*INFORMS Journal on Computing*
Vol. 18, No. 4, Fall 2006, pp. 455–465
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 1804 | 0455

DOI 10.1287/ijoc.1040.0124


# Exact Algorithms for a Loading Problem with Bounded Clique Width

Linda S. Moonen, Frits C. R. Spieksma
Research Center for Operations Research and Business Statistics (ORSTAT), Katholieke Universiteit Leuven, Naamsestraat69, B-3000 Leuven, Belgium,
{linda.moonen@econ.kuleuven.be, frits.spieksma@econ.kuleuven.be}

In this paper we discuss a special pallet-loading problem, which we encountered at a manufacturing company. In graph-theoretical terms, the problem is equivalent to partitioning a permutation graph into bounded-size cliques. We formulate the problem as an integer program, and present two exact algorithms for solving it. The first algorithm is a branch-and-price algorithm based on the integer-programming formulation; the second one is an algorithm based on the concept of bounded clique width. The latter algorithm was motivated by the structure present in the real-world instances. Test results are given, both for real-world instances and randomly generated instances. As far as we are aware, this is the first implementation of an algorithm based on bounded clique width.

*Key words*: integer programming; analysis of algorithms
*History*: Accepted by William Cook, Area Editor for Design and Analysis of Algorithms; received November 2003; revised August 2004; accepted November 2004.

## 1. Introduction

Consider the following situation. Given is a set $S$ of distinct points (or items) in the plane, $S = \{1, 2, \ldots, n\}$. For any pair of points $i, j \in S$, we say that $i$ is *smaller* than $j$ ($i \prec j$) iff

$$(x_i \leq x_j) \wedge (y_i \leq y_j),$$

where $x_i$ and $y_i$ denote the $x$- and $y$-coordinate of $i$, $i \in S$. Furthermore, we call $R \subseteq S$ a *feasible subset* (or a *stable pallet*, see Section 1.1) iff for any two points $i, j \in R$ either $i \prec j$ or $j \prec i$.

The problem we consider is as follows: given the set $S$ and an integer $B \leq n$, partition $S$ into as few feasible subsets as possible such that each subset contains no more than $B$ points. For Section 1.1, we refer to this problem as the *loading problem*.

This loading problem is intimately related to problems in graph theory. Indeed, when we build a graph $G = (V, E)$ as follows: for each point $i$ in $S$ there is a node in $V$, and two nodes are adjacent iff $i \prec j$ or $j \prec i$, a so-called permutation graph arises (see for instance Golumbic 1980). Observe that a feasible subset in $S$ corresponds to a clique in $G$. Thus, in graph-theoretic terms, the loading problem boils down to finding a partition of a permutation graph $G$ into a minimum number of cliques such that each clique has no more than $B$ vertices. In fact, the two problems are equivalent.

Of course, if $B$ is not present in the input of our problem, the resulting problem is solvable in polynomial time since it is a special case of Dilworth's chain decomposition theorem (Dilworth 1950). However, Jansen (2003) proves that for each fixed $B \geq 6$, our loading problem is $\mathcal{NP}$-hard. We now proceed by describing the industrial application that motivates our problem.

### 1.1. Motivation

Our motivation comes from a real-world setting encountered at Bruynzeel Storage Systems (BSS), a manufacturing company in the Netherlands. BSS produces mobile storage systems that are delivered worldwide. To construct such a system, BSS produces many rectangular shaped boxes, each with a specific length and a specific width. We refer to such a rectangular shaped box as an *item*. A single storage system may consist of up to 200 items. Furthermore, there are no standard sizes, so each customer specifies his or her own requirements. The height of an item, however, is identical for all items. The items have to be loaded onto pallets for transportation to the clients. It is allowed to place items on top of each other in layers; however, the number of items per layer is restricted to one. Since the items all have identical heights, it follows that the height of the trucks that transport the pallets determines the maximum number of layers of each pallet. We denote this number by $B$ (in the case of BSS, $B = 12$). A crucial feature involves the *stability* of the pallets (see for example Bischoff 1991). BSS stipulated that no larger item could be placed on top of a smaller item. More precisely, both the length and the width of an item placed in some layer must be smaller

than or equal to the length and the width of the item placed in the layer directly under it. This restriction ensures that pallets arrive in good shape at their final destination (Moonen 2001). In order to achieve an efficient usage of the trucks it is important to minimize the total number of pallets used.

Thus, our problem can be seen as a type of pallet-loading problem (PLP). Pallet- or container-loading problems concern the optimal packing of small items into large containers or pallets. The terms *pallets* and *containers* are used interchangeably in most studies, although there is an important difference between them. When loading goods onto a pallet, the notion of the *stability* of the loading schemes is far more important than when the goods are to be loaded into a container. When loading items on a pallet, we cannot make use of the upstanding walls that we have when loading items into a container, so the stability of the loading schemes must be guaranteed (Bischoff 1991).

Although the problem discussed in this paper is a type of pallet-loading problem, it is quite different from customary PLPs. Usually, PLPs are three-dimensional packing problems that concern the optimal packing of a number of small items into a large container, with the objective to minimize the volume of product packed on the pallet, while the problem we discuss is a two-dimensional problem. Also, restricting the pallets such that there can only be one item on each layer is unusual for general PLPs. Indeed, when it is possible to store multiple items on a layer, the resulting packings may have a very complex structure. Thus, our loading problem is a very specific pallet-loading problem, and different from ordinary PLPs.

Most of the research on PLPs has concentrated on the case where a set of identical items has to be loaded onto a single pallet. Dyckhoff (1990) gives a detailed overview of the different types of PLPs and proposes a number of solution approaches for solving them. In more recent work, Morabito and Morales (1998) developed a heuristic based on a recursive procedure to solve the problem, and G and Kang (2001) propose a heuristic that can be applied to relatively large instances (more than 5,000 items). Letchford and Amaral (2001) give a detailed analysis of upper bounds for the PLP. Also, some heuristics have been suggested for solving the PLP with non-identical items. Scheithauer and Terno (1996b) developed a heuristic combining a general branch-and-bound framework with optimal two-dimensional loading patterns. More recently, Terno et al. (2000) proposed an algorithm that uses the G4-heuristic introduced in Scheithauer and Terno (1996a), and combine this with a branch-and-bound procedure.

Notice that the application described here allows for identical items, whereas we assume in the loading problem that all items are pairwise distinct. It is not difficult to see, however, that all results presented later are valid for the case where identical items are allowed.

### 1.2. Related Work

Our loading problem also occurs in the field of mutual exclusion scheduling problems (Baker and Coffman 1996, Jansen 2003). In such a scheduling problem a graph is given such that each vertex corresponds to a job, and an edge between two vertices indicates that the two corresponding jobs are incompatible, i.e., cannot be processed at the same time. Assuming that we have $B$ processors available, and that each job needs a single time unit, computing a schedule such that the latest job finishes as soon as possible is an instance of the loading problem (provided that the conflict graph is a permutation graph).

This type of problem is also known under the name of *batch scheduling with job compatibilities*. Batch scheduling involves a machine that can process multiple jobs simultaneously. Often, jobs within a batch need to be pairwise compatible, and these compatibilities can be expressed using a compatibility graph. Boudhar (2003) and Finke et al. (2004) study different variants of these batch scheduling problems when the compatibility graph is bipartite or an interval graph.

Shum and Trotter (1996) investigate a generalization of the loading problem, namely the problem of decomposing a partial order in chains of bounded size. They show that this problem is $\mathcal{NP}$-hard and study the facial structure of a formulation of this problem.

Another related problem, described in Felsner and Wernisch (1998), involves covering as many points in a planar point set as possible, using a given number of chains.

### 1.3. Outline

The goal of this work is

- to propose two exact algorithms for solving the loading problem: a branch-and-price algorithm and an enumerative algorithm based on the concept of bounded clique width, and
- to assess the quality of these algorithms by performing computational experiments on instances from practice as well as on randomly generated instances.

The paper is organized as follows. Section 2 proposes a branch-and-price approach based on a set-partitioning formulation of the loading problem (see Barnhart et al. 1998 for a description of branch-and-price algorithms). We show that the pricing problem is solvable in polynomial time, and that we can generalize this approach to partial orders. Section 3 is devoted to an exact enumeration algorithm for a special case of the loading problem. This algorithm is based on the concept of bounded clique width.

In Section 4, we show computational results from the branch-and-price algorithm and from the algorithm based on bounded clique width. Section 5 contains the conclusions.

## 2. A Branch-and-Price Algorithm

In this section we formulate the loading problem as an integer program and we describe a branch-and-price algorithm for solving it (see eg. Barnhart et al. 1998). We use terminology corresponding to the application, i.e., we use "items" (instead of points), and "stable pallets" (instead of feasible subsets).

### 2.1. Problem Formulation

We introduce a decision variable $x_k$ for every possible stable pallet $k$, such that:

$$x_k = \begin{cases} 1 & \text{if stable pallet } k \text{ is in the solution} \\ 0 & \text{otherwise.} \end{cases}$$

Next, we define $\mathcal{J}_k$ as the set of all items contained in pallet $k$. Using a set-partitioning formulation, we get the following model:

$$\min \sum_k x_k \tag{1}$$

subject to

$$\sum_{k:\, i \in \mathcal{J}_k} x_k = 1 \quad \forall i \tag{2}$$

$$x_k \in \{0, 1\} \quad \forall k \tag{3}$$

The objective (1) is to minimize the total number of pallets needed to pack all items. Constraints (2) state that each item has to be in exactly one pallet, and constraints (3) are the zero-one constraints on the $x_k$ variables.

### 2.2. Column Generation

Since the number of variables in formulation (1)–(3) is exponentially large, we employ column generation to find the LP-relaxation of (1)–(3) without having to enumerate all variables. In the column-generation process, we start with a small subset of the variables that contains a feasible solution. All other variables are implicitly assigned the value zero. The subproblem constructed in this way is called the *restricted master problem* (RMP). We solve the LP-relaxation of RMP, and then we have to determine whether the solution found is optimal for the master problem. To do this, we have to answer the question: do there exist variables with negative reduced costs? Let $u_i$ $(i = 1, \ldots, n)$ be the dual variables corresponding to constraints (2) from our formulation. We can now formulate an expression for the reduced costs of a variable $x_k$:

$$1 - \sum_{i \in \mathcal{J}_k} u_i$$

Thus, given a feasible solution to the LP-relaxation and the corresponding dual variables, the pricing problem boils down to the following question:

$$\exists k \quad \text{such that} \quad \sum_{i \in \mathcal{J}_k} u_i > 1?$$

Lemma 1. *The pricing problem can be solved in polynomial time.*

Proof. We construct a directed graph $D = (V, A)$. There is a node in $V$ for each item, and there is a source $s$ in $V$. We draw an arc from node $i$ to node $j$ if for the corresponding items $i \prec j$ holds; this arc has length $u_j$. Also, there is an arc from $s$ to each node $i \in V$ with length $u_i$. Observe that the constructed graph is acyclic. We now define $d^p(j)$ to be the length of a longest path from $s$ to $j$ using at most $p$ arcs$(j = 1, \ldots, n)$. These longest paths can be calculated in polynomial time using the following dynamic-programming recursion:

$$\begin{gathered} d^p(j) = \max\Big(\max_{i:\,(i,j)\in A} d^{p-1}(i) + u_j,\, d^{p-1}(j)\Big) \quad \text{with} \\ d^1(j) = u_j \quad \forall\, j \neq s \;\; (p = 2, \ldots, B). \quad \square \end{gathered} \tag{4}$$

Remark. One could consider a situation where a weight $p_k$ is given for each possible pallet $k$, and next minimize total weight. For instance, in terms of the application, it would be quite natural to define the weight of pallet $k$ as the area of its largest item. Indeed, it is easy to exhibit examples where minimizing total area is not equivalent to minimizing the number of pallets needed. Notice that in this case the efficient solvability of the pricing problem is preserved since by computing $d^B(j)$ using (4), and next comparing each value with the corresponding area of item $j$ determines whether a variable with negative reduced costs exists.

The solution found by applying the column-generation procedure will in general be a fractional solution. We now sketch a branching structure in order to find the integer optimum.

### 2.3. Branching Procedure

Ryan and Foster (1981) proposed a general branching rule for set-partitioning problems, where two rows have to be covered by the same column in one branch, and by different columns in the other branch. For our problem, this would mean that two items have to be packed onto the same pallet in one branch, and on different pallets in the other branch. Since it is difficult to force two items to appear on the same pallet, or to appear in different pallets, we use a slightly modified version of the Ryan-Foster branching rule (see also Vanderbeck 1994). We partition the solution space based on *the order* in which items are packed onto a pallet. Two items are called *successors* if they

are packed on the same pallet such that one item lies directly above the other. The branching rule we use is then as follows: in one branch two items $i$ and $j$ are forced to appear as successors on a pallet, and in the other branch items $i$ and $j$ cannot be successors on a pallet. Similar modifications of the Ryan-Foster rule have also been used in the field of airline crew scheduling (see for example Desrosiers et al. 1991 or Vance et al. 1997).

When an optimal, fractional LP-solution has been found, we identify two items $i$ and $j$ for which the sum of all pallets where $i$ and $j$ are successors lies between 0 and 1. In the integer optimum, these two items will either be successors on a pallet, or they will not. So, given two items $i$ and $j$, we branch as follows. In one branch we modify the directed graph $D$ in such a way that items $i$ and $j$ have to be successors. We do this by deleting all arcs$(i, p)$ for $p \neq j$ and all arcs$(p, j)$ for $p \neq i$. Observe that, when solving the pricing problem in case $j$ is a successor of $i$, the value of $d^B(i)$ is no longer relevant since a pallet with item $i$ not followed by item $j$ is not allowed in this branch. Therefore, we record in each node of the tree which items cannot serve as a last item in a pallet, and for these items $j$ we do not consider $d^B(j)$. In the second branch, we make sure that items $i$ and $j$ can never be successors in a solution by deleting arc$(i, j)$ from $D$. In our algorithm we employ this branching step repeatedly to find an integer solution to our problem. Notice that this branching scheme keeps the problem structure intact, which allows us to use column generation throughout the branch-and-bound tree.

Remark. The branch-and-price approach sketched in Sections §2.2 and 2.3 remains valid for so-called partial orders. Consider the problem of decomposing a partial order into a minimum number of chains, such that each chain contains no more than $B$ elements (see Shum and Trotter 1996). We will refer to such a chain as a *B-chain*. We claim that this problem can be tackled using the approach sketched here. First, one easily verifies that the formulation (1)–(3) goes through by substituting the word "$B$-chain" for "stable pallet" in the definition of the $x_k$-variables. Second, the efficient solvability of the pricing problem (Lemma 1) depends on the fact that the digraph contains no directed cycles. This property is preserved when we consider partial orders. Finally, notice that the branching rule also holds in this more general setting, and it follows that the branch-and-price approach remains valid.

## 3. An Algorithm Based on Bounded Clique Width

In this section we propose an enumeration algorithm that is based on a property of some of the instances encountered at BSS. It turns out that, in some instances, many items have the same length. We exploit this property in this section by assuming that the number of different lengths in an instance is bounded by a given parameter $K$. In other words, we assume that in the input of the problem an additional parameter $K$ is present; we refer to this variant of our loading problem as problem $P(K)$.

As a motivating example we first explore the case $K = 2$. We define $n_j$ as the number of items of length $j$, and we assume that $L_1 < L_2$, where $L_j$ is the $j$th length. Furthermore, let $p = n_1 \mod B$ and $q = n_2 \mod B$. Consider now the items with length $L_1$, and let $w_1$ be the width that corresponds to the $p$th smallest item. Then consider the items of length 2, and let $w_2$ be the width that corresponds to the $q$th largest item. Notice that the optimal solution of problem $P(2)$ has value $\lceil n/B \rceil$ or $\lceil n/B \rceil + 1$ since $\lceil n_1/B \rceil + \lceil n_2/B \rceil \leq \lceil n/B \rceil + 1$. In fact, we characterize below in Proposition 1 when instances of problem $P(2)$ have value $\lceil n/B \rceil$ or $\lceil n/B \rceil + 1$.

Proposition 1. *The optimal solution of problem $P(2)$ has value $\lceil n/B \rceil$ if and only if either $\lceil n/B \rceil = \lceil n_1/B \rceil + \lceil n_2/B \rceil$ or $w_1 \leq w_2$.*

We now consider problem $P(K)$ in case of a fixed value of $K$. We assume that the lengths are ordered such that $L_1 < L_2 < \cdots < L_K$. In Section 3.1 we focus on the concept of (bounded) clique width. Section 3.2 describes an exact algorithm for problem $P(K)$.

### 3.1. Clique Width

A property of graphs that has received wide attention recently is clique width. This property was first introduced by Courcelle et al. (1993); a related concept called NLC-width has been introduced by Wanke (1994). The reason for this attention is the fact that important graph-theoretic problems (like maximum clique or independent set) can be solved in polynomial time for graphs with bounded clique width.

Informally, the notion of clique width of a graph $G$ can be described using the following operations (see Courcelle and Olariu 2001 or Brandstädt et al. 2004 for formal definitions):

— creation of a vertex labeled with some integer $i$ (the vertex is said to have label $i$); we refer to this as Operation 1;

— disjoint union of two vertex-labeled graphs (given $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, let $G = (V_1 \cup V_2, E_1 \cup E_2)$); we refer to this as Operation 2;

— adding an edge between each vertex with label $i$ and each vertex with label $j$, $i \neq j$; we refer to this as Operation 3;

— relabel each vertex with label $i$ by label $j$; we refer to this as Operation 4.

The minimum number of labels needed to construct $G$ using these operations is the clique width

of $G$. Notice that permutation graphs in general have unbounded clique-width (Brandstädt and Lozin 2003). However, in case of $P(K)$ we have the following:

LEMMA 2. *A graph associated to an instance of $P(K)$ has clique width at most $K+1$.*

PROOF. We prove the lemma by exhibiting a sequence of operations. First, we order the vertices according to the width of the associated item in decreasing order. In case of a tie, the vertex with the highest length goes first.

For each vertex $i = 1, \ldots, n$, letting the vertex correspond to an item with length $L_j$, $1 \le j \le K$, we perform the following operations:

— create vertex $i$ and label it $K+1$, using Operation 1;

— add vertex $i$ to the graph, using Operation 2, i.e., $G := (V \cup \{i\}, E)$;

— connect the vertex with label $K+1$ to all vertices with label $j, j+1, \ldots, K$, using Operation 3, repeatedly;

— relabel the vertex with label $K+1$ by label $j$ using Operation 4.

Observe that this construction guarantees that each vertex that corresponds to an item with length $L_j$ is connected to all vertices that correspond to items that have length $L_j$ or larger. Thus, the resulting permutation graph corresponds to an instance of $P(K)$. □

REMARK. It is easy to verify that the graphs corresponding to instances of $P(K)$ do not have bounded tree width.

We can now state the following theorem:

THEOREM 1. *Problem $P(K)$ is solvable in polynomial time.*

PROOF. This result follows from Lemma 2 above and Theorem 2 in Espelage et al. (2001), which states that the problem of partitioning a graph into cliques of bounded size is solvable in polynomial time for graphs with bounded clique width. □

Notice, however, that the approach in Espelage et al. (2001) is quite general, and does not lead to a ready-to-use algorithm. We propose such an algorithm in the next section.

### 3.2. An Algorithm for $P(K)$

We describe an exact algorithm for problem $P(K)$ that, given $B$ and $K$, runs in polynomial time. We now state some preliminaries.

DEFINITION 1. A pallet is called *pure* when it contains $B$ items of the same length; otherwise a pallet is called *mixed*.

Notice that a pallet with fewer than $B$ items of the same length is called a *mixed* pallet.

DEFINITION 2. The length of an item $i$ is denoted by $l_i$, and its width by $w_i$.

PROPERTY 1. A solution of problem $P(K)$ is said to have Property 1 if it contains no more than $2^K$ mixed pallets.

PROPERTY 2. A solution of problem $P(K)$ is said to have Property 2 if no item $r$ in a mixed pallet can be replaced by an item $s$ from a pure pallet, with $l_s = l_r$ and $w_s < w_r$.

DEFINITION 3. We call a solution to problem $P(K)$ *minimal* if it simultaneously satisfies Properties 1 and 2.

LEMMA 3. *There exists an optimal solution to problem $P(K)$ that is minimal.*

PROOF. Consider some optimal solution to problem $P(K)$. By interchanging and transferring items, we show that there is an optimal solution that is minimal. If there exists an item $r$ occurring in a mixed pallet that can be replaced by an item $s$ from a pure pallet with $l_s = l_r$ and $w_s < w_r$, we interchange these items so that Property 2 is satisfied. To see that there exists an optimal solution that satisfies Property 1, observe that an upper bound on the maximum number of mixed pallets with different length sets is equal to $2^K$. Therefore, if we have found a solution containing more than $2^K$ mixed pallets, there exist at least two pallets with identical length sets. We now show that, by interchanging some items between these pallets, we can alter the solution such that no pallets with identical length sets are present in the solution.

If, in an optimal solution, the number of mixed pallets that contain items of one single length exceeds $K$, we can transfer items in a straightforward way, and reduce to $K$ the number of mixed pallets that have items of a same length.

If, in an optimal solution, the number of mixed pallets that contain items of at least two different lengths exceeds $2^K - K$, we can reduce the number of mixed pallets that have items of at least two different lengths by transferring items. For this, we first define

$p^A_{L_i}$: the smallest width of an item of length $L_i$ from pallet $A$;

$q^A_{L_i}$: the largest width of an item of length $L_i$ from pallet $A$.

Observe that, when we discard the size requirement of a pallet, all items of length $L_i$ can be transferred from a pallet $A$ to a pallet $C$ if the following two conditions hold:

$$q^A_{L_i} \le p^C_{L_{i+1}} \tag{5}$$

$$p^A_{L_i} \ge q^C_{L_{i-1}} \tag{6}$$

Now, consider an optimal solution that contains more than $2^K - K$ mixed pallets with items of at least two different lengths. Then there exist two pallets $A$ and $C$ with identical length sets, say $L_1, L_2, \ldots, L_m$ $(m \ge 2)$. We claim that there exist two lengths $L_i$ and $L_j$ such

that either all items of $L_i$ can be transferred from pallet $A$ to pallet $C$, or all items of $L_j$ can be transferred from $C$ to $A$. This implies that we can construct an alternative optimal solution by interchanging items between $A$ and $C$ such that these pallets no longer have identical length sets.

Without loss of generality we assume that

$$p^A_{L_m} \geq q^C_{L_{m-1}}.$$

(If this would not be the case, we have $p^A_{L_m} < q^C_{L_{m-1}}$; we know, by feasibility of pallets $A$ and $C$, that $q^A_{L_{m-1}} \leq p^A_{L_m}$ and $q^C_{L_{m-1}} \leq p^C_{L_m}$, and it follows from this that $p^C_{L_m} \geq q^A_{L_{m-1}}$, and we can simply change the order of the two pallets to arrive at our assumption that $p^A_{L_m} \geq q^C_{L_{m-1}}$.)

Since $p^A_{L_m} \geq q^C_{L_{m-1}}$, the items of length $L_m$ from pallet $A$ can be transferred to pallet $C$. Now, we have to find a length such that items from pallet $C$ can be transferred to pallet $A$. In order to do so, we have to find a length for which conditions (5) and (6) hold. Assume that we cannot find such a length; we then show that we will ultimately arrive at a contradiction, proving that such a length does exist.

Claim 1. *If items of length $L_1, \ldots, L_j$ cannot be transferred from $C$ to $A$, it follows that $q^C_{L_j} > p^A_{L_{j+1}}$, $j = 1, \ldots, m-1$.*

Proof. we use induction to prove this claim. Consider the case $j = 1$. We can transfer the items of $L_1$ from pallet $C$ to pallet $A$ if $q^C_{L_1} \leq p^A_{L_2}$. Since the items of $L_1$ are the smallest items, condition (6) does not apply, since there is no length smaller than $L_1$. We assumed that we could not transfer items from pallet $C$ to pallet $A$, so it must hold that $q^C_{L_1} > p^A_{L_2}$. Next, suppose the claim is true for $j = l-1$; is it then true for $j = l$? Since we are not able to transfer the items of $L_l$ from $C$ to $A$, at least one of the inequalities $q^C_{L_l} \leq p^A_{L_{l+1}}$ and $p^C_{L_l} \geq q^A_{L_{l-1}}$ must be violated. But we know by induction that $q^C_{L_{l-1}} > p^A_{L_l}$, which, together with $p^C_{L_l} \geq q^C_{L_{l-1}}$ and $p^A_{L_l} \geq q^A_{L_{l-1}}$, implies $p^C_{L_l} \geq q^A_{L_{l-1}}$. Hence, it follows that $q^A_{L_l} > p^A_{L_{l+1}}$, proving the claim. □

Lemma 3 implies that there exists an optimal solution such that for each $j = 1, \ldots, K$ the number of items of length $L_j$ present in mixed pallets (denoted by $s_j$) equals

$$s_j = n_j - \alpha_j B, \quad \text{for some } \alpha_j \in \left\{0, 1, \ldots, \left\lceil \frac{n_j}{B} \right\rceil \right\},$$

and satisfies $\sum_{j=1}^K s_j \leq B2^K$. Now, given a set of possible $s_j$ values with $\sum_{j=1}^K s_j \leq B2^K$, we enumerate all possible minimal solutions. We do this using the concept of a *partial* solution.

Definition 4. A *partial* solution is a family of $2^K$ sets of items such that each set corresponds to a stable pallet and such that each item occurs at most once in the family.

To each partial solution we associate a length. That is, the minimum length $L_j$ for which fewer than $s_j$ items are present in the current partial solution. Furthermore, we associate a *minimal item* to each stable pallet with fewer than $B$ items in the partial solution. This minimal item is the item of minimum length that has minimal width, and can be feasibly added to that pallet.

We now give an algorithm that finds an optimal minimal solution to problem $P(K)$, assuming that a set of $s_j$ values, satisfying $\sum_{j=1}^K s_j \leq B2^K$, is given. First, we deal exclusively with constructing the mixed pallets. For this, we start with a partial solution that has $2^K$ empty pallets, and we gradually fill—in many different ways—these pallets.

**Algorithm ENUM:**

*Step* 1. Start with the initial partial solution that consists of $2^K$ empty pallets. We associate length $L_1$ to this solution (assuming $s_1 > 0$), and set as minimal item for each pallet the smallest item of $L_1$. Go to Step 2.

*Step* 2. Generate (at most) $2^K$ new partial solutions by adding for each pallet in the old partial solution its minimal item. Notice that we get $2^K$ new partial solutions, since there are $2^K$ pallets in the old partial solution. Go to Step 3.

*Step* 3. Associate to each partial solution the new minimum length $L_j$ for which fewer than $s_j$ items are present, and associate to each pallet in all partial solutions its new minimal item. If $\sum_{j=1}^K s_j$ items are present in the new partial solution, go to Step 4. Otherwise, go to Step 2.

*Step* 4. For each final partial solution, i.e., for each partial solution where $\sum_{j=1}^K s_j$ items are assigned, verify whether each pallet in the solution is a mixed pallet. If not, we simply discard the solution. Else, go to Step 5.

*Step* 5. Complete each final partial solution to a feasible solution by adding the remaining items in pure pallets in a straightforward way. Output the solution that contains the smallest number of pallets. Stop.

By associating a node to each partial solution and connecting two nodes if one partial solution is constructed by adding a single minimal item to the other, a tree results. We refer to this as the *tree of partial solutions*.

Lemma 4. *A solution produced by algorithm ENUM is minimal.*

Proof. We verify whether a solution found by ENUM satisfies Properties 1 and 2. Obviously, it satisfies Property 1. Now suppose that the solution found does not satisfy Property 2; that is, there exists at least one item $r$ that is present in a mixed pallet, that could be interchanged with an item $s$ satisfying $l_s = l_r$ and $w_s < w_r$. Let $r$ be the smallest interchangeable item and consider the step in the algorithm when

we added item $r$ to a pallet. Apparently, we could have added item $s$ at that time. But that implies that item $r$ was not a minimal item for that pallet. Hence, such a solution cannot have been generated by the algorithm. □

Lemma 5. *Any minimal solution is generated by algorithm ENUM.*

Proof. Consider a minimal solution that is not generated by ENUM. Remove from this solution all pure pallets, so that a final partial solution $S$ remains. So each final partial solution generated by ENUM differs from $S$. Consider the tree of partial solutions. Let us find a set of paths in this tree: starting with the initial solution, follow a branch to a next partial solution if it puts an item in a pallet if in $S$ the same item is in the same pallet. Notice that no path makes it until the end (since $S$ was not generated by ENUM). So let us consider a partial solution that has no outgoing branches and that is not final. To this partial solution a length is associated, say the current length.

Consider now the minimal item of the current length of that partial solution that is used in $S$, and that has not been considered when we followed branches. Say that this is item $d$ and that it is in pallet $j$ in solution $S$. This pallet $j$ has another item, say item $c$, serving as minimal item when we look at the branch from our current partial solution to the partial solution where pallet $j$ receives an item (if $c = d$, we would have followed that branch). Thus, $c \prec d$. Now, since $S$ is minimal it must use item $c$ somewhere else (if $S$ did not use $c$ at all, we could replace $d$ by $c$ in $S$, contradicting the minimality of $S$ (Property 2)). Say item $c$ is used in pallet $j'$ ($j' \neq j$). If we look at the branch from our current partial solution to the partial solution that gives $j'$ another item, we know there is a minimal item that cannot be item $c$ (otherwise, we would have followed that branch). Thus, there is another item present in that partial solution, say item $b$, $b \prec c$. Again, $b$ must be somewhere in $S$, say in pallet $j''$. Notice that $j'' \neq j'$ (for obvious reasons) but also $j'' \neq j$ (since $c$ is minimal for $j$ and $b \prec c$). Let us look at the pallet $j''$ and its minimal item given our current partial solution. It cannot be $b$ (else we would have followed this branch), so it must be less than $b$, say $a$. Thus, $a$ must be in $S$ (otherwise, we can interchange contradicting the minimality of $S$), say in $j'''$. Again, this pallet $j'''$ is different from the previously considered pallets $j''$, $j'$, and $j$ (otherwise, each of the pallets wouldn't have the minimal item they have). Continuing in this way, it leads to the conclusion that $S$ has more than $2^K$ pallets, contradicting Property 1; hence $S$ is not minimal. □

Theorem 2. *The running time of algorithm ENUM is bounded by* $(2^K)^{B2^K} n^K$.

Proof. The complexity of ENUM depends on the number of solutions generated. This number depends on the number of items that are present in mixed pallets. Property 1 implies that

$$\sum_{j=1}^{K} s_j \leq B2^K$$

Hence, ENUM cannot generate more than $(2^K)^{B2^K}$ different solutions. Furthermore, ENUM has to be executed for each possible set of $s_j$ values. Observe that for each $s_j$ there are $O(n_j/B)$ possible values, $j = 1, \ldots, K$, leading to $O(n^K)$ possible sets of $s_j$ values for a fixed $B$. The result follows. Notice that, for a fixed $B$ and a fixed $K$, this is a polynomial-time algorithm. □

## 4. Computational Experiments

In this section we discuss some issues concerning the implementation of the algorithms described in this paper, and we show the computational results.

### 4.1. Implementation Issues

Both algorithms described in this paper are implemented on a 733 MHz computer with 128 Mb of physical memory. The algorithms are coded in C++, and in the branch-and-price algorithm, we use LINDO to solve the restricted master problems.

We use three data sets for the computational experiments. The first data set contains 50 real-world instances where the length and the width of each item was provided to us by BSS; the second data set contains 50 randomly generated instances where the length and the width of each item is uniformly distributed between 0 and 3,000. The third data set also contains 50 randomly generated instances, but in this data set all instances have small clique width. More specifically, the number of different lengths is uniformly distributed between 5 and 15, whereas the width of the items is uniformly distributed between 0 and 3,000. In all three data sets the number of items ranges from 0 to 200 (see Table 1). The number of items for an instance of the second and third data set follows a uniform distribution between 0 and 200 items per instance. We use different values for $B$, ranging from 3 to 15. In the real-world setting from BSS, $B = 12$.

In the branch-and-price algorithm, we use a heuristic to find a good starting solution, before starting the actual branch-and-price procedure. This starting solution is computed in a very straightforward way: all items are ordered, first according to their length (increasing) and second according to their width (also increasing). We start with the first item and put it in a pallet. Then we simply go down the list, and if an item can be added to the current pallet, we add it,

**Table 1 Characteristics of the Data Sets**

| No. of items | No. of instances (data set 1) | No. of instances (data set 2) | No. of instances (data set 3) |
|---|---|---|---|
| 0–40 | 10 | 9 | 4 |
| 40–80 | 15 | 13 | 13 |
| 80–120 | 10 | 7 | 15 |
| 120–160 | 7 | 12 | 8 |
| 160–200 | 8 | 9 | 10 |

and otherwise we continue with the next item. If a pallet contains $B$ items, or if we are at the end of the list, we start a new pallet with the first available item and start this procedure over. To determine whether a solution generated by this heuristic is optimal, we use the lower bound $\lceil n/B \rceil$.

In the pricing problem, when trying to find new variables with negative reduced costs, we add one variable in each iteration of the longest-path procedure. This is the variable with reduced costs that are the most negative.

In the enumeration algorithm, we first compute a lower and an upper bound. The lower bound equals $\lceil n/B \rceil$, and the upper bound is equal to $\lceil n_1/B \rceil + \cdots + \lceil n_K/B \rceil$. If these bounds coincide, there exists an optimal solution with value $\lceil n/B \rceil$, and we do not need to run ENUM to find a solution.

Apart from computing the LP-relaxation and $\lceil n/B \rceil$, we compute a third lower bound, AC. AC stands for the size of a maximum antichain. In other words, AC is the optimal value of the loading problem if there is no restriction on $B$ (i.e., $B = n$).

### 4.2. Results

For data set 1, the value of $K$ ranges from 2 to 9, and in most instances (approximately 85%) $K$ equals 2, 3, or 4. For the second data set, however, the value of $K$ is very close to $n$. Thus, the clique width of the instances of data set 1 is small; this is not guaranteed for the instances of data set 2. Since the running time of the enumeration algorithm is exponential in $K$, the instances from data set 2 are very hard to solve for ENUM. In fact, none of the instances could be solved by ENUM in less than one hour of computation time, so for ENUM we present only the results of the first and third data set.

The results for the three data sets are shown in Tables 2, 3, and 4. In the first two columns we give the value of $B$ and a range for the number of items. The following three columns give the values of three lower bounds, namely $\lceil n/B \rceil$, the size of a maximum antichain, and the value of the LP-relaxation. The number between brackets is the gap (%) between the lower bound and the integer optimum, i.e., (OPT − LB)/OPT. The column labelled "OPT" denotes the value of the optimal integer solution. In the last columns we give the number of branching nodes visited in the search tree (for ENUM, this is the number of nodes in the tree of partial orders), and the computation time in seconds. For Tables 2 and 4, these last two values are given both for the branch-and-price algorithm as well as for ENUM; Table 3 shows only the results of the branch-and-price algorithm. Notice that all values are average values over all test instances in the specific range.

In the tables with the results we see that, in a number of cases, the number of branching nodes is equal to 0. For the branch-and-price algorithm, this means that the solution found by the heuristic equals $\lceil n/B \rceil$ (this happened 217 out of 250 times in Table 2, 12 out of 250 times in Table 3, and 45 out of 250 times in Table 4). For the ENUM algorithm it means that the lower and upper bounds computed at the start of the algorithm are the same, which means that there exists an optimal solution with value $\lceil n/B \rceil$ (this happened 90 out of 250 times in Table 2, and never in Table 4).

From Table 2 we conclude that the instances from data set 1 can be solved to optimality very quickly by both algorithms; 90% of the instances are solved in less than one second for the branch-and-price algorithm, and for ENUM, 99% of all instances are solved in less than one second. One reason for the good performance of the branch-and-price algorithm is that in 86.8% of the instances, the heuristic for finding an initial solution in the branch-and-price algorithm provides us with an optimal solution that equals $\lceil n/B \rceil$. Indeed, the quality of the lower bound $\lceil n/B \rceil$ for data set 1 is striking. Another reason for the success of the branch-and-price algorithm is that the matrices are very sparse (for example, for $B = 15$, each column has fewer than 15 nonzeros). For the ENUM algorithm, an optimal solution is found without having to branch in 36.0% of the instances. Thus, in most cases ENUM has to be executed, and then it finds an optimal solution very quickly, i.e., usually faster than the branch-and-price algorithm. As described before, from the results it is also clear that both $\lceil n/B \rceil$ and LP are good lower bounds for the integer optimum; the value of AC, however, is in many cases far from the optimum. The small gap between the LP and IP solutions is not surprising, since this is the case for set-partitioning models in general (Byun 2001).

When we look at the results from the random instances in Table 3, we see that the computation times of the branch-and-price algorithm are slower than those from the real-world instances. Furthermore, the lower bound from the LP relaxation and the value of AC are very close to the integer optimum; for large $B$ ($B \geq 9$) they even coincide. Not surprisingly, the lower bound $\lceil n/B \rceil$ performs much worse here, especially for large $B$. The heuristic for finding an initial solution performs much worse compared to the results from

**Table 2 Results for Real-World Instances**

| | | | | | | B&P | | ENUM | |
|---|---|---|---|---|---|---|---|---|---|
| $B$ | $n$ | $\lceil n/B \rceil$ (Gap) | AC (Gap) | LP (Gap) | OPT | BN | Time | BN | Time |
| 3 | ≤40 | 11.90 (0.00) | 2.50 (78.99) | 11.47 (3.61) | 11.90 | 0.00 | 0.00 | 3.70 | 0.00 |
| | ≤80 | 19.13 (0.00) | 3.33 (82.59) | 18.91 (1.15) | 19.13 | 29.07 | 0.69 | 33.93 | 2.52 |
| | ≤120 | 36.90 (0.00) | 2.90 (92.14) | 36.40 (1.36) | 36.90 | 0.00 | 0.01 | 2.90 | 0.01 |
| | ≤160 | 45.86 (0.00) | 3.00 (93.46) | 45.43 (0.94) | 45.86 | 0.00 | 0.02 | 3.14 | 2.04 |
| | ≤200 | 60.50 (0.00) | 2.38 (96.07) | 60.08 (0.69) | 60.50 | 0.00 | 0.05 | 0.63 | 0.00 |
| 6 | ≤40 | 6.20 (0.00) | 2.50 (59.68) | 5.72 (7.74) | 6.20 | 0.00 | 0.00 | 7.40 | 0.00 |
| | ≤80 | 9.73 (0.71) | 3.33 (66.02) | 9.48 (3.27) | 9.80 | 0.07 | 0.03 | 23.87 | 0.00 |
| | ≤120 | 18.70 (0.00) | 2.90 (84.49) | 18.20 (2.67) | 18.70 | 0.00 | 0.01 | 11.80 | 0.00 |
| | ≤160 | 23.29 (0.00) | 3.00 (87.12) | 22.71 (2.49) | 23.29 | 0.00 | 0.02 | 4.29 | 0.01 |
| | ≤200 | 30.38 (0.00) | 2.38 (92.17) | 30.04 (1.12) | 30.38 | 0.00 | 0.05 | 4.63 | 0.00 |
| 9 | ≤40 | 4.40 (0.00) | 2.50 (43.18) | 3.97 (9.77) | 4.40 | 0.10 | 0.02 | 11.00 | 0.00 |
| | ≤80 | 6.67 (0.00) | 3.33 (50.07) | 6.35 (4.80) | 6.67 | 10.08 | 4.87 | 24.27 | 0.00 |
| | ≤120 | 12.80 (0.00) | 2.90 (77.34) | 12.18 (4.84) | 12.80 | 5.40 | 0.23 | 5.80 | 0.00 |
| | ≤160 | 15.57 (0.00) | 3.00 (80.73) | 15.16 (2.63) | 15.57 | 1.43 | 2.26 | 20.57 | 0.00 |
| | ≤200 | 20.38 (0.00) | 2.38 (88.32) | 20.04 (1.67) | 20.38 | 7.75 | 0.65 | 11.00 | 0.00 |
| 12 | ≤40 | 3.50 (5.41) | 2.50 (32.43) | 3.23 (12.70) | 3.70 | 4.80 | 0.56 | 14.00 | 0.00 |
| | ≤80 | 5.13 (2.66) | 3.33 (36.81) | 4.82 (8.54) | 5.27 | 9.00 | 6.64 | 25.94 | 0.00 |
| | ≤120 | 9.60 (0.00) | 2.90 (69.79) | 9.10 (5.21) | 9.60 | 0.00 | 0.01 | 25.90 | 0.00 |
| | ≤160 | 11.86 (0.00) | 3.00 (74.70) | 11.48 (3.20) | 11.86 | 7.57 | 0.96 | 31.43 | 0.00 |
| | ≤200 | 15.50 (0.00) | 2.38 (84.65) | 15.02 (3.10) | 15.50 | 8.75 | 2.60 | 51.88 | 0.01 |
| 15 | ≤40 | 2.70 (12.90) | 2.50 (19.35) | 2.81 (9.35) | 3.10 | 8.10 | 5.72 | 22.10 | 0.00 |
| | ≤80 | 4.33 (1.59) | 3.33 (24.32) | 3.96 (10.00) | 4.40 | 0.07 | 1.35 | 29.07 | 0.00 |
| | ≤120 | 7.60 (0.00) | 2.90 (61.84) | 7.29 (4.08) | 7.60 | 4.90 | 5.13 | 28.00 | 0.00 |
| | ≤160 | 9.43 (0.00) | 3.00 (68.19) | 9.10 (3.50) | 9.43 | 10.71 | 6.89 | 61.14 | 0.00 |
| | ≤200 | 12.63 (2.55) | 2.38 (81.64) | 12.19 (5.94) | 12.96 | 15.50 | 7.78 | 55.88 | 0.01 |

**Table 3 Results for Random Instances**

| $B$ | $n$ | $\lceil n/B \rceil$ (Gap) | AC (Gap) | LP (Gap) | OPT | BN | Time |
|---|---|---|---|---|---|---|---|
| 3 | ≤40 | 9.89 (5.27) | 8.67 (16.95) | 10.11 (3.16) | 10.44 | 15.11 | 0.10 |
| | ≤80 | 18.77 (0.42) | 11.69 (37.98) | 18.54 (1.64) | 18.85 | 46.08 | 7.23 |
| | ≤120 | 33.57 (0.00) | 16.14 (51.92) | 33.38 (0.57) | 33.57 | 101.14 | 12.92 |
| | ≤160 | 47.92 (0.00) | 20.83 (56.53) | 47.47 (0.94) | 47.92 | 165.75 | 41.43 |
| | ≤200 | 61.11 (0.00) | 23.22 (62.00) | 60.85 (0.43) | 61.11 | 305.56 | 120.10 |
| 6 | ≤40 | 5.22 (39.79) | 8.67 (0.00) | 8.67 (0.00) | 8.67 | 1.00 | 0.02 |
| | ≤80 | 9.62 (18.27) | 11.69 (0.68) | 11.72 (0.42) | 11.77 | 12.31 | 4.50 |
| | ≤120 | 17.00 (1.68) | 16.14 (6.65) | 17.10 (1.10) | 17.29 | 69.14 | 82.53 |
| | ≤160 | 24.17 (0.33) | 20.83 (14.10) | 23.90 (1.44) | 24.25 | 51.83 | 65.36 |
| | ≤200 | 30.67 (0.00) | 23.22 (24.29) | 30.50 (0.55) | 30.67 | 62.67 | 112.78 |
| 9 | ≤40 | 3.78 (56.40) | 8.67 (0.00) | 8.67 (0.00) | 8.67 | 1.00 | 0.02 |
| | ≤80 | 6.62 (43.37) | 11.69 (0.00) | 11.69 (0.00) | 11.69 | 1.00 | 0.14 |
| | ≤120 | 11.57 (28.31) | 16.14 (0.00) | 16.14 (0.00) | 16.14 | 9.57 | 9.25 |
| | ≤160 | 16.33 (23.76) | 20.83 (2.75) | 21.42 (0.00) | 21.42 | 10.67 | 25.40 |
| | ≤200 | 20.67 (10.98) | 23.22 (10.69) | 26.00 (0.00) | 26.00 | 15.22 | 97.62 |
| 12 | ≤40 | 2.78 (67.94) | 8.67 (0.00) | 8.67 (0.00) | 8.67 | 1.00 | 0.03 |
| | ≤80 | 5.08 (56.54) | 11.69 (0.00) | 11.69 (0.00) | 11.69 | 1.00 | 0.37 |
| | ≤120 | 8.71 (46.03) | 16.14 (0.00) | 16.14 (0.00) | 16.14 | 1.00 | 3.68 |
| | ≤160 | 12.42 (42.02) | 20.83 (0.00) | 20.83 (0.00) | 20.83 | 13.17 | 43.54 |
| | ≤200 | 15.56 (32.99) | 23.22 (0.00) | 23.22 (0.00) | 23.22 | 1.00 | 78.54 |
| 15 | ≤40 | 2.44 (71.86) | 8.67 (0.00) | 8.67 (0.00) | 8.67 | 1.00 | 0.03 |
| | ≤80 | 4.23 (63.82) | 11.69 (0.00) | 11.69 (0.00) | 11.69 | 1.00 | 0.37 |
| | ≤120 | 7.29 (54.83) | 16.14 (0.00) | 16.14 (0.00) | 16.14 | 1.00 | 3.98 |
| | ≤160 | 10.00 (53.31) | 20.83 (0.00) | 20.83 (0.00) | 20.83 | 1.00 | 16.20 |
| | ≤200 | 12.67 (45.43) | 23.22 (0.00) | 23.22 (0.00) | 23.22 | 1.00 | 85.93 |

**Table 4 Results for Random Instances with Small Clique Width**

| | | | | | | B&P | | ENUM | |
|---|---|---|---|---|---|---|---|---|---|
| $B$ | $n$ | $\lceil n/B \rceil$ (Gap) | AC (Gap) | LP (Gap) | OPT | BN | Time | BN | Time |
| 3 | ≤40 | 10.25 (0.00) | 6.00 (36.31) | 9.67 (6.16) | 10.25 | 5.50 | 0.02 | 12.75 | 0.00 |
| | ≤80 | 21.69 (0.00) | 7.75 (63.71) | 21.36 (1.58) | 21.69 | 65.00 | 2.18 | 15.00 | 0.12 |
| | ≤120 | 34.33 (0.00) | 8.93 (73.56) | 34.13 (0.56) | 34.33 | 118.87 | 14.06 | 19.47 | 1.93 |
| | ≤160 | 44.75 (0.00) | 10.63 (76.14) | 44.46 (0.66) | 44.75 | 107.38 | 17.47 | 18.13 | 0.06 |
| | ≤200 | 62.90 (0.00) | 9.50 (84.76) | 62.57 (0.52) | 62.90 | 93.10 | 77.80 | 19.90 | 8.83 |
| 6 | ≤40 | 5.00 (20.00) | 6.00 (0.00) | 6.00 (0.00) | 6.00 | 4.75 | 0.33 | 27.00 | 0.00 |
| | ≤80 | 11.15 (1.65) | 7.77 (30.82) | 10.72 (5.65) | 11.38 | –(10) | — | 45.85 | 0.02 |
| | ≤120 | 17.47 (0.32) | 8.93 (48.27) | 17.07 (2.63) | 17.53 | –(13) | — | 156.60 | 16.50 |
| | ≤160 | 22.63 (0.00) | 10.63 (52.82) | 22.23 (1.78) | 22.63 | –(5) | — | 52.75 | 0.20 |
| | ≤200 | 31.80 (0.00) | 9.50 (69.85) | 31.28 (1.60) | 31.80 | –(4) | — | 43.10 | 0.93 |
| 9 | ≤40 | 3.75 (39.29) | 6.00 (0.00) | 6.00 (0.00) | 6.00 | 1.00 | 0.03 | 29.00 | 0.00 |
| | ≤80 | 7.54 (8.45) | 7.77 (6.41) | 8.25 (0.74) | 8.31 | 42.31 | 47.07 | 49.85 | 0.00 |
| | ≤120 | 11.93 (1.56) | 8.93 (25.47) | 11.63 (3.93) | 12.13 | –(13) | — | 66.87 | 0.01 |
| | ≤160 | 15.25 (0.83) | 10.63 (30.62) | 14.83 (3.49) | 15.38 | –(3) | — | 67.13 | 0.07 |
| | ≤200 | 21.50 (0.00) | 9.50 (55.49) | 20.86 (2.99) | 21.50 | –(1) | — | 58.10 | 0.06 |
| 12 | ≤40 | 3.00 (51.43) | 6.00 (0.00) | 6.00 (0.00) | 6.00 | 1.00 | 0.04 | 29.00 | 0.00 |
| | ≤80 | 5.85 (26.41) | 7.77 (4.12) | 7.86 (2.93) | 8.08 | –(11) | — | 57.00 | 0.00 |
| | ≤120 | 8.93 (11.83) | 8.93 (13.88) | 9.99 (2.83) | 10.27 | –(10) | — | 77.20 | 0.00 |
| | ≤160 | 11.75 (5.80) | 10.63 (14.81) | 11.68 (6.50) | 12.50 | –(3) | — | 118.25 | 0.12 |
| | ≤200 | 16.20 (1.11) | 9.50 (41.99) | 15.67 (4.36) | 16.40 | –(1) | — | 168.80 | 16.10 |
| 15 | ≤40 | 2.50 (58.57) | 6.00 (0.00) | 6.00 (0.00) | 6.00 | 1.00 | 0.04 | 29.00 | 0.00 |
| | ≤80 | 4.69 (38.20) | 7.77 (1.10) | 7.77 (1.02) | 7.85 | –(11) | — | 59.15 | 0.00 |
| | ≤120 | 7.27 (22.95) | 8.93 (9.48) | 9.35 (4.31) | 9.73 | –(9) | — | 82.53 | 0.00 |
| | ≤160 | 9.38 (16.10) | 10.63 (5.62) | 10.97 (2.69) | 11.25 | –(5) | — | 117.88 | 0.02 |
| | ≤200 | 13.10 (3.48) | 9.50 (30.56) | 12.93 (4.85) | 13.60 | –(0) | — | 100.90 | 0.01 |

the first data set: for only 4.8% of the instances does the heuristic find an optimal solution equal to $\lceil n/B \rceil$.

When we look at Table 4, we see that the branch-and-price algorithm is no longer capable of solving the larger instances to optimality in a reasonable amount of time (we use a time limit of one hour to solve a single problem instance). If a number of instances in a specific range could not be solved within this time limit, this is denoted by "$-^{(\alpha)}$" where $\alpha$ denotes the number of instances that could be solved by the branch-and-price algorithm. So we present only these figures from these groups of instances that could all be solved optimally. We see that, for the smaller instances, the branch-and-price algorithm is still very fast, but as the number of items increases, the number of instances that cannot be solved also increases. This is in sharp contrast to the performance of ENUM. From these results we see that ENUM is able to solve all instances, and 95.6% of all instances are solved within one second of computation time. Also for this data set, the value of the LP relaxation is a good lower bound for the integer optimum.

## 5. Conclusion

In this paper we described two exact algorithms for a pallet-loading problem. The first algorithm is a branch-and-price algorithm, based on an integer-programming formulation. The pricing problem can be formulated as a longest-path problem and can be solved efficiently by dynamic programming. The second algorithm is an enumeration algorithm based on the concept of bounded clique width. This algorithm was motivated by a special structure that is present in the real-world instances that were used for computational experiments. From the computational results we conclude that the problem instances from data sets 1 and 2 can be solved satisfactorily by the branch-and-price algorithm, while approximately 30% of the instances from data set 3 cannot be solved after one hour of computation time. The enumeration algorithm performs really well in case of the real-world instances (99% of the instances are solved within a second). Also, the instances from data set 3 can be solved efficiently by ENUM, due to the small clique width of these problem instances (95% of these instances are solved in less than one second), but the random instances cannot be solved efficiently, due to the large number of different lengths in the input. From the results we also see that the LP relaxation provides us with a good lower bound on the integer optimum.

### Acknowledgments

This research was partially supported by EU Thematic Network APPOL II, IST-2001-32007. We would like to thank

Eric Stinges from Bruynzeel Storage Systems for providing us with the data for the test instances.

## References

Baker, B. S., E. G. Coffman, Jr. 1996. Mutual exclusion scheduling. *Theoret. Comput. Sci.* **162** 225–245.

Barnhart, C., E. D. Johnson, G. L. Nemhauser, M. W. P. Salvelsbergh, P. H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* **46** 316–329.

Bischoff, E. E. 1991. Stability aspects of pallet loading. *OR Spektrum* **13** 189–197.

Boudhar, M. 2003. Scheduling a batch processing machine with bipartite compatibility graphs. *Math. Methods Oper. Res.* **57** 513–527.

Brandstädt, A., V. V. Lozin. 2003. On the linear structure and clique-width of bipartite permutation graphs. *Ars Combinatoria* **LXVII** 273–281.

Brandstädt, A., F. F. Dragan, H.-O. Le, R. Mosca. 2005. New graph classes of bounded clique-width. *Theory Comput. Systems*. **38** 623–645.

Byun, C. 2001. Lower bounds for large-scale set partitioning problems. ZIB-Report 01-06, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany.

Courcelle, B., S. Olariu. 2001. Upper bounds to the clique-width of graphs. *Discrete Appl. Math.* **101** 77–114.

Courcelle, B., J. Engelfriet, G. Rozenberg. 1993. Handle-rewriting hypergraph grammars. *J. Comput. System Sci.* **46** 218–270.

Desrosiers, J., Y. Dumas, M. Desrochers, F. Soumis, B. Sanso, P. Trudeau. 1991. A breakthrough in airline crew scheduling. Cahiers du GERAD G-91-11. Proc. 26th Annual Meeting of the Canadian Transportation Res. Forum, Montreal, Québec, Canada, 464–478.

Dilworth, R. P. 1950. A decomposition theorem for partially ordered sets. *Ann. Math.* **51** 161–166.

Dyckhoff, H. 1990. A typology of cutting and packing problems. *Eur. J. Oper. Res.* **44** 145–159.

Espelage, W., F. Gurski, E. Wanke. 2001. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. *Proc. 27th Workshop on Graph-Theoretic Concepts in Comput. Sci.* (*WG 2001*), *Lecture Notes in Comput. Sci.*, Vol. 2204. Springer Verlag, Heidelberg, Germany, 117–128.

Felsner, S., L. Wernisch. 1998. Maximum $k$-chains in planar point sets: Combinatorial structure and algorithms. *SIAM J. Comput.* **28** 192–209.

Finke, G., V. Jost, M. Queyranne. 2004. Batch processing with interval graph compatibilities between tasks. *Proc. Discrete Optim. Methods in Production and Logist.*, Omsk-Irkutsk, Russia, 88–94.

G, Y.-G., M.-K. Kang. 2001. A fast algorithm for two-dimensional pallet loading problems of large size. *Eur. J. Oper. Res.* **134** 193–202.

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

Jansen, K. 2003. The mutual exclusion scheduling problem for permutation and comparability graphs. *Inform. Comput.* **180** 71–81.

Letchford, A. N., A. Amaral. 2001. Analysis of upper bounds for the pallet loading problem. *Eur. J. Oper. Res.* **132** 582–593.

Moonen, L. S. 2001. Optimaliseren van een productieproces. Master's thesis, Department of Mathematics, Maastricht University, Maastricht, The Netherlands (in Dutch).

Morabito, R., S. Morales. 1998. A simple and effective procedure for the manufacturer's pallet loading problem. *J. Oper. Res. Soc.* **49** 819–828.

Ryan, D. M., B. A. Foster. 1981. An integer programming approach to scheduling. A. Wren, ed. *Computer Scheduling of Public Transport: Urban Passenger, Vehicle and Crew Scheduling*. North-Holland, Amsterdam, The Netherlands, 269–280.

Scheithauer, G., J. Terno. 1996a. The G4-heuristic for the pallet loading problem. *J. Oper. Res. Soc.* **47** 511–522.

Scheithauer, G., J. Terno. 1996b. A heuristic approach for solving the multi-pallet packing problem. Working paper, Institute for Numerical Mathematics, Dresden University of Technology, Dresden, Germany.

Shum, H., L. E. Trotter, Jr. 1996. Cardinality-restricted chains and antichains in partially ordered sets. *Discrete Appl. Math.* **65** 421–439.

Terno, J., G. Scheithauer, U. Sommerweiß, J. Riehme. 2000. An efficient approach for the multi-pallet loading problem. *Eur. J. Oper. Res.* **123** 372–381.

Vance, P. H., A. Atamtürk, C. Barnhart, E. Gelman, E. L. Johnson, A. Krishna, D. Mahidhara, G. L. Nemhauser, R. Rebello. 1997. A heuristic branch-and-price approach for the airline crew pairing problem. Technical Report LEC-97-06, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

Vanderbeck, F. 1994. Decomposition and column generation for integer programs. Ph.D. thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

Wanke, E. 1994. $k$-NLC graphs under polynomial algorithms. *Discrete Appl. Math.* **54** 251–266.