JMMA manuscript No.
(will be inserted by the editor)

MPQ-trees for the orthogonal packing problem

Cédric Joncour, Arnaud Pêcher, Petru Valicov

July 26, 2011

Abstract Given a set of rectangular items of different sizes and a rectangular container, the aim of the bi-dimensional Orthogonal Packing Problem (OPP-2 for short) is to decide whether there exists a non-overlapping packing of the items in this container. The rotation of items is not allowed. In this paper we present a new exact algorithm for solving OPP-2, based upon the characterization of solutions using interval graphs proposed by Fekete and Schepers. The algorithm uses MPQ-trees, which were introduced by Korte and Möhring to recognize interval graphs.

Keywords Orthogonal Packing Problem · MPQ-trees

Let V be a set of D-dimensional rectangular shapes. For $d \in \{1,\ldots,D\}$ and every $v \in V$, let $w_d(v) \in \mathbb{Q}^+$ (resp. $w_d(C)$) be the length of v (resp. of the container C) with respect to the dimension d. For every subset of items $S \subseteq V$, let $w_d(S) = \sum_{v\in S} w_d(v)$. Let $W(v) = \prod_{d=1}^{D} w_d(v)$ and $W(C) = \prod_{d=1}^{D} w_d(C)$ be the volumes of the item v and of the container C respectively.

The *D-dimensional orthogonal packing problem* (OPP-D) is to decide if the set of items V fits into the container C without overlapping (if true, V is said to be *feasible*). Formally speaking, we have to find out whether $\forall d \in \{1,\ldots,D\}$ there exists a function $x_d : V \to \mathbb{Q}^+$, such that:

$$\forall v \in V, x_d(v) + w_d(v) \leq w_d(C)$$
$$\forall v_1, v_2 \in V, (v_1 \neq v_2), [x_d(v_1), x_d(v_1) + w_d(v_1)) \cap [x_d(v_2), x_d(v_2) + w_d(v_2)) = \emptyset$$

Let $p_v \in \mathbb{Q}^+$ be the value associated with an item $v \in V$. The *d-dimensional knapsack problem* (OKP-d) consists in computing a feasible set $V' \subseteq V$ such that $\sum_{v\in V'} p_v$ is maximal.

In this paper, we consider the bi-dimensional case, which has been the most studied so far [11, 9, 6, 12, 7, 3, 5, 8, 1]. This paper is organized as follows. In the first section we describe Fekete and Schepers' model and point out some issues which motivated our work. In the second section we define the model of MPQ-trees (introduced in [17]) and give an algorithm to check feasibility. In the third section, we present the computational results compared with other algorithms on standard benchmarks. The fourth section is a short conclusion.

This research was partially supported by the ANR Project GraTel ANR-09-blan-0373-01.

C. Joncour
IMB / INRIA Bordeaux - Sud-Ouest, University of Bordeaux, 351 Cours de la Libération, 33405 Talence Cedex, France
E-mail: cedric.joncour@math.u-bordeaux1.fr

A. Pêcher
IRIT, University of Toulouse, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France
E-mail: pecher@irit.fr

P. Valicov
LaBRI, 351 Cours de la Libération, 33405 Talence Cedex, France
E-mail: valicov@labri.fr

1 Fekete and Schepers' model

In this paper, all graphs are simple, undirected and finite. We denote by $N(u)$ the set of neighbours of the vertex u. C_n denotes a cycle on n vertices. A *2-chordless cycle* is a cycle $v_0,\ldots,v_{n-1},v_0$ such that there is no edge v_iv_j for $i,j \in \{0,\ldots,n-1\}$ and $|i-j| \mod n = 2$. A *stable* or *independent* set of a graph is a subset of pairwise non-adjacent vertices of this graph. A *clique* is a set of pairwise adjacent vertices. An *interval graph* is a graph $G=(V,E)$ such that there is an assignment of intervals I_v $(v \in V)$ of the real line to the vertices of G such that for every pair of vertices u and v, uv is an edge if and only if $I_u \cap I_v \neq \emptyset$.

Given a feasible packing of a set of items V, for every dimension d, let $G_d=(V,E_d)$ be the interval graph with vertex set V and edge set E_d, such that ij is an edge if and only if the projections of the packing of items $i \in V$ and $j \in V$ onto the dimension d intersect (see Fig. 1 for an illustration).

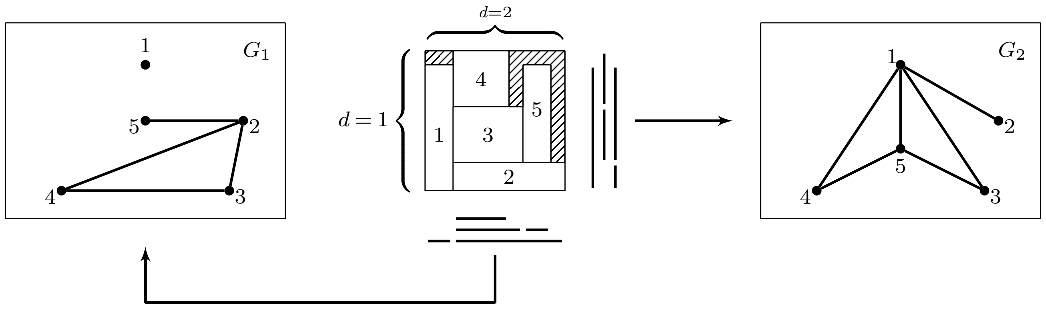


Fig. 1: Example of 2D packing and its associated interval graphs

Fekete and Schepers proved the following result, which is crucial for our algorithm:

Theorem 1 (Fekete and Schepers [10]) *Given a D-dimensional container C, a set of items V is feasible, if and only if there is a set of D graphs $G_d=(V,E_d)$, with $d \in \{1,\ldots,D\}$, such that:*

(P1) *Every graph G_d is an interval graph*
(P2) *For every stable set S of G_d, $w_d(S) \leq w_d(C)$*
(P3) $\bigcap_{d=1}^{D} E_d = \emptyset$

The tuple of the D graphs G_d is called a *packing class.*

Fekete, Schepers and van der Veen gave an efficient algorithm for solving OKP-D by solving its subproblem OPP-D [12]. Their algorithm is based upon the packing classes. To do so, they used the following characterization of interval graphs:

Theorem 2 (Ghouilà-Houri [15], Gilmore and Hoffman [16]) *A graph G is an interval graph if and only if it does not contain an induced C_4 and its complement does not contain a 2-chordless cycle of odd length.*

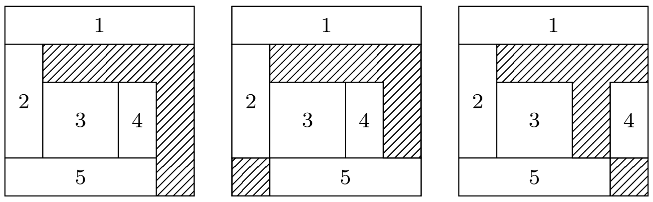


Fig. 2: Symmetrical solutions in Fekete and Schepers' model

Despite its efficiency, their algorithm may enumerate symmetrical solutions. An example is given in Figure 2 where "almost" similar packing configurations are modeled by different couples of interval graphs. Moreover, there are some degeneracy issues of Fekete and Schepers' algorithm pointed out in [13], implying the generation of some unnecessary couples of interval graphs.

2 Our approach

Our aim is to handle those symmetry issues more efficiently, by taking advantage of the nice algorithmic properties of the MPQ-tree data structure. Fekete and Schepers' model considers only the intersection of the projections of the items onto each dimension. In our approach we maintain some extra information that enables the early detection of some symmetrical solutions or some infeasible packing configurations. This was the motivation to use a different representation of interval graphs and we chose MPQ-trees for this purpose. In this section we recall the definition of an MPQ-tree by presenting the idea of the algorithm of recognition of interval graphs using this data structure introduced in [17] and give a new characterization of feasible items. We give an algorithm to solve OPP-2 using MPQ-trees and provide some optimizations by exploiting some properties of feasible sets of items.

2.1 MPQ-trees

To mention the relationship between MPQ-trees and interval graphs we need the following definition:

Definition 3 Let $G = (V, E)$ be a graph and $Q_1, \ldots, Q_m$ its maximal cliques. A consecutive arrangement of the maximal cliques of G is an order $\prec$ over the maximal cliques such that $Q_i \prec Q_j$ if:
$\forall v \in V$, if $v \in Q_i$ and $v \in Q_j$, then $v \in Q_k$ for all k s.t. $Q_i \prec Q_k \prec Q_j$.

Theorem 4 (Fulkerson and Gross [14]) *A graph G is an interval graph if and only if the maximal cliques of G can be linearly ordered to obtain a consecutive arrangement.*

Let M be a set of elements and $\mathcal{M}$ a collection of subsets of M. A PQ-tree is a data-structure representing all permutations of M that are consistent with constraints of consecutiveness given by $\mathcal{M}$ with the convention that the elements of each $M' \in \mathcal{M}$ must occur consecutively in a permutation. In the case of interval graphs, M is the set of maximal cliques, and $\mathcal{M}$ is the set of all $\mathscr{C}(v)$, $v \in V$, where $\mathscr{C}(v)$ denotes the set of all maximal cliques containing v.

A PQ-tree is a planar drawing of a rooted tree with two types of internal vertices: P and Q, represented by circles and rectangles respectively. The leaves of a PQ-tree are labelled 1-1 with the maximal cliques of an interval graph G. To avoid ambiguity, we will call *nodes*, the vertices of a PQ-tree.

The *frontier* $F(T)$ of a PQ-tree T, represents the permutation of the maximal cliques obtained by the ordering of the leaves of T from left to right. A PQ-tree T' is *equivalent* to T, if one can be obtained from the other by applying the following rules a finite number of times:

1. Arbitrarily permute the children of a P-node
2. Reverse the order of children of a Q-node

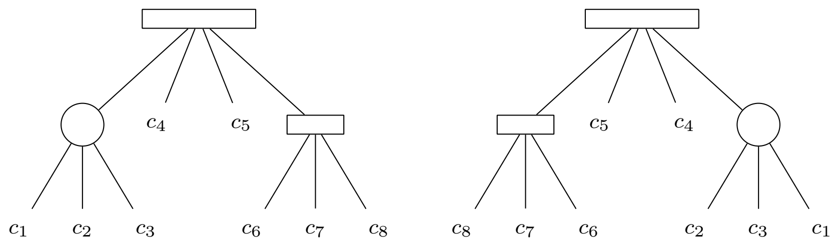


Fig. 3: Equivalent PQ-trees

Figure 3 shows an example of two equivalent PQ-trees.

A *proper* PQ-tree is one for which the P-nodes have at least two children and the Q-nodes have at least three children. From now on, by the term PQ-tree we will consider a proper PQ-tree.

Theorem 5 (Booth and Lueker [4]) *A graph G is an interval graph if and only if there exists a PQ-trees T such that $F(T)$ is a consecutive arrangement of the maximal cliques of G.*

A modified PQ-tree (MPQ-tree, introduced by Korte and Möhring in [17])) associated with a graph G is an extension of a PQ-tree where the nodes of the tree are labelled with some subsets of vertices of G, such that each branch of the MPQ-tree represents a maximal clique. A P-node is assigned only one set, while a Q-node is assigned a set for each of its children. Here are the rules of the labelling:

– A P-node is labelled by the set of vertices of G which are *only* contained in *all* cliques represented by the subtree of T rooted in this node.
– A leaf is labelled by the set of vertices of G contained *only* in the clique represented by this leaf.
– A Q-node, with m children $F_1, \ldots, F_m$, is labelled by a list of sets S_k, for $k \in \{1, \ldots, m\}$, each of them being called a *section* such that a section S_k corresponds to the child F_k in the left-to-right order. Each section S_k ($k \in \{1, \ldots, m\}$) contains the vertices of G contained in *all* cliques represented by the subtree rooted in F_k and also in *all* cliques represented by the subtree rooted in another child F_l ($l \in \{1, \ldots, m\}$ and $l \neq k$).

We will say that a node N of an MPQ-tree associated with a graph G *contains* a vertex v of G if $v \in V_N$, where V_N is the vertex set or section (in case of a Q-node) of the label of N. Notice that from the definition of an MPQ-tree, a vertex v of G is contained in exactly one MPQ-tree node. However, v can be contained in more than one section of a Q-node.

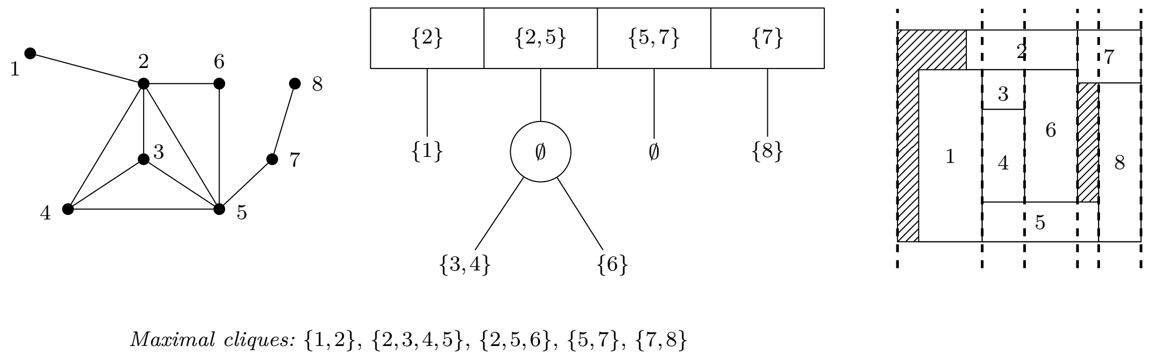


Fig. 4: An interval graph, an associated MPQ-tree and the packing configuration

Figure 4 shows an example of an MPQ-tree associated with an interval graph G, where both representations modelize the intersections in dimension 1 of a feasible packing in terms of maximal cliques (the intersections are given by the strips in the picture).

We have the following characterization of interval graphs, which is the basis of our algorithm:

Theorem 6 (Korte and Möhring [17]) *A graph G is an interval graph if and only if there exists an MPQ-tree associated with G.*

Hence we may consider MPQ-trees instead of interval graphs since Theorem 6 gives an equivalence between the two structures.

Lemma 7 (Korte and Möhring [17]) *Let G be an interval graph and T its associated MPQ-tree. Then $G+u$ (where u is a vertex added to G) is an interval graph if and only if the following holds:*

1. *All vertices adjacent to u are contained in a unique path of T.*
2. *For each Q-node N, labelled with sections $S_1, \ldots, S_m$, let $S = S_1 \cup \ldots \cup S_m$. Then $S \cap N(u) \in S_1$ or $S \cap N(u) \in S_m$*

In other words, Lemma 7 says that while building an MPQ-tree incrementally from an interval graph (i.e. adding vertices of the graph one by one) only one path of this tree must be updated and due to the properties of the reading orders of children of nodes of an MPQ-tree, this path can be chosen to be the leftmost one. This restricts considerably the number of cases to consider while updating the MPQ-tree. This result is crucial for our algorithm.

Using Theorem 6, Korte and Möhring gave an algorithm recognizing if a graph G is an interval graph, by constructing an associated MPQ-tree. It uses a LexBFS-ordering (introduced in [18]) $\lambda = [v_1, \dots, v_n]$ of the vertices of G to iteratively build the MPQ-tree.

Let $G = (V, E)$ be an interval graph with an associated MPQ-tree T_G and u the vertex to be inserted in T_G to obtain the MPQ-tree T_{G+u} associated with the graph $G + u$. Since vertices are inserted in LexBFS-order, $N(u)$ must induce a clique. Let N be a node of T_G. If N is of type P or a leaf, let V_N be its associated vertex set and if N is a Q-node let V_N be the vertex set corresponding to the first section. The principle of Korte and Möhring's algorithm is the following:

- Find the unique path P of the current MPQ-tree having a node N such that $V_N \cap N(u) \neq \emptyset$. This path can be either leftmost or rightmost.
- Find the first node N_* in the bottom-up traversal such that $\exists j \in V_{N_*}, j \in N(u)$.
- Select the corresponding pattern and apply the suitable replacement: let $V_{N_*} = A \cup B$ be the partition of V_{N_*} such that $A = V_{N_*} \cap N(u)$ and $B = V_{N_*} \setminus A$. Let N^* be the highest node in P such that $V_{N^*} \setminus N(u) \neq \emptyset$ if it exists and let $N^* = N_*$ otherwise. Due to Lemma 7, the patterns which can be applied are described in Figures 5, 6 and 7. In the case when $N_* \neq N^*$ the patterns are applied recursively by rewriting the current tree starting from N_* up to N^*, to obtain a valid MPQ-tree. We omit details which are rather technical. For an elaborate explanation, see the original paper [17]. Notice that the process is deterministic e.g. at each step only one pattern can apply. If no pattern can be applied to the current configuration, then $G + u$ is not an interval graph.

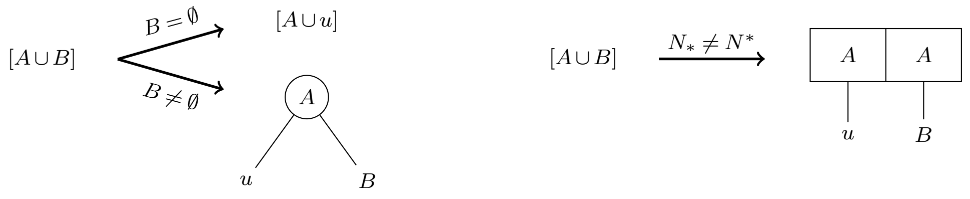


Fig. 5: Templates of a leaf ($V_{N_*} = A \cup B$)

Definition 8 For every node N of an MPQ-tree associated with a dimension d of a packing configuration, we define the *width* λ_N of N as described below.

- If N is a **leaf** L labelled with the set V_L, $\lambda_L = \max_{i \in V_L}\{w_d(i)\}$ if $V_L \neq \emptyset$ and $\lambda_L = 0$ otherwise.
- If N is a **P-node** P labelled with the set V_P, and $\lambda_{f_1}, \dots, \lambda_{f_m}$ are the widths of each of its children, $\lambda_P = \max\left\{\sum_{j=1}^{m} \lambda_{f_j}, \max_{i \in V_P}\{w_d(i)\}\right\}$
- If N is a **Q-node** Q, let $S_1, \dots, S_m$ be its sections and $\lambda_{f_1}, \dots, \lambda_{f_m}$ the widths of each of the children of the node. In order to define the width λ_Q, we first define, recursively from m to 1, the widths of its sections: $\lambda_{S_j}, \forall 1 \leq j \leq m$:

$$\lambda_{S_m} = \lambda_{f_m}$$

Suppose $\lambda_{S_{k+1}}, \dots, \lambda_{S_m}$ are defined, then

$$\lambda_{S_k} = \max\left\{\lambda_{f_k}, \max_{i \in S_k, i \notin S_{k-1}}\left\{w_d(i) - \sum_{h>k,\, i \in S_h} \lambda_{S_h}\right\}\right\}$$

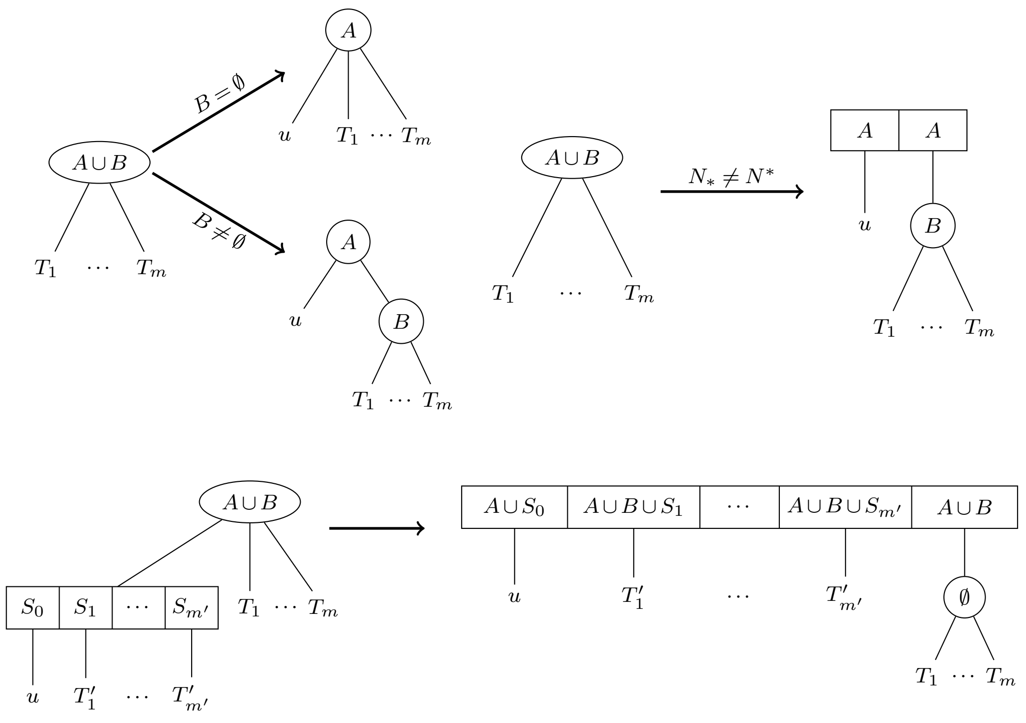


Fig. 6: Templates of a P-node

The width of N is given by the following formula

$$\lambda_Q = \sum_{k=1}^{m} \lambda_{S_k}$$

It follows from definition that for every node N with children $f_1, \ldots, f_m$, $\lambda_N \geq \sum_{k=1}^{m} \lambda_{f_k}$.

Lemma 9 *Let $G_d = (V, E_d)$ be an interval graph of a packing class and T_d an associated MPQ-tree. Let v be an item of V. For every P-node or leaf N such that v belongs to the label of N, we have $w_d(v) \leq \lambda_N$. For every Q-node N containing v in the labels of some of its sections, we have $w_d(v) \leq \sum_{h=k}^{l} \lambda_{S_h}$, where $S_k, \ldots, S_l$ are the sections of N containing v.*

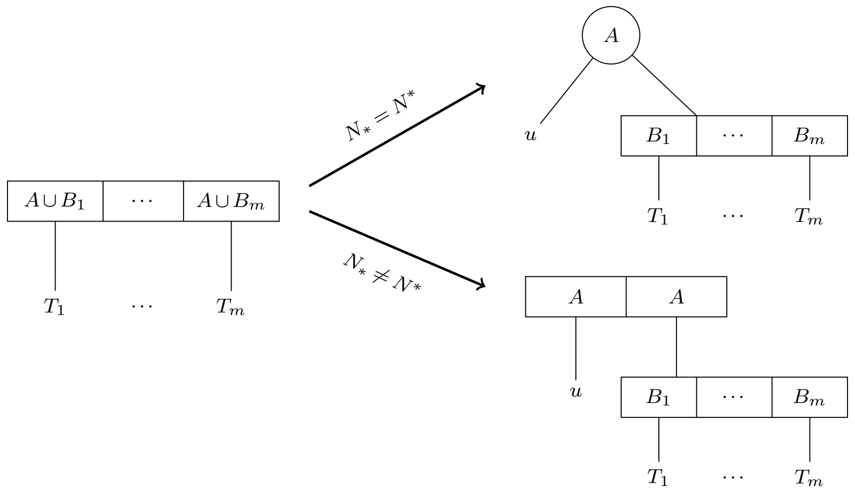
Proof The cases of a P-node and of a leaf are obvious. Suppose N is a Q-node with sections $S_1, \ldots, S_m$ and children $f_1, \ldots, f_m$. Since branches associated with sections $S_k, \ldots, S_l$ of N are all the cliques containing v, we have $\lambda_{S_k} \geq w_d(v) - \sum_{h=k+1}^{l} \lambda_{S_h}$. Hence, $w_d(v) \leq \sum_{h=k}^{l} \lambda_{S_h}$. □

The following proposition translates property (*P2*) of Theorem 1.

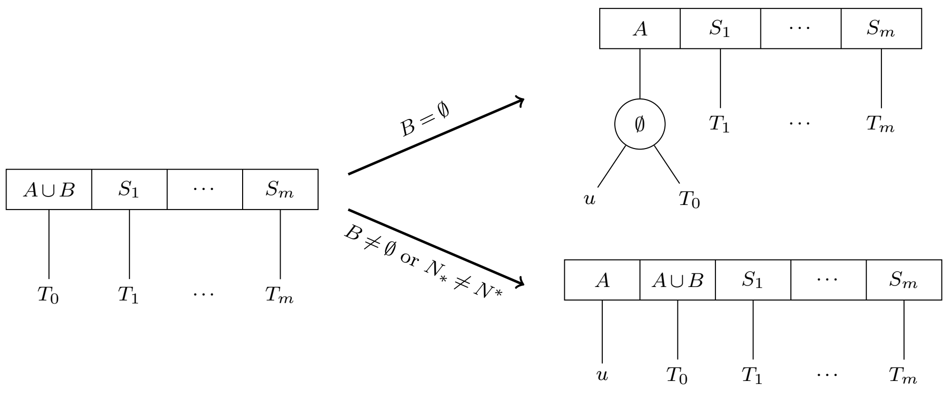
Proposition 10 *Let G be an interval graph. G satisfies property P2 if and only if for every associated MPQ-tree T of G, the width λ_R of the root of T verifies $\lambda_R \leq w_d(C)$.*

Proof
Suppose G satisfies property *P2* and consider an MPQ-tree T of G with root R. We will prove that $\lambda_R \leq w_d(C)$.

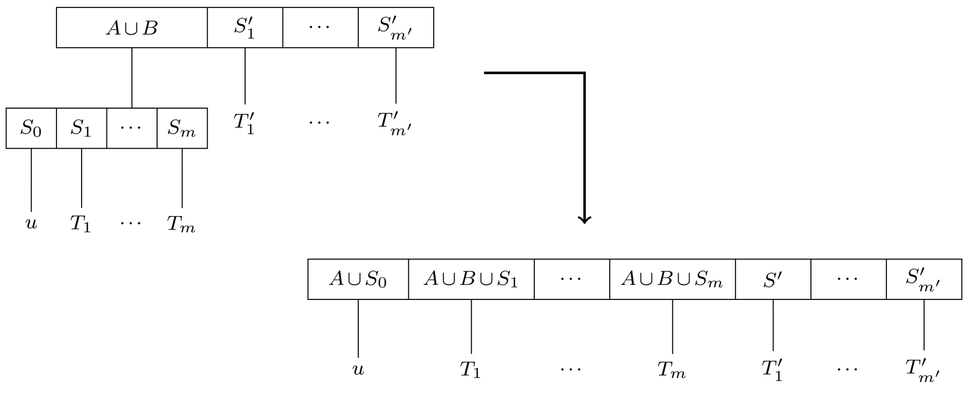
The proof is by induction on the distance between R and the nodes of T. Let $H(k)$ be the assertion: "If N is a node of T at distance k from R then there is a stable set S of $G[N]$ such that $\lambda_N \leq w_d(S)$, where $G[N]$ is the subgraph of G induced by the set of vertices of G contained in the labels of the subtree of T rooted in N."



(a) Q1



(b) Q2



(c) Q3

Fig. 7: Templates of a Q-node

- Let M be the maximal distance from R in T. Let N be any node of T at distance M from R. Hence N is a leaf. Due to the definition of a leaf, there is i in V_N such that $w_d(i) = \lambda_N$. Hence, $S = \{i\}$ and $H(M)$ is true.
- Assume that $H(k)$ is true for $k \leq M$ (induction hypothesis). Let N be any node of T at distance $k-1$ from R.
  - If N is a leaf then the above argument applies again.

– If N is a P-node with m sons, then we distinguish two cases. If $\max_{j \in V_N}\{w_d(j)\} \leq \sum_{k=1}^{m} \lambda_{f_k}$, then let $S = \bigcup_{k=1}^{m} Z_k$, where Z_k is the stable set associated to son f_k such that, due to induction hypothesis, $\lambda_{f_k} \leq w_d(Z_k)$. Since in the graph G_d there are no edges between two vertices contained in two different children of N, S is clearly an independent set of $G[N]$. Applying the induction hypothesis, $\lambda_N \leq \sum_{k=1}^{m} \lambda_{f_k} \leq w_d(S)$ and we are done. If $\max_{j \in V_N}\{w_d(j)\} > \sum_{k=1}^{m} \lambda_{f_k}$ then let $S = \{i\}$ where i is such that $w_d(i) = \max_{j \in V_N}\{w_d(j)\}$. Thus $\lambda_N = w_d(S)$ and we are done.
– If N is a Q-node with sections $S_1, \ldots, S_m$ then let Z_k be the stable set of G built by iteration of k from 1 to m:
  - $k = 1$. If the width of S_1 is equal to the width of its child f_1, then due to the induction hypothesis, Z_1 is a stable set of $G[N]$ such that $\lambda_{S_1} \leq w_d(Z_1)$; otherwise there is a vertex $b \in S_1$ such that $\lambda_{S_1} = w_d(b) - \sum_{h>1, b \in S_h} \lambda_{S_h}$, in which case define $Z_1 = \{b\}$.
  - $k \geq 2$. If there is a vertex $b \in S_k, b \in S_{k-1}$ and $Z_{k-1} = \{b\}$, define $Z_k = \{b\}$; otherwise, if there is a vertex $b \in S_k, b \notin S_{k-1}$ and $\lambda_{S_k} = w_d(b) - \sum_{h>k, b \in S_h} \lambda_{S_h}$, then $Z_k = \{b\}$; otherwise, necessarily $\lambda_{S_k} = \lambda_{f_k}$, in which case, due to the induction hypothesis, Z_k can be chosen to be the stable set of $G[f_k]$ such that $\lambda_{f_k} \leq w_d(Z_k)$.

  Let $S = \bigcup_{k=1}^{m} Z_k$. Clearly, S is a stable set of G_N such that $\lambda_N = \sum_{k=1}^{m} \lambda_{S_k} \leq \sum_{s \in S} w_d(s) = w_d(S)$.

Hence $H(k-1)$ is true.

Due to the induction, $H(0)$ is true, and since $w_d(S) \leq w_d(C)$ (property *P2* of Theorem 1), we have $\lambda_R \leq w_d(C)$.

Conversely, suppose $\lambda_R \leq w_d(C)$. Consider an independent set S of G_d. By definition of an MPQ-tree, $\forall u \in S$, in T_d there exist exactly two distinct cases for the node N of T_d containing u:

1. N is a Q-node. Let $S_k, \ldots, S_l$ be the sections of N containing u. In this case, let $\Lambda_u = \sum_{h=k}^{l} \lambda_{S_h}$.
2. N is a P-node or a leaf. In this case, let $\Lambda_u = \lambda_{N_u}$.

Since S is an independent set, no two vertices of S are contained in the same branch of T_d. Hence, by definition of λ_R, we have $\sum_{s \in S} \Lambda_s \leq \lambda_{R_d}$. By Lemma 9, we have $\forall x \in V$, $w_d(x) \leq \Lambda_x$. Hence, $w_d(S) \leq \lambda_{R_d}$. Applying the hypothesis, $w_d(S) \leq w_d(C)$ which is exactly the property (P2) of Theorem 1. □

Notice that property (P3) of Theorem 1 may be easily translated with respect to MPQ-trees, since the branches of an MPQ-tree represent the maximal cliques of the interval graphs.

2.2 The core of the algorithm to check feasibility

For a given dimension and a given order on the items (which are the vertices of the associated interval graph), the (pseudo) Algorithm 1 constructs all the MPQ-trees to get the one representing this dimension. Thus, it may have to be executed for all possible orders σ (in the next subsection, we explain that we have in fact to consider only a subset of orders). The items are added one by one in the MPQ-tree such that for every node N, $\lambda_N \leq w_d(C)$.

2.3 Optimizations

The structure of an MPQ-tree provides some partial information about the position of the items which can be easily exploited. For instance, its frontier gives the order of appearance of items in the packing

```
Require: order $\sigma$, (int) n, MPQ-tree $T$
Ensure: TRUE if there exists a feasible packing, FALSE otherwise
  recurse(MPQ-node currentNode, int nrVertex)
  if nrVertex $>$ n then
    return TRUE
  end if
  repeat
    for all corresponding patterns do
      apply the modification with the vertex $\sigma$(nrVertex)
      currentNode $\leftarrow$ new created leaf L
      for all $N$ in leftmost branch of $T$ do
        update $\lambda_N$
      end for
      if $\lambda_R \leq w_d(C)$ then
        if recurse(currentNode,nrVertex+1) then
          return TRUE
        end if
      end if
      for all $N$ in leftmost branch of $T$ do
        backup $\lambda_N$
      end for
      undo the modification
    end for
    currentNode $\leftarrow$ currentNode.getFather()
  until currentNode $\neq$ NULL
  return FALSE
```

Algorithm 1: recursive enumeration of MPQ-trees to check feasibility

configuration. In addition, the higher an item is contained in the tree associated with a dimension d, the more items it "covers" in d (i.e. the more are the intersections of its projection on d with the projections of the other items on d). Being able to compute at each step the widths of each node, we provide some optimizations of the algorithm by adding some valid constraints, breaking some symmetries and early detecting some infeasible configurations.

2.3.1 Bi-dimensional case: remaining areas, widths and branches heights

In the bi-dimensional case, a few basic valid constraints may be applied when enumerating MPQ-trees in dimension 1:

1. Assume that at the step i of the algorithm, an MPQ-tree T_1 containing $\sigma(1),\ldots,\sigma(i)$ was constructed with the root having m children $f_1,\ldots,f_m$. After the step i, the widths of the children $f_2,\ldots,f_m$ cannot be modified (since only the leftmost branch can be modified), therefore by Proposition 10, the remaining items $\sigma(i+1),\ldots,\sigma(n)$ have to be packed in the area $W(C) - w_2(C) * \sum_{h=2}^{m} \lambda_{f_h}$. Hence the following constraint is valid:

$$\sum_{h=i+1}^{n} W(\sigma(h)) + w_2(C) * \sum_{h=2}^{m} \lambda_{f_h} \leq W(C)$$

2. The width of any item among $\sigma(i+1),\ldots,\sigma(n)$ cannot exceed the remaining width of the container after removing $\lambda_{f_2},\ldots,\lambda_{f_m}$:

$$\sum_{h=2}^{m} \lambda_{f_h} + \max\{w_1(\sigma(i+1)),\ldots,w_1(\sigma(n))\} \leq w_1(C)$$

3. Let $\mathcal{B}$ be a branch of T_1. Since all the items contained in this branch overlap in the dimension 2, the sum of the size of these items in dimension 2 cannot exceed $w_2(C)$:

$$\sum_{i \in \mathcal{B}} w_2(i) \leq w_2(C)$$

2.3.2 Positive widths

The left part of the Figure 8 is an example of packing which could be avoided by considering only the right part. Next lemma establishes a sufficient condition for avoiding the enumeration of this type of equivalent packing configurations. Hence, our algorithm will consider only the right packing of Figure 8.

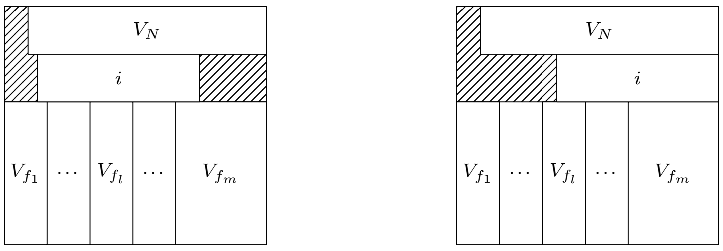


Fig. 8: Example of "similar" packing configurations

Lemma 11 *Let G_d be an interval graph of a packing class. There is an associated MPQ-tree of G_d such that for every internal node N with m sons $f_1, \dots, f_m$, the following holds:*

– *If N is a P-node, $\forall i \in V_N$, $w_d(i) - \sum_{k=2}^{m} \lambda_{f_k} > 0$*

– *If N is a Q-node with sections $S_1, \dots, S_m$, $\forall k \in \{1, \dots, m\}$, $\forall i \in S_k$, $w_d(i) - \sum_{h>k, i \in S_h} \lambda_{S_h} > 0$*

Proof Recall that, from Definition 8, for every node or section X, $\lambda_X \geq 0$.

Let T' be an MPQ-tree associated with G_d and let N' be an internal node of T' not satisfying the statement.

– Suppose N' is a P-node. Then there is $i \in V_{N'}$, such that $w_d(i) - \sum_{k=2}^{m} \lambda_{f_k} \leq 0$.

We choose i such that $w_d(i) = \min_{j \in V_{N'}} \{w_d(j)\}$. Let $l = \max\{h \geq 2 \mid w_d(i) \leq \sum_{k=h}^{m} \lambda_{f_k}\}$. We distinguish two cases for the value of l.

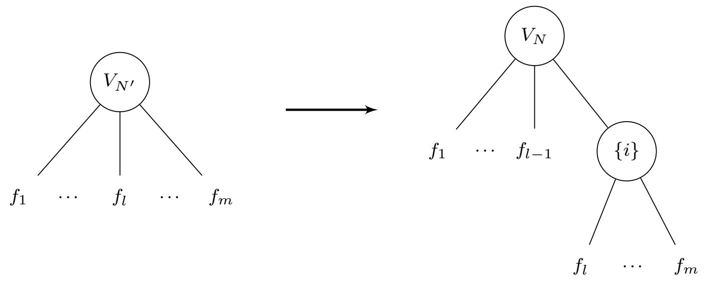


Fig. 9: P-node replacement

- Suppose $l < m$. Let T be the MPQ-tree obtained from T' by replacing N' as shown in Figure 9. Observe that the MPQ-trees associated to dimension 1 of the packing configurations of Figure 8 are the MPQ-trees of Figure 9 when $f_1, \dots, f_m$ are leaves. In T, N is the P-node replacing N' of T' and having $V_N = V_{N'} \setminus \{i\}$ as a labelling set, $\{i\}$ is the labelling set of a new P-node, say A, having $f_l, \dots, f_m$ as children. In T, $\lambda_A = \sum_{k=l}^{m} \lambda_{f_k}$ and, therefore, $\sum_{k=1}^{m} \lambda_{f_k} = \sum_{k=1}^{l-1} \lambda_{f_k} + \lambda_A$. Hence, $\lambda_N = \lambda_{N'}$. Finally, we note that $w_d(i) - \sum_{k=l+1}^{m} \lambda_{f_k} > 0$.

- Suppose $l = m$. Let T be the MPQ-tree obtained from T' by removing i from the set $V_{N'}$ and modifing f_m as follows. If f_m is a leaf or a P-node, then insert i in the labelling set of f_m. If f_m is a Q-node insert i in every section of f_m. Now, if f_m is a leaf we are done. Otherwise, if i does not satisfy the statement for f_m, then apply the proof on f_m.

– Suppose N' is a Q-node with sections $S'_1, \dots, S'_m$. Then there are i and a section S'_k with $i \in S'_k$, such that $w_d(i) - \sum_{h>k, i \in S'_h} \lambda_{S'_h} \leq 0$. We choose k such that $k = \min\{h \mid i \in S_h\}$. Let $l = \min\{h \mid w_d(i) > \sum_{h'>h, i \in S'_{h'}} \lambda_{S'_{h'}}\}$. We have $w_d(i) - \sum_{h \geq l, i \in S'_h} \lambda_{S_h} \leq 0$. We distinguish two cases for the value of l.

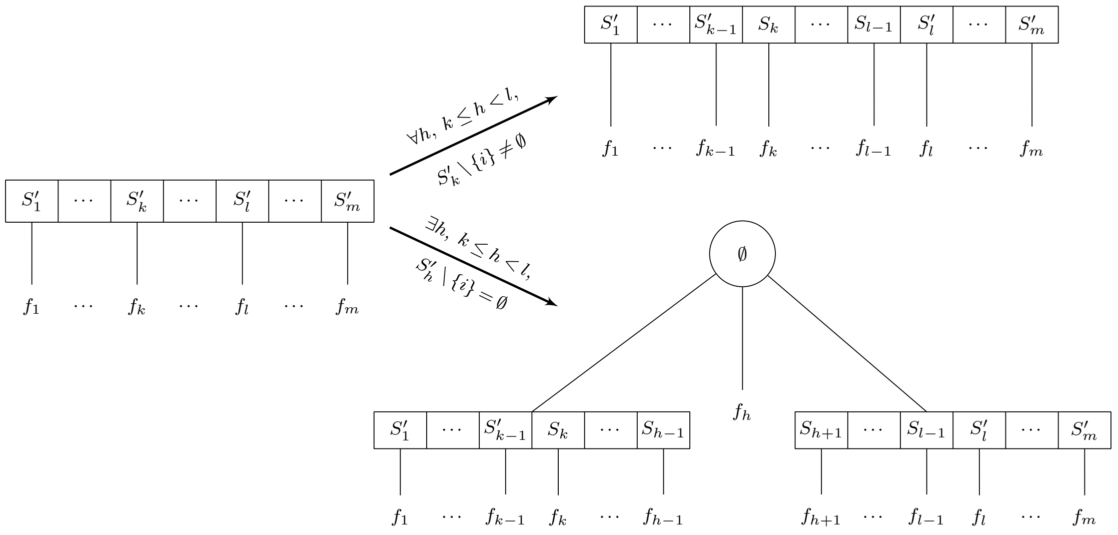


Fig. 10: Q-node replacement

- Suppose $l < m$. Intuitively, one has to remove i from every section of N' which does not satisfy the statement of the lemma in case of a Q-node. Apply the replacement of Figure 10 to construct from T' an MPQ-tree T, in which $\forall h \in \{k, \dots, l-1\}$, $S_h = S'_h \setminus \{i\}$ and the other sections do not change. To distinguish the widths of T and T', let λ^T_M (resp. $\lambda^{T'}_M$) denote the width of any node (or section) M of T (resp. of T').
  Consider the upper case of the replacement of Figure 10. We have $\forall h \in \{l, \dots, m\}$, $\lambda^{T'}_{S'_h} = \lambda^T_{S'_h}$. Since $w_d(i) - \sum_{h \geq l, i \in S_h} \lambda^{T'}_{S_h} \leq 0$, we have $\forall h \in \{k, \dots, l-1\}$, $\lambda^T_{S_h} = \lambda^{T'}_{S'_h}$ and, therefore, $\forall h \in \{1, \dots, k-1\}$, $\lambda^T_{S'_h} = \lambda^{T'}_{S'_h}$. Hence $\lambda^T_N = \lambda^{T'}_N$.
  Consider the bottom case. We suppose that there exists only one section S_h containing only item i. If there are more than one section of this type, one can easily use the same technique of replacement. Let N be the new created P-node and N_1 and N_2 be the two created Q-nodes. Obviously, $\forall h' \in \{l, \dots, m\}$, $\lambda^{T'}_{S'_{h'}} = \lambda^T_{S'_{h'}}$. By the same arguments as in the upper case of the figure, we have $\forall h' \in \{h+1, \dots, l-1\}$, $\lambda^{T'}_{S'_{h'}} = \lambda^T_{S_{h'}}$. Now, since $S_{h-1} \cap S_{h+1} = \emptyset$, $\forall h' \in \{k, \dots, h-1\}$, $\lambda^{T'}_{S'_{h'}} = \lambda^T_{S_{h'}}$ and, therefore, $\forall h' \in \{1, \dots, k-1\}$, $\lambda^{T'}_{S'_{h'}} = \lambda^T_{S'_{h'}}$. In T', $S'_h = \{i\}$ and the subtree

rooted in f_h is not empty, because otherwise the branch of T' containing S'_h and f_h would not represent a maximal clique. Thus $\lambda^{T'}_{S'_h} = \lambda^{T}_{f_h} > 0$ and $\lambda^{T}_{f_h} = \lambda^{T'}_{f_h}$. We have $\lambda_{N_1} = \sum_{h'=1}^{h-1} \lambda^{T'}_{S_{h'}}$ and $\lambda_{N_2} = \sum_{h'=h+1}^{m} \lambda^{T'}_{S_{h'}}$. Hence, $\lambda_N = \lambda_{N_1} + \lambda^{T}_{f_h} + \lambda_{N_2} = \lambda^{T'}_{N'}$.
Notice that in both cases of the replacement, every item satisfying the condition of the lemma in T', satisfies it also in T.

- Suppose $l = m$. Similarly to the case when $l = m$ and N' is a P-node, build an MPQ-tree T from T'. For this purpose, remove i from every section of N' containing i and modify f_m by inserting i as in the case when N' is a P-node. Now, if $S'_m = \{i\}$, then apply the replacement of Figure 11. The case when $m > 3$ (the upper case of the Figure) is straightforward. Consider the case when $m = 3$. Recall that by definition of a Q-node, every item of any section must be contained in at least two sections of the same Q-node. Hence, $S'_1 \subset S'_2$, $S'_3 \subset S'_2$ and $S'_2 = S'_1 \cup S'_3$. Therefore, $S'_1 = S'_2 \setminus \{i\} = A$ and N' is replaced by a P-node as shown in the bottom case of the replacement of Figure 11.
Finally, if f_m is a leaf we are done. Otherwise, if i does not satisfy the statement of the lemma for f_m, apply the proof for the new node f_m.

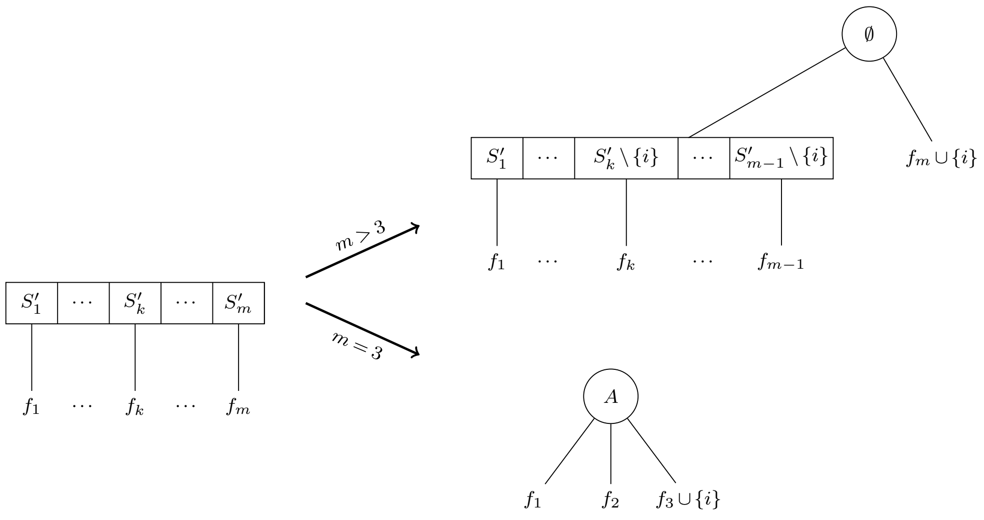


Fig. 11: Q-node replacement when $S'_m = \{i\}$

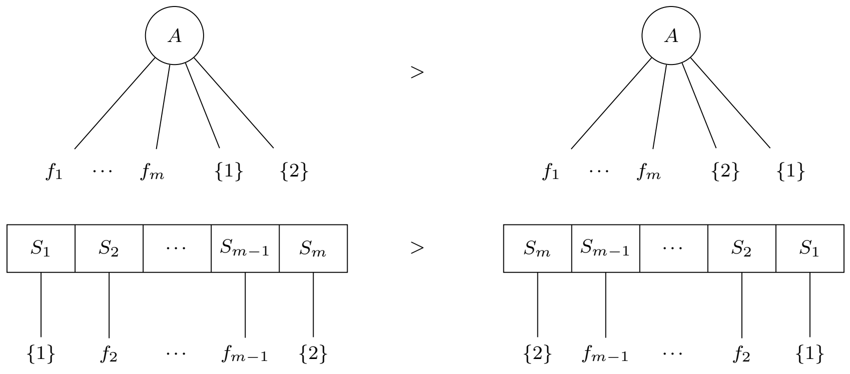
By applying the replacement of Figures 9, 10, 11 for all internal nodes (P-type or Q-type) of T', for every item i not satisfying the statement of the lemma, we can construct another MPQ-tree associated with a feasible packing in which every item satisfies this statement. □

2.3.3 Order on the children of a node

Using the general properties of PQ-trees, we impose an order on the children of a node during the generation. We define for any subtree T_s of an MPQ-tree T a variable $m_T = \min\{i \mid i \in T_s\}$. We say that T is *lexicographically ordered* if:

– For any two subtrees T_1 and T_2 children of the same P-node such that T_1 is the first in the right to left reading order, we have $m_{T_1} > m_{T_2}$.
– For two subtrees T_1 and T_2 children of the same Q-node where T_1 is the rightmost and T_2 is the leftmost, we have $m_{T_1} > m_{T_2}$.

An example of lexicographically ordered MPQ-trees is depicted in Figure 12. Notice that due to the definition of PQ-trees, for every MPQ-tree, there is an equivalent lexicographically ordered MPQ-tree. Therefore, our algorithm to check feasibility generates all lexicographically ordered possible MPQ-trees.



A
f_1 $\cdots$ f_m $\{1\}$ $\{2\}$
$>$
A
f_1 $\cdots$ f_m $\{2\}$ $\{1\}$
S_1 S_2 $\cdots$ S_{m-1} S_m
$\{1\}$ f_2 $\cdots$ f_{m-1} $\{2\}$
$>$
S_m S_{m-1} $\cdots$ S_2 S_1
$\{2\}$ f_{m-1} $\cdots$ f_2 $\{1\}$

Fig. 12: Lexicographic order between equivalent MPQ-trees

3 Computational Results

We report the performance of our algorithm on 37 classical benchmarks for OKP-2 from [3, 2, 7, 12] (Tables 1a) and on 42 benchmarks for OPP-2 defined in [8] (Table 1b). For OKP-2 instances, we used a basic branch-and-bound procedure to select the items to be checked for feasibility.

The program was implemented in Java 6 and was tested on a PC (Pentium IV, 3GHz). These conditions of experimentation are quite similar to the ones used by Fekete and Schepers [12] (PC with Pentium IV processor, 2,8 GHz, using C++). In contrary to Fekete and Schepers, we do not use some heuristics prior to launching the main algorithm to check feasibility.

Table 1a shows the running times of our algorithm along with the ones existing in the literature, as reported in [12]. The first column (JPV) gives our runtimes. The column FS gives the runtimes of the algorithm from [12]. The column BB corresponds to the algorithm of Baldacci and Boschetti [1], implemented in Visual Digital Fortran 6.0 and run on a Laptop equipped with an Intel Pentium IV, 2.5 GHz. The columns A0, A1, A2 and A3 correspond to the algorithms of Caprara and Monaci, as depicted in [5] and which were implemented in ANSI C and run on a Pentium III 800 MHz. We also report the number of unsolved benchmarks (within the time limit of 1800 seconds) and the average time (computed on the set of instances `cgcut`, `gcut` and `okp`), with the convention that an unsolved benchmark counts for 1800 seconds.

The running times of our algorithm are of interest since it is one of the two algorithms to solve all, but one, of the benchmarks within the time limit of 1800 seconds (`gcut13` is still open, the optimal value being unknown). Compared to Fekete and Schepers', on an average, our running times turned out to be smaller and were significantly better for 6 instances (`cgcut2`, `gcut3`, `gcut8`, `gcut11`, `gcut12` and `okp1`), though Fekete and Schepers' algorithm outperforms ours for the 2 instances `okp2` and `okp5`. Additionally, compared to other authors', our running times are considerably better. Note that considering the big difference between our processors and the ones used by Caprara and Monaci for their experiments, one can see that algorithms A1 and A3 are also competitive.

Table 1b shows the running times on benchmarks defined in [8]. The third column corresponds to the algorithm of Clautiaux, Carlier and Moukrim [8] implemented in C on a PC (Pentium IV, 2.6 GHz). The size of the containers is (20,20) and there are 10 to 23 items to be packed. These benchmarks are designed to check feasibility (OPP-2) and therefore relevant to the approach described in this paper. However, the size of containers being small, our approach as well as Fekete and Schepers' is less appropriate compared to the methods using a space discretization. The F (resp. N) character in the name of the instance stand for "feasible" (resp. "non feasible") and X character is used when there exists another instance being of the same type (feasible or infeasible) and with the same number of items to be packed. In the case of feasible instances the algorithm has a good performance, giving better execution times than other algorithms for four of them and being significantly worse for only two instances. For unfeasible instances, the running times of about half of them are equivalent to those of the other authors. However, for six instances the running times are less competitive, while only one instance is not solved. In average, our algorithm is thrice faster than Fekete and Schepers', but significantly slower than Clautiaux, Carlier and Moukrim's.

Benchmark	JPV[1]	FS[2]	BB[3]	A0[4]	A1[4]	A2[4]	A3[4]
ngcut1 … ngcut11	0	0	0				
hccut2 … hccut5	0	0	0				
wang20	0	0	-	6	6	17	2
cgcut1	0	0	0	0	1	1	1
cgcut2	**96**	>1800	>1800	>1800	>1800	533	531
cgcut3	1	0	95	23	23	4	4
gcut1	0	0	0	0	0	0	0
gcut2	0	0	0	0	0	25	0
gcut3	**0**	4	2	>1800	2	276	3
gcut4	137	195	46	>1800	346	>1800	376
gcut5	0	0	0	0	0	0	0
gcut6	0	0	1	0	0	9	0
gcut7	0	2	3	1	0	354	1
gcut8	**60**	253	186	1202	136	>1800	168
gcut9	0	0	0	0	0	0	0
gcut10	0	0	0	0	0	6	0
gcut11	**2**	8	3	16	14	>1800	16
gcut12	**6**	109	12	63	16	>1800	25
gcut13	>1800	>1800	>1800	>1800	>1800	>1800	>1800
okp1	**1**	10	779	24	25	72	35
okp2	49	20	288	>1800	>1800	1535	1559
okp3	1	5	0	21	1	465	10
okp4	1	2	14	40	2	0	4
okp5	210	11	190	40	>1800	513	488
# unsolved	1	2	2	5	4	5	1
Average time	64	114	248	474	353	582	228

(a) OKP-2 benchmarks

Benchmark	JPV[1]	FS[2]	CCM[5]
E02F17	30	7	12
E02F20	**2**	> 1800	12
E02F22	**2**	167	4
E04F15	60	0	1
E04F17	**9**	13	26
E04F19	14	560	7
E04F20	**0**	22	3
E05F15	**0**	0	3
E05F18	0	0	126
E05F20	6	491	2
E07F15	0	0	1
E08F15	0	0	117
E20F15	1	0	1
E00X23	> 1800	> 1800	289
E03X18	24	0	22
E05X15	110	2	0
E07X15	79	0	1
E10X15	50	0	1
E13X15	2	0	0
E20X15	8	0	44
E00N10	0	0	0
E00N15	2	0	2
E00N23	87	> 1800	86
E02N20	0	0	1
E03N10	0	0	0
E03N15	21	0	1
E03N16	51	2	32
E03N17	45	0	4
E04N15	7	0	1
E04N17	3	0	1
E04N18	7	10	7
E05N15	4	0	0
E05N17	1	0	1
E07N10	0	0	0
E07N15	0	0	0
E08N15	3	0	1
E10N10	0	0	0
E10N15	0	0	0
E13N10	0	0	0
E13N15	0	0	0
E15N10	0	0	0
E15N15	0	0	0
# unsolved	1	3	0
Average time	57	158	19

(b) OPP-2 benchmarks

Table 1: Running times in seconds

[1] Java, Pentium IV, 3GHz

[2] C++, Pentium IV, 2.8GHz

[3] Visual Digital Fortran 6.0, Pentium IV, 2.5 GHz

[4] ANSI C, Pentium III, 800 MHz

[5] C, Pentium IV, 2.6 GHz

Solving the instance `gcut13` is still a challenging issue. The particularity of this benchmark is that the size of the container is big (3000 by 3000) while there are many items (32) to be packed, which are of relatively small sizes.

4 Conclusion

In this paper we presented a new algorithm for solving the OPP-2 problem by introducing new characterization of feasible packings based on the properties of MPQ-trees. Our approach is of interest since the computational results are competitive and as well as in Fekete and Schepers' case it can be easily extended to the multidimensional case.

References

1. Baldacci R, Boschetti M (2007) A cutting plane approach for the two-dimensional orthogonal non-guillotine cutting stock problem. European Journal of Operational Research 183(3):1136–1149
2. Beasley J (1985) Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 36(4):297–306
3. Beasley J (1985) An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33(1):49–64
4. Booth K, Lueker G (1975) Linear algorithms to recognize interval graphs and test for the consecutive ones property. In: Proceedings of the seventh Annual ACM Symposium on Theory of Computing (STOC'75), pp 255–265
5. Caprara A, Monaci M (2004) On the two-dimensional knapsack problem. Operations Research Letters 32(1):5–14
6. Carlier J, Clautiaux F, Moukrim A (2007) New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. Computers and Operations Research 34(8):2223–2250
7. Christofides N, Hadjiconstantinou E (1995) An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts. European Journal of Operational Research 83(1):21–38
8. Clautiaux F, Carlier J, Moukrim A (2007) A new exact method for the orthogonal packing problem. European Journal of Operational Research 183(3):1196–1211
9. Clautiaux F, Jouglet A, Carlier J, Moukrim A (2008) A new constraint programming approach for the orthogonal packing problem. Computers and Operations Research 35(3):944–959
10. Fekete S, Schepers J (1997) On more-dimensional packing i: Modeling. Tech. rep., University of Köln, Germany
11. Fekete S, Schepers J (1997) On more-dimensional packing iii: Exact algorithms. Tech. rep., University of Köln, Germany
12. Fekete SP, Schepers J, van der Veen J (2007) An exact algorithm for higher-dimensional orthogonal packing. Operations Research 55:569–587
13. Ferreira E, Oliveira J (2005) A note on fekete and schepers' algorithm for the non-guillotinable two-dimensional packing problem. Technical Report `http://paginas.fe.up.pt/~jfo/techreports/Fekete%20and%20Schepers%20OPP%20degeneracy.pdf`
14. Fulkerson D, Gross O (1965) Incidence matrices and interval graphs. Pacific Journal of Mathematics 15(3):835–855
15. Ghouila-Houri A (1962) Caractérisation des graphes non orientes dont on peut orienter les aretes de maniere à obtenir le graphe d'une rélation d'ordre. Comptes Rendus Mathématique Académie des Sciences Paris 254:1370–1371
16. Gilmore P, Hoffman A (1964) A characterization of comparability graphs and of interval graphs. Canadian Journal of Mathematics 16:539–548
17. Korte N, Möhring R (1989) An incremental linear-time algorithm for recognizing interval graphs. SIAM Journal on Computing 18(1):68–81
18. Rose D, Tarjan R, Lueker G (1976) Algorithmic aspects of vertex eliminationon graphs. SIAM Journal on Computing 5:266–283