A new graph-theoretical model for the guillotine-cutting problem

François Clautiaux[+,*], Antoine Jouglet[†] and Aziz Moukrim[†],
+: Université des Sciences et Technologies de Lille,
LIFL UMR CNRS 8022, INRIA,
†: Université de Technologie de Compiègne,
HeuDiaSyC, UMR CNRS 6599

October 29, 2010

Abstract

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 term 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.

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 a cut 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 [22] 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 (UTDGC). This problem is *NP-complete*, as it generalizes the classical bin-packing problem $1BP$ (see [14]).

A guillotine-cutting instance D is a pair (I, B). 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 *guillotine* (see Figure 1) if it can be obtained using guillotine cuts only. It can be checked in $O(n^2)$ time [6].

The two-dimensional guillotine-cutting problem has been widely studied in the OR literature. One approach to tackling this problem efficiently is to use restrictions on the cutting patterns, such as *staged cutting patterns* (see [15, 1, 5]), or *two-section cutting patterns* (see [11]). This paper addresses the standard guillotine-cutting problem. In the literature to date we find two alternative methods for solving the guillotine problem (see [17]). The first approach [8] consists

*Corresponding author: F. Clautiaux, IUT A, département informatique, Cité Scientifique, 59650 Villeneuve d'Ascq, francois.clautiaux@univ-lille1.fr

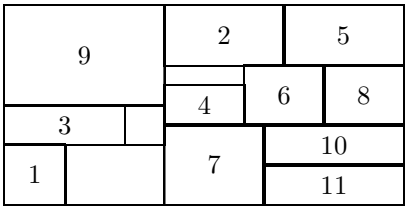


Figure 1: A guillotine pattern.

in iteratively cutting the bin into two rectangles, using horizontal or vertical cuts, until all the required rectangles are obtained. The second approach [20] recursively merges items into larger rectangles, using so-called horizontal or vertical *builds* [21]. To our knowledge the most recent work on the subject is by Bekrar *et al.* [2], and provides an adaptation of the branch-and-bound method of Martello *et al.* [19].

Graph-theoretical approaches can be found in the literature to solve some packing problems. In particular, for the two-dimensional packing problem $2OPP$, Fekete and Schepers [12] show that a pair of interval graphs (see [16]) 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.* propose a method [13] for seeking a pair of interval graphs with the properties described above. In comparison with the classical algorithms their method avoids a large number of redundancies, and is still competitive compared to more recent methods [9, 10].

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 [12], 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, which relies heavily on the properties of guillotine graphs.

In Section 2, we introduce the concept of guillotine-cutting classes. Section 3 is devoted to definitions and properties of guillotine graphs. In Section 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 Section 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 [17] and [2] on randomly-generated guillotine strip-packing instances from the literature. When applied to the difficult instances discussed in the literature, our method significantly outperforms previous ones.

2 Guillotine-cutting classes

In order to avoid equivalent patterns in the non-guillotine orthogonal-packing problem ($2OPP$), Fekete and Schepers [12] 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, since two different packing classes may give rise to patterns having the same combinatorial structure. We introduce the concept of a *guillotine-cutting class* which takes into account the fact that exchanging the positions of two rectangular

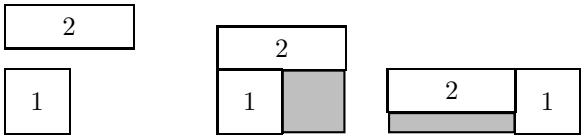


Figure 2: Vertical and horizontal builds of two items 1 and 2.

blocks of items does not change the combinatorial structure of the solution. The definition uses the notion of so-called *builds*.

A build [21] 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 $\{i,j\}^h$, 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 $\{i,j\}^v$, 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 [20]). 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, pattern-wise, to any sequence of builds in set $\{(\{i_1,i_3\}^d, \{i_1,i_2\}^d), (\{i_2,i_3\}^d, \{i_1,i_2\}^d)\}$.

From now on, instead of using sequences of builds, we shall use expressions that we call *recursive multi-builds*. These expressions allow us to eliminate this kind of redundancy by replacing successive sequences of builds in the same direction by 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 multi-build.

Definition 2.1. *A recursive multi-build (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 multi-builds, comprising either a single item, or a recursive multi-build of dimension $d' \neq d$.*

For example, the following recursive multi-build is related to Figure 1:

$$\Big\{\{1,3,9\}^v, \big\{\{2,5\}^h, \{4,6,8\}^h, \{7,\{10,11\}^v\}^h\big\}^v\Big\}^h \tag{1}$$

In the following, the set of all RMBs which 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 with the same RMB. Indeed, as we have already seen, an RMB represents a lot of different sequences of builds, each of the sub-sequences 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 as 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 multi-build.*

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

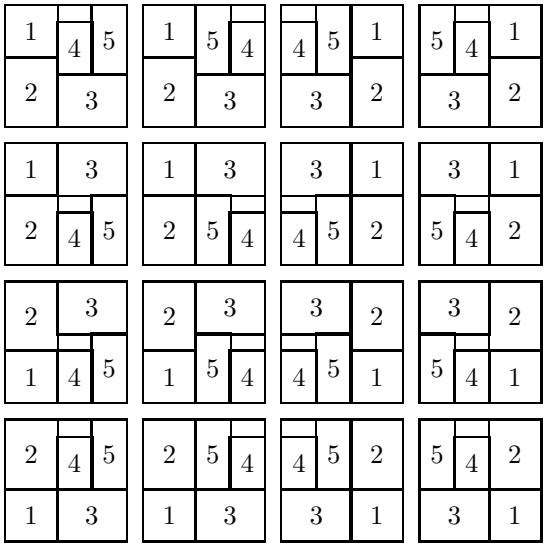


Figure 3: Some elements of a guillotine-cutting class.

if the bottom-left rule is not used. For example, sliding item 4 up can lead to another equivalent pattern.

Note that since any pattern can be obtained by the application of a sequence of builds [21], 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 in the 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, since they correspond to different sequence of cuts.

It is of particular concern 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.

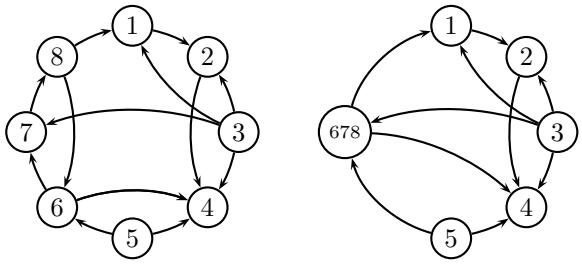
3 A new graph model for the guillotine-cutting problem

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 we 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 Section 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 re-use classical definitions and properties from graph theory.

Our graph model represents RMBs, and thus the **recursive structure** of the problem. This is different from [12], where graphs are used to represent the relative coordinates of the items.

In order 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 (WSNG)*. We show that there is a bijection between the set of WSNGs and the set of guillotine cutting classes.



(a) initial graph

(b) after circuit-contraction of $(6, 7, 8, 6)$

Figure 4: Circuit-contraction.

In this section, we focus on theoretical results (definitions, 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 multi-builds 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 vertex u and vertex 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* (resp. circuit) μ is a path (resp. 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.

In order 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 in deleting u, v and all arcs incident to u or incident to 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 incident to v. In Figure 4, contracting the circuit $\mu = [(6, 7), (7, 8), (8, 6)]$ in the left-hand graph yields the right-hand graph.

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

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

Definition 3.2. *Let G be an arc-bicolored directed graph. G is a* guillotine graph *if G 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*
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 degree strictly greater than two.*

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.

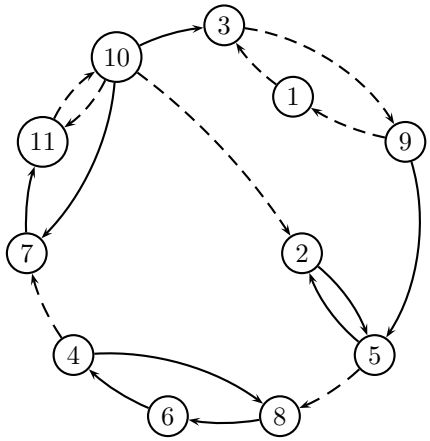


Figure 5: Modeling the guillotine-cutting class associated with the pattern of Figure 1 with a guillotine graph. The doted arcs correspond to the vertical dimension, whereas the plain arcs correspond to the horizontal dimension.

Contracting a circuit of a guillotine graph is equivalent to building a new RMB including 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).

Let $\mathcal{G}_I$ be the set of guillotine graphs of vertex set $I = \{1, \ldots, n\}$. The association between guillotine graphs and guillotine-cutting classes can be formally expressed with the application $\kappa : \mathcal{G}_I \to \mathcal{R}_I$. 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 $G = (I, A)$
while $|I| > 1$ **do**
  $U \leftarrow$ set of monochromatic circuits of G ;
  foreach $\mu = (u_1, \ldots, u_k) \in U$ **do**
    $d \leftarrow \chi(u_1, u_2)$ contract μ in G and label the resulting vertex with $r = \{u_1, \ldots, u_k\}^d$;
return *the label of the unique vertex of* G

In a graph G, contracting a circuit of color h (resp. v) corresponds to a horizontal (resp. 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 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 respectively by single vertices labeled $\{1, 3, 9\}^v$, $\{2, 5\}^h$, $\{4, 6, 8\}^h$, and $\{10, 11\}^v$. Then, a new monochromatic circuit including vertex 7 and the newly created vertex labeled $\{10, 11\}^v$ appears, and is contracted again, giving a vertex labeled $\{7, \{10, 11\}^v\}^h$. Then, a new vertex is created with 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.*

Proof. At each contraction the number of vertices in G strictly decreases. From the definition of a guillotine graph we know that there must exist a monochromatic circuit at each step of the algorithm. Consequently, Algorithm 1 terminates.

We now prove that the algorithm returns an RMB related to I. We prove the following loop invariant: *at the end of each iteration of the "while" loop, G is a guillotine graph and each vertex label is an RMB.* The property is initially satisfied: each vertex of G has a value $i \in I$ (which is an RMB) as a label, and G is a guillotine graph by hypothesis.

The contraction of a monochromatic μ of G has no impact on the vertices which do not belong to μ. Thus, either the graph obtained after contraction is guillotine, or G did not satisfy Definition 3.2, which contradicts the fact that G is guillotine. By construction, if the labels of the different elements of μ were RMBs, the obtained label is an RMB. This is true because in a guillotine graph a vertex cannot belong to two monochromatic circuits, and thus there cannot be an RMB $x = (x_1, \ldots, x_k)^d$ with a x_j $(j = 1, \ldots, k)$ such that $x_j = (x'_1, \ldots, x'_l)^d$ has the same direction as x. □

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

Proof. Our proof consists of constructing from any RMB r a graph G such that $\kappa(G) = r$, using an iterative algorithm. For this purpose, we introduce the following κ^{-1} algorithm.

Algorithm 2: Inverse algorithm κ^{-1}

```
Data: An RMB r
V ← {r};
E ← ∅;
while |V| < [I] do
    select a vertex v ∈ V labeled by an RMB containing x = (x_1, ..., x_k)^d, k > 1;
    V' ← V \ {v};
    create k items v_1, ..., v_k, respectively labeled x_1, ... x_k;
    V' ← V' ∪ {v_1, ..., v_k} ;
    create k arcs (v_1, v_2), ..., (v_{k-1}, v_k), (v_k, v_1) of color d;
    E' ← E ∪ {(v_1, v_2), ..., (v_{k-1}, v_k), (v_k, v_1)};
    for (w, v) ∈ E of color d' do
        create an arc (w, v_1) of color d';
        E' ← (w, v_1);
    for (v, w) ∈ E of color d' do
        create an arc (v_k, w) of color d';
        E' ← (v_k, w);
    V ← V';
    E ← E';
return G=(V,E);
```

The algorithm terminates because the number of vertices strictly increases at each step of the loop, and since r is a valid RMB, when $|V| < n$, there is a vertex whose label is not a single item.

We now prove the following loop invariant: *at each step of the "While" loop, G is guillotine, and $\kappa(G) = r$.* Initially, the invariant is satisfied, since G contains a single item (thus it is a guillotine graph), and $\kappa(G)$ is simply r.

Now, suppose the invariant is satisfied at step i of the algorithm, and consider step $i + 1$. At the beginning of the step, either vertex v does not belong to a monochromatic circuit or it belongs to a monochromatic circuit of color d' (since $G = (V, E)$ is guillotine by assumption). If v does not belong to a monochromatic circuit in G, then new vertices $v_1, \ldots, v_z$ belong to only one monochromatic circuit of color d. Conversely, if x belongs to a monochromatic circuit μ of color d' in G, let vertices p and s be respectively the predecessor and the successor of v in this circuit (in some cases, $p = s$). In $G' = (V', E')$, the monochromatic quality of μ is broken by the insertion of the circuit of color d built with vertices $v_1, \ldots, v_z$. Thus, the vertices belonging to

$\mu \setminus \{v\}$ no longer belong to a monochromatic circuit, while vertices $v_1, \ldots, v_z$ belong to exactly one monochromatic circuit of color d. In all cases, we obtain a new graph G' whose vertices belong to at most one monochromatic circuit and on which the application of one iteration of the outer loop of Algorithm 1 yields G, which is guillotine by assumption. Therefore G' is guillotine and applying Algorithm 1 to G' leads to r.

Consequently, at the end of the algorithm, G contains n vertices, each labeled by a single item, and $\kappa(G) = r$, which is the result sought.

□

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.*

Proof. We consider the iterative contraction process of Algorithm 1. When a circuit μ is contracted, the number of deleted arcs in the graph is n_1, where n_1 is the number of vertices in this circuit. Let r be the number of iterations needed. When Algorithm 1 is applied, it removes $(n_1 - 1) + (n_2 - 1) + \ldots + (n_r - 1)$ vertices, and $n_1 + n_2 + \ldots + n_r$ arcs corresponding to circuits $\mu_1, \mu_2, \ldots, \mu_r$.

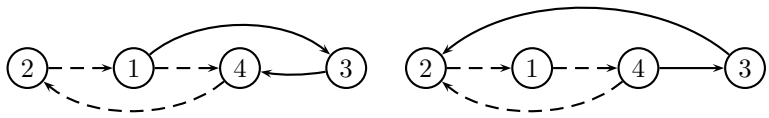
After r iterations, $n-1$ vertices have been removed, so $n-1+r$ arcs have also been removed. Since each arc is removed once, the initial number of arcs is equal to $n-1+r$. The minimum possible value for r is 1 (when all items are cut side by side), and the maximum possible value is $n-1$ (when only one vertex is removed at each step). Thus we obtain the following relation: $n \leq m \leq 2n-2$. □

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 degree greater than two 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 degree strictly greater than two, such that $(x_i, x_j) \in \mu$, $|N^+(x_i)| = 1$ and $|N^-(x_j)| = 1$.*



(a) non-normal (b) normal

Figure 6: Two equivalent guillotine graphs. The first is non-normal, the second is normal.

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

(vertex 4 in Figure 6), 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.*

Proof. We prove this result by recurrence, with reference to the number of vertices $|V|$ in the graph. If $|V| = 1$, the result is direct (the Hamiltonian circuit of size zero consists only of the item 1).

Suppose that the result is true for any normal guillotine graph such that $|V| \geq n$, and consider a normal guillotine graph G with $n+1$ elements. By definition, G contains a monochromatic circuit $\mu = (v_1, \ldots, v_k, v_1)$. Let G' be the graph obtained by contracting μ into a single vertex v. By definition G' is a normal guillotine graph with at most n vertices, and thus by assumption it contains a single Hamiltonian circuit $\mu_{G'} = (v'_1, \ldots, v'_i, v, v'_j, \ldots, v'_{n-k+2})$. From this circuit, we can construct a Hamiltonian circuit μ_G for G by replacing v by the path $(v_1, \ldots, v_k)$.

Now we have to prove that μ_G is unique. First, note that since $v_2, \ldots, v_{k-1}$ are of degree two (because the graph is normal), any Hamiltonian circuit includes $(v_1, \ldots, v_k)$. Suppose that G contains two different Hamiltonian circuits $\mu_G = (w_1 = v_k, \ldots, w_2, \ldots, v_1, v_2, \ldots, v_{k-1}, v_k)$ (defined above) and $\mu^2_G = (z_1 = v_k, \ldots, v_1, \ldots, v_k)$. This means that there is an index $l < n+1$ such that $z_j = w_j$ for any $j < l$ and $z_l \neq w_l$. We consider two possibilities for z_l: either $z_l \in \{v_1, \ldots, v_k\}$ or it is not. If $z_l \in \{v_1, \ldots, v_k\}$, then $z_l = v_1$ (otherwise G would not be normal). Since the out-degree of vertices $v_1, \ldots, v_{k-1}$ is one, the path in construction will be $(z_1 = v_k, \ldots, z_{l-1}, v_1, \ldots, v_k, \ldots z_{n+1}, z_1)$, and therefore, the circuit cannot be Hamiltonian (because there is a subtour). The second possibility is that $z_l \notin \{v_1, \ldots, v_k\}$. In this case, the path has the following structure: $(z_1 = v_k, \ldots, z_{l-1}, z_l \neq w_l, \ldots, v_1, \ldots, v_k)$. This means that G' contains two different Hamiltonian circuits $(v, \ldots, w_{l-1}, w_l, \ldots, v)$ and $(v, \ldots, z_{l-1}, z_l, \ldots, v)$ with $w_l \neq z_l$. This contradicts the assumption. Therefore, the Hamiltonian circuit contained within G is unique. □

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 the 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 $\mu_G = (v_1, \ldots, v_n)$ its 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.*

Proof. We prove this result by constructing this ordering. It can be done during the contraction process of G by iteratively adding arcs between vertices in the circuit currently being contracted until a path is obtained. When the path is obtained, the corresponding ordering is normal.

Initially, we consider a graph $P = (I, A_P = \emptyset)$ containing all the vertices of G but no arcs. Each time a monochromatic circuit of G is contracted, we add a partial ordering between the vertices of the circuit by constructing a path between these vertices in P. Consider a monochromatic circuit $\mu = (x_1, x_2, \ldots, x_{k-1}, x_k, x_1)$ to be contracted in G, such that x_1 and x_k are the only vertices of degree strictly greater than two in μ with $|N^+(x_1)| = 1$ and $|N^-(x_k)| = 1$. Each vertex of μ has a corresponding path in P. Then for each pair $(x_i, x_{i+1}), i \in \{1, \ldots k-1\}$, an arc between the last vertex of the path associated with x_i and the first vertex of the path associated with x_{i+1}) is added in A_P. At the last step of the contraction process the monochromatic circuit μ to be contracted is such that all vertices are of degree 2. Then one vertex is chosen to be x_1 in $\mu = (x_1, x_2, \ldots, x_{k-1}, x_k, x_1)$ and the same procedure applied. We then obtain a path P, corresponding to a unique ordering σ.

By construction, when there is an arc (i, j) that is not in the Hamiltonian circuit, the vertex j comes before i in σ (because i played the role of x_1 and j the role of x_k in the contraction process). Consequently, the ordering obtained is normal.

□

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" since 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 σ 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, 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 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, $\sigma_1 = 2$ 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 $i < k \leq j < l$*
2. *color properties*
   (a) *for each vertex $\sigma_i, i \in \{1, \ldots, n-1\}$ 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 $\sigma_i, i \in \{2, \ldots n\}$ consider $A^+(\sigma_i) = \{u_1, \ldots, u_z\}$ as constructed above, then $\chi(\sigma_{i-1}, \sigma_i) = \chi(u_1)$ and $\forall k \in \{1, \ldots, z-1\}, \chi(u_k) \neq \chi(u_{k+1})$.*

Proof. **Necessary conditions.**

The fact that G contains a Hamiltonian circuit is proved in Lemma 3.1.

Condition 1. First, note that in any normal guillotine graph G no vertex i can be both the head of a backward arc and the tail of another backward arc. Indeed, a vertex i has to be contracted in a monochromatic circuit to terminate the contraction process, and in this circuit, either $|N^-(i)| < 2$ or $|N^+(i)| < 2$ according to Definition 3.3.

We know that G contains a unique Hamiltonian circuit μ_G, and that it admits a normal ordering σ (Proposition 3.4). Take four vertices i, j, k and l such that $\mu_G = (\sigma_1, \ldots, i, \ldots, j, \ldots, k, \ldots, l, \ldots, \sigma_n)$ and assume that there exist in G two backward arcs (k, i) and (l, j). We show that in this case G cannot be reduced to a single vertex by iterative contraction of monochromatic circuits fulfilling the conditions of Definition 3.3.

Note that because of the forward arcs between vertices in μ_G and because of backward arcs (k,i) and (l,j), the following property holds:

$$N^-(i) \geq 2 \text{ and } N^-(j) \geq 2 \text{ and } N^+(k) \geq 2 \text{ and } N^+(l) \geq 2. \tag{2}$$

We first show that any contraction of monochromatic circuits containing at most one vertex from among $\{i,j,k,l\}$ conserves (2). At each possible contraction step, if none of the four vertices above are contained in the circuit contracted, then (2) directly remains true. If only one vertex $x \in \{i,j,k,l\}$ is contained in the contracted circuit, consider without loss of generality that the residual vertex takes the same label x. Consequently, after the contraction of these circuits, we still have a Hamiltonian circuit $(\sigma'_1, \ldots, i, \ldots, j, \ldots, k, \ldots, l, \ldots, \sigma'_{n'})$ containing backward arcs (k,i) and (l,j). Therefore (2) remains true.

After all contractions involving either zero or one of the four considered vertices, a circuit containing more than 2 vertices in $\{i,j,k,l\}$ is contracted. Otherwise, the contraction process stops with more than one vertex. Let μ be such a circuit. According to Definition 3.3, the guillotine graph obtained after successive contractions should be normal and μ should contain exactly one item of out-degree 2 and one item of in-degree 2. Consequently, we cannot have both i and j in μ and we cannot have both k and l in μ. Thus, if μ contains more than 1 vertex in $\{i,j,k,l\}$, it can only be j and k. Since $N^-(j) \geq 2$ and $N^+(k) \geq 2$, the only possibility for a circuit involving both items is a circuit with an arc (k,j), otherwise there would be a third vertex x with either $N^-(x) \geq 2$ or $N^+(x) \geq 2$, which again is not allowed in a normal graph. The vertex y resulting from the contraction is such that $N^-(y) \geq 2$ and $N^+(y) \geq 2$, which means that y is both the head of a backward arc and the tail of another backward arc, which contradicts the fact that G is normal.

Conditions 2. Note that any contracted monochromatic circuit contains one or several forward arcs and at most one backward arc. Otherwise, the circuit contains a vertex which is both the tail of a backward arc and the head of another backward arc.

Condition 2(a). Consider a vertex σ_i with $A^-(\sigma_i) = \{u_1, \ldots, u_z\}$ as constructed above such that $|A^-(\sigma_i)| > 1$. Suppose that $\chi(\sigma_i, \sigma_{i+1}) \neq \chi(u_1)$. Then the first monochromatic circuit to be contracted containing σ_i cannot contain the arc $u_1 = (\sigma_j, \sigma_i)$, $j > i$ while it has to contain vertex σ_j and an arc $u_x = (\sigma_k, \sigma_i) \in \{u_2, \ldots, u_{z-1}\}$ with $x > j > i$. It means that $N(\sigma_i) >= 2$, $N(\sigma_j) >= 2$ and $N(\sigma_k) >= 2$, which contradicts the fact that G is normal. The same reasoning serves to deduce that the different contractions of monochromatic circuits containing vertex σ_i have to be performed with arcs in $\{u_1, \ldots, u_z\}$ in increasing order of u_i. Moreover, suppose there exists a k such that $\chi(u_k) = \chi(u_{k+1})$. Since the two arcs are contracted consecutively, this would mean that vertex σ_i must belong to two monochromatic circuits of same color, contradicting the fact that the graph is guillotine.

Condition 2(b). This condition can be demonstrated analogously to Condition 2(a), by considering a vertex σ_i with $A^+(\sigma_i) = \{u_1, \ldots, u_z\}$ as constructed above such that $|A^+(\sigma_i)| > 1$.

Sufficient. We now prove that the conditions of the theorem are sufficient for the graph to be a normal guillotine graph. We prove this result by recurrence on the size of the graph. If the graph is of size one, then the result is trivially true, because the graph is already contracted to one vertex.

Now suppose the result is true for any graph with k vertices or less, and consider a graph containing $k+1$ vertices such that the conditions of the theorem hold. If this graph is a monochromatic circuit, then the result is directly true, since it fulfills the conditions of Definition 3.3. If the graph is not a circuit, then there is a backward arc (σ_j, σ_i) connecting two vertices of the Hamiltonian circuit. Consider the circuit $\mu = (\sigma_i, \ldots, \sigma_j, \sigma_i)$. This circuit contains k or fewer vertices and, by hypothesis, it can be reduced to a single vertex by iterative contraction of monochromatic circuits fulfilling the conditions of Definition 3.3. This iterative process of contraction can be safely applied to G, since every arc outside μ connected to a vertex of μ is connected either to σ_i or to σ_j (otherwise there would be two arcs, thus contradicting condition 1.) Moreover, conditions 2(a) and 2(b) ensure that the color of the last circuit contracted in μ is different from the color of the circuit containing σ_i when the contraction has been performed. Once the iterative contraction

described above is performed, there is still a unique Hamiltonian circuit (μ is replaced by σ_i), no backward arcs have been added, and so there cannot be two backward arcs, which contradicts condition 2(b). Finally, the colors of the backward arcs remain alternated, since only their extremities have changed. Consequently, a graph containing fewer than k vertices is obtained, for which the conditions of the theorem hold. By construction, this new graph is a normal guillotine graph. Therefore, the initial graph can be reduced to a single vertex by iterative contraction of circuits, thus fulfilling the conditions of Definition 3.3.

□

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 due to the fact that 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 yet 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 above. 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* well-sorted normal guillotine graph (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 $\sigma_1 = 1$.*

Proof. The only well-sorted ordering is computed using the same algorithm as that used in Proposition 3.4. By definition 3.6, if vertex 1 belongs to a contracted circuit with vertices of degree greater than two, then it is the head of the backward arc. Therefore, there is never a vertex v such that $(v, 1)$ is added to the current path.

When a circuit with vertices of degree 2 only is reached, vertex 1 belongs to the set of vertices that can be chosen for σ_1. For this last circuit to be well-sorted, we must have $\sigma_1 = 1$. □

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 σ 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 $\eta(x) = \eta(\sigma_i)$.*

Proof. Suppose that during the contraction process there is a vertex resulting from the contraction of a monochromatic circuit $\mu = (x_1, \ldots, x_k, x_1)$, such that $\eta(x) \neq \eta(x_1)$. Since by definition we have $\forall i \in \{1, \ldots, k\}, \eta(x) \leq \eta(x_i)$, μ contains at least one forward arc (x_i, x_{i+1}) such that $\eta(x_i) > \eta(x_{i+1})$, contradicting the fact that G is well sorted. □

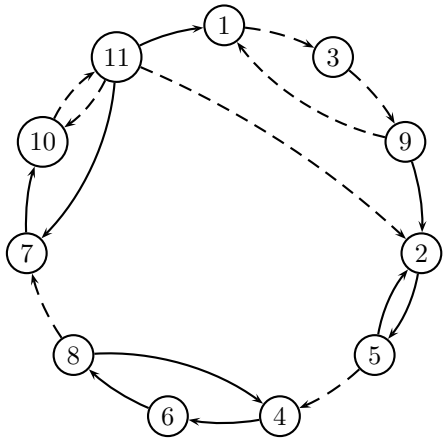


Figure 7: Modeling the guillotine-cutting class associated with the pattern of Figure 1 with a well-sorted normal guillotine graph.

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 $l < j$.*

Proof. Let G be a normal guillotine graph. Suppose there exists a forward arc (i, j) such that $i > j$. Suppose also that there are no backward arcs outgoing from vertex i. Then, at a given step of the contraction process, there is a monochromatic circuit $(x_1, \ldots, i, j', \ldots, x_k, x_1)$ with a forward arc (i, j') and $i > \eta(j') = j$ and G is not well-sorted. Now suppose there exists a backward arc (i, l), such that $l > j$. During the contraction process i will be contracted in a monochromatic circuit $\mu = (r, \ldots, i, r)$ where r is an RMB containing l (possibly $r = l$). If $\eta(r) \geq l$ then μ contains at least one arc (u, v) with $\eta(u) > \eta(v)$ and G is not well-sorted. If $\eta(r) < l$ then l has been contracted in a monochromatic circuit in which there was an arc (l, v) such that $l > \eta(v)$ and G is not well-sorted. Therefore the condition is necessary for a normal guillotine graph G to be well-sorted.

Now consider a normal guillotine graph G fulfilling the condition. Suppose that G is not well-sorted. This means that at some iteration of the contraction process there must exist a monochromatic circuit $\mu = (x_1, \ldots, i, j, x_k, x_1)$ with a forward arc (i, j) such that $\eta(i) > \eta(j)$. Note that this monochromatic circuit can be contracted only if there is no backward arc outgoing from i. Both i or j can be labeled either with a single item ($i = \eta(i)$ or $j = \eta(j)$), or with the result of a contraction. In all cases the arc (i, j) corresponds to an arc (u, v) in G such that $u \geq \eta(i)$ and $v = \eta(j)$ (see Proposition 3.2). Moreover, in G, either there is a backward arc from u to $\eta(i)$, or $u = \eta(i)$ and there is no backward arc outgoing from u. In all cases we have $u \geq \eta(i) > \eta(j) \geq v$ and no backward arc (u, l) with $l < v$, which is contradictory with G fulfilling the condition of the theorem. Therefore the condition is sufficient for a normal guillotine graph G to be well-sorted. □

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.*

Proof. Suppose that there exist two well-sorted guillotine graphs $G_1, G_2 \in \mathcal{G}_I$ such that $G_1 \neq G_2$, and such that the application of Algorithm 1 gives the same RMB, *i.e.* $\kappa'(G_1) = \kappa'(G_2)$. Now suppose that we apply the algorithm in parallel on G_1 and G_2. At some iteration a monochromatic

circuit μ_1 to be contracted in G_1 will be different from a monochromatic circuit μ_2 to be contracted in G_2. Either μ_1 and μ_2 contain the same vertices but in a different order, implying that at least one of the circuits is not well-sorted, and thus contradicting the assumption that both G_1 and G_2 are well-sorted, or there is at least one vertex which appears in only one of the circuits. In this case, the contraction of μ_1 and μ_2 cannot yield the same RMB, contradicting the assumption that $\kappa'(G_1) = \kappa'(G_2)$. Therefore κ' is injective.

The proof of the surjectivity of κ' is similar to the proof of the surjectivity of κ' in Proposition 3.2. The vertices $x_1, x_2, \ldots, x_z$ are sorted in increasing order of $\eta(x_1), \ldots, \eta(x_2)$ each time a vertex of label $\{x_1, x_2, \ldots, x_z\}$ is replaced by its associated monochromatic circuit. □

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 Section 5), since 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, since 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 WSNG in linear time. Algorithm 3 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.

Proposition 4.1. *Algorithm 3 returns* ***true*** *if the input graph is a well-sorted normal guillotine graph,* ***false*** *otherwise.*

Proof. Theorem 3.1 states that four conditions are necessary and sufficient for a graph to be a normal guillotine graph. Phase 1 of Algorithm 3 is related to condition 1(a), Phase 2 is related to condition 1(b) and Phase 3 is related to conditions 2(a) and 2(b). We show that the algorithm returns true if and only if the four conditions are fulfilled. The condition of well-sortedness can be verified in a straightforward manner, with reference to Theorem 3.2.

- First we show that Phase 1 finds the unique Hamiltonian circuit of the graph if it exists, or returns **false** otherwise. Since $\sigma_1 = 1$, the well-sorted normal ordering directly follows.
  - A Hamiltonian circuit is found if the graph is a well-sorted normal guillotine graph. Initially, only vertex 1 is attained ($\sigma_1^{-1} = 1$ and $\forall i \neq 1, \sigma_i^{-1} = -1$). We know that $\sigma_1 = 1$ by Proposition 3.5. At each iteration, the vertex added to the circuit is the vertex w that can be reached by an arc from the last attained vertex v (initially 1), and which has not yet been seen ($\sigma_w = -1$), which means that subtours are avoided. By construction, after n iterations, if the graph can be a WSNG, a Hamiltonian circuit is constructed.
  - Uniqueness of the Hamiltonian circuit. Suppose there are two Hamiltonian circuits μ and μ' in the graph : $\mu = (w_1 = 1, w_2, \ldots, w_n, w_1)$ and $\mu' = (w'_1 = 1, w'_2, \ldots, w'_n, w'_1)$.

Algorithm 3: Recognizing a well-sorted normal guillotine graph

```
   Data: G: a graph;
 1 Let σ and σ^{-1} be 2 arrays of n integers whose cells are initialized with −1
   /*Phase 1: find the unique Hamiltonian path and the unique well-sorted normal ordering */
 2 v ← 1, σ_1 ← 1, σ_1^{-1} ← 1;
 3 next ← 1;
 4 for i : 2 → n do
       /*determine the only forward arc outgoing from v                          */
 5     next ← −1;
 6     for w ∈ N^+(v) do
 7         if σ_w^{-1} = −1 or (i = n and w = 1) then
 8             if next ≠ −1 then return false;
 9             next ← w;
10     if next = −1 then return false;
       /*return false if the graph is not well-sorted                            */
11     if next < v then
12         well−sorted ← false;
13         for w ∈ N^+(v) \ {next} do
14             if w < next then
15                 well−sorted ← true
16         if well−sorted = false then return false;
17     v ← next ;
18     σ_i ← v;
19     σ_v^{-1} ← i;
20 if v ≠ 1 then return false
   /*Phase 2: check the structure of the graph                                   */
21 Let S be a stack of vertices initially empty;
22 for i : 1 → n do
23     foreach backward arc (σ_i, w) by decreasing value of σ_w^{-1} do
24         while S is not empty and the top element of S does not equal w do
25             remove the top element of S ;
26         if S is empty then return false
27     push σ_i on S;
   /*Phase 3: check the colors of the arcs                                       */
28 for i : 1 → n do
29     if i < n then
30         d ← χ(σ_i, σ_{i+1});
31         foreach arc u_j ∈ A^−(σ_i) do
32             if χ(u_j) ≠ d then return false;
33             d ← d̄ ;
34     if i > 1 then
35         d ← χ(σ_{i−1}, σ_i);
36         foreach arc u_j ∈ A^+(σ_i) do
37             if χ(u_j) ≠ d then return false;
38             d ← d̄ ;
39 return true;
```

Let k be the index such that $w_i = w'_i$, $\forall i \leq k$ and $w_{k+1} \neq w'_{k+1}$. This index exists because $w_1 = w'_1$. When vertex $w_k = w'_k$ is reached by the algorithm, there are two arcs $(w_k = w'_k, w_{k+1})$ and $(w_k = w'_k, w'_{k+1})$ leading to vertices that have not been seen before. In this case Phase 1 returns **false** (see line 8). Consequently, the algorithm returns **false** if there is more than one Hamiltonian circuit in the graph.

- We now show that Phase 2 is able to verify the second structural property, once a unique Hamiltonian circuit has been found in Phase 1.

  The algorithm returns false if there are crossing arcs. The second part of the proof states that step 2 of the algorithm checks that there are no "crossing arcs" in the graph. Suppose there are two crossing arcs (σ_j, σ_i) and (σ_l, σ_k) with $i < k \leq j < l$. In this case, σ_i is first pushed onto the stack, then σ_k is pushed onto the stack above (not necessarily immediately after) σ_i. When σ_j is reached, vertices are removed from the stack until σ_i is reached (this means that σ_k will be removed. When σ_l is reached, the stack will have been emptied, since σ_k will never be found. Consequently the algorithm will return **false**.

- Finally, we show that Phase 3 verifies the color property. The algorithm checks step by step that conditions 2(a) and 2(b) of Theorem 3.1 are fulfilled, so the algorithm will return **true** if and only if the colors are consistent with the graph.

□

Now that we have an algorithm for recognizing WSNG, the size of the associated guillotine class remains to be computed. Algorithm 4 computes the width and the height of the guillotine-cutting class associated with the input WSNG G. First the ordering σ is computed using Algorithm 3. 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 out-going 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 which have to be done in decreasing order of σ_j. They are done by merging the current RMB with some RMBs which 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 $w(t) \times h(t)$.

Proposition 4.2. *Algorithm 4 computes the size of the pattern associated with the normal guillotine graph G.*

Proof. Let $G^k = G[\{\sigma_1, \ldots, \sigma_k\}]$ be the subgraph induced by the set $\{\sigma_1, \ldots, \sigma_k\}$, and H^k the graph obtained by G^k by recursively circuit-contracting all possible monochromatic circuits (note that H^k is a path, and H^n has one vertex).

Loop invariant: *"at the end of each step i of the outer loop, the stack S contains one triplet $t = (label(t), w(t), h(t)$ per vertex of H^i, sorted from the bottom to the top by increasing value of $\sigma^{-1}(\eta(label(t)))$ and for which $label(t)$ is an RMB and $w(t) \times h(t)$ its associated size".*

Before the beginning of step 2 of the outer loop, H^1 is composed of one vertex 1, and the stack contains this vertex. The size associated with 1 is (w_1, h_1). Consequently, the loop invariant is initially verified.

Suppose the loop invariant is verified after step $i-1$. Now consider step i. If there is no backward arc (σ_i, j), then after step i, stack S contains the elements that were present at step $i-1$ plus one element $(\sigma_i, w_{\sigma_i}, h_{\sigma_i})$ on its top, which conserves the loop invariant.

If there are backward arcs, it is sufficient to prove the following result: *"If the stack S initially verifies the loop invariant, then the "foreach" loop contracts recursively all monochromatic circuits $(v, \ldots, \sigma_i, v)$ of G s.t. (σ_i, v) is a backward arc, removes the corresponding vertices from S and replaces them by the new vertex obtained".*

Let (σ_i, v) be the current backward arc with the largest value of $\sigma^{-1}(v)$. This circuit must be monochromatic, otherwise it would not be possible to contract any circuit containing these vertices with the properties of normal guillotine graphs which contradicts the fact that the graph

Algorithm 4: 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 σ its corresponding ordering on the vertices;
Let S be an initially empty stack of 3-tuples $t = (label(t), w(t), h(t))$;
$d \leftarrow \chi(\sigma_1, \sigma_2)$;
/*σ_1 is an RMB of size $w_{\sigma_1} \times h_{\sigma_1}$ */
push $(\sigma_1, w_{\sigma_1}, h_{\sigma_1})$ on S;
for $i : 2 \rightarrow n$ **do**
  /*σ_i is an RMB of size $w_{\sigma_i} \times h_{\sigma_i}$ */
  $rmb \leftarrow \sigma_i$;
  $w \leftarrow w_{\sigma_i}$;
  $h \leftarrow h_{\sigma_i}$;
  foreach *backward arc* $(\sigma_i, v) \in A^+(\sigma_i)$ *by decreasing value of* σ_v^{-1} **do**
    /*compute the RMB associated with the monochromatic circuit to which arc (σ_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) */
    $X \leftarrow \{rmb\}$;
    while $\eta(rmb) \neq v$ **do**
      $(rmb_t, w_t, h_t) \leftarrow$ top of S;
      remove top of S;
      $X \leftarrow X + rmb_t$;
      /*rmb_t is an RMB to merge with the other RMBs in set X */
      if $d = h$ **then** $w \leftarrow w + w_t$ and $h \leftarrow \max(h, h_t)$;
      else $w \leftarrow \max(w, w_t)$ and $h \leftarrow h + h_t$;
    $rmb \leftarrow X^d$;
    $d \leftarrow \bar{d}$;
  push (rmb, w, h) on S;
$(rmb_t, w_t, h_t) \leftarrow$ top of S;
return (w_t, h_t);

is a well-sorted normal guillotine graph. Therefore, it can be contracted. Let $\mu = (v_1 = v, v_2, \ldots, v_z = \sigma_i, v)$ be this circuit. By assumption, the stack contains (from the top to the bottom): $v_{z-1}, \ldots, v_2, v_1, \ldots$. Consequently the "while" loop terminates and computes a valid RMB containing $v, v_2, \ldots, v_i$. By construction, the size associated with this RMB is valid. By definition of normal guillotine graphs, the graph obtained after contraction of μ is still a normal guillotine graph. Consequently, the same argument holds iteratively for all circuits whose last vertex is σ_i. Once all backward arcs (σ_i, v) have been considered, the stack S contains the triplets corresponding with vertices of H^{i-1} that were not built with σ_i (in the same order as before), and the new triplet t obtained from the contractions including σ_i. Since the graph is a well-sorted normal guillotine graph, $\sigma^{-1}(\eta(label(t))) > \sigma^{-1}(\eta(label(u)))$ for any element u of the stack. Therefore, the loop invariant is satisfied.

After step n, the graph H^n obtained is reduced to a single vertex (because the graph is guillotine). According to the loop invariant, the size of the RMB associated is computed by the algorithm. □

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 3), 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.*

Proof. In Algorithm 3, there are three phases, all of which can be run in linear time. In phases one and two, each vertex and each arc is visited exactly once, and twice in phase three. This is true if one is able to construct the adjacency lists where the arcs (u, v) are sorted by decreasing values of σ_v^{-1}. This can be realized by reconstructing the adjacency list as follows. Consider an initial empty set of successor lists. Using the Hamiltonian circuit, add the forward arcs in the successor lists ($O(n)$). Then, using the Hamiltonian circuit again, for each item v, compute its set of predecessors u, and insert v at the first position in the successor list of any predecessor u ($O(m)$). A copy of the initial graph is obtained, where the successor lists of arcs (u, v) are sorted by decreasing value of σ_v^{-1}, and the only forward arc in the list is the last element.

This implies a complexity of $O(n + m)$. Since $m \in O(n)$, the complexity is $O(n)$.

In the main phase of Algorithm 4, each vertex is considered once, and then each arc is considered twice: when the corresponding build is added to the stack, and when it is removed from the stack. Consequently, computing the guillotine pattern from each guillotine graph is in $O(n + m)$.

Proposition 3.3 indicates that $m \in O(n)$. Consequently, the overall complexity is $O(n)$. □

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 since it has been shown to be efficient for the non-guillotine rectangle packing problem (see [4, 10] 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 $\{v_1, v_2, \ldots, v_k\}$, 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, while the second 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) $(i, j \in \{1, \ldots, n\})$ is therefore governed by the following variables:

- $existence(i, j)$; its domain is initially $\{true, false\}$;
- $orientation(i, j)$; its domain is initially $\{forward, backward\}$;
- $direction(i, j)$; 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. In order 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 multi-build 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 $[1..i]$ (since 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 $[w_i..W]$, since it contains at least item i
- $height(i)$: 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 (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 following variable is associated:

- $before(i, j)$: a boolean that indicates if $\sigma_i^{-1} < \sigma_j^{-1}$.

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 which 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: bottommost and side by side, in increasing order of labels, from left to right;
- in a vertical contraction: leftmost and one above the other, in 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 non-overlap constraint (see for example [3, 10]) 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 $[u_0, u_1, u_2, \ldots, u_k]$ 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. Since 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 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 behavior of the approach with filtering algorithms

During the search, constraint-propagation techniques are used to reduce the search space by eliminating non-relevant 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; 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

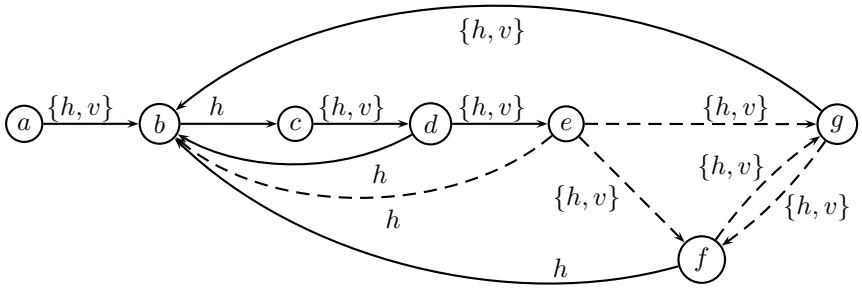
Each time a circuit $[u_0, u_1, u_2, \ldots, u_k]$ is contracted the quantity of lost space is updated. The variable *lost* is updated according to the size of aggregates. In particular, if a *horizontal* contraction is performed between two aggregates i and j, then the created gap is of dimensions $width(j)^- \times (height(i)^- - height(j)^-)$ if $(height(i)^- > height(j)^-)$, $width(i)^- \times (height(j)^- - height(i)^-)$ 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 $(width(i)^- > width(j)^-)$,

$height(i)^- \times (width(j)^- - width(i)^-)$ if $(width(i)^- < width(j)^-)$. The gap is used to adjust the lower bound of $domain(lost)$ with $lost^- = lost^- + gap$.

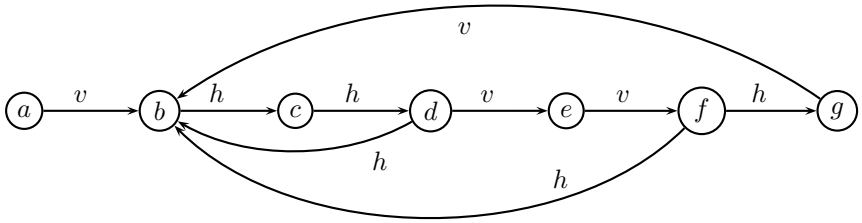
Each time a contraction is performed, an aggregate replaces a set of items and a certain amount of lost space. The lower bounds of [7, 9] are used in a feasibility test to ensure that the new set of aggregates will fit into the bin. These bounds are designed for the non-guillotine case, and thus do not take into account the guillotine constraint. However they are useful for pruning a large number of non-useful 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.



(a) initial graph



(b) after propagation

Figure 8: Propagation from arcs to arcs: a chain propagation removes a whole set of potential arcs.

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 mono-directional contractions.

Figure 8 provides an example of these kinds of deductions. The mandatory arcs are shown as continuous lines, while the possible arcs are dotted lines. If the direction of an arc is known, it is marked h or v respectively for *horizontal* and *vertical*. 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 *sigma*.

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 non-contracted 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.

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 Section 5.1). Thus, some constraints on the coordinates are deduced from the existence of forward arcs in the graph.

The labels can also be used to filter the possible values for the coordinates. Let j be an aggregate which is too tall to be packed above aggregate 0. j must therefore 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 mono-directional (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 [18], 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 unfeasible 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.* [9], which were initially generated for the non-guillotine $2OPP$ problem. An instance "M, X" contains M items. These instances are available on the web: *http://www.lifl.fr/~clautiau/*. This test bed was chosen because it contains both easy instances 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)"). Symbol "_" means that an optimal solution was not found within 3600 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 $H_{min} = \frac{1}{W} \sum_{i \in I} h_i w_i$ 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 non-guillotine version of the problem is feasible.

The first graph-based method, IGG_{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 to find 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 unfeasible problem. The optimal height H^* is the height of the last search problem for which a solution exists. For this method only one unfeasible problem is considered (the problem in which we search for a solution with a height strictly lower than H^*).

The second method, IGG_{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 be solved is to find a solution with a height lower than or equal to $H_{min}+1$. While 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^*) while all unfeasible 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 non-guillotine case is first checked for each decision problem to solve (using [10]). 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 [10] 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 non-guillotine version or the guillotine version of the problem, apart from the last one, which is feasible at least for the non-guillotine version), while 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 [10], originally designed for the non-guillotine case. This method enumerates solutions for the non-guillotine case until the pattern found is guillotine. We use the algorithm of [6] to prune non-guillotine 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 [10] only considers the X-coordinates in a first phase (and thus the guillotine test is only useful in the deep nodes of the search).

A first remark is that the computing time required by the algorithms is large compared to the time required by [10] to solve the non-guillotine version. This fact justifies the third method, which first checks the feasibility of the non-guillotine 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 brackets, so the reader can verify that the time needed to solve the non-guillotine 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 $2SP + IGG_{lb}$, it will be remarked that when a search problem is not feasible for either the non-guillotine 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 $0.11s$, while IGG takes more than $31s$. For this instance, the last search problem for a guillotine solution with a height set to 21 takes only $0.08s$ 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 unfeasible. However, it should be noted that this is not usually the case.

By comparing the results obtained by the three guillotine-based methods with $2SP + 2SCP$, we see that our model really does provide useful information. Method $2SP + 2SCP$ is seldom able to improve on the methods that use the guillotine model, and is unable to solve most of the instances, although the method is efficient for these instances in the non-guillotine case. The ineffectiveness of the method comes the fact than it was originally designed for the non-guillotine problem. In particular, as explained before, the method of [10] 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 non-guillotine problem are considered (since they are not filtered by the method), while 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 compu-

tation 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 [17] 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 [20] algorithm (IMVB and BMVB) proposed by Hifi [17] 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. Then the new item obtained is added to the list and the algorithm is run recursively. The branching scheme consists in finding which pair of current items is built at each step. We also consider the method of Bekrar *et al.* [2], which is an adaption of Martello *et al.* [19].

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 [2], we provide the number of nodes ("nodes") and the computing time in seconds ("cpu(s)"). Unfortunately, only the computational times are provided in [2]. Note that there is a time limit of 1000 seconds for the method of [2]. The instances which are not solved within this time limit are marked with a * in the table. The value $1000s$ is then taken for these instances to determine the average computation time.

Comparing the computing times of the different methods is tricky, since the results of the IMVB and BMVB methods are those provided by [17], obtained in 1998 using a Sparc-Server 20/712. In order 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, 7174 MIPS for the machine used by [2] and 13207 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 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 IMVB method of [17] needs 40909 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 [17]. These results show that the additional information added 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 [2].

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 methods needs a few more nodes than the IMVB method to find a feasible solution. This suggests that our branching heuristic was not suited to 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 in order to handle them. Computational experiments show that our model provides useful information that can be used in an exact algorithm, and leads to results that outperforms those of the literature.

instance	H_{min}	H_{max}	H^+	H^*	IGG_{ub} nodes	IGG_{ub} $cpu(s)$	IGG_{lb} nodes	IGG_{lb} $cpu(s)$	$2SP + IGG_{lb}$ nodes	$2SP + IGG_{lb}$ $cpu(s)$	$2SP + 2SCP$ nodes	$2SP + 2SCP$ $cpu(s)$
23, A	20	106	21	21	348457	449.25	113090	202.234	32982 (11945)	30.52 (2.43)	2246142	206.906
23, B	20	108	21	21	18347	35.1719	13877	31.4844	196 (136)	0.19 (0.11)	-	-
22, A	20	122	20	20	1252332	1633.92	364558	539.031	364590 (32)	471.79 (0.04)	223	0.015625
20, A	20	126	20	21	-	-	898690	1065.42	909236 (10546)	967.24 (0.77)	-	-
20, B	20	134	22	22	-	-	344622	437.688	703 (46)	0.82 (0.05)	-	-
20, C	20	142	20	20	358166	333.625	1376	2.07813	1421 (45)	2.16 (0.04)	13381014	1578.09
20, D	19	115	20	20	285334	342.688	178617	231.516	178643 (27)	209.25 (0.04)	-	-
19, A	20	117	20	20	121200	125.484	95532	106.453	95561 (29)	93 (0.1)	-	-
18, A	20	114	20	22	1845495	1636.48	1844271	1796.83	1844294 (23)	1570.47 (0.05)	-	-
18, B	20	124	21	?	-	-	-	-	-	-	-	-
18, C	19	119	20	21	211421	183.047	103725	108.422	103747 (23)	95.83 (0.04)	-	-
17, A	20	112	20	21	142921	138.375	96020	104.984	96042 (22)	94.65 (0.02)	-	-
17, B	20	105	21	22	10463	10.9531	9557	10.0469	9089 (27)	8.36 (0.05)	-	-
17, C	20	97	20	21	48027	44.9219	43383	42.4375	43406 (23)	37.63 (0.02)	-	-
17, D	20	141	21	?	-	-	-	-	-	-	-	-
17, E	20	115	21	22	404003	317.813	307684	273.438	306724 (92)	247.47 (0.05)	-	-
16, A	20	118	21	22	2093	2.0625	893	1,00	10987 (10230)	2.63 (1.93)	-	-
15, A	20	87	21	22	32784	22.3125	3357	3.09375	3631 (275)	2.71 (0.13)	8094280	840.813
15, B	20	116	21	21	16877	15.5781	5016	4.76563	4987 (90)	4.24 (0.05)	9203561	1150.14
15, C	20	127	22	24	14005	10.4531	10085	8.67188	12720 (2803)	8.25 (0.68)	-	-
15, D	20	82	20	22	86903	64.7031	87798	68.2031	87984 (186)	60.75 (0.07)	-	-
15, E	19	76	20	22	24260	17.875	27572	21.1875	28166 (737)	18.91 (0.22)	-	-
15, F	19	142	21	23	705324	400.406	821428	506.516	824000 (2578)	453.55 (1.11)	-	-
15, G	19	97	21	21	67581	46.4531	45834	37.5469	21766 (1332)	15.19 (0.36)	-	-
15, H	19	91	21	22	404173	222.953	201898	153.047	149571 (404)	101.93 (0.16)	-	-
15, I	19	118	20	21	9069	6.23438	5116	4.15625	5144 (28)	3.79 (0.05)	-	-
15, J	19	147	21	21	5025	3.03125	196	0.125	218 (23)	0.29 (0.04)	80	0.0625
15, K	19	113	21	21	752717	384.781	135752	82.4219	135773 (22)	76.54 (0.01)	140	0.078125
15, L	19	115	20	21	3356	2.375	1148	1.01563	1276 (131)	0.82 (0.11)	-	-
15, M	18	165	22	22	2627	1.92188	323	0.234375	1541 (1220)	0.68 (0.54)	-	-
15, N	18	112	21	22	4744	3.34375	393	0.40625	902 (653)	0.38 (0.19)	45070463	3036.72
15, O	18	106	22	22	10065	7.4375	9463	7.14063	233 (158)	0.13 (0.11)	-	-
15, P	18	85	22	22	2605	2.53125	1333	1.03125	1357 (26)	1.08 (0.04)	142	0.109375
15, Q	18	143	22	22	40174	19.4844	9638	4.3125	10021 (385)	4.66 (0.10)	441	0.15625
15, R	16	77	17	21	1219272	749.563	1220724	767.938	1220750 (27)	713.36 (0.02)	-	-
15, S	17	93	20	21	1469	1.84375	90	0.109375	114 (24)	0.07 (0.04)	-	-
10, A	20	88	22	23	223	0.34375	139	0.109375	599 (496)	0.19 (0.15)	705993	42.4531
10, B	20	62	21	24	360	0.390625	183	0.0625	196 (14)	0.15 (0.02)	33877520	2251.47
10, C	19	88	22	23	539	0.40625	142	0.09375	313 (176)	0.11 (0.04)	-	-
10, D	18	86	21	25	431	0.390625	331	0.15625	346 (16)	0.19 (0.05)	-	-
10, E	18	69	21	22	266	0.453125	292	0.125	262 (39)	0.24 (0.08)	-	-

Table 1: Experimental results on [9] instances

		$IMVB$ [17]			$BMVB$ [17]			[2]		IGG_{lb}		
	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	0.016	206
SCP2	11	1797	3.4	857	409	1.2	302	0.01	72	63	0.031	413
SCP3	15	3427	6.8	1714	1650	19.4	4889	7.1	50935	56	0.047	619
SCP4	11	6356	78.6	19807	1125	4.1	1033	2.45	17576	494	0.203	2683
SCP5	8	5	0.1	25	141	0.1	25	0.01	72	13	0.000	0
SCP6	7	8012	54.6	13759	536	3.4	857	0.01	72	19	0.000	0
SCP7	8	1195	1.8	454	458	1.4	353	0.01	72	30	0.016	206
SCP8	12	484	0.4	101	721	2.3	580	1.57	11263	20	0.000	0
SCP9	12	60	0.7	176	72	0.1	25	0.14	1004	32	0.016	206
SCP10	8	4	0.1	25	9	0.5	126	0.18	1291	13	0.000	0
SCP11	10	11036	221.5	55818	530	0.7	176	4.89	35081	21	0.016	206
SCP12	18	908	1.3	328	2029	2.7	680	1000*	7174000	58	0.078	1032
SCP13	7	4359	39.5	9954	1278	16.8	4234	0.38	2726	213	0.031	413
SCP14	10	4782	41.7	10508	2475	48.7	12272	15.57	111699	600	0.250	3302
SCP15	14	673	0.7	176	26728	165.7	41756	1000*	7174000	88	0.063	825
SCP16	14	85627	654.8	165010	4409	181.4	45713	606.12	4348305	809	0.469	6191
SCP17	9	13668	227.3	57280	4751	323.6	81547	2.07	14850	815	0.266	3508
SCP18	10	22087	321.5	81018	7113	327.8	82606	9.43	67651	1781	0.750	9905
SCP19	12	39550	1794.3	452164	10441	473.0	119196	0.01	72	1040	0.484	6397
SCP20	10	36577	874.3	220324	12326	673.9	169823	1.73	12411	1127	0.406	5365
SCP21	11	26748	1757.6	442915	17552	2001.8	504454	24.31	174400	2440	1.031	13620
SCP22	22	40909	606.0	152712	104013	757.8	190966	1000*	7174000	293	0.328	4334
SCP23	12	29512	691.9	174359	48612	1031.9	260039	37.97	272397	1545	0.672	8873
SCP24	10	117013	6265.0	1578780	45143	5585.7	1407596	19.34	138745	6458	2.453	32398
SCP25	15	69434	3735.8	941422	27344	2662.9	671051	1000*	7174000	3105	2.156	28478
Average		20972	695	175188	12796	571	144013	189	1358271	846	0.391	5167

Table 2: Comparison with methods in the literature

We wish to emphasize that this model may also be used for the three-dimensional case, using three different colors.

Acknowledgments

The authors would like to thank Abdelghani Bekrar for sending us detailed information about his computational results.

References

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

[2] A. Bekrar, I. Kacem, C. Chu, and C. Sadfi, *An improved heuristic and an exact algorithm for the 2d strip and bin packing problem*, International Journal of Product Development **10** (2010) no. 1-3, 217–240.

[3] N. Beldiceanu and M. Carlsson, *Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints*, Principles and Practice of Constraint Programming (CP'2001), Lecture Notes in Computer Science **2239** (2001), 377–391.

[4] N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet, *A generic geometrical constraint kernel in space and time for handling polymorphic k -dimensional objects*, Principles and Practice of Constraint Programming - CP 2007, LNCS 4741, 2007, pp. 180–194.

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

[6] S. Ben Messaoud, C. Chu, and M.-L. Espinouse, *Characterization and modelling of guillotine constraints*, European Journal of Operational Research **191** (2008), 112–126.

[7] J. Carlier, F. Clautiaux, and A. Moukrim, *New reduction procedures and lower bounds for the two-dimensional bin-packing problem with fixed orientation*, Computers and Operations Research **34** (2007), no. 8, 2223–2250.

[8] N. Christofides and E. Hadjiconstantinou, *An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts*, European Journal of Operational Research **83** (1995), 21–38.

[9] F. Clautiaux, J. Carlier, and A. Moukrim, *A new exact method for the two-dimensional orthogonal packing problem*, European Journal of Operational Research **183** (2007), no. 3, 1196–1211.

[10] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim, *A new constraint programming approach for the orthogonal packing problem*, Computers and Operations Research **35** (2008), no. 3, 944–959.

[11] Y. Cui, H. Dongli, and X Song, *Generating optimal two-section cutting patterns for rectangular blanks*, Computers and Operations Research **33** (2006), 1505,1520.

[12] S. Fekete and J. Schepers, *A combinatorial characterization of higher-dimensional orthogonal packing*, Mathematics of Operations Research **29** (2004), 353–368.

[13] S. Fekete, J. Schepers, and J. Van Der Ween, *An exact algorithm for higher-dimensional orthogonal packing.*, Operations Research **55** (2007), no. 3, 569–587.

[14] M. R. Garey and D. S. Johnson, *Computers and intractability, a guide to the theory of NP-completeness*, Freeman, New York, 1979.

[15] P. Gilmore and R. Gomory, *Multistage cutting stock problems of two and more dimensions*, Operations Research **13** (1965), 94–120.

[16] M. C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.

[17] M. Hifi, *Exact algorithms for the guillotine strip cutting/packing problem*, Computers and Operations Research **25** (1998), no. 11, 925–940.

[18] Ilog, *Ilog solver reference manual*, Gentilly, France, 2004.

[19] S. Martello, M. Monaci, and D. Vigo, *An exact approach to the strip-packing problem*, INFORMS Journal on Computing **15** (2003), 310–319.

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

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

[22] G. Wäscher, H. Haussner, and H. Schumann, *An improved typology of cutting and packing problems*, European Journal of Operational Research **183** (2007), 1109–1130.