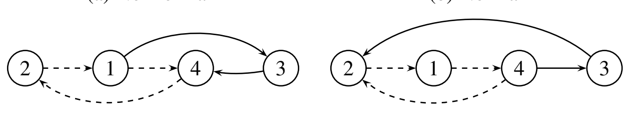

Figure 6 **Two Equivalent Guillotine Graphs**

unique Hamiltonian circuit. A *normal ordering* σ of I is a mapping from I to $\{1, \ldots, n\}$ such that for $i = 1, \ldots, n-1$, $(\sigma_i, \sigma_{i+1}) \in \mu_G$, and for any arc $(\sigma_i, \sigma_j) \notin \mu_G$, we have $j < i$.

Proposition 3.4. *A normal guillotine graph always admits a normal ordering.*

In the following, σ will always refer to a normal ordering. Hereafter, when a graph G has a unique Hamiltonian circuit, and for a given ordering σ, we shall refer to any arc that is not included in the circuit as a *backward arc*. We name these arcs "backward" because Definition 3.4 implies that an arc (σ_i, σ_k) that is not in the circuit is such that $k < i$ in a normal ordering. Note that (σ_n, σ_1) is also backward because it entails the property that each contracted circuit contains exactly one backward arc in a normal guillotine graph.

Definition 3.5. Let G be a normal guillotine graph, and let σ be a normal ordering of its vertices. A *forward* arc is an arc of form (σ_i, σ_{i+1}) with $i < n-1$. A *backward* arc is an arc that is not forward.

The following theorem gives a precise definition of normal guillotine graphs, which can be used in algorithms to recognize them. The theorem states that there is a Hamiltonian circuit in the graph and that there are no "crossing arcs" (two different circuits with common vertices, neither of which is included in the other). Moreover, the fact that a vertex does not belong to two monochromatic circuits can be verified locally by considering the neighborhood of each vertex.

As stated in condition 1 of the following theorem, in a normal guillotine graph a vertex cannot be both the head and the tail of backward arcs. For the sake of simplicity, we sort the backward arcs incident to a vertex in a certain order, and we introduce the following notations. Let $A^-(\sigma_i) = \{u_1, u_2, \ldots, u_z\}$ be the list of arcs whose head is σ_i, where $\{u_1, \ldots, u_{z-1}\}$ are the backward arcs (σ_j, σ_i) ordered by *increasing* value of j and $u_z = (\sigma_{i-1}, \sigma_i)$ is the incoming forward arc to σ_i. In addition, let $A^+(\sigma_i) = \{u_1, u_2, \ldots, u_z\}$ be the list of arcs whose tail is σ_i, where $\{u_1, \ldots, u_{z-1}\}$ are the backward arcs (σ_i, σ_j) ordered by *decreasing* value of j and $u_z = (\sigma_i, \sigma_{i+1})$ is the outgoing forward arc from σ_i. For example, in the normal graph of Figure 6(b), $\sigma_1 = 2$ and $A^-(\sigma_1) = \{(4,2), (3,2)\}$.

Theorem 3.1. *Let $G = (I, E)$ be a graph. G is a normal guillotine graph if and only if it contains a unique Hamiltonian circuit, and for any normal ordering σ of I,*

1. *G does not include two backward arcs (σ_j, σ_i) and (σ_l, σ_k) such that $i < k \le j < l$.*
2. *Color properties:*

(a) *For each vertex σ_i, $i \in \{1, \ldots, n-1\}$, consider $A^-(\sigma_i) = \{u_1, \ldots, u_z\}$ as constructed above. Then $\chi(\sigma_i, \sigma_{i+1}) = \chi(u_1)$ and $\forall k \in \{1, \ldots, z-1\}$, $\chi(u_k) \neq \chi(u_{k+1})$.*

(b) *For each vertex σ_i, $i \in \{2, \ldots, n\}$, consider $A^+(\sigma_i) = \{u_1, \ldots, u_z\}$ as constructed above. Then $\chi(\sigma_{i-1}, \sigma_i) = \chi(u_1)$ and $\forall k \in \{1, \ldots, z-1\}$, $\chi(u_k) \neq \chi(u_{k+1})$.*

3.3. Well-Sorted Normal Guillotine Graphs

Focusing on normal graphs restricts the set of guillotine graphs to be considered. However, some redundancies remain. This is because RMBs are sets, and thus unordered, whereas circuits are ordered. For example, circuits $(1,2,3,4,1)$ and $(1,3,2,4,1)$ are equivalent.

We now restrict further the set of guillotine graphs to be considered by introducing the notion of well-sorted normal guillotine graphs. These graphs are normal and thus retain all the properties listed previously. The extra property possessed by well-sorted normal guillotine graphs is that vertices have to be sorted by increasing index in any monochromatic circuit and at any step of the contraction process.

We show below that for any instance, there is a bijection between the subset of well-sorted normal guillotine graphs in $\mathcal{G}_I$ and the set $\mathcal{R}_I$ of RMBs.

Recall that during the contraction process (Algorithm 1), vertices are labeled with RMBs. Given an RMB r, we denote by $\eta(r)$ the item of smallest label in r. This operator is useful for defining well-sorted graphs properly.

Definition 3.6. Let $G = (I, A)$ be a normal guillotine graph. G is a WSNG if there is a normal ordering σ of I such that at each step of Algorithm 1 applied to G, any forward arc (i, j) belonging to a monochromatic circuit is such that $\eta(i) < \eta(j)$. We say that such a σ is a *well-sorted normal ordering*.

Proposition 3.5. *In a well-sorted guillotine graph, there is a unique well-sorted normal ordering, and $\sigma_1 = 1$.*

Figure 7 shows the well-sorted normal guillotine graph that models the guillotine-cutting class associated with the pattern of Figure 1. It can be compared with the guillotine graph of Figure 5, which is neither normal nor well sorted.

Theorem 3.2 adds a condition to Theorem 3.1 to offer a way of recognizing well-sorted dominant graphs. To simplify the proof, we first show that the "first" vertex of a circuit (i.e., the first vertex in the ordering σ) contains the vertex of smallest label in the circuit.

Lemma 3.2. *Let G be a well-sorted normal guillotine graph, and let σ be a corresponding well-sorted normal order. A vertex x resulting from the contraction of a monochromatic circuit $\mu = (\sigma_i, \sigma_{i+1}, \ldots, \sigma_{i+k}, \sigma_i)$ contracted during the contraction process is such that $\eta(x) = \eta(\sigma_i)$.*

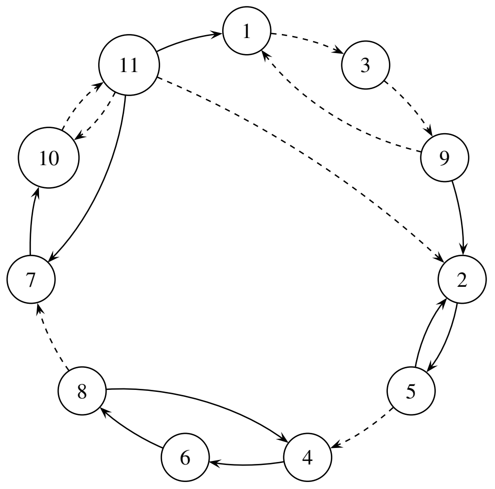

Figure 7 **Modeling the Guillotine-Cutting Class Associated with the Pattern of Figure 1 with a Well-Sorted Normal Guillotine Graph**

Theorem 3.2. *Let $G = (I, A)$ be a normal guillotine graph; G is a well-sorted normal guillotine graph; if and only there is a normal ordering of I such that for each forward arc (i, j), $i > j$, there exists a backward arc (i, l) such that $l < j$.*

We now show that our application κ yields a bijection when it is restricted to the set of well-sorted normal guillotine graphs. For this purpose we introduce $\mathcal{G}_I^*$, the set of well-sorted guillotine graphs of $\mathcal{G}_I$.

Theorem 3.3. *Let κ' be the restriction of κ defined from $\mathcal{G}_I^*$ to $\mathcal{R}_I$. Function κ' is bijective.*

We have shown that it is sufficient to consider well-sorted normal guillotine graphs. This subset of graphs is well suited to enumerative exact methods or constraint programming-based methods (see §5), because it does not allow any redundancies in terms of RMBs. In the following section, we describe efficient algorithms for handling these graphs.

4. Manipulating Well-Sorted Normal Guillotine Graphs

In this section we focus on methods for manipulating guillotine graphs efficiently. Our goal is to provide a "toolbox" of appropriate complexity for dealing with these graphs. We shall consider only well-sorted normal graphs because these are capable of being recognized by algorithms of low complexity, making the related guillotine cutting class easier to compute. First, we show how a WSNG can be recognized in linear time. Then, we show how the corresponding pattern can be computed with the same complexity.

We first describe an algorithm that uses Theorems 3.1 and 3.2 to recognize WSNGs in linear time. Algorithm 2 works as follows. First, it computes the Hamiltonian circuit of the graph (if it exists). It proceeds iteratively by appending the vertices to the circuit one by one. When a vertex v is considered, its only successor not yet visited is its successor in the Hamiltonian circuit (the other successors are reached by backward arcs). Then, the well-sorted property is checked with a direct application of Theorem 3.2. The second phase checks that there are no pairs of "crossing arcs" (see Theorem 3.1), i.e., that the recursive structure of circuits is respected. This algorithm resembles a classical algorithm for verifying that an expression is correctly parenthesized. Finally, the color property is verified in a straightforward manner.

Algorithm 2 (Recognizing a well-sorted normal guillotine graph G)

```
1  Let $\sigma$ and $\sigma^{-1}$ be 2 arrays of $n$ integers whose
       cells are initialized with $-1$
   /*Phase 1: find the Hamiltonian path and the
       well-sorted normal ordering */
2  $v \leftarrow 1$, $\sigma_1 \leftarrow 1$, $\sigma_1^{-1} \leftarrow 1$, $next \leftarrow 1$;
3  for $i$: $2 \rightarrow n$ do
       /*determine the only forward arc outgoing
         from $v$ */
4      $next \leftarrow -1$;
5      for $w \in N^+(v)$ do
6          if $\sigma_w^{-1} = -1$ or ($i = n$ and $w = 1$) then
7              if $next \neq -1$ then return false;
8              $next \leftarrow w$;
9      if $next = -1$ then return false;
       /*return false if the graph is not
         well-sorted */
10     if $next < v$ then
11         well-sorted $\leftarrow$ false;
12         for $w \in N^+(v) \backslash \{next\}$ do
13             if $w < next$ then well-sorted $\leftarrow$ true;
14         if well-sorted = false then return false;
15     $v \leftarrow next$, $\sigma_i \leftarrow v$, $\sigma_v^{-1} \leftarrow i$;
16 if $v \neq 1$ then return false
   /*Phase 2: check the structure of the graph */
17 Let $S$ be a stack of vertices initially empty;
18 for $i$: $1 \rightarrow n$ do
19     foreach backward arc $(\sigma_i, w)$ by decreasing
           value of $\sigma_w^{-1}$ do
20         while $S$ is not empty and the top element of $S$
                 does not equal $w$ do
21             remove the top element of $S$;
22         if $S$ is empty then return false
23     push $\sigma_i$ on $S$;
   /*Phase 3: check the colors of the arcs */
```

```
24 for $i$: $1 \rightarrow n$ do
25     if $i < n$ then
26         $d \leftarrow \chi(\sigma_i, \sigma_{i+1})$;
27         foreach arc $u_j \in A^-(\sigma_i)$ do
28             if $\chi(u_j) \neq d$ then return false;
29             $d \leftarrow \bar{d}$;
30     if $i > 1$ then
31         $d \leftarrow \chi(\sigma_{i-1}, \sigma_i)$;
32         foreach arc $u_j \in A^+(\sigma_i)$ do
33             if $\chi(u_j) \neq d$ then return false;
34             $d \leftarrow \bar{d}$;
35 return true;
```

Proposition 4.1. *Algorithm* 2 *returns* true *if the input graph is a well-sorted normal guillotine graph and false otherwise.*

Now that we have an algorithm for recognizing WSNGs, the size of the associated guillotine class remains to be computed. Algorithm 3 computes the width and height of the guillotine-cutting class associated with the input WSNG G. First, the ordering σ is computed using Algorithm 2. Then the vertices are considered according to the ordering σ. At each iteration the RMB containing only item σ_i is created. The size associated with this RMB is then $w_{\sigma_i} \times h_{\sigma_i}$. If there is no backward arc outgoing from σ_i, the associated RMB is pushed on a stack S. If there are one or several backward arcs (σ_i, σ_j) with $j < i$, they correspond to successive contractions that have to be done in decreasing order of σ_j. They are done by merging the current RMB with some RMBs that have been previously pushed on the stack. At the end of the algorithm, the stack contains only one element, whose label *label*(t) is the RMB associated with the graph and for which the size is $w(t) \times h(t)$.

Algorithm 3 (Computing the size of a pattern of the guillotine class related to a well-sorted normal guillotine graph)

```
   Data: $G$, a normal guillotine graph with $\sigma$ its
     corresponding ordering on the vertices;
 1 Let $S$ be an initially empty stack of 3-tuples
     $t = (label(t), w(t), h(t))$;
 2 $d \leftarrow \chi(\sigma_1, \sigma_2)$;
 3 push $(\sigma_1, w_{\sigma_1}, h_{\sigma_1})$ on $S$
     //$\sigma_1$ is an RMB of size $w_{\sigma_1} \times h_{\sigma_1}$
 4 for $i$: $2 \rightarrow n$ do
       /*$\sigma_i$ is an RMB of size $w_{\sigma_i} \times h_{\sigma_i}$ */
 5     $rmb \leftarrow \sigma_i$, $w \leftarrow w_{\sigma_i}$, $h \leftarrow h_{\sigma_i}$;
 6     foreach backward arc $(\sigma_i, v) \in A^+(\sigma_i)$ by
         decreasing value of $\sigma_v^{-1}$ do
           /*compute the RMB associated with the
             monochromatic circuit to which arc $(\sigma_i, v)$
             belongs and its corresponding size */
           /*set $X$ will contain all the RMBs that are
             merged during the circuit contraction
             ($rmb$ and vertices from the stack $S$) */
 7         $X \leftarrow \{rmb\}$;
 8         while $\eta(rmb) \neq v$ do
 9             $(rmb_t, w_t, h_t) \leftarrow$ top of $S$;
10             remove top of $S$;
11             $X \leftarrow X + rmb_t$;
               /*$rmb_t$ is an RMB to merge with the
                 other RMBs in set $X$
12             if $d = h$ then $w \leftarrow w + w_t$ and
                 $h \leftarrow \max(h, h_t)$;
13             else $w \leftarrow \max(w, w_t)$ and $h \leftarrow h + h_t$;
14         $rmb \leftarrow X^d$, $d \leftarrow \bar{d}$;
15     push $(rmb, w, h)$ on $S$;
16 $(rmb_t, w_t, h_t) \leftarrow$ top of $S$;
17 return $(w_t, h_t)$;
```

Proposition 4.2. *Algorithm* 3 *computes the size of the pattern associated with the normal guillotine graph G.*

Both algorithms presented in this section can be executed in linear time. In fact, they are executed in $O(m)$, but Proposition 3.3 states that guillotine graphs have fewer than $2n$ arcs. Therefore, if only graphs with fewer than $2n$ arcs are considered (by adding a test at the beginning of Algorithm 2), the complexity becomes $O(n)$.

Theorem 4.1. *Recognizing a well-sorted normal guillotine graph, and computing the guillotine-cutting class related to this graph takes $O(n)$ time and space.*

5. An Application of Guillotine Graphs

In this section, we provide numerical evidence of the usefulness of our model. For this purpose, we describe an exact approach based on our new model. The basic idea of the method is to seek a well-sorted normal guillotine graph corresponding to a configuration that fits within the boundary of the input bin. Our model is embedded into a *constraint programming* scheme, which seeks a suitable set of arcs.

First, we recall the paradigm of constraint programming before showing how the guillotine-cutting problem can be expressed as a constraint satisfaction problem. Then we describe propagation techniques related to the model. Finally, we discuss our computational experiments.

5.1. A Constraint Programming Approach

Constraint programming is a paradigm aimed at solving combinatorial problems that can be described by a set of variables, a set of possible values for each variable, and a set of constraints between the variables.

We chose this approach because it has been shown to be efficient for the nonguillotine rectangle packing problem (see Beldiceanu et al. 2007, Clautiaux et al. 2008, for example). However, algorithms designed for the bin packing problem cannot be applied directly to the guillotine case. Our graph model can be used for this purpose.

The set of possible values of a variable V is called the *variable domain*, denoted as $domain(V)$. It might, for example, be a set of numeric or symbolic values $\{v_1, v_2, \dots, v_k\}$ or an interval of consecutive integers $[\alpha..\beta]$. In the latter case the lower bound of $domain(V)$ is denoted as $V^- = \alpha$, and the upper bound is denoted as $V^+ = \beta$.

A constraint between variables expresses which combinations of values are permitted for the different variables. The question is whether there exists an assignment of values to variables such that all constraints are satisfied. The power of the constraint-programming method lies mainly in the fact that constraints can be used in an active process, termed "constraint propagation," where certain deductions are performed, in order to reduce computational effort. Constraint propagation removes values from the domains, deduces new constraints, and detects inconsistencies.

In this section we describe how the guillotine-cutting problem can be modeled in terms of two sets of variables and constraints. The first variable set is related to the graph underlying the pattern to ensure that the configuration is guillotine, whereas the second variable is related to geometric considerations to ensure that the rectangles can be placed within the boundaries of the large rectangle.

5.1.1. Arc-Related Variables.

The first set of variables is related to the arcs of the guillotine graph to be built. It specifies the *state* of each arc. The state of an arc is determined by its *existence*, its *orientation* (backward or forward), and its *direction* (horizontal or vertical). Each potential arc (i, j) $(i, j \in \{1, \dots, n\})$ is therefore governed by the following variables:

- $existence(i, j)$: its domain is initially $\{true, false\}$,
- $orientation(i, j)$: its domain is initially $\{forward, backward\}$, and
- $direction(i, j)$: its domain is initially $\{horizontal, vertical\}$.

For the k-dimensional cases, $k > 2$, only the domain of $direction(i, j)$ has to be modified (k possible directions are needed).

5.1.2. Vertex-Related Variables.

Recall that a guillotine graph can be reduced to a single vertex by iterative contractions of monochromatic circuits. Thus, each time such a monochromatic circuit μ is found in the graph under construction, μ is contracted. To prevent contracted vertices from being revisited, a state is associated with each vertex, specifying whether or not it has been contracted and giving its current dimensions. Thus, *a vertex i represents an item or an aggregation of items*. Its dimensions are either the dimensions of the original rectangle i or the dimensions of the multibuild corresponding to the contractions in the graph. Each vertex i therefore has the following associated variables:

- $contraction(i)$: The label of the aggregate that contains i; its domain is initially $[1..i]$ (because the label of an aggregate is always the smallest label among the included vertices).
- $width(i)$: The width of the aggregate i; its domain is initially $[w_i..W]$.
- $height(i)$: The height of the aggregate i; its domain is initially $[h_i..H]$.

Note that during the search, $width(i)^-$ and $height(i)^-$ are equal to the current dimensions of aggregate i.

When a build is performed (a contraction in the graph), gaps (i.e., holes in which no items can be placed) are generally created. A variable *lost* keeps track of the accumulated lost space. The lower bound of *lost* is updated at each contraction so as to equal the current lost space induced by the partial graph. The domain of *lost* is initially $[0..(W \times H - \sum_{i \in I} w_i \times h_i)]$.

Recall that any well-sorted normal guillotine graph has a corresponding Hamiltonian circuit. We now consider its associated well-sorted normal ordering σ. To each pair of vertices i, j, the Boolean variable $before(i, j)$, indicating whether $\sigma_i^{-1} < \sigma_j^{-1}$, is associated.

5.1.3. Geometric Variables.

A valid well-sorted normal guillotine graph may yield a guillotine-cutting class that does not fit within the boundaries of the bin. Consequently, we use a second set of variables that represent the coordinates of a specific member of the guillotine-cutting class under construction. Each item i has the following associated variables:

- $X(i)$: The horizontal position of i; its domain is initially $[0..W - w_i]$.
- $Y(i)$: The vertical position of i; its domain is initially $[0..H - h_i]$.

These are equal to the coordinates of the items in the pattern of the guillotine-cutting class related to the well-sorted normal guillotine graph whose items are packed as follows: In a horizontal contraction, they are packed bottommost and side by side, in an increasing order of labels, from left to right. In a vertical contraction, they are packed leftmost and one above the other, in an increasing order of labels, from bottom to top. Consequently, $X(1) = Y(1) = 0$, since item 1 is always cut out of the bottom left-hand corner of the pattern.

A nonoverlap constraint (see, for example, Beldiceanu and Carlsson 2001, Clautiaux et al. 2008) is then used to adjust or determine the positions of the items in such a way that this solution is valid with respect to the dimensions of the bin. The associated propagation techniques are adapted to aggregates of items.

5.1.4. Implementation Details. Suppose that a circuit $[u_0, \ldots, u_k]$ is contracted. Each vertex u_i is either a vertex or an aggregate of vertices resulting from previous contractions. The variable *contraction*, for vertices in the circuit, is adjusted. The variables $width(u_0)$ and $height(u_0)$ are adjusted according to the sizes of the aggregates in the circuit. Because there are no crossing arcs in a normal guillotine graph, any arc adjacent to vertices of the circuit other than u_0 and u_k is eliminated. We also take into account the fact that a contraction of direction d in a vertex u_0 cannot immediately be followed by a contraction including u_0 using a circuit of direction d. A contraction in the other direction must first take place.

5.1.5. Exploration of the Search Space. In our method, the branching scheme modifies only the graph variables directly: the values of the geometric variables X and Y are deduced from constraint propagation. At each node of the search tree, an arc must be chosen for possible inclusion in the graph. We use a depth-first strategy, giving priority to the inclusion of backward arcs in the current partial Hamiltonian path $\sigma = \sigma_1, \ldots, \sigma_k$. The backward arc (σ_j, σ_i) is selected from among all the possible backward arcs, with j and i being the smallest and largest indexes, respectively. If no backward arc is possible in σ, then σ is expanded by adding a forward arc between σ_k and another vertex.

5.2. Improving the Approach with Filtering Algorithms

During the search, constraint-propagation techniques are used to reduce the search space by eliminating nonrelevant values from the domain of the variables. These techniques perform different deductions: they eliminate potential arcs that cannot lead to a WSNG or to a valid solution, they eliminate potential coordinates that cannot lead to a valid solution, they add some arcs that are mandatory for obtaining a WSNG and a valid solution, and they update the possible orientations or the backward status of arcs.

These techniques are used to adjust the domains of graph variables to the domains of coordinate variables, and vice versa. We now look at the main ideas underlying the propagation rules that we have integrated in our method.

5.2.1. Updating the Lost Space During Contractions. Each time a circuit $[u_0, u_1, u_2, \ldots, u_k]$ is contracted, the quantity of lost space is updated. The variable *lost* is updated according to the size of aggregates. In particular, if a *horizontal* contraction is performed between two aggregates i and j, then the created gap is of dimensions $width(j)^- \times (height(i)^- - height(j)^-)$ if $(height(i)^- > height(j)^-)$ and $width(i)^- \times (height(j)^- - height(i)^-)$ if $(height(i)^- < height(j)^-)$. Similarly, if the contraction is *vertical*, then the created gap is of dimensions $height(j)^- \times (width(i)^- - width(j)^-)$ if $(width(i)^- > width(j)^-)$ and $height(i)^- \times (width(j)^- - width(i)^-)$ if $(width(i)^- < width(j)^-)$. The gap is used to adjust the lower bound of $domain(lost)$ with $lost^- = lost^- + gap$.

Each time a contraction is performed, an aggregate replaces a set of items and a certain amount of lost space. The lower bounds of Carlier et al. (2007) and Clautiaux et al. (2007) are used in a feasibility test to ensure that the new set of aggregates can fit into the bin. These bounds are designed for the nonguillotine case and thus do not take into account the guillotine constraint. However, they are useful for pruning a large number of nonuseful solutions.

5.2.2. Propagation from Arcs to Arcs. In a partial solution, the information on the arcs is generally not complete, but some deductions are nevertheless performed, even if some subset of their existence, direction, and orientation remains unknown. We might know that some arcs have to be in the guillotine graph but without knowing their direction or their orientation. For other arcs, we might know their only possible direction, even though their existence has not yet been established. In certain cases neither the existence nor the direction of an arc will be known.

The deductions we perform are based on the following two properties: (1) a vertex cannot be in two consecutive contractions with the same direction, and (2) the graph has to be reduced with monochromatic contractions. Moreover, we know from Theorem 3.1 that two backward arcs cannot "cross" each other. Suppose that we have a backward arc from j to i. For any noncontracted vertex k between i and j in σ, all arcs from k to a vertex before i or after j in σ are then precluded.

Figure 8 provides an example of these kinds of deductions. The mandatory arcs are shown as continuous lines, and the possible arcs are dotted lines. If the direction of an arc is known, it is marked h or v for *horizontal* or *vertical*, respectively. Otherwise, it is marked $\{h, v\}$. Note that the relative positions of vertices are known (because of the forward arcs), except for vertices g and f. Thus we do not know whether g or f is the successor of e in σ.

5.2.3. Filtering Coordinate Variables. The arc-to-coordinates propagation techniques rely on the fact that a fixed rule is chosen a priori to determine the placing of items when a build is performed (see §5.1). Thus, some constraints on the coordinates are deduced from the existence of forward arcs in the graph.

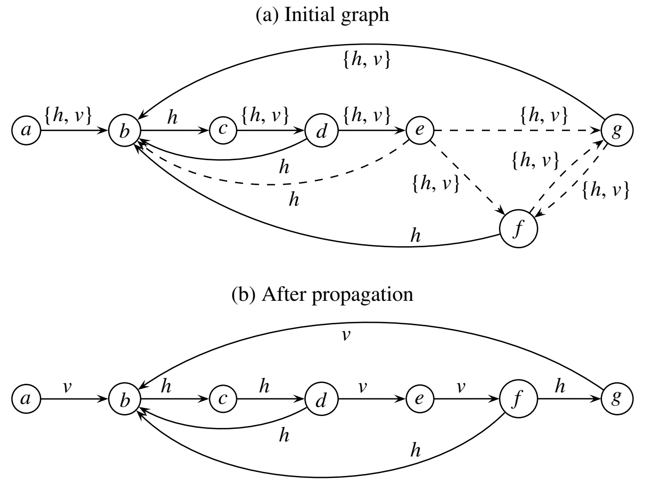

Figure 8 **Propagation from Arcs to Arcs: A Chain Propagation Removes a Set of Potential Arcs**

The labels can also be used to filter the possible values for the coordinates. Let j be an aggregate that is too tall to be packed above aggregate 0. Therefore j must be contracted in an aggregate that will be placed to the right of aggregate 0 at y-coordinate 0. Thus, the condition $\exists k \in \{1, \ldots, j\}$ with $Y(k) = Y(0) = 0$ has to be satisfied. It can be generalized to each pair of aggregates i and j with $i < j$. If $X(j)^- \geq X(i)^+ \wedge Y(j)^- \geq Y(i)^+ \wedge Y(i)^- + height(i) + height(j) > height(i)^+$, then the condition $k \in \{0, \ldots, j\}$ with $Y(k) \leq Y(i)$ has to be satisfied. The same kind of condition holds for the other dimension. If no item k meets this condition, a failure occurs. If only one item k can meet the condition, coordinates of k are adjusted according to the condition.

Geometric considerations are also used for filtering possible values for the arcs and the coordinates. Let $\mu = [u_1, \ldots, u_k]$ be a monochromatic (horizontal or vertical) subpath in σ of the guillotine graph (at any contraction step). The following condition should be satisfied: $X(u_1)^- + \sum_{i=1}^{k-1} width(i)^- \leq X(u_k)^+$ if it is a horizontal path or $Y(u_1)^- + \sum_{i=1}^{k-1} height(i)^- \leq Y(u_k)^+$ if it is a vertical path. Using this constraint allows us to eliminate aggregates as potential successors of u_k in σ and to adjust the coordinates of the aggregates belonging to the subpath.

5.2.4. Breaking Symmetries. Supposing we have a well-sorted normal guillotine graph for which the associated guillotine pattern fits into the boundaries of the bin. If at any step of the contraction process there exist two aggregates i and j such that $width(i)^- = width(j)^-$ and $height(i)^- = height(j)^-$, then there exists a well-sorted normal guillotine graph associated with a valid pattern such that i is before j in the related Hamiltonian path. We can use this dominance rule in the following way. If during the search two aggregates i and j $(i < j)$ have the same dimensions, a lexicographic constraint can be added. Using our model, this constraint is straightforward to include $before(i, j)$ is set to *true*; i.e., we ensure that i is before j in σ.

5.3. Computational Experiments

The constraint programming approach was implemented in C++ using the ILOG Solver (ILOG 2004) and was run on a Genuine Intel 2.16 GHz T2600 CPU. We tested our method on the strip-cutting problem in which the width of the bin is fixed and the minimal feasible height for the bin must be determined. This problem gives rise to a set of feasible or infeasible decision problems.

5.3.1. Validation of Our Method. In Table 1 we compare several of our methods using the benchmarks proposed by Clautiaux et al. (2007), which were initially generated for the nonguillotine 2OPP problem. An instance "M, X" contains M items. These instances are available on the Web at http://www2.lifl.fr/~clautiau/pmwiki/pmwiki.php?n=Research.Benchmarks. This test bed was chosen because it contains both easy and hard instances, most of them being solved in a reasonable time by our best methods. Thus, it offers an excellent compromise for comparing a number of our methods.

For each method, we provide the number of nodes ("Nodes") and the computing time in seconds ("CPU (s)"). The symbol "–" means that an optimal solution was not found within 3,600 seconds. Four methods were tested, three of which are based on the graph model. Let $[H_{\min}, H_{\max}]$, respectively, be a lower and an upper bound of the height of the bin (initially, $H_{\min} = (1/W) \sum_{i \in I} h_i w_i$ and $H_{\max} = \sum_{i \in I} h_i$). The column H^* provides the optimal height for each instance of the problem. Moreover, the column H^+ provides the smallest height for which the nonguillotine version of the problem is feasible.

The first graph-based method, IGG$_{\text{ub}}$, solves the search problems from the upper bound to the lower bound. First, it searches for a solution whose height is lower than or equal to $H_{\max}$. If a solution with height H is found, then the next search problem to be solved is finding a solution with a height strictly lower than H. As long as solutions continue to be found, the procedure is repeated, stopping only when it encounters an infeasible problem. The optimal height H^* is the height of the last search problem for which a solution exists. For this method only one infeasible problem is considered (the problem in which we search for a solution with a height strictly lower than H^*).

The second method, IGG$_{\text{lb}}$, goes from the lower bound to the upper bound and stops as soon as a feasible solution is found. First, it searches for a solution whose height is lower than or equal to $H_{\min}$. If no solution is found, then the next search problem to

Table 1 Experimental Results on Clautiaux et al. (2007) Instances

					IGG_{ub}		IGG_{lb}		$2SP + IGG_{lb}$		$2SP + 2SCP$	
Instance	H_{min}	H_{max}	H^+	H^*	Nodes	CPU (s)	Nodes	CPU (s)	Nodes	CPU (s)	Nodes	CPU (s)
23, A	20	106	21	21	348,457	449.25	113,090	202.23	32,982 (11,945)	30.52 (2.43)	2,246,142	206.90
23, B	20	108	21	21	18,347	35.1719	13,877	31.48	196 (136)	0.19 (0.11)	—	—
22, A	20	122	20	20	1,252,332	1,633.92	364,558	539.03	364,590 (32)	471.79 (0.04)	223	0.01
20, A	20	126	20	21	—	—	898,690	1,065.42	909,236 (10,546)	967.24 (0.77)	—	—
20, B	20	134	22	22	—	—	344,622	437.68	703 (46)	0.82 (0.05)	—	—
20, C	20	142	20	20	358,166	333.62	1,376	2.07	1,421 (45)	2.16 (0.04)	13,381,014	1,578.09
20, D	19	115	20	20	285,334	342.68	178,617	231.51	178,643 (27)	209.25 (0.04)	—	—
19, A	20	117	20	20	121,200	125.48	95,532	106.45	95,561 (29)	93 (0.1)	—	—
18, A	20	114	20	22	1,845,495	1,636.48	1,844,271	1,796.83	1,844,294 (23)	1,570.47 (0.05)	—	—
18, B	20	124	21	?	—	—	—	—	—	—	—	—
18, C	19	119	20	21	211,421	183.04	103,725	108.42	103,747 (23)	95.83 (0.04)	—	—
17, A	20	112	20	21	142,921	138.37	96,020	104.98	96,042 (22)	94.65 (0.02)	—	—
17, B	20	105	21	22	10,463	10.95	9,557	10.04	9,089 (27)	8.36 (0.05)	—	—
17, C	20	97	20	21	48,027	44.92	43,383	42.43	43,406 (23)	37.63 (0.02)	—	—
17, D	20	141	21	?	—	—	—	—	—	—	—	—
17, E	20	115	21	22	404,003	317.81	307,684	273.43	306,724 (92)	247.47 (0.05)	—	—
16, A	20	118	21	22	2,093	2.06	893	1.00	10,987 (10,230)	2.63 (1.93)	—	—
15, A	20	87	21	22	32,784	22.31	3,357	3.09	3,631 (275)	2.71 (0.13)	8,094,280	840.81
15, B	20	116	21	21	16,877	15.57	5,016	4.76	4,987 (90)	4.24 (0.05)	9,203,561	1,150.14
15, C	20	127	22	24	14,005	10.45	10,085	8.67	12,720 (2,803)	8.25 (0.68)	—	—
15, D	20	82	20	22	86,903	64.70	87,798	68.20	87,984 (186)	60.75 (0.07)	—	—
15, E	19	76	20	22	24,260	17.87	27,572	21.18	28,166 (737)	18.91 (0.22)	—	—
15, F	19	142	21	23	705,324	400.40	821,428	506.51	824,000 (2,578)	453.55 (1.11)	—	—
15, G	19	97	21	21	67,581	46.45	45,834	37.54	21,766 (1,332)	15.19 (0.36)	—	—
15, H	19	91	21	22	404,173	222.95	201,898	153.04	149,571 (404)	101.93 (0.16)	—	—
15, I	19	118	20	21	9,069	6.23	5,116	4.15	5,144 (28)	3.79 (0.05)	—	—
15, J	19	147	21	21	5,025	3.03	196	0.12	218 (23)	0.29 (0.04)	80	0.06
15, K	19	113	21	21	752,717	384.78	135,752	82.42	135,773 (22)	76.54 (0.01)	140	0.07
15, L	19	115	20	21	3,356	2.37	1,148	1.01	1,276 (131)	0.82 (0.11)	—	—
15, M	18	165	22	22	2,627	1.92	323	0.23	1,541 (1,220)	0.68 (0.54)	—	—
15, N	18	112	21	22	4,744	3.34	393	0.40	902 (653)	0.38 (0.19)	45,070,463	3,036.72
15, O	18	106	22	22	10,065	7.43	9,463	7.14	233 (158)	0.13 (0.11)	—	—
15, P	18	85	22	22	2,605	2.53	1,333	1.03	1,357 (26)	1.08 (0.04)	142	0.10
15, Q	18	143	22	22	40,174	19.44	9,638	4.31	10,021 (385)	4.66 (0.10)	441	0.15
15, R	16	77	17	21	1,219,272	749.56	1,220,724	767.93	1,220,750 (27)	713.36 (0.02)	—	—
15, S	17	93	20	21	1,469	1.84	90	0.10	114 (24)	0.07 (0.04)	—	—
10, A	20	88	22	23	223	0.34	139	0.10	599 (496)	0.19 (0.15)	705,993	42.45
10, B	20	62	21	24	360	0.39	183	0.06	196 (14)	0.15 (0.02)	33,877,520	2,251.47
10, C	19	88	22	23	539	0.40	142	0.09	313 (176)	0.11 (0.04)	—	—
10, D	18	86	21	25	431	0.39	331	0.15	346 (16)	0.19 (0.05)	—	—
10, E	18	69	21	22	266	0.45	292	0.12	262 (39)	0.24 (0.08)	—	—

be solved is finding a solution with a height lower than or equal to $H_{min} + 1$. Although no solution has been found, the procedure is repeated and stops as soon as it reaches a feasible problem. The optimal height H^* is the height of the first search problem for which a solution exists. For this method only one feasible problem is considered (the problem in which we search for a solution with a height lower than or equal to H^*), whereas all infeasible problems with a maximal height in $\{H_{min}, \ldots, H^* - 1\}$ are considered.

The last graph-based method, $2SP + IGG_{lb}$, is a variant of IGG_{lb} in which the feasibility of the nonguillotine case is first checked for each decision problem to solve (using Clautiaux et al. 2008). If the problem obtained by removing the guillotine constraint has no solution, then the guillotine-cutting solver is not run for the current size. In other words, the method of Clautiaux et al. (2008) is run to adjust the lower bound H_{min}. Thus, method 2SP may be used to solve search problems with a fixed height in $\{H_{min}, \ldots, H^+\}$. These problems are not feasible for either the nonguillotine version or the guillotine version of the problem, apart from the last one, which is feasible at least for the nonguillotine version, whereas method IGG_{lb} is used to solve search problems with a fixed height in $\{H^+, \ldots, H^*\}$.

The final method, $2SP + 2SCP$, is an adaptation of the algorithm of Clautiaux et al. (2008), originally designed for the nonguillotine case. This method enumerates solutions for the nonguillotine case until the pattern found is guillotine. We use the algorithm of Ben Messaoud et al. (2008) to prune nonguillotine solutions. Practically speaking, the most efficient method was obtained by using the method at the leaf nodes only. This can explained by the fact that the method of Clautiaux et al. (2008) only considers the X-coordinates in the first phase (and thus the guillotine test is only useful in the deep nodes of the search).

The computing time required by the algorithms is large compared to the time required by Clautiaux et al. (2008) to solve the nonguillotine version. This fact justifies the third method, which first checks the feasibility of the nonguillotine case. For this method, we report the computation time and the number of nodes for the total method, with the proportion taken by 2SP shown in parentheses, so the reader can verify that the time needed to solve the nonguillotine case is generally significantly shorter than the time needed to solve the guillotine case. This method is better than the other versions tested for these particular instances. Comparing IGG_{lb} and $2\text{SP} + \text{IGG}_{\text{lb}}$, we note that when a search problem is not feasible for either the nonguillotine case or the guillotine case, method 2SP is generally significantly more efficient than IGG. For example, for instance 23, B, to prove that there is no solution with a height equal to 20, method 2SP takes only 0.11 seconds, whereas IGG takes more than 31 seconds. For this instance, the last search problem for a guillotine solution with a height set to 21 takes only 0.08 seconds to be solved. The heuristic used at the separation of the enumerative method is more efficient in this last search problem than in the previous one, which was infeasible. However, it should be noted that this is not usually the case.

By comparing the results obtained by the three guillotine-based methods with $2\text{SP} + 2\text{SCP}$, we see that our model really does provide useful information. Method $2\text{SP} + 2\text{SCP}$ is seldom able to improve on the methods that use the guillotine model, and it is unable to solve most of the instances, although the method is efficient for these instances in the nonguillotine case. The ineffectiveness of the method comes the fact than it was originally designed for the nonguillotine problem. In particular, as explained earlier, the method of Clautiaux et al. (2008) only considers the X-coordinates in a first phase, and thus the guillotine test can be done only in the deep nodes of the search. The consequence is that a lot of feasible solutions for the nonguillotine problem are considered (because they are not filtered by the method), whereas they are not feasible for the guillotine case, this unfeasability being detected only at the leaves of the search tree.

Method IGG_{lb} is (other than in five instances) better than IGG_{ub} both in terms of computation time and search space. This is explained by the fact that IGG_{lb} usually has fewer search subproblems to solve than IGG_{ub}. It can also be explained by the fact that method IGG (which searches for a feasible solution for a fixed height H) is generally better at proving that a problem has no solutions than at finding a feasible solution, suggesting that the heuristic used at the separation of the enumerative method might be improved so as to find a feasible solution more quickly.

5.3.2. Comparison with the Methods in the Literature. In Table 2 we compare our methods with algorithms in the literature, using 25 instances Hifi (1998) derived from strip-cutting problems. In the instances considered, the size of the bin varies from 4 to 100, and the number of items is 22 for the largest instance.

The two modified versions of Viswanathan and Bagchi's (1993) algorithm (IMVB and BMVB) proposed by Hifi (1998) are considered. The basic idea of these algorithms is to maintain a list of builds, initialized by the set of original items, and to try to achieve a build using two items/builds in the list. The new item obtained is then added to the list, and the algorithm is run recursively. The branching scheme consists of finding which pair of current items is built at each step. We also consider the method of Bekrar et al. (2010), which is an adaption of Martello et al. (2003).

The results reported for our method are those obtained by solving the decision problems from below (from a known lower bound for the height of the bin, and upward). For each method, except for the method of Bekrar et al. (2010), we provide the number of nodes ("Nodes") and the computing time in seconds ("CPU (s)"). Unfortunately, only the computational times are provided in Bekrar et al. (2010). Note that there is a time limit of 1,000 seconds for the method of Bekrar et al. (2010). The instances that are not solved within this time limit are noted in the table. The value 1,000 seconds is then taken for these instances to determine the average computation time.

Comparing the computing times of the different methods is tricky because the results of the IMVB and BMVB methods are those provided by Hifi (1998), obtained in 1998 using a Sparc-Server 20/712. To give a fair comparison, we also provide a third column ("nbMI") for all methods, which represents the CPU time multiplied by the number of millions of instructions per second (MIPS) of the machine (252 MIPS for the Sparc-Server 20/712, 7,174 MIPS for the machine used by Bekrar et al. 2010, and 13,207 MIPS for our machine). This indicator should be used for approximative comparison purposes only. The number of nodes, on the other hand, is a reliable indicator, and it is consistent with the computing times observed. We were able to solve each test case in less than three seconds. For several instances, the difference in terms of nodes in the search tree is large. For example, the

Table 2 Comparison with Methods in the Literature

		IMVB (Hifi 1998)			BMVB (Hifi 1998)			Bekrar et al. (2008)		IGG_{lb}		
Instance	*n*	Nodes	CPU (s)	nbMI	Nodes	CPU (s)	nbMI	CPU (s)	nbMI	Nodes	CPU (s)	nbMI
SCP1	10	69	0.1	25	36	0.1	25	0.01	72	15	0.01	206
SCP2	11	1,797	3.4	857	409	1.2	302	0.01	72	63	0.03	413
SCP3	15	3,427	6.8	1,714	1,650	19.4	4,889	7.1	50,935	56	0.04	619
SCP4	11	6,356	78.6	19,807	1,125	4.1	1,033	2.45	17,576	494	0.20	2,683
SCP5	8	5	0.1	25	141	0.1	25	0.01	72	13	0.00	0
SCP6	7	8,012	54.6	13,759	536	3.4	857	0.01	72	19	0.00	0
SCP7	8	1,195	1.8	454	458	1.4	353	0.01	72	30	0.01	206
SCP8	12	484	0.4	101	721	2.3	580	1.57	11,263	20	0.00	0
SCP9	12	60	0.7	176	72	0.1	25	0.14	1,004	32	0.01	206
SCP10	8	4	0.1	25	9	0.5	126	0.18	1,291	13	0.00	0
SCP11	10	11,036	221.5	55,818	530	0.7	176	4.89	35,081	21	0.01	206
SCP12	18	908	1.3	328	2,029	2.7	680	1,000[a]	7,174,000	58	0.07	1,032
SCP13	7	4,359	39.5	9,954	1,278	16.8	4,234	0.38	2,726	213	0.03	413
SCP14	10	4,782	41.7	10,508	2,475	48.7	12,272	15.57	111,699	600	0.25	3,302
SCP15	14	673	0.7	176	26,728	165.7	41,756	1,000[a]	7,174,000	88	0.06	825
SCP16	14	85,627	654.8	165,010	4,409	181.4	45,713	606.12	4,348,305	809	0.46	6,191
SCP17	9	13,668	227.3	57,280	4,751	323.6	81,547	2.07	14,850	815	0.26	3,508
SCP18	10	22,087	321.5	81,018	7,113	327.8	82,606	9.43	67,651	1,781	0.75	9,905
SCP19	12	39,550	1,794.3	452,164	10,441	473.0	119,196	0.01	72	1,040	0.48	6,397
SCP20	10	36,577	874.3	220,324	12,326	673.9	169,823	1.73	12,411	1,127	0.40	5,365
SCP21	11	26,748	1,757.6	442,915	17,552	2,001.8	504,454	24.31	174,400	2,440	1.03	13,620
SCP22	22	40,909	606.0	152,712	104,013	757.8	190,966	1,000[a]	7,174,000	293	0.32	4,334
SCP23	12	29,512	691.9	174,359	48,612	1,031.9	260,039	37.97	272,397	1,545	0.67	8,873
SCP24	10	117,013	6,265.0	1,578,780	45,143	5,585.7	1,407,596	19.34	138,745	6,458	2.45	32,398
SCP25	15	69,434	3,735.8	941,422	27,344	2,662.9	671,051	1,000[a]	7,174,000	3,105	2.15	28,478
Average		20,972	695	175,188	12,796	571	144,013	189	1,358,271	846	0.39	5,167

[a]Instances not solved within the time limit.

IMVB method of Hifi (1998) needs 40,909 branching points to solve SCP22, whereas our method needs only 293. On average, our method needs 15 times fewer nodes that the best approach, BMVB of Hifi (1998). These results show that the additional information provided by our model enhances the exploration of branches in a tree search. Moreover, when computation times are compared, our method would appear to outperform that of Bekrar et al. (2010).

For several easy cases, the number of MIPS is larger for our method. This happens when the computing time is small enough to make the setup time (creating the constraint programming variables and constraints) significant in relation to the total time. For the instance SCP10, our method needs a few more nodes than the IMVB method to find a feasible solution. This suggests that our branching heuristic was not suited for this instance.

6. Conclusion

We have proposed a new graph-theoretical model for the two-dimensional guillotine-cutting problem. We propose an analysis of the combinatorial structure of these graphs and several algorithms of linear complexity to handle them. Computational experiments show that our model provides useful information that can be used in an exact algorithm, and it leads to results that outperform those of the literature. We wish to emphasize that this model may also be used for the three-dimensional case, using three different colors.

Electronic Companion

An electronic companion to this paper is available as part of the online version at http://dx.doi.org/10.1287/ijoc.1110.0478.

Acknowledgments

The authors thank Abdelghani Bekrar for providing detailed information about his computational results. The authors also thank the referees, whose comments helped improve the quality of the paper.

References

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

Bekrar, A., I. Kacem, C. Chu, C. Sadfi. 2010. An improved heuristic and an exact algorithm for the 2D strip and bin packing problem. *Internat. J. Product Development* **10**(1–3) 217–240.

Beldiceanu, N., M. Carlsson. 2001. Sweep as a generic pruning technique applied to the nonoverlapping rectangles constraints. T. Walsh, ed. *Principles and Practice of Constraint Programming (CP 2001)*. Lecture Notes in Computer Science, Vol. 2239. Springer, Berlin, 377–391.

Beldiceanu, N., M. Carlsson, E. Poder, R. Sadek, C. Truchet. 2007. A generic geometrical constraint kernel in space and time for handling polymorphic *k*-dimensional objects. C. Bessière, ed. *Principles and Practice of Constraint Programming* (*CP* 2007). Lecture Notes in Computer Science, Vol. 4741. Springer, Berlin, 180–194.

Belov, G., G. Scheithauer. 2006. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *Eur. J. Oper. Res.* **171**(1) 85–106.

Ben Messaoud, S., C. Chu, M.-L. Espinouse. 2008. Characterization and modelling of guillotine constraints. *Eur. J. Oper. Res.* **191**(1) 112–126.

Carlier, J., F. Clautiaux, A. Moukrim. 2007. New reduction procedures and lower bounds for the two-dimensional bin-packing problem with fixed orientation. *Comput. Oper. Res.* **34**(8) 2223–2250.

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

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

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

Cui, Y., D. He, X. Song. 2006. Generating optimal two-section cutting patterns for rectangular blanks. *Comput. Oper. Res.* **33**(6) 1505–1520.

Fekete, S. P., J. Schepers. 2004. A combinatorial characterization of higher-dimensional orthogonal packing. *Math. Oper. Res.* **29**(2) 353–368.

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

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Gilmore, P. C., R. E. Gomory. 1965. Multistage cutting stock problems of two and more dimensions. *Oper. Res.* **13**(1) 94–120.

Golumbic, M. C. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

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

ILOG. 2004. *ILOG Solver Reference Manual*. ILOG, Gentilly, France.

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

Viswanathan, K. V., A. Bagchi. 1993. Best-first search methods for constrained twodimensional cutting stock problem. *Oper. Res.* **41**(4) 768–776.

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

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