Datalogisk Institut
KØBENHAVNS UNIVERSITET

The Off-line Group Seat Reservation Problem

Tommy Clausen, Allan Norlunde Hjorth,
Morten Nielsen, David Pisinger

Technical Report no. 07/02
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

The Off-line Group Seat Reservation Problem

Tommy Clausen, Allan Nordlunde Hjorth, Morten Nielsen, David Pisinger

DIKU, University of Copenhagen*
{tclausen,wolfie,mnielsen,pisinger}@diku.dk

Abstract

In this paper we address the problem of assigning seats in a train for a group of people traveling together. We consider two variants of the problem. One is a variant of two-dimensional knapsack where we consider the train as having fixed size and the objective is to maximize the utilization of the seats in the train. The other is a variant of two-dimensional bin packing where all requests must be accommodated while trying to minimize the number of passenger cars needed. We present a number of bounds for these two variants and develop exact algorithms for solving the problems. Computational results are presented for various instances known from the packing literature adapted to the problems addressed.

1 Introduction

We are considering the off-line group seat reservation problem (GSRP). In this problem it is the objective to maximize the use of the seats in a train subject to a number of constraints: A train consists of a number of seats which are numbered consecutively. A seat reservation brings a person from a station a to a station b without changing seat. A group of people, all having the same station of origin and destination, may wish to sit together, i.e. being assigned to seats with consecutive numbers. Since the problem is off-line, it is assumed that all data are given in advance.

The GSRP can be interpreted geometrically in the following way. A (group) reservation can be represented by a rectangle having width equal to the number of seats reserved and height equal to the distance travelled. For the train, the entire route corresponds to the height and the availability of seats is represented by the width. This corresponds to a two-dimensional orthogonal packing problem where no rotation is allowed and where the position of each reservation is fixed in height.

The example in Figure 1 illustrates the geometric interpretation of the GSRP. A train with three seats travels four stations from y_1 to y_4. The five given reservations and the corresponding packing is illustrated in the figure. We shall in the following use the terms reservation and rectangle interchangedly.

*Technical Report 02/2007, DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen

reservation	no. seats	from	to
1	1	y_1	y_3
2	2	y_1	y_2
3	1	y_2	y_3
4	1	y_2	y_4
5	2	y_3	y_4

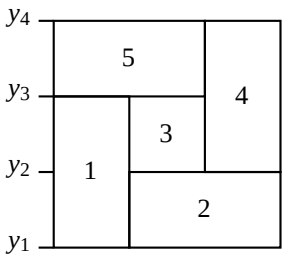


Figure 1: The geometric interpretation of a group seat reservation problem instance. The train travels from station y_1 to station y_4. Five reservations are given in the table to the left and a packing of the train is shown to the right.

The group seat reservation problem has numerous applications. A straightforward application is the reservation of hotel rooms, where people wish to have adjacent rooms. Also some job scheduling problems can be seen as a group seat reservation problem. Each job has a fixed starting time and a completion time, and it should be carried out on a number of adjacent machines. Finally, the group seat reservation problem finds application in the cutting of two-dimensional material, where the quality of the material varies from top to bottom. This is e.g. the case for coloring of textiles, where the color may be more intensive in the beginning of a roll, while it is less intensive at the end. In a similar way, a sheet of metal may have varying qualities, if the melted metal is the result of a chemical process which varies over time.

In the on-line version of the group seat reservation problem, reservation requests arrive one by one, and should be assigned a group of seats immediately. The on-line algorithm must be *fair*, i.e. it should only reject a request if it cannot be satisfied at the present moment. Various performance measures for on-line algorithms have been presented. The most common measure is the *competitive ratio* [15] which is defined as the worst-case ratio over all possible input sequences of the on-line performance to the optimal off-line performance. In the *relative worst-order ratio* one compares two on-line algorithms by considering the ratio on how the two algorithms perform on their respective worst ordering of the sequence. Boyar and Medvedev [6] considered the single-customer version of the on-line problem, and showed that if all tickets have the same price, first-fit and best-fit are better than worst-fit in the relative worst-order ratio. This also holds for the case where the price of the ticket is proportional to the distance traveled. Moreover, they showed that the off-line version of the single-customer group seat reservation problem where all tickets have the same price, is equivalent to a maximum k-colorable subgraph problem for interval graphs, which can be solved in polynomial time, as shown by Yannakakis and Gavril [28]. Helliesen [18] presented a new algorithm, scan-fit, for the on-line single-customer seat reservation problem, that was shown to perform better than first-fit and best-fit in the competitive ratio, both theoretically and empirically. Helliesen also considered the on-line version of the GSRP. In his version it is the objective to utilize the seats as effectively as possible while trying to minimize the distance between the people in a group. The developed algorithms also take into account how seats are arranged in the train by adding distance tables.

In this paper we describe two variants of the GSRP derived from two variants of

two-dimensional packing. Definitions and terminology follow in Section 2. The first variant, the group seat reservation knapsack problem, is considered in Section 3, upper bounds are derived in Section 3.1 and an exact algorithm is presented in Section 3.2. In addition, a method for testing the feasibility of reservation packings is presented in Section 3.3. The second variant, the group seat reservation bin packing problem is described in Section 4. Lower bounds for this problem are described in Section 4.1 and an exact algorithm is presented in Section 4.2. Finally, computational results for both variants are presented in Section 5.

2 Definitions and Terminology

Using the terminology from bin packing we may assume that the train has H stations, and W seats. Let $N = \{1, \cdots, n\}$ be the set of requests, each request j asking for a number w_j of seats, traveling h_j stations from station y_j to station $y_j + h_j$. Without loss of generality we may assume that $w_j \leq W$.

Although H in principle may be large, we may reduce the problem to only consider the active stations, i.e.

$$Y := \{y_j \mid j \in N\} \cup \{y_j + h_j \mid j \in N\}$$

and let N_y be the set of requests using a seat at station $y \in Y$

$$N_y := \{j \in N \mid y_j \leq y < y_j + h_j\}.$$

We associate with each station $y \in Y$ a "height" H_y to represent the distance from station y to the next active station in Y.

By considering the train as a single row of seats, we formulate the group seat reservation knapsack problem (GSR-KP) as the problem of choosing the set of requests that maximizes the utilization of the train. We measure the utilization of the train as the number of seats times the distance travelled for all chosen reservations. This problem is a variant of the two-dimensional knapsack problem (2DKP), in which a number of rectangles have to be placed inside a larger sheet. Each rectangle has an associated profit and the objective is to place the rectangles on the sheet so as to maximize the overall profit. For the GSR-KP the profit is the area of the rectangles. A well-known variant of the 2DKP is the addition of so called guillotine pattern constraints. A two-dimensional packing is said to have a guillotine cutting pattern if the sheet can be separated by a horizontal or vertical line such that none of the rectangles are cut in two by the line. This property must hold recursively for the two sheets obtained by separating the original along the line. It is noted that guillotine cutting patterns cannot generally be applied to the GSR-KP, as seen by the packing in Figure 1 where no guillotine cuttable solution exists.

The problem may be formulated as the following integer-programming model. Let $\delta_i = 1$ if request i is selected. Let x_i be the first seat (left coordinate) of request i. Let $E = \{(i, j)\}$ be the set of rectangles which in some way share a station (y-coordinate).

Finally, let $\ell_{ij} = 1$ iff request i is located left of j.

$$\max \quad \sum_{j \in N} w_j h_j \delta_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in N_y} w_j \delta_j \leq W, \qquad y \in Y \tag{2}$$

$$\delta_i + \delta_j - \ell_{ij} - \ell_{ji} \leq 1 \qquad (i,j) \in E \tag{3}$$

$$x_i - x_j + W\ell_{ij} \leq W - w_i \qquad (i,j) \in E \tag{4}$$

$$0 \leq x_j \leq W - w_j \qquad j \in N \tag{5}$$

$$\ell_{ij} \in \{0,1\} \qquad (i,j) \in E \tag{6}$$

$$\delta_j \in \{0,1\} \qquad j \in N \tag{7}$$

Here (2) says that we may not exceed the capacity W of the train at any station. Constraint (3) says that if both requests i and j are selected, then one of them should be to the left of the other i.e. $(\delta_i = 1 \wedge \delta_j = 1) \Rightarrow (\ell_{ij} = 1 \vee \ell_{ji} = 1)$. Constraint (4) says that if request i is located to the left of request j then this should be reflected in the coordinates, i.e. $\ell_{ij} = 1 \Rightarrow x_i + w_i \leq x_j$. Constraint (5) says that all requests should be placed inside the train and finally constraints (6) and (7) say that the variables should be binary.

The problem is $\mathcal{NP}$-hard, which can easily be shown by reduction of the subset sum problem to a group seat reservation knapsack problem with no intermediate stations.

Trains usually consist of several passenger cars, rather than a single passenger compartment. This poses additional restrictions on the placement of the reservations in the train. Not only must reservations be assigned consecutive seats, they must also be assigned seats in the same car to be adjacent. This defines another problem, namely the group seat reservation bin packing problem (GSR-BPP) as the problem of assigning reservations to cars such that all requests are fulfilled and the number of cars used are minimized. The GSR-BPP is a variant of the two-dimensional bin packing problem (2DBPP), in which a number of rectangles must be assigned to identical bins, such that no bin is overfilled and the number of bins used is minimized. Regarded as a 2DBPP variant, the train cars correspond to bins and the reservations correspond to the rectangles that must be assigned to bins. Additionally, the rectangles have fixed y-coordinates in the GSR-BPP.

More formally, we define the GSR-BPP as follows. Let W determine the number of seats in a car, and let all other variables from formulation (1)–(7) have the same interpretation in the GSR-BPP as in the GSR-KP. Furthermore, let m_i identify the car that request i is in, and let $p_{ij} = 1$ if request i is placed in a car closer to the front of the train than request j (i.e. if $m_i < m_j$). Finally v denotes the number of cars used and n is

the number of requests. The problem may then be formulated as:

$$\begin{aligned}
\min \quad & v && && (8)\\
\text{s.t.} \quad & \ell_{ij}+\ell_{ji}+p_{ij}+p_{ji} \geq 1 && (i,j)\in E, i<j && (9)\\
& x_i - x_j + W\ell_{ij} \leq W - w_i && (i,j)\in E && (10)\\
& m_i - m_j + np_{ij} \leq n-1 && (i,j)\in E && (11)\\
& 0 \leq x_j \leq W - w_j && j \in N && (12)\\
& 0 \leq m_j \leq v && j \in N && (13)\\
& \ell_{ij}, p_{ij} \in \{0,1\} && (i,j)\in E && (14)\\
& m_j \in \mathbb{N} && j \in N && (15)
\end{aligned}$$

Constraint (9) states that either requests i and j may not overlap, or they must be in different cars. Constraint (10) enforces that x-coordinates must reflect the values of the ℓ variables, i.e. $\ell_{ij} = 1 \Rightarrow x_i + w_i < x_j$. Similarly, constraint (11) enforces that if request i is placed in front of request j (car-wise), it must have a lower car number, i.e. $p_{ij} = 1 \Rightarrow m_i < m_j$. Constraint (12) states that a request must be placed entirely inside a car, and constraint (13) bounds the number of cars used.

The formulation (8)–(15) is $\mathcal{NP}$-hard, which can be realized by reducing from one-dimensional bin packing to the special case where all requests travel the entire train route.

To the authors' knowledge, no previous literature exist on the off-line GSR-KP or GSR-BPP. However, much related work has been done on the two-dimensional knapsack problem and the two-dimensional bin packing problem.

A number of algorithms have been presented for the 2DKP. Hadjiconstantinou and Christofides [17] studied various IP formulations, but were only able to solve instances of moderate size. Fekete and Schepers [14] presented a two-phase algorithm which first selects a subset of rectangles with large profits, and then tests feasibility through an enumerative algorithm based on isomorphic packing classes. Caprara and Monaci [7] presented an $(\frac{1}{3}-\varepsilon)$-approximation algorithm for the 2DKP and developed four exact algorithms based on various enumeration schemes. Belov [3] consider a special case of the 2DKP in which the placement of the rectangles should follow a guillotine packing pattern. This more restricted version of the problem can be solved through dynamic programming. Finally, Pisinger and Sigurd [26] used constraint programming to solve the general and guillotine-restricted version of the 2DKP. The algorithm follows a two-phase approach as in Fekete and Schepers [14] by first choosing a subset of rectangles to pack in the sheet (by solving a one-dimensional knapsack problem), and then testing whether the rectangles actually fit into the sheet. If the rectangles do not fit into the sheet a new constraint is added to the one-dimensional knapsack problem.

Berkey and Wang [2] presented an extensive computational review of heuristics for finding upper bounds for two-dimensional bin packing problems (2DBPP). The heuristics considered were mostly similar to well-known on-line algorithms like best-fit or next-fit. For a thorough presentation of on-line packing algorithms we refer to Csirik and Woeginger [9].

A number of lower bounds for the 2DBPP exist, mostly based on adaptations of bounds for one-dimensional bin packing (see e.g. Martello and Vigo [23]). More recent

bounds include Fekete and Schepers [12] and Boschetti and Mingozzi [4], [5]. The latter also present a heuristic based upper bound.

Martello and Vigo [23] present an exact algorithm for the 2DBPP. The algorithm consists of an outer branch-decision tree in which rectangles are assigned to bins. At every node a heuristic is used to find a feasible solution for the assignment. If none is found an inner branch-decision tree is created to test the feasibility of the assignment. An additional heuristic is used to close bins if no unassigned rectangles can fit into the bin. A similar approach was used by Martello, Pisinger and Vigo [22] to solve the three-dimensional bin packing problem.

Finally, Lodi, Martello and Vigo [21] provides a detailed survey of bounds, exact algorithms and heuristics for the 2DBPP.

3 The Group Seat Reservation Knapsack Problem

As mentioned in the introduction we consider the group seat reservation knapsack problem where the train is considered as having all seats in a single row. In this section we present a number of upper bounds that we use to develop an exact algorithm for the problem.

3.1 Upper Bounds

A first upper bound may be obtained by LP-relaxing the integer programming model for GSR-KP defined by (1)–(7). The optimal solution to this model gives us the upper bound u_1.

Notice that in any LP-optimal solution to (1)–(7) we may set $x_i := 0$ and $\ell_{ij} := (W - w_i)/W$. In this way constraint (3) is always satisfied when (1) is satisfied since

$$\ell_{ij} + \ell_{ji} = \frac{W - w_i}{W} + \frac{W - w_j}{W} = 2 - \frac{w_i + w_j}{W},$$

and (4) is also satisfied since

$$W\ell_{ij} = W\frac{W - w_i}{W} = W - w_i.$$

As constraint (5) is obviously satisfied, what remains is the problem

$$\begin{aligned} \max \quad & \sum_{j \in N} w_j h_j \delta_j \\ \text{s.t.} \quad & \sum_{j \in N_y} w_j \delta_j \leq W, \quad y \in Y \\ & 0 \leq \delta_j \leq 1 \qquad j \in N \end{aligned} \tag{16}$$

Now, suppose we create a variable δ_j^k for each single seat by splitting each variable δ_j

into w_j variables. This yields the formulation

$$\begin{aligned} \max \quad & \sum_{j \in N} \sum_{k=1}^{w_j} h_j \delta_j^k \\ \text{s.t.} \quad & \sum_{j \in N_y} \sum_{k=1}^{w_j} \delta_j^k \leq W, \quad y \in Y \\ & 0 \leq \delta_j^k \leq 1 \qquad\qquad j \in N,\ k = \{1, \cdots, w_j\}, \end{aligned} \tag{17}$$

which is equivalent to (16).

Let A be the constraint matrix of (17). In each column of A the ones will appear consecutively, since the reservations are connected. This is known as the "consecutive ones" property which implies that A is totally unimodular. Consequently, $\delta_j^k \in \{0,1\}$ in the optimal solution to (17). This means that (17), and thereby (16), is equivalent to splitting the reservations into single-seat reservations.

Denote by G the intersection graph of A, i.e. G contains a vertex for each column in A and an edge between two vertices if there exists a row in A that contains ones in both the corresponding columns. Equivalently, G contains a vertex for each single-seat reservation, and two vertices in G are connected by an edge if the corresponding reservations share a station. By considering the reservations as intervals on a real line, we note that G is an interval graph (and is thereby perfect).

If we assign the reservation travel distances as weights to the vertices of G, we may formulate (17) as a weighted k-independent set problem. The weighted k-independent set problem considers a graph with a non-negative weight associated with each vertex. The problem is then to find k independent sets such that the weight sum of all vertices in the independent sets is maximized. The problem is $\mathcal{NP}$-hard in general, but can be solved polynomially if the graph is an interval graph (see e.g. [25]). This graph interpretation of the single-seat reservation problem is also noted in [6], although they noted it for the also equivalent k-colorable subgraph problem.

For the weighted k-independent set problem on interval graphs, Pal and Bhattacharjee [25] present an $O(km\sqrt{\log c} + \gamma)$ time algorithm, where m is the number of vertices in the interval graph, c is the weight of the longest path in the graph and γ is the total size of all maximal cliques in the graph. Using this algorithm to solve (16), we can rewrite the running time using the notation of the GSR-KP. The number of independent sets k is then the number of seats in the train W. The number of vertices m is the total number of seats in all reservations, i.e. $m = \sum_{j=1}^{n} w_j$. The weight of the longest path c is bounded above by the total travel length of all reservations, i.e. $c \leq \sum_{j=1}^{n} h_j$. The number of cliques in an interval graph is at most the number of vertices (see e.g. [16]), so γ is bounded by $m^2 = \left(\sum_{j=1}^{n} w_j\right)^2$. Thus, the complexity is

$$O\left(W \cdot \sum_{j=1}^{n} w_j \cdot \sqrt{\log \sum_{j=1}^{n} h_j} + \left(\sum_{j=1}^{n} w_j\right)^2\right). \tag{18}$$

By noting that $\sum_{j=1}^{n} w_j \leq N \cdot W$ and $\sum_{j=1}^{n} h_j \leq N \cdot H$ expression (18) can be reduced to

$$O\left(NW^2\sqrt{\log(NH)} + (NW)^2\right)$$

which is clearly pseudo-polynomial in terms of the GSR-KP.

We have described two methods for calculating the single-seat relaxation. We shall denote by $\mathcal{U}_1$ the bound calculated by solving the LP-relaxation of (1) – (7) and denote by $\mathcal{U}_2$ the bound calculated by the weighted k-colorable subgraph problem as described above. Although the bounds are identical, the time complexities of calculating them are different and no dominance exists between the time bounds.

A third upper bound is obtained by relaxing the problem to the case where the passengers may need to change seats at every station.

$$\begin{aligned} \max \quad & \sum_{j \in N} w_j h_j \delta_j \\ \text{s.t.} \quad & \sum_{j \in N_y} w_j \delta_j \leq W \quad y \in Y \\ & \delta_j \in \{0,1\} \qquad j \in N \end{aligned} \tag{19}$$

The above problem is a multidimensional knapsack problem with $|Y|$ knapsack constraints. Let us denote the optimal value of this problem $\mathcal{U}_3$. The multidimensional knapsack problem is strongly $\mathcal{NP}$-hard (see. e.g. [19]).

A fourth upper bound may be found by for every station (y-coordinate), to calculate how well the train may be filled at this station, by solving the following subset sum problem:

$$F_y = \max\left\{ \sum_{j \in N_y} w_j \delta_j \,\middle|\, \sum_{j \in N_y} w_j \delta_j \leq W, \delta_j \in \{0,1\} \right\}.$$

We may now calculate an upper bound as

$$\mathcal{U}_4 = \sum_{y \in Y} F_y H_y.$$

Calculating $\mathcal{U}_4$ is $\mathcal{NP}$-hard as it contains $|Y|$ subset sum problems.

A fifth upper bound can be calculated by looking at the transition at any station, i.e. taking into account that the seats being left should be filled by new passengers. For a given station $y \in Y$ we consider three disjunctive sets of requests:
P_y is the set of requests that pass through station y, i.e.

$$P_y = \{ j \in N \mid y_j < y < y_j + h_j \}$$

S_y is the set of requests which start at station y, i.e.

$$S_y = \{ j \in N \mid y_j = y \}$$

T_y is the set of requests which terminate at station y, i.e.

$$T_y = \{ j \in N \mid y_j + h_j = y \}$$

For a given station $y \in Y$ consider the problem

$$\begin{aligned} L_y = \min \quad & \Big\{ H_y \left(W - \textstyle\sum_{j \in P_y} w_j \delta_j - \sum_{j \in S_y} w_j \delta_j \right), \\ & \quad H_{y-1} \left(W - \textstyle\sum_{j \in P_y} w_j \delta_j - \sum_{j \in T_y} w_j \delta_j \right) \Big\} \\ \text{s.t.} \quad & \textstyle\sum_{j \in P_y} w_j \delta_j + \sum_{j \in S_y} w_j \delta_j \leq W \\ & \textstyle\sum_{j \in P_y} w_j \delta_j + \sum_{j \in T_y} w_j \delta_j \leq W \\ & \delta_j \in \{0,1\}, j \in P_y \cup S_y \cup T_y \end{aligned}$$

In other words, L_y is the minimum loss we get at station y. Summing the loss at all stations, we get the upper bound

$$\mathcal{U}_5 = WH - \sum_{y \in Y} L_y.$$

L_y may be determined through dynamic programming. Through a straightforward subset sum recursion we first find all weight sums which can be obtained using requests from P_y. In a similar way we derive dynamic programming tables for weight sums which can be obtained using requests from S_y and for requests from T_y. Now, for every obtainable weight sum corresponding to requests in P_y, we match it with the best weight sums from S_y (resp. T_y) so that the total sum does not exceed W. Note that the first and the last station are special cases, which needs to be handled separately. At the first station there is no previous station, so T_y and P_y are both empty sets. Thus L_0 can be determined by solving a subset sum problem on S_y. On the other hand, the last station will never generate any profit since no passenger will be on board afterward, so any potential loss can safely be ignored. L_y can be determined in $O(nW)$ time, where $n = |P_y \cup S_y \cup T_y|$. Thus, the running time of $\mathcal{U}_5$ is $O(nWY)$.

3.1.1 Bound Dominance

The bound $\mathcal{U}_4$ is calculated by splitting the reservations into smaller reservations each traveling only one station. In the interpretation of allowing seat changes, this corresponds to allowing seat changes to "outside the 'train", i.e. passengers are allowed to leave the train and join it again at a later station. This is clearly less restricted than $\mathcal{U}_3$ in which seat changes must be to other seats within the train. Thus, $\mathcal{U}_3$ dominates $\mathcal{U}_4$. Bound $\mathcal{U}_5$ considers the train before and after each station and chooses the best obtainable packing. This may be done by calculating a subset-sum problem at the station and its predecessor and choosing the highest value. Clearly, $F_y \leq W - L_y$ so $\mathcal{U}_4$ dominates $\mathcal{U}_5$. That $\mathcal{U}_3$ dominates $\mathcal{U}_1$ and $\mathcal{U}_2$ is seen by comparing formulations (16) and (19). Clearly, $\mathcal{U}_1$ and $\mathcal{U}_2$ are the LP-relaxation of $\mathcal{U}_3$.
The bounds $\mathcal{U}_1$ and $\mathcal{U}_4$ split reservations in different ways: $\mathcal{U}_1$ splits into single-seat reservations and $\mathcal{U}_4$ splits into single-station reservations. Thus, $\mathcal{U}_1$ and $\mathcal{U}_4$ does not dominate each other. But since the problem is already restricted with regard to stations, we may expect $\mathcal{U}_1$ to be the tighter of the two bounds in general.

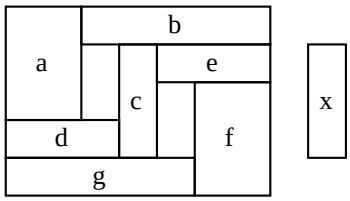
Theorem 1 *The bound $\mathcal{U}_1$ does not dominate $\mathcal{U}_4$ and $\mathcal{U}_4$ does not dominate $\mathcal{U}_1$.*

Proof. Consider a train with height H and width W and two reservations r_1 and r_2 which both have height H (i.e. they travel from the first to the last station) and require $\lfloor W/2 \rfloor + 1$ seats. Clearly, both r_1 and r_2 cannot be assigned seats in the train. By splitting the reservations into single-seat reservations, the entire train can be filled, so $u_1 = H \cdot W$. If the reservations into single-station reservations, each station can still only accommodate r_1 or r_2. Thus, $u_4 = H \cdot (\lfloor W/2 \rfloor + 1)$.
Conversely, consider the reservations s_1 and s_2 that both request W seats. Let s_1 travel from station 1 to station $\lfloor H/2 \rfloor + 1$ and s_2 from station $\lfloor H/2 \rfloor$ to station H. Splitting s_1 and s_2 into single-seat reservations will not change that station $\lfloor H/2 \rfloor$ can accommodate only W seats. The W largest single-seat reservations will originate from s_1, so $u_1 = W \cdot (\lfloor H/2 \rfloor + 1)$. If the reservations are split into single-station reservations, only station $\lfloor H/2 \rfloor$ will be overfilled. This, and all other stations can be filled completely, so $u_4 = W$. □

As u_3 dominates all the other bounds and is itself *NP*-hard, it may be that u_3 is simply a more simple formulation of the GSR-KP. This is not the case as is shown in Theorem 2.

Theorem 2 *There exists an instance of the GSR-KP for which $u_3 > OPT$, where OPT denotes the optimal solution.*

Proof. Consider a train with 7 seats and the reservations *a–g* and *x* as illustrated below. The illustrated packing of *a-g* represents an optimal solution. The bound u_3 will split *x* into single-station reservations which may be placed next to *c*. Thus, $OPT < u_3$ for this instance.



□

3.2 Exact Algorithm

In Section 3.1 we have used different ideas to develop upper bounds for the GSR-KP. For all the bounds it is possible that the calculated value is optimal for the GSR-KP, but it is more likely to be larger than optimum. Since we wish to develop an exact algorithm for solving the GSR-KP we use branch and bound.

In each node of the branching tree we choose a rectangle j and divide the solution space into two subtrees. In one subtree we demand that the chosen rectangle is in the packing and in the other subtree, we exclude the rectangle from the packing. In the integer programming model (1)–(7) this corresponds to branching on the δ variables fixing δ_j to 1 respectively 0. We thereby get two new subproblems. The first subproblem is equivalent to saying that the width of the train in reduced by w_j on all stations

covered by the rectangle. The second subproblem corresponds to removing the chosen rectangle from the set of rectangles that should be packed.

We start by fixing the largest rectangles. This way we will very early in the branching tree get to a point where there is no more room for extra rectangles and we can prune large parts of the branching tree.

When we fix a rectangle we check each station separately to see if there is enough room for the rectangle, but this does not ensure that there exist a legal packing of the fixed rectangles. Therefore at some point we need to test for feasibility i.e. find out if there exist a legal packing of the chosen rectangles. One can consider different schemes for testing feasibility. One possibility is to test for feasibility at each node, but since we need to find a packing to ensure that it is legal, this scheme requires solving an $\mathcal{NP}$-hard problem at each node, thus we choose to only test for feasibility when all rectangles have been fixed. How the actual testing of feasibility is done, is described in Section 3.3.

Since it is very important that we quickly find some feasible solutions in order to prune parts of the branching tree we use a depth first strategy in our branching.

3.3 Testing Feasibility

When testing the feasibility of a packing, we are given a set of reservations and we then wish to determine if the reservations can be placed in a way to make them fit into the train. Placing the reservations within the train corresponds to assigning values to the binary ℓ-variables of equations (3) and (9) for the GSR-KP and GSR-BPP, respectively. Recall that $\ell_{ij} = 1$ means that reservation i is positioned to the left of j.
The test for feasibility consists of two parts. An algorithm that determines the feasibility of a single packing (i.e. assignment of the ℓ-variables) is described in section 3.4 and an enumeration scheme for applying this algorithm to every possible packing is described in section 3.5.

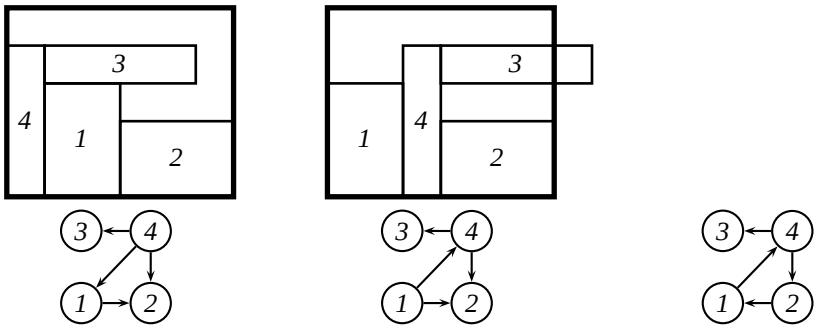
3.4 Feasibility of a Packing

. We shall first consider the feasibility of a packing where the relative positions of the rectangles are known, i.e. the left-right ordering of overlapping rectangles is predetermined. To test the feasibility of the packing, we will construct a graph representing the packing in a way similar to the constraint graph considered by Pisinger and Sigurd [26] and Fekete and Schepers [13]. By exploiting that the rectangles are fixed in the y-dimension, we may represent a packing by using a graph as follows: Let $G = (V, E)$ be a directed graph with a vertex for each reservation and the edge $(i, j) \in E$ iff $\ell_{ij} = 1$. For ease of notation we shall not distinguish between a vertex and its corresponding rectangle. The following properties are necessary and sufficient for determining if the packing represented by G is feasible. Without loss of generality we will assume that all rectangles are placed as far to the left as possible.

P1 G is acyclic.

P2 For every path $p = \langle v_1, \ldots, v_n \rangle$ in G, $\sum_{i \in p} w_i \leq W$.

Example 1 *Consider the instance given by* $N = \{(2,3,0),(3,2,0),(4,1,3),(1,4,0)\}$, *where* $i = (w_i, h_i, y_i)$, $i \in N$ *and* $H = 6, W = 6$. *Below are shown graph representations of three different packings of the instance.*



The leftmost graph represents a feasible packing with 1 *to the left of* 2 *and* 4 *to the left of all other rectangles. The middle graph contains the path* $p = \langle 1,4,3 \rangle$ *with* $\sum_{i \in p} w_i = 7$, *so this is infeasible by P2. The rightmost graph contains a cycle and does not represent a legal packing by P1. By transitivity,* 1 *must be both to the left and right of* 2, *but this cannot result in a feasible arrangement of the rectangles.*

That the properties P1 and P2 are necessary follows from the middle and right graph of Example 1. That the properties are also sufficient can be seen by the following:

Theorem 3 *The properties P1 and P2 are sufficient for describing a feasible packing*

Proof. We will show that every graph G that satisfies properties P1 and P2 corresponds to an arrangement of rectangles that comprises a feasible packing, i.e. the rectangles satisfies the following:

(1) None of the rectangles overlap

(2) All rectangles are placed within the box representing the train

To show (1) consider two nodes $i, j \in V$. Since G is acyclic (by P1), exactly one of the three following cases occur:

i) There is a path p from i to j in G.
Assume wlog. that p has length k and $p = \langle v_0, v_1, \ldots, v_k \rangle$ with $v_0 = i$ and $v_k = j$. For each edge $(v_m, v_{m+1}) \in E$ we can place v_m and v_{m+1} so that $x_m + w_m \leq x_{m+1}$. By transitivity this will ensure $x_i + w_i \leq x_j$, yielding a packing where i and j does not overlap.

ii) There is a path from j to i in G.
This case is analogous to i) and yields $x_j + w_j \leq x_i$.

iii) There is no path between i and j in G.
i and j are positioned at different heights in the box, i.e. $y_i + h_i \leq y_j$ or $y_j + h_j \leq y_i$. Since the y-coordinates are fixed there is no way for i and j to overlap.

Since each of the three cases can lead to a packing without overlap and i and j were chosen arbitrarily, (1) is proven.
To show (2) consider a rectangle $j \in N$ and assume for the sake of contradiction that $x_j + w_j > W$. Since $w_j \leq W$, $x_j > 0$. By assumption, j is positioned as far left as possible, so there must exist a rectangle $i \in N$ with $x_i + w_i = x_j$ and $(i, j) \in E$, i.e. rectangle i blocks j from being pushed further to the left. Similarly, another such rectangle will exist for i unless $x_i = 0$. This leads to a sequence of rectangles $\{m_0, m_1, \ldots, m_k\}$ where $j = m_0$ and $i = m_1$. Each rectangle in the sequence will touch its immediate successor and predecessor in the sequence.
By considering the sequence in reverse order (such that the rectangles will be ordered left to right), we get a path in G. Since the rectangles touch each other, $x_{m_s} = \sum_{t=s+1}^{k} w_t$. Rectangle j is the last rectangle on the path, so $x_j = m_0 = \sum_{t=1}^{k} w_t$. But $x_j + w_j = \sum_{v \in p} w_v \leq W$ by property P2, which leads to the desired contradiction. Since j was chosen arbitrarily, this proves (2). □

The following algorithm determines if a graph satisfies the properties P1 and P2 by computing the x_j values for all reservations $j \in N$. By the assumption of leftmost placement we must set

$$x_j = \begin{cases} \max\{x_i + w_i \mid (i, j) \in E\}, & \exists i \in N : (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

The algorithm works as follows: Initially all rectangles are placed all the way to the left. We topologically sort G to find a left-right ordering of the rectangles. If G is cyclic, and thus infeasible by P1, this is discovered during the topological sort. Starting with the leftmost rectangles, the rectangle pushes all overlapping rectangles to the right until they no longer overlap. Since G is topologically sorted, no overlap will exist when all rectangles have been considered. If a rectangle is pushed beyond the width of the box, i.e. for some j we have $x_j + w_j > W$, the packing is infeasible by P2.
The main loop of the algorithm visits every vertex and every edge, so the main loop runs in $\Theta(V + E)$ time. The topological sort can be implemented to run in $O(V + E)$ time, so the complexity of Algorithm 1 is $\Theta(V + E)$.

3.5 Enumeration Scheme

Algorithm 1 describes testing the feasibility of a packing, given the individual placements of the reservations. To decide if a feasible packing exists for a set of reservations, all such placements must be considered. Since each placement is represented by a specific orientation of all edges in the graph, there exist an exponential number of placements. We consider all placements by using a branch-decision tree to enumerate the edge orientations in the graph. At each node in the tree we add an edge to the graph and branch on the orientation of the edge. If the graph at some node describes an infeasible packing, that subtree is eliminated since adding more edges to the graph will not make the packing feasible. If a feasible packing is found at a leaf node, the algorithm returns **true**. If no more nodes exists, the algorithm returns **false**.

Algorithm 1 Assigning position coordinates to reservations in packing

```
if G is acyclic then
  L ← topologic_sort(G)
else
  return false
end if
x_i ← 0,   ∀i ∈ N {Initialize x_i}
while L ≠ ∅ do
  v_i = first item in L
  for all v_j : (i, j) ∈ E do
    x_j ← max{x_j, x_i + w_i} {Push all overlapping rectangles right}
    if x_j + w_j > W then
      return false
    end if
  end for
  L ← L \ {v_i}
end while
return true
```

4 The Group Seat Reservation Bin Packing Problem

So far, the train has been considered as having a single line of seats. However, most trains will consist of several cars or coupes, and it would seem unreasonable to split a group of passengers traveling together into different cars, even if their seat numbers are consecutive. Thus we now consider the group seat reservation bin packing problem. For this problem we will, as for the GSR-KP, develop an exact algorithm, but first we present some lower bounds we can use in the algorithm.

4.1 Lower Bounds

A first lower bound $\mathcal{L}_1$ may be found by solving the LP-relaxation of (8)–(15). Similar to the GSR-KP upper bound $\mathcal{U}_1$ we may set $\ell_{ij} = (W - w_i)/W$ and $p_{ij} = (n-1)/n$. Thus constraint (8) is always satisfied, since

$$\ell_{ij} + \ell_{ji} + p_{ij} + p_{ji} = 4 - \frac{w_i + w_j}{W} - \frac{2}{n}.$$

Again, similar to $\mathcal{U}_1$, setting $x_i := 0$ and $m_i := 0$ satisfies constraints (9) and (10), since

$$W\ell_{ij} = W\frac{W - w_i}{W} = W - w_i \quad \text{and} \quad np_{ij} = \frac{n-1}{n} = n - 1.$$

As $m_i = 0$ for all i, $v = 0$ and $\mathcal{L}_1 = 0$ will hold for any instance. This means that $\mathcal{L}_1$ will always pack all items into the first bin.

A second lower bound $\mathcal{L}_2$ is found by for every station (y-coordinate), to calculate a lower bound on the number of cars at this station. Let the binary variable $x_{ij} = 1$ iff

request j is assigned to car i and let B denote the set of cars. Moreover let $\delta_i = 1$ iff car i is used.

$$\min \quad \xi_y^2 = \sum_{i=1}^{n} \delta_i \tag{20}$$

$$\text{s.t.} \quad \sum_{j \in N_y} w_j x_{ij} \leq W\delta_i \quad i \in B \tag{21}$$

$$\sum_{i \in B} x_{ij} = 1 \qquad j \in N_y \tag{22}$$

$$\delta_i \in \{0,1\} \qquad i \in B \tag{23}$$

$$x_{ij} \in \{0,1\} \qquad i \in B,\ j \in N_y \tag{24}$$

We may now calculate the lower bound as

$$\mathcal{L}_2 = \max_{y \in Y} \xi_y^2$$

The model (20)–(24) is recognized as an ordinary bin packing problem (BPP) which is $\mathcal{NP}$-hard to solve. Hence, to find a polynomial bound for the GSR-BPP we may use any lower bound from the literature of BPP. For each $y \in Y$ let ξ_y^3 be such a lower bound chosen as maximum of the bounds presented by Martello and Toth [24], Dell'Amico and Martello [11].

$$\begin{aligned} \xi_y^3 &= \left| \left\{ j \in N_y : w_j > \frac{W}{2} \right\} \right| \\ &+ \max_{1 \leq p \leq \frac{W}{2}} \left\{ \left\lceil \frac{\sum_{j \in N_s(p)} w_j - (|N_\ell(p)|W - \sum_{j \in N_\ell(p)} w_j)}{W} \right\rceil, \right. \\ &\qquad \left. \left\lceil \frac{|N_s(p)| - \sum_{j \in N_\ell(p)} \lfloor \frac{W - w_j}{p} \rfloor}{\lfloor \frac{W}{p} \rfloor} \right\rceil \right\} \end{aligned} \tag{25}$$

where

$$N_\ell(p) = \{ j \in N_y : W - p \geq w_j > \tfrac{W}{2} \} \tag{26}$$

$$N_s(p) = \{ j \in N_y : \tfrac{W}{2} \geq w_j \geq p \} \tag{27}$$

This leads to the third lower bound

$$\mathcal{L}_3 = \max_{y \in Y} \xi_y^3$$

which according to [11] can be calculated in $O(N_y)$ time for each $y \in Y$ leading to an overall time complexity of $O(N^2)$.

Finally, we may choose to consider two stations $y, y' \in Y$ at the same time. Let $x_{ij} = 1$ iff request $j \in N_y$ is assigned to car i at station y and $x'_{ij} = 1$ iff request $j \in N_{y'}$ is assigned to car i at station y'. As before let $\delta_i = 1$ iff car i is used and let B denote

the set of cars.

$$\begin{aligned}
\min \quad & \xi^4_{y,y'} = \textstyle\sum_{i=1}^{n} \delta_i && && (28)\\
\text{s.t.} \quad & \textstyle\sum_{j \in N_y} w_j x_{ij} \leq W \delta_i && i \in B && (29)\\
& \textstyle\sum_{j \in N_{y'}} w_j x'_{ij} \leq W \delta_i && i \in B && (30)\\
& x_{ij} = x'_{ij} && j \in N_y \cap N_{y'} && (31)\\
& \textstyle\sum_{i \in B} x_{ij} = 1 && j \in N_y && (32)\\
& \delta_i \in \{0,1\} && i \in B && (33)\\
& x_{ij}, x'_{ij} \in \{0,1\} && i \in B,\ j \in N_y && (34)
\end{aligned}$$

Here constraint (29) and (30) are the ordinary bin packing constraints for each of the stations y, y' while (31) demands that a request is assigned to the same car at both stations. Constraint (32) enforces that each request $j \in N_y$ must be assigned a car.

We may now calculate the lower bound as

$$\mathcal{L}_4 = \max_{y,y' \in Y} \xi^4_{y,y'}$$

Calculating ξ^4 is $\mathcal{NP}$-hard, which is seen by reduction from bin packing to the special case $y = y'$. Therefore calculating $\mathcal{L}_4$ is also $\mathcal{NP}$-hard.

4.2 Exact Algorithm

For solving the group seat reservation bin packing problem, we construct a two-phase branching algorithm as proposed by Martello and Vigo [23].

The first phase is an outer branch-decision tree that assigns rectangles to bins, considering rectangles ordered by decreasing size. At each node the next unassigned rectangle is assigned to each of the *open bins*, i.e. bins that already contain rectangles. If the number of open bins is less than the upper bound, a new bin is opened by assigning the rectangle to it. Initially the upper bound is set as the number of rectangles, and is updated when a better feasible solution is found. The tree is traversed in a depth first manner, as this postpones the parts of the solution space using a large number of bins to a point where they may hopefully be pruned. Each rectangle is only allowed to open one new bin, which specifically restricts the assignments of the first rectangles. Since there are no open bins initially, the first rectangle can only be assigned to the first bin (by opening it), the second rectangle can only be assigned to the first and second bin, and so on. This helps limit the symmetry inherent to bin packing problems.

The second phase is run at each node of the outer tree to test the feasibility of the assigned rectangles. We use the branch-decision tree described in Section 3.3 to test the feasibility.

4.2.1 Closing Bins

In addition to the two-phase branching scheme we attempt at each node to close one or more bins. If it can be determined that for some bin i, none of the unassigned rectangles

fit into the bin, we mark the bin as closed. In the subtree rooted at that node, rectangles are not assigned to the closed bin.

In order to avoid creating a feasibility branch-decision tree for each unassigned rectangle in combination with each node, the following method is employed instead. For each station (y-coordinate) of the bin, the width of all rectangles that cover that station is added. If the sum exceeds the width of the bin, the packing is infeasible. Since the method is heuristic in nature, it may occur that a bin is kept open even though no feasible packing may be obtained from adding any of the remaining unassigned rectangles. This method is identical to the test performed when fixing a rectangle in the branch and bound tree for the GSR-KP (see Section 3.2).

4.3 Dantzig Wolfe Decomposition

Like most other bin packing like problems, the GSR-BPP can easily be decomposed. Let S be the set of possible assignments of requests in a single car. Let $a_{ij} = 1$ iff request j was assigned to the car in assignment $i \in S$. Finally let $\delta_i = 1$ iff assignment i is used. Then the problem can be formulated as

$$\min \quad \sum_{i \in S} \delta_i \tag{35}$$

$$\text{s.t.} \quad \sum_{i \in S} a_{ij}\delta_i \geq 1 \quad j \in N \tag{36}$$

$$\delta_i \in \{0,1\} \qquad i \in S \tag{37}$$

The objective function (35) minimizes the number of assignments (i.e. cars) used, while constraint (36) ensures that every request was fulfilled.

As the above model may become super-polynomially large, delayed column generation can be used to solve the LP-relaxation of the model. Let $S' \subseteq S$ be a subset of the assignments, satisfying that all request can be fulfilled. Then the restricted master problem is

$$\min \quad \sum_{i \in S'} \delta_i \tag{38}$$

$$\text{s.t.} \quad \sum_{i \in S'} a_{ij}\delta_i \geq 1 \quad j \in N \tag{39}$$

$$\delta_i \in \{0,1\} \qquad i \in S' \tag{40}$$

Let v_j for $j \in N$ be the dual variables corresponding to (39). The pricing problem then

becomes

$$\max \quad \sum_{j \in N} v_j \delta_j \tag{41}$$

$$\text{s.t.} \quad \sum_{j \in N_y} w_j \delta_j \leq W, \qquad y \in Y \tag{42}$$

$$\delta_i + \delta_j - \ell_{ij} - \ell_{ji} \leq 1 \qquad (i,j) \in E \tag{43}$$

$$x_i - x_j + W\ell_{ij} \leq W - w_i \qquad (i,j) \in E \tag{44}$$

$$0 \leq x_j \leq W - w_j \qquad j \in N \tag{45}$$

$$x_j \geq 0 \qquad j \in N \tag{46}$$

$$\ell_{ij} \in \{0,1\} \qquad (i,j) \in E \tag{47}$$

$$\delta_j \in \{0,1\} \qquad j \in N \tag{48}$$

which is recognized as a generalization of (1)–(7), in which the profit of a request is not proportional to $w_j h_j$. The reduced cost of the assignment found by solving (41)–(48) is

$$r = 1 - \sum_{j \in N} v_j \delta_j.$$

If $r < 0$ we extend S' with the new assignment, i.e. setting $a_{ij} = \delta_j$ in the next column $i = |S'| + 1$ and extending S' accordingly. If $r \geq 0$ then the process terminates, and we have solved the LP-relaxation of (35) – (37).

5 Computational Results

We have implemented bounds and exact algorithms for the GSR-KP and the GSR-BPP as described in Sections 3 and 4. The algorithms and bounds have been implemented in ANSI C except bound u_2 which has been implemented in C++ since it makes use of some algorithms from LEDA 4.5 [20]. All tests have been performed on an Intel Pentium 4 with 2 GB of memory running at 3GHz. For all GSR-BPP tests we have limited the time consumption to 30 minutes. Tests not completed within this time limit are terminated with no result.

We start this section by looking at the upper bounds for the GSR-KP followed by the exact algorithm for the GSR-KP. Then we look at lower bounds for GSR-BPP and the performance of the exact algorithm for this problem. All computational times in the tables are in seconds.

5.1 Upper bounds for GSR-KP

Since we have no knowledge of any prior work on this problem new test instances for testing the algorithms had to be created. All instances were created by modifying the 2D knapsack packing instances also considered by Caprara and Monaci [7]. A detailed description of the `CGCUT` instances can be found in [8]. The `GCUT` instances are described in [1] and the `OKP` instances are described in [14]. Finally the `WANG` instances are described in [27].

We needed to add a starting station for all the reservations, while making sure that the ending stations were still valid. The CGCUT, OKP and WANG instances allowed several reservations of the same type. These reservations were split up into individual reservations and assigned separate starting stations. The starting stations are generated randomly and we therefore generate five new instances from each of the original instances to make sure we get some variation.

To evaluate the quality of u_1 we used CPLEX 7.0 [10] to solve the problem as a linear programming problem. u_3 is a multiple knapsack problem with no further constraints and can relatively easily be solved as an integer programming problem. Thus this bound is also solved using CPLEX. We have made own implementations of the upper bounds u_2, u_4 and u_5 described in Section 3.1.

We have compared the quality and performance of the bounds implemented and chosen the tightest bound and the fastest bound to be used in the bounding part of the exact algorithm.

The results for the upper bounds are summarized in Tables 1 and 2. In the first two columns we see the IP-optimal value found by CPLEX and the time used in seconds. The next columns are the time used and the relative gap in percent to the optimal value for each of the upper bounds.

u_3 is the tightest of the considered bounds, as it finds the optimal solution for all of the instances. For most problem instances u_1 and u_2 are the second best upper bounds and as expected u_1 and u_2 yield exactly the same bound values. u_2 is faster to calculate on the CGCUT, OKP and WANG instances. For these instances the train is relatively small, having at most 100 seats. For the GCUT instances u_1 performs better, being up to 10 seconds faster than u_2 on some instances. For these instances H and W are significantly larger. GCUT_13, which is the largest, has $H = W = 3000$, so the pseudo-polynomial nature of u_2 is clearly observed.

Table 3 gives an overview of the overall quality of the bounds. It is clear that u_3 is the tightest bound considered and the corresponding solution times are reasonable, so we will choose this bound as one of the bounds for the exact algorithm. It is also obvious that u_4 is significantly faster to calculate so we choose this upper bound as the second bound for the exact algorithm.

Due to the very high running times of u_2 on the large instances we shall not consider it further, but use u_1 for comparisons with the remaining bounds.

Figure 2 illustrates the quality of the different bounds without taking the time aspect into consideration. The *x*-axis indicates the gap in percent and the *y*-axis indicates the number of instances solved within the given gap. u_3 is hardly visible since all instances are solved with a gap of 0% to the optimal solution. Therefore the u_3 line is at 100% all of the time. It is seen that although bound u_1 on average provides upper bounds of very good quality, it has a problems closing the gap to the optimal solution for a number of instances.

5.2 Results from the GSR-KP

The results from the exact algorithm for the GSR-KP are summarized in Table 4. The algorithm is run twice for each instance, using the bounds u_3 and u_4. The results are compared to those of the CPLEX IP-solver.

	CPLEX		u_1		u_2		u_3		u_4		u_5	
Instance	Time	Opt	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap
CGCUT01_0	0.07	119	0.01	2.52	0.00	2.52	0.00	0.00	0.00	3.36	0.00	8.40
CGCUT01_1	0.03	124	0.03	8.06	0.00	8.06	0.01	0.00	0.00	9.68	0.00	13.71
CGCUT01_2	0.01	119	0.01	0.00	0.00	0.00	0.00	0.00	0.00	10.08	0.00	15.13
CGCUT01_3	0.06	107	0.01	2.80	0.00	2.80	0.00	0.00	0.00	13.08	0.00	17.76
CGCUT01_4	0.02	112	0.01	6.25	0.00	6.25	0.00	0.00	0.00	8.04	0.00	15.18
CGCUT02_0	2.24	2137	0.04	6.50	0.00	6.50	0.01	0.00	0.00	19.09	0.00	21.10
CGCUT02_1	1.24	2033	0.04	7.38	0.00	7.38	0.04	0.00	0.00	13.92	0.00	17.36
CGCUT02_2	0.97	2085	0.03	5.90	0.01	5.90	0.01	0.00	0.00	27.91	0.00	29.50
CGCUT02_3	0.42	2042	0.03	4.51	0.00	4.51	0.00	0.00	0.00	13.42	0.00	15.87
CGCUT02_4	1.44	1990	0.03	8.49	0.00	8.49	0.01	0.00	0.00	18.79	0.00	22.31
CGCUT03_0	2.07	2277	0.22	5.49	0.03	5.49	0.03	0.00	0.00	22.22	0.00	22.62
CGCUT03_1	0.31	2391	0.27	1.17	0.03	1.17	0.01	0.00	0.00	16.31	0.00	17.11
CGCUT03_2	3.84	2277	0.23	5.97	0.05	5.97	0.10	0.00	0.01	22.75	0.00	22.79
CGCUT03_3	2.38	2373	0.22	6.62	0.04	6.62	0.04	0.00	0.00	17.02	0.00	17.02
CGCUT03_4	1.44	2323	0.22	8.52	0.05	8.52	0.04	0.00	0.00	20.28	0.00	20.49
GCUT01_0	0.01	31404	0.00	32.94	0.03	32.94	0.01	0.00	0.00	32.31	0.00	33.09
GCUT01_1	0.00	36840	0.02	40.17	0.02	40.17	0.02	0.00	0.00	27.62	0.00	28.28
GCUT01_2	0.00	31404	0.01	44.22	0.02	44.22	0.02	0.00	0.00	38.94	0.00	39.72
GCUT01_3	0.01	36840	0.01	30.98	0.02	30.98	0.01	0.00	0.00	17.73	0.00	18.40
GCUT01_4	0.00	31404	0.01	38.96	0.02	38.96	0.01	0.00	0.00	33.72	0.00	34.51
GCUT02_0	0.05	40328	0.03	7.23	0.07	7.23	0.02	0.00	0.00	33.89	0.00	34.51
GCUT02_1	0.04	50934	0.05	8.30	0.03	8.30	0.02	0.00	0.00	14.43	0.00	14.92
GCUT02_2	0.09	42341	0.03	17.46	0.07	17.46	0.02	0.00	0.00	32.72	0.00	33.32
GCUT02_3	0.03	46568	0.05	14.68	0.04	14.68	0.02	0.00	0.00	17.05	0.00	17.59
GCUT02_4	0.03	49997	0.05	8.66	0.07	8.66	0.00	0.00	0.00	10.64	0.00	11.14
GCUT03_0	0.15	46614	0.08	14.62	0.10	14.62	0.02	0.00	0.00	26.33	0.00	26.87
GCUT03_1	0.05	51714	0.06	11.35	0.11	11.35	0.02	0.00	0.00	13.89	0.00	14.38
GCUT03_2	0.09	44963	0.08	9.25	0.12	9.25	0.02	0.00	0.00	28.18	0.00	28.74
GCUT03_3	0.13	47614	0.05	15.86	0.11	15.86	0.03	0.00	0.00	24.19	0.00	24.49
GCUT03_4	0.14	47053	0.07	14.24	0.09	14.24	0.03	0.00	0.00	24.85	0.00	25.39
GCUT04_0	0.63	52509	0.20	11.55	0.24	11.55	0.07	0.00	0.00	15.73	0.00	16.21
GCUT04_1	0.94	50937	0.19	13.43	0.24	13.43	0.09	0.00	0.00	19.03	0.00	19.32
GCUT04_2	0.48	52673	0.17	14.42	0.29	14.42	0.07	0.00	0.00	16.25	0.00	16.38
GCUT04_3	0.59	56225	0.21	2.55	0.27	2.55	0.04	0.00	0.00	8.63	0.00	9.08
GCUT04_4	0.31	50902	0.21	10.08	0.35	10.08	0.08	0.00	0.00	17.40	0.00	17.89
GCUT05_0	0.03	169939	0.03	17.31	0.03	17.31	0.00	0.00	0.00	33.28	0.02	33.58
GCUT05_1	0.03	169939	0.02	6.44	0.05	6.44	0.01	0.00	0.00	29.52	0.01	29.68
GCUT05_2	0.03	169939	0.02	9.73	0.04	9.73	0.01	0.00	0.00	26.58	0.01	26.87
GCUT05_3	0.02	169939	0.03	7.51	0.05	7.51	0.00	0.00	0.00	19.71	0.01	20.00
GCUT05_4	0.02	169939	0.02	9.73	0.03	9.73	0.00	0.00	0.00	27.75	0.01	28.04
GCUT06_0	0.06	193017	0.06	9.25	0.23	9.25	0.02	0.00	0.00	16.17	0.00	16.43
GCUT06_1	0.04	224996	0.06	2.66	0.15	2.66	0.03	0.00	0.00	8.30	0.01	8.53
GCUT06_2	0.05	169866	0.06	3.81	0.23	3.81	0.03	0.00	0.00	32.98	0.01	33.27
GCUT06_3	0.06	174603	0.07	11.78	0.21	11.78	0.03	0.00	0.00	29.64	0.01	29.93
GCUT06_4	0.05	191258	0.07	5.73	0.24	5.73	0.03	0.00	0.00	20.18	0.01	20.44
GCUT07_0	0.10	179087	0.12	12.26	0.69	12.26	0.02	0.00	0.00	23.45	0.01	23.73
GCUT07_1	0.06	191817	0.10	7.40	0.66	7.40	0.03	0.00	0.00	19.40	0.00	19.67
GCUT07_2	0.08	189689	0.12	17.15	0.59	17.15	0.03	0.00	0.00	24.51	0.01	24.78
GCUT07_3	0.08	206839	0.12	8.31	0.61	8.31	0.04	0.00	0.00	16.18	0.01	16.42
GCUT07_4	0.11	193296	0.09	16.99	0.65	16.99	0.04	0.00	0.00	19.25	0.01	19.31
GCUT08_0	1.04	202333	0.22	13.80	1.21	13.80	0.10	0.00	0.00	19.40	0.01	19.65
GCUT08_1	0.72	217062	0.21	5.58	1.51	5.58	0.11	0.00	0.00	12.52	0.01	12.75
GCUT08_2	0.64	215983	0.24	4.46	0.78	4.46	0.06	0.00	0.00	12.93	0.01	13.06
GCUT08_3	0.81	195827	0.21	8.14	1.83	8.14	0.07	0.00	0.00	19.14	0.00	19.40
GCUT08_4	0.29	213144	0.26	6.06	1.65	6.06	0.06	0.00	0.01	12.20	0.01	12.43

Table 1: Comparison of the five bounds for GSR-KP on the first half of the instances. The first column shows the instance name. The second column reports the time used in seconds by CPLEX for solving the problem to IP-optimality, and the found optimal solution. The remaining columns show the time required to compute the upper bound and the gap between it and the optimal solution for each of the bounds.

	CPLEX		u_1		u_2		u_3		u_4		u_5	
Instance	Time	Opt	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap
GCUT09_0	0.02	689944	0.04	19.11	0.15	19.11	0.02	0.00	0.00	21.09	0.03	21.24
GCUT09_1	0.01	608874	0.04	21.01	0.26	21.01	0.00	0.00	0.00	23.55	0.03	23.71
GCUT09_2	0.02	579884	0.03	28.01	0.27	28.01	0.02	0.00	0.01	33.88	0.03	34.05
GCUT09_3	0.01	674052	0.03	4.71	0.18	4.71	0.01	0.00	0.00	27.82	0.03	27.97
GCUT09_4	0.02	644262	0.06	8.54	0.25	8.54	0.01	0.00	0.00	23.68	0.03	23.83
GCUT10_0	0.07	684206	0.10	6.49	1.47	6.49	0.03	0.00	0.01	26.94	0.05	27.09
GCUT10_1	0.07	684206	0.12	10.62	1.19	10.62	0.04	0.00	0.00	32.84	0.04	32.99
GCUT10_2	0.07	684206	0.10	9.80	1.26	9.80	0.04	0.00	0.01	30.45	0.04	30.59
GCUT10_3	0.07	684206	0.11	11.83	1.18	11.83	0.05	0.00	0.00	30.94	0.03	31.08
GCUT10_4	0.08	684206	0.11	16.29	1.37	16.29	0.06	0.00	0.01	32.93	0.05	33.08
GCUT11_0	0.18	725928	0.13	11.08	2.62	11.08	0.07	0.00	0.00	27.23	0.05	27.37
GCUT11_1	0.21	823381	0.14	12.26	2.24	12.26	0.06	0.00	0.00	17.16	0.02	17.28
GCUT11_2	0.27	801744	0.16	12.41	1.95	12.41	0.08	0.00	0.00	17.71	0.01	17.83
GCUT11_3	0.34	738006	0.16	17.13	1.74	17.13	0.09	0.00	0.01	23.20	0.05	23.34
GCUT11_4	0.20	751244	0.17	14.00	2.71	14.00	0.08	0.00	0.00	23.84	0.04	23.97
GCUT12_0	0.50	886038	0.27	7.02	9.36	7.02	0.12	0.00	0.02	10.87	0.13	10.98
GCUT12_1	0.55	860063	0.28	7.49	8.21	7.49	0.12	0.00	0.03	10.12	0.15	10.24
GCUT12_2	0.70	839682	0.32	11.42	4.84	11.42	0.16	0.00	0.03	16.18	0.13	16.30
GCUT12_3	0.54	755831	0.32	16.49	5.59	16.49	0.16	0.00	0.03	24.79	0.14	24.92
GCUT12_4	0.37	835296	0.28	9.74	9.85	9.74	0.09	0.00	0.03	13.32	0.15	13.44
GCUT13_0	1.65	6306124	0.22	11.79	3.25	11.79	0.11	0.00	0.01	23.41	1.71	23.46
GCUT13_1	0.64	6235136	0.20	7.68	4.53	7.68	0.08	0.00	0.01	23.25	1.25	23.31
GCUT13_2	1.18	6162026	0.20	6.22	3.56	6.22	0.10	0.00	0.01	26.47	1.39	26.55
GCUT13_3	0.75	6634536	0.18	5.40	3.15	5.40	0.10	0.00	0.01	21.37	1.92	21.44
GCUT13_4	2.73	6131618	0.18	8.83	4.86	8.83	0.10	0.00	0.01	27.11	1.77	27.19
OKP01_0	5.82	8462	0.14	8.11	0.02	8.11	0.05	0.00	0.00	16.90	0.00	17.17
OKP01_1	20.61	8904	0.14	9.06	0.04	9.06	0.09	0.00	0.00	11.58	0.01	12.20
OKP01_2	4.98	8544	0.15	5.03	0.03	5.03	0.05	0.00	0.00	13.78	0.01	14.37
OKP01_3	4.79	8640	0.16	7.55	0.03	7.55	0.10	0.00	0.00	13.18	0.01	13.76
OKP01_4	5.54	8380	0.18	9.01	0.03	9.01	0.05	0.00	0.00	14.42	0.01	15.56
OKP02_0	0.54	7967	0.04	11.22	0.02	11.22	0.04	0.00	0.00	16.81	0.00	17.81
OKP02_1	0.23	7881	0.05	7.13	0.01	7.13	0.02	0.00	0.00	10.86	0.00	11.78
OKP02_2	0.37	8335	0.04	10.19	0.01	10.19	0.03	0.00	0.00	16.09	0.00	17.07
OKP02_3	0.37	8498	0.04	11.04	0.01	11.04	0.04	0.00	0.00	15.53	0.00	15.56
OKP02_4	0.40	7845	0.05	13.54	0.01	13.54	0.03	0.00	0.00	23.11	0.01	24.19
OKP03_0	0.18	7830	0.04	18.35	0.02	18.35	0.05	0.00	0.00	23.14	0.00	23.28
OKP03_1	0.25	7767	0.04	12.59	0.02	12.59	0.02	0.00	0.00	21.94	0.00	23.12
OKP03_2	0.14	7880	0.05	15.94	0.02	15.94	0.02	0.00	0.00	22.20	0.00	23.38
OKP03_3	0.28	7846	0.07	18.17	0.02	18.17	0.04	0.00	0.00	25.68	0.00	26.87
OKP03_4	0.25	7408	0.05	15.59	0.02	15.59	0.03	0.00	0.00	24.61	0.00	25.85
OKP04_0	12.63	8848	0.20	7.15	0.05	7.15	0.16	0.00	0.00	10.04	0.00	11.42
OKP04_1	5.69	9356	0.22	4.42	0.05	4.42	0.15	0.00	0.00	6.87	0.01	6.87
OKP04_2	24.54	9045	0.19	5.78	0.05	5.78	0.12	0.00	0.00	8.27	0.01	8.27
OKP04_3	2.91	9226	0.18	5.54	0.05	5.54	0.05	0.00	0.00	7.00	0.01	7.00
OKP04_4	8.55	8660	0.23	10.35	0.05	10.35	0.11	0.00	0.00	12.91	0.00	14.55
OKP05_0	162.51	9793	0.49	1.59	0.10	1.59	0.10	0.00	0.01	2.06	0.01	2.07
OKP05_1	6.03	9799	0.48	1.65	0.10	1.65	0.06	0.00	0.00	1.84	0.01	1.96
OKP05_2	8.84	9787	0.49	1.20	0.10	1.20	0.02	0.00	0.00	1.96	0.01	2.18
OKP05_3	5.42	9787	0.42	1.66	0.10	1.66	0.06	0.00	0.00	2.15	0.01	2.17
OKP05_4	33.06	9793	0.47	1.65	0.09	1.65	0.06	0.00	0.00	2.09	0.01	2.10
WANG20_0	0.16	2190	0.10	17.76	0.01	17.76	0.02	0.00	0.00	24.06	0.01	24.57
WANG20_1	0.11	2237	0.10	12.65	0.01	12.65	0.01	0.00	0.00	16.18	0.01	17.97
WANG20_2	0.16	2378	0.08	6.85	0.01	6.85	0.03	0.00	0.00	12.45	0.00	12.66
WANG20_3	0.04	2398	0.12	12.55	0.01	12.55	0.02	0.00	0.00	12.39	0.01	12.59
WANG20_4	0.10	2166	0.08	17.17	0.01	17.17	0.03	0.00	0.00	22.25	0.01	22.62

Table 2: Comparison of the five bounds for GSR-KP on the second half of the instances. The first column shows the instance name. The second column reports the time used by CPLEX in seconds for solving the problem to IP-optimality, and the found optimal solution. The remaining columns show the time required to compute the upper bound and the gap between it and the optimal solution for each of the bounds.

Bound	CPLEX-IP	u_1	u_2	u_3	u_4	u_5
Average time	3.19	0.13	0.83	0.05	0.00	0.09
Spread / time	16.03	0.11	1.81	0.04	0.01	0.34
Average gap	0.00	10.92	10.92	0.00	19.23	19.93
Spread / gap	0.00	7.72	7.72	0.00	8.32	8.12

Table 3: Overall comparison of the five bounds for GSR-KP. Average time (in seconds) and average gap in percent to the optimal solution and the matching spread for the CPLEX IP-solver and the five upper bounds.

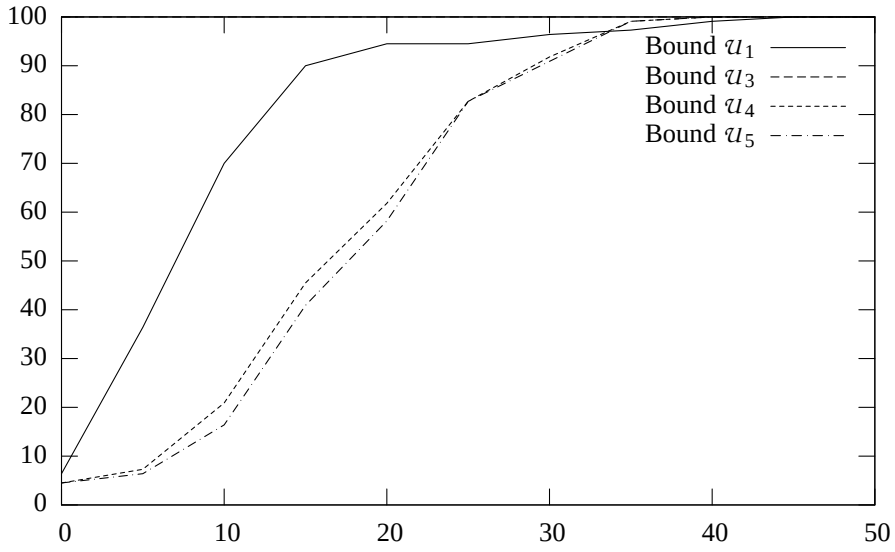


Figure 2: Quality of the implemented upper bounds. "u_1" represents u_1 and the identical u_2. The *x*-axis indicates the gap to the optimal solution in percent, and the *y*-axis indicates how many instances are solved within the given gap (sampled in 5% intervals)

Instance	Requests	Opt	CPLEX time	A_3 time	A_4 time	Instance	Requests	Opt	CPLEX time	A_3 time	A_4 time
CGCUT01_0	16	119	0.07	0.01	**0.00**	CGCUT02_0	23	2137	2.24	**0.17**	0.67
CGCUT01_1	16	124	0.03	**0.01**	**0.01**	CGCUT02_1	23	2033	1.24	**0.10**	0.79
CGCUT01_2	16	119	**0.01**	**0.01**	**0.01**	CGCUT02_2	23	2085	0.97	**0.08**	1.10
CGCUT01_3	16	107	0.06	**0.01**	0.02	CGCUT02_3	23	2042	0.42	**0.09**	0.39
CGCUT01_4	16	112	0.02	**0.01**	**0.01**	CGCUT02_4	23	1990	1.44	**0.12**	0.84
CGCUT03_0	62	2277	2.07	**1.35**	5.48	GCUT01_0	10	31404	**0.01**	0.04	**0.01**
CGCUT03_1	62	2391	**0.31**	0.81	6.14	GCUT01_1	10	36840	**0.00**	0.05	0.03
CGCUT03_2	62	2277	3.84	**2.05**	6.43	GCUT01_2	10	31404	**0.00**	0.03	0.03
CGCUT03_3	62	2373	2.38	**1.49**	4.46	GCUT01_3	10	36840	**0.01**	0.05	0.03
CGCUT03_4	62	2323	**1.44**	1.60	5.78	GCUT01_4	10	31404	**0.00**	0.03	0.03
GCUT02_0	20	40328	**0.05**	0.13	0.22	GCUT03_0	30	46614	**0.15**	0.41	0.97
GCUT02_1	20	50934	**0.04**	0.11	0.16	GCUT03_1	30	51714	**0.05**	0.44	0.90
GCUT02_2	20	42341	**0.09**	0.22	0.32	GCUT03_2	30	44963	**0.09**	0.43	0.64
GCUT02_3	20	46568	**0.03**	0.08	0.25	GCUT03_3	30	47614	**0.13**	0.38	0.71
GCUT02_4	20	49997	**0.03**	0.08	0.07	GCUT03_4	30	47053	**0.14**	0.21	0.75
GCUT04_0	50	52509	**0.63**	1.66	6.35	GCUT05_0	10	169939	**0.03**	0.06	0.10
GCUT04_1	50	50937	**0.94**	2.13	9.02	GCUT05_1	10	169939	**0.03**	0.06	0.09
GCUT04_2	50	52673	**0.48**	0.92	4.75	GCUT05_2	10	169939	**0.03**	0.05	0.07
GCUT04_3	50	56225	**0.59**	1.71	5.25	GCUT05_3	10	169939	**0.02**	0.05	0.07
GCUT04_4	50	50902	**0.31**	1.58	4.85	GCUT05_4	10	169939	**0.02**	0.05	0.09
GCUT06_0	20	193017	**0.06**	0.28	0.45	GCUT07_0	30	179087	**0.10**	0.51	0.70
GCUT06_1	20	224996	**0.04**	0.20	0.40	GCUT07_1	30	191817	**0.06**	0.64	1.29
GCUT06_2	20	169866	**0.05**	0.22	0.58	GCUT07_2	30	189689	**0.08**	0.71	1.29
GCUT06_3	20	174603	**0.06**	0.29	0.67	GCUT07_3	30	206839	**0.08**	0.76	0.76
GCUT06_4	20	191258	**0.05**	0.35	0.61	GCUT07_4	30	193296	**0.11**	0.67	1.28
GCUT08_0	50	202333	**1.04**	3.53	15.05	GCUT09_0	10	689944	**0.02**	0.03	0.14
GCUT08_1	50	217062	**0.72**	3.50	14.12	GCUT09_1	10	608874	**0.01**	0.07	0.11
GCUT08_2	50	215983	**0.64**	2.26	8.73	GCUT09_2	10	579884	**0.02**	0.05	0.11
GCUT08_3	50	195827	**0.81**	2.00	16.79	GCUT09_3	10	674052	**0.01**	0.06	0.10
GCUT08_4	50	213144	**0.29**	2.29	9.54	GCUT09_4	10	644262	**0.02**	0.04	0.11
GCUT10_0	20	684206	**0.07**	0.27	0.79	GCUT11_0	30	725928	**0.18**	2.19	5.56
GCUT10_1	20	684206	**0.07**	0.29	1.07	GCUT11_1	30	823381	**0.21**	1.56	3.23
GCUT10_2	20	684206	**0.07**	0.28	0.83	GCUT11_2	30	801744	**0.27**	2.02	4.21
GCUT10_3	20	684206	**0.07**	0.31	0.96	GCUT11_3	30	738006	**0.34**	1.79	5.62
GCUT10_4	20	684206	**0.08**	0.32	1.00	GCUT11_4	30	751244	**0.20**	0.61	4.32
GCUT12_0	50	886038	**0.50**	5.20	14.97	GCUT13_0	32	6306124	**1.65**	4.69	145.29
GCUT12_1	50	860063	**0.55**	3.89	16.35	GCUT13_1	32	6235136	**0.64**	1.36	45.87
GCUT12_2	50	839682	**0.70**	4.77	22.63	GCUT13_2	32	6162026	**1.18**	4.29	162.28
GCUT12_3	50	755831	**0.54**	3.64	21.42	GCUT13_3	32	6634536	**0.75**	4.11	83.77
GCUT12_4	50	835296	**0.37**	3.51	10.06	GCUT13_4	32	6131618	**2.73**	3.00	81.02
OKP01_0	50	8462	5.82	**1.23**	5.93	OKP02_0	30	7967	0.54	**0.14**	0.26
OKP01_1	50	8904	20.61	**0.74**	5.26	OKP02_1	30	7881	0.23	**0.14**	0.15
OKP01_2	50	8544	4.98	**1.30**	5.87	OKP02_2	30	8335	0.37	0.29	**0.27**
OKP01_3	50	8640	4.79	**1.65**	4.75	OKP02_3	30	8498	0.37	**0.24**	0.31
OKP01_4	50	8380	5.54	**1.02**	4.89	OKP02_4	30	7845	0.40	**0.13**	0.34
OKP03_0	30	7830	**0.18**	0.26	0.26	OKP04_0	61	8848	12.63	**1.05**	9.11
OKP03_1	30	7767	0.25	**0.10**	0.28	OKP04_1	61	9356	5.69	**3.29**	11.45
OKP03_2	30	7880	**0.14**	0.22	0.33	OKP04_2	61	9045	24.54	**2.70**	11.13
OKP03_3	30	7846	**0.28**	0.31	0.38	OKP04_3	61	9226	2.91	**0.99**	3.55
OKP03_4	30	7408	0.25	**0.20**	0.26	OKP04_4	61	8660	8.55	**1.06**	4.86
OKP05_0	97	9793	162.51	**7.27**	111.31	WANG20_0	42	2190	**0.16**	0.19	0.43
OKP05_1	97	9799	**6.03**	8.06	1332.44	WANG20_1	42	2237	**0.11**	0.48	0.52
OKP05_2	97	9787	8.84	**8.05**	302.16	WANG20_2	42	2378	**0.16**	0.42	0.24
OKP05_3	97	9787	**5.42**	5.77	385.83	WANG20_3	42	2398	**0.04**	0.20	0.40
OKP05_4	97	9793	33.06	**7.69**	1303.85	WANG20_4	42	2166	**0.10**	0.38	0.46

Table 4: Comparison between CPLEX IP-solver and the exact algorithm for the GSR-KP. The columns show the instance name, the number of request and the optimal solution to the problem. Next, we report the time used by CPLEX in seconds and the exact algorithm to compute the solution. A_3 is the exact algorithm using $\mathcal{U}_3$ and A_4 is the exact algorithm using $\mathcal{U}_4$. The fastest time for each instance is shown in bold face.

The algorithm is able to solve all the test instances using both of the applied bounds. Using $\mathcal{U}_3$ all instances are solved within reasonable time. A slight increase in computational time is noted as the number of requests increase, up to slightly more than eight seconds for `OKP_05` which has 97 requests. The algorithm using $\mathcal{U}_4$ also shows good computational times for the smaller instances, but degrades significantly for the larger instances. This seems reasonable as $\mathcal{U}_4$ is not very tight, so the solution space becomes too large, even though the bound is faster to compute.

It is clear from the table that the performance of our algorithm depends on the type of problem instance it is solving. With bound $\mathcal{U}_3$ our algorithm is faster than the CPLEX IP-solver for most `CGCUT` and `OKP` instances. This is an interesting result, as the `OKP` were introduced by Fekete and Schepers as particularly difficult instances. For the `GCUT` and `WANG` instances the CPLEX IP-solver is faster than our algorithm and for some of them quite considerably. Our algorithm is the fastest at solving 30 out of 100 instances when using bound $\mathcal{U}_3$. In six of the instances the algorithm is the fastest when using bound $\mathcal{U}_4$.

	CPLEX	A_3	A_4
Total time	351.38	141.60	4279.25
Average time	3.19	1.29	38.90
Spread	16.03	1.88	182.44

Table 5: Summary of results for the CPLEX IP-solver and the exact algorithm for the GSR-KP. A_3 and A_4 denotes the exact algorithm using bounds $\mathcal{U}_3$ and $\mathcal{U}_4$ respectively. Times are in seconds.

Table 5 summarizes the results in Table 4. The exact algorithm using $\mathcal{U}_3$ displays the lowest total running time despite being fastest on only 30 of the 110 instances. It also has a low standard deviation (spread) indicating that this algorithm is robust to the different instances. Although being fastest on the majority of the instances, the CPLEX IP-solver has a higher total running time and a larger standard deviation. This indicates that the CPLEX IP-solver is more susceptible to fluctuations in the solution time required. This is most obvious on `OKP05_0` where the CPLEX IP-solver uses 162 seconds, but also `OKP01_1`, `OKP04_2` and `OKP05_4` displays this tendency.

5.3 Lower bounds for the GSR-BPP

The test instances used in the GSR-BPP is a further modification of the instances from Caprara and Monaci [7] used in the GSR-KP. The maximum number of available seats in the train is ignored, and instead we introduce W, the number of seats in a train car. The train car sizes chosen are 10, 20 and 40. To make the new instances valid we adjust the number of seats asked for by a request. To ensure $0 < w_j \leq W$ we set

$$w_j = \begin{cases} W & \text{if } w_j \bmod W = 0, \\ w_j \bmod W & \text{otherwise.} \end{cases}$$

Of the lower bounds considered $\mathcal{L}_1$ is the LP-relaxation of the problem. We saw in section 4.1 that $\mathcal{L}_1 = 0$ for all instances, so this was not implemented. $\mathcal{L}_2$ is an ordinary

bin packing problem and hence it will be $\mathcal{NP}$-hard to compute this bound. $\mathcal{L}_3$ is a lower bound of $\mathcal{L}_2$ which is polynomially solvable. It is quickly computed and is generally expected to produce very tight lower bounds (see e.g. [24]). $\mathcal{L}_4$ is similar to $\mathcal{L}_2$ with some additional constraints and hence still $\mathcal{NP}$-hard to solve. This led us to suspect that $\mathcal{L}_4$ would be somewhat slower than $\mathcal{L}_2$ but possibly give tighter bounds.

In a branch and bound algorithm the time required to calculate a bound is very important. $\mathcal{L}_3$ is expected to be considerably faster than the other two and is at the same time believed to give quite good bounds. Consequently only $\mathcal{L}_3$ was chosen as the lower bound implemented for use in the exact algorithm.

The results of the lower bound applied to the test instances are shown in Table 6. As can be seen all bounds were calculated using a negligible amount of time. Moreover, in the cases where we have an optimal solution, $\mathcal{L}_3$ matched the upper bound in every instance except one (`GCUT10_1` - bin size 40), indicating that $\mathcal{L}_3$ is very tight for the considered instances.

5.4 Results from the GSR-BPP

Since we had no previous results to compare to, we used the CPLEX IP-solver as a reference solution. CPLEX displayed some difficulties solving the test instances. In fact the CPLEX IP-solver was only able to solve between 18 and 20 (depending on bin size) of the 110 instances in less than 30 minutes. On some of the instances CPLEX was imposed a time limit of 18 hours but was still unable to solve the instances.

The exact algorithm was not able to solve all of the instances, but performed significantly better than the CPLEX IP-solver as can be seen in Table 7 and Table 8. It solved between 49 and 54 of the instances to optimality within the 30 minute time limit. Moreover, for the instances where both algorithms solved the problem to optimality, the presented exact algorithm was considerably faster than CPLEX.

The complexity of the problem depends in large parts on the number of requests. The CPLEX IP-solver is able to solve all instances with 10 requests and most of the instances with 16 requests but none of the instances with more requests. The presented exact algorithm solves most of the instances with up to about 30 requests and a few with more than 30 requests.

It is interesting to notice that the considered instances are either solved very fast or not at all (given the imposed time limit). Considering that $\mathcal{L}_3$ was able to find the correct solution to almost every instance which was solved to optimality and in very little time, we expect that it is reasonably easy to find a good solution, but quite difficult to prove optimality. In many cases both the CPLEX IP-solver and the exact algorithm probably have found an optimal solution, but are unable to prove optimality within the given time frame.

6 Further Work

The GSR-KP considers the profit of each reservation to be the product of the number of seats occupied and the distance travelled. A more general approach would be to represent the profit of a reservation as a separate parameter, as is the case in two-dimensional

	Bin size 10		Bin size 20		Bin size 40			Bin size 10		Bin size 20		Bin size 40	
Instance	Bound	time	Bound	time	Bound	time	Instance	Bound	time	Bound	time	Bound	time
CGCUT01_0	3	0.00	2	0.00	1	0.00	CGCUT02_0	10	0.00	7	0.00	5	0.00
CGCUT01_1	3	0.00	2	0.00	1	0.00	CGCUT02_1	10	0.00	7	0.00	5	0.00
CGCUT01_2	3	0.00	2	0.00	1	0.00	CGCUT02_2	9	0.00	7	0.00	6	0.00
CGCUT01_3	4	0.00	2	0.00	1	0.00	CGCUT02_3	9	0.00	7	0.00	6	0.00
CGCUT01_4	4	0.00	2	0.00	1	0.00	CGCUT02_4	10	0.00	7	0.00	6	0.00
CGCUT03_0	27	0.00	40	0.01	39	0.00	GCUT01_0	6	0.00	5	0.01	7	0.00
CGCUT03_1	28	0.00	39	0.01	40	0.00	GCUT01_1	4	0.00	4	0.01	5	0.00
CGCUT03_2	27	0.01	38	0.01	39	0.00	GCUT01_2	5	0.00	4	0.00	5	0.00
CGCUT03_3	27	0.01	39	0.01	39	0.00	GCUT01_3	5	0.00	4	0.00	6	0.00
CGCUT03_4	27	0.01	39	0.01	40	0.00	GCUT01_4	5	0.00	5	0.01	6	0.00
GCUT02_0	14	0.01	14	0.01	13	0.00	GCUT03_0	15	0.01	14	0.01	14	0.00
GCUT02_1	12	0.01	13	0.01	12	0.00	GCUT03_1	15	0.01	15	0.01	14	0.00
GCUT02_2	13	0.01	12	0.01	12	0.00	GCUT03_2	16	0.01	15	0.01	15	0.00
GCUT02_3	13	0.00	13	0.00	13	0.00	GCUT03_3	14	0.01	14	0.01	15	0.00
GCUT02_4	13	0.01	14	0.01	13	0.00	GCUT03_4	15	0.01	15	0.01	15	0.00
GCUT04_0	23	0.01	22	0.01	23	0.00	GCUT05_0	5	0.01	5	0.01	6	0.00
GCUT04_1	25	0.01	24	0.01	25	0.00	GCUT05_1	6	0.01	6	0.01	6	0.00
GCUT04_2	25	0.01	24	0.01	23	0.00	GCUT05_2	6	0.01	5	0.01	6	0.00
GCUT04_3	24	0.01	22	0.00	23	0.00	GCUT05_3	6	0.01	5	0.01	6	0.00
GCUT04_4	28	0.01	27	0.01	26	0.00	GCUT05_4	6	0.01	5	0.01	6	0.00
GCUT06_0	10	0.01	12	0.01	8	0.00	GCUT07_0	16	0.00	16	0.00	18	0.01
GCUT06_1	10	0.01	11	0.01	7	0.00	GCUT07_1	14	0.01	14	0.01	17	0.01
GCUT06_2	11	0.01	12	0.01	8	0.00	GCUT07_2	15	0.01	16	0.01	17	0.00
GCUT06_3	10	0.01	12	0.01	8	0.00	GCUT07_3	15	0.01	14	0.01	17	0.00
GCUT06_4	11	0.00	12	0.01	8	0.00	GCUT07_4	16	0.01	15	0.01	17	0.00
GCUT08_0	27	0.00	24	0.01	20	0.01	GCUT09_0	5	0.01	4	0.01	4	0.00
GCUT08_1	26	0.01	21	0.01	20	0.01	GCUT09_1	5	0.01	4	0.01	5	0.00
GCUT08_2	26	0.01	22	0.00	19	0.01	GCUT09_2	6	0.01	5	0.01	4	0.00
GCUT08_3	27	0.01	23	0.00	21	0.01	GCUT09_3	5	0.01	4	0.01	5	0.00
GCUT08_4	29	0.00	25	0.01	20	0.01	GCUT09_4	6	0.01	5	0.01	4	0.00
GCUT10_0	14	0.01	12	0.01	10	0.02	GCUT11_0	16	0.01	15	0.01	18	0.01
GCUT10_1	13	0.01	11	0.01	9	0.01	GCUT11_1	15	0.01	16	0.02	17	0.01
GCUT10_2	14	0.01	12	0.00	10	0.01	GCUT11_2	14	0.00	14	0.01	16	0.01
GCUT10_3	14	0.01	12	0.01	10	0.01	GCUT11_3	15	0.00	15	0.02	16	0.01
GCUT10_4	13	0.01	11	0.01	9	0.01	GCUT11_4	14	0.01	14	0.01	18	0.01
GCUT12_0	31	0.02	25	0.01	28	0.02	GCUT13_0	9	0.01	7	0.02	6	0.01
GCUT12_1	30	0.01	26	0.01	27	0.03	GCUT13_1	11	0.01	11	0.01	10	0.01
GCUT12_2	29	0.02	26	0.02	28	0.03	GCUT13_2	9	0.01	7	0.02	6	0.01
GCUT12_3	29	0.02	25	0.01	27	0.03	GCUT13_3	8	0.01	7	0.01	6	0.01
GCUT12_4	31	0.01	27	0.01	28	0.03	GCUT13_4	9	0.01	7	0.02	6	0.01
OKP01_0	16	0.00	11	0.01	14	0.00	OKP02_0	8	0.00	10	0.01	8	0.00
OKP01_1	16	0.01	11	0.01	11	0.00	OKP02_1	9	0.00	11	0.01	8	0.00
OKP01_2	15	0.01	10	0.01	11	0.00	OKP02_2	9	0.00	10	0.00	8	0.00
OKP01_3	16	0.00	11	0.00	12	0.00	OKP02_3	7	0.00	8	0.01	6	0.00
OKP01_4	16	0.00	11	0.01	12	0.00	OKP02_4	9	0.00	10	0.00	8	0.00
OKP03_0	14	0.00	8	0.01	8	0.00	OKP04_0	21	0.00	18	0.01	15	0.00
OKP03_1	10	0.00	6	0.01	7	0.00	OKP04_1	20	0.00	18	0.01	14	0.00
OKP03_2	13	0.00	7	0.01	10	0.00	OKP04_2	19	0.00	18	0.01	14	0.00
OKP03_3	14	0.00	9	0.01	9	0.00	OKP04_3	21	0.00	20	0.01	16	0.00
OKP03_4	11	0.00	6	0.01	8	0.00	OKP04_4	24	0.00	22	0.01	18	0.00
OKP05_0	27	0.00	27	0.01	19	0.00	WANG20_0	16	0.00	14	0.01	20	0.00
OKP05_1	24	0.00	27	0.00	17	0.00	WANG20_1	13	0.00	14	0.01	20	0.00
OKP05_2	25	0.00	27	0.01	18	0.00	WANG20_2	13	0.00	12	0.01	19	0.00
OKP05_3	26	0.00	27	0.01	19	0.00	WANG20_3	15	0.00	12	0.01	17	0.00
OKP05_4	23	0.00	23	0.01	14	0.00	WANG20_4	15	0.00	14	0.01	18	0.00

Table 6: Computational results for $\mathcal{L}_3$, illustrating the value of the lower bound and the time required to compute it in seconds, using three different bin sizes.

		Bin size 10			Bin size 20			Bin size 40		
Instance	Requests	Opt	CPLEX time	b&b time	Opt	CPLEX time	b&b time	Opt	CPLEX time	b&b time
CGCUT01_0	16	3	8.25	0.00	2	0.45	0.14	1	0.07	0.00
CGCUT01_1	16	3	0.58	0.00	2	0.47	0.07	1	0.07	0.00
CGCUT01_2	16	3	0.36	0.00	2	0.46	0.08	1	0.06	0.00
CGCUT01_3	16	4	-	0.00	2	0.29	0.07	1	0.07	0.00
CGCUT01_4	16	4	-	0.00	2	0.34	0.05	1	0.08	0.00
CGCUT02_0	23	-	-	-	7	-	4.77	5	-	0.02
CGCUT02_1	23	10	-	326.58	7	-	0.07	-	-	-
CGCUT02_2	23	-	-	-	7	-	0.03	6	-	0.02
CGCUT02_3	23	-	-	-	7	-	0.04	6	-	0.06
CGCUT02_4	23	10	-	351.74	7	-	0.04	6	-	9.36
CGCUT03_0	62	-	-	-	-	-	-	-	-	-
CGCUT03_1	62	-	-	-	-	-	-	-	-	-
CGCUT03_2	62	-	-	-	-	-	-	-	-	-
CGCUT03_3	62	-	-	-	-	-	-	-	-	-
CGCUT03_4	62	-	-	-	-	-	-	-	-	-
GCUT01_0	10	6	1.10	0.00	5	0.43	0.00	7	9.42	0.01
GCUT01_1	10	4	0.13	0.00	4	0.21	0.00	5	0.88	0.01
GCUT01_2	10	5	5.54	0.00	4	0.33	0.00	5	0.71	0.00
GCUT01_3	10	5	0.22	0.00	4	2.64	0.00	6	4.15	0.03
GCUT01_4	10	5	0.35	0.01	5	0.70	0.00	6	7.08	0.00
GCUT02_0	20	14	-	6.69	14	-	1.19	13	-	0.23
GCUT02_1	20	12	-	56.71	13	-	6.41	12	-	6.55
GCUT02_2	20	13	-	11.26	12	-	0.04	12	-	0.37
GCUT02_3	20	13	-	42.44	13	-	2.84	13	-	4.04
GCUT02_4	20	13	-	2.46	14	-	1.01	13	-	0.65
GCUT03_0	30	-	-	-	-	-	-	-	-	-
GCUT03_1	30	-	-	-	-	-	-	-	-	-
GCUT03_2	30	-	-	-	-	-	-	-	-	-
GCUT03_3	30	14	-	167.27	-	-	-	-	-	-
GCUT03_4	30	-	-	-	-	-	-	15	-	978.01
GCUT04_0	50	-	-	-	-	-	-	-	-	-
GCUT04_1	50	-	-	-	-	-	-	-	-	-
GCUT04_2	50	-	-	-	-	-	-	-	-	-
GCUT04_3	50	-	-	-	-	-	-	-	-	-
GCUT04_4	50	-	-	-	-	-	-	-	-	-
GCUT05_0	10	5	0.42	0.00	5	4.23	0.01	6	7.54	0.01
GCUT05_1	10	6	6.54	0.01	6	2.84	0.01	6	1.41	0.00
GCUT05_2	10	6	5.74	0.01	5	4.07	0.00	6	3.98	0.02
GCUT05_3	10	6	1.01	0.02	5	1.88	0.00	6	0.46	0.00
GCUT05_4	10	6	3.17	0.00	5	12.67	0.01	6	1.02	0.00
GCUT06_0	20	10	-	0.03	12	-	2.27	8	-	5.74
GCUT06_1	20	10	-	0.50	11	-	0.11	-	-	-
GCUT06_2	20	11	-	0.05	12	-	0.06	-	-	-
GCUT06_3	20	10	-	0.03	12	-	1.16	8	-	0.11
GCUT06_4	20	11	-	0.94	12	-	0.06	8	-	62.14
GCUT07_0	30	-	-	-	-	-	-	-	-	-
GCUT07_1	30	14	-	9.19	14	-	75.70	-	-	-
GCUT07_2	30	-	-	-	-	-	-	-	-	-
GCUT07_3	30	15	-	24.92	14	-	4.48	-	-	-
GCUT07_4	30	-	-	-	-	-	-	-	-	-
GCUT08_0	50	-	-	-	-	-	-	-	-	-
GCUT08_1	50	-	-	-	-	-	-	-	-	-
GCUT08_2	50	-	-	-	-	-	-	-	-	-
GCUT08_3	50	-	-	-	-	-	-	-	-	-
GCUT08_4	50	-	-	-	-	-	-	-	-	-

Table 7: Comparison between CPLEX IP-solver and the proposed branch-and-bound algorithm on the first half of the GSR-BPP instances. We report the instance name and the number of requests for the given instance. For each of the three bin sizes we report the optimal solution and the running time in seconds for CPLEX and our algorithm. A hyphen indicates that the instance could not be solved within a 30 minute time limit.

		Bin size 10			Bin size 20			Bin size 40		
Instance	Requests	Opt	CPLEX time	b&b time	Opt	CPLEX time	b&b time	Opt	CPLEX time	b&b time
GCUT09_0	10	5	0.18	0.00	4	0.87	0.00	4	0.08	0.00
GCUT09_1	10	5	12.62	0.00	4	1.63	0.01	5	0.46	0.01
GCUT09_2	10	6	8.03	0.01	5	3.41	0.00	4	0.10	0.01
GCUT09_3	10	5	0.16	0.01	4	0.09	0.01	5	2.55	0.02
GCUT09_4	10	6	25.64	0.01	5	1.11	0.01	4	1.99	0.01
GCUT10_0	20	14	-	0.32	12	-	16.69	10	-	0.06
GCUT10_1	20	13	-	0.56	11	-	17.53	10	-	647.89
GCUT10_2	20	14	-	0.35	12	-	18.01	10	-	0.05
GCUT10_3	20	14	-	2.23	12	-	162.85	10	-	0.06
GCUT10_4	20	13	-	0.15	11	-	0.06	9	-	0.05
GCUT11_0	30	-	-	-	-	-	-	-	-	-
GCUT11_1	30	-	-	-	-	-	-	17	-	245.09
GCUT11_2	30	-	-	-	-	-	-	-	-	-
GCUT11_3	30	-	-	-	-	-	-	-	-	-
GCUT11_4	30	14	-	557.32	-	-	-	-	-	-
GCUT12_0	50	-	-	-	-	-	-	-	-	-
GCUT12_1	50	-	-	-	-	-	-	-	-	-
GCUT12_2	50	-	-	-	-	-	-	28	-	495.56
GCUT12_3	50	-	-	-	-	-	-	-	-	-
GCUT12_4	50	-	-	-	-	-	-	-	-	-
GCUT13_0	32	9	-	336.81	7	-	0.08	6	-	0.05
GCUT13_1	32	-	-	-	11	-	653.65	-	-	-
GCUT13_2	32	-	-	-	7	-	0.06	6	-	5.21
GCUT13_3	32	8	-	2.48	7	-	0.09	6	-	0.06
GCUT13_4	32	-	-	-	-	-	-	-	-	-
OKP01_0	50	-	-	-	-	-	-	-	-	-
OKP01_1	50	-	-	-	-	-	-	-	-	-
OKP01_2	50	-	-	-	-	-	-	-	-	-
OKP01_3	50	-	-	-	-	-	-	-	-	-
OKP01_4	50	-	-	-	-	-	-	-	-	-
OKP02_0	30	8	-	142.82	10	-	70.94	8	-	249.07
OKP02_1	30	-	-	-	11	-	0.07	-	-	-
OKP02_2	30	9	-	0.06	10	-	0.06	8	-	8.16
OKP02_3	30	-	-	-	8	-	523.68	-	-	-
OKP02_4	30	-	-	-	-	-	-	-	-	-
OKP03_0	30	14	-	0.15	-	-	-	8	-	0.05
OKP03_1	30	-	-	-	-	-	-	7	-	0.04
OKP03_2	30	-	-	-	-	-	-	10	-	52.78
OKP03_3	30	14	-	11.72	-	-	-	9	-	0.06
OKP03_4	30	11	-	3.42	-	-	-	8	-	0.05
OKP04_0	61	21	-	1.77	-	-	-	15	-	0.80
OKP04_1	61	-	-	-	-	-	-	-	-	-
OKP04_2	61	-	-	-	-	-	-	-	-	-
OKP04_3	61	-	-	-	-	-	-	-	-	-
OKP04_4	61	-	-	-	-	-	-	18	-	1.11
OKP05_0	97	-	-	-	-	-	-	-	-	-
OKP05_1	97	-	-	-	-	-	-	-	-	-
OKP05_2	97	-	-	-	-	-	-	-	-	-
OKP05_3	97	-	-	-	-	-	-	-	-	-
OKP05_4	97	-	-	-	-	-	-	-	-	-
WANG20_0	42	-	-	-	-	-	-	-	-	-
WANG20_1	42	-	-	-	-	-	-	20	-	1.37
WANG20_2	42	-	-	-	-	-	-	-	-	-
WANG20_3	42	-	-	-	-	-	-	-	-	-
WANG20_4	42	-	-	-	-	-	-	18	-	19.26

Table 8: Comparison between CPLEX IP-solver and the proposed branch-and-bound algorithm on the second half of the GSR-BPP instances. We report the instance name and the number of requests for the given instance. For each of the three bin sizes we report the optimal solution and the running time in seconds for CPLEX and our algorithm. A hyphen indicates that the instance could not be solved within a 30 minute time limit.

knapsack problems. In this way, route segments can be priced individually, which more realistically models the pricing presently done by many railway companies. However, the upper bounds presented cannot be used without modification.

For the GSR-BPP, no upper bounds have been considered. As the feasibility computations are much more efficient for the GSR-BPP than for ordinary two-dimensional packing problems, it is not known what effect an upper bound would have on the efficiency of the exact algorithm presented.

The Dantzig Wolfe decomposition formulation stated in Section 4.3 has not been tested empirically. As the formulation will lead to tighter bounds one could hope that even more instances could be solved to optimality by this approach.

7 Conclusion

This is, to the best of our knowledge, the first paper to study the exact solution of the off-line GSRP. Two variants have been considered: the GSR-KP which is a variant of the 2DKP and the GSR-BPP which is a variant of the 2DBPP. For each problem, a number of bounds has been proposed and exact algorithms to solve the problems have been implemented. Additionally, necessary and sufficient conditions for feasible GSRP solutions based on a graph representation of the reservations has been described. An enumerative algorithm using these conditions has been implemented and is used in the exact algorithms of both the GSR-KP and the GSP-BPP.

For the GSR-KP we have proposed five upper bounds, which have been compared theoretically and computationally. Of these, the tightest bound and the fastest bound has been used in an exact algorithm. For comparison, the instances was also solved using the CPLEX IP-solver.

Of the implemented algorithms, the algorithm using the fastest bound showed the poorest performance, and was the fastest algorithm on very few instances. The algorithm using the tightest bound solved all instances within reasonable time, and was significantly faster than the CPLEX IP-solver on some of the hardest instances. For many of the medium-sized instances, however, the CPLEX IP-solver showed the best performance.

For the GSR-BPP four lower bounds were considered. Of these, $\mathcal{L}_3$ was expected to perform well, and it was implemented with promising results. The GSR-BPP was more complex to solve than the GSR-KP and both the CPLEX IP-solver and the exact algorithm implemented had difficulties solving some of the instances. Neither was able to solve all of the instances within a 30 minute time limit, but the exact algorithm solved several instances which the CPLEX IP-solver was unable to solve. For the instances solvable by the CPLEX IP-solver, the exact algorithm was the fastest in all cases. Thus for this version of the GSRP the presented algorithm is clearly the best choice, when an exact solution is desired.

References

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

[2] J.O. Berkey and P.Y. Wang. Two Dimensional Finite Bin Packing Algorithms. J. Oper. Res. Soc. **38** 423–429 (1987).

[3] G. Belov. Problems, Models and Algorithms in One- and Two-Dimensional Cutting. PhD thesis, Technischen Universität Dresden, 2003. `http://www.math.tu-dresden.de/~belov/publ/text030908_SUBMIT.pdf`.

[4] M. A. Boschetti and A. Mingozzi Two-Dimensional Finite Bin Packing Problem. Part I: New lower bounds for the oriented case 4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies **1**, 27–42 (2003).

[5] M. A. Boschetti and A. Mingozzi Two-Dimensional Finite Bin Packing Problem. Part II: New lower and upper bounds 4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies **1**, 135–147 (2003).

[6] J. Boyar and P. Medvedev. The relative worst order ratio applied to seat reservation Proceedings of SWAT 2004 Lecture Notes in Computer Science **3111** Springer, Heidelberg (2004) T. Hagerup and J. Katajainen (eds.)

[7] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. Operations Research Letters, **32** 5–14 (2004).

[8] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research, **25** 30–44 (1977).

[9] J. Csirik and G. Woeginger. On-line Packing and Covering Problems. Online algorithms. Lecture Notes in Computer Science **1442** 147-177 Springer, Heidelberg (1998) A. Fiat and G. Woeginger (eds.)

[10] ILOG. CPLEX 7.0, Reference Manual, ILOG, S.A., France, 2004

[11] M. Dell'Amico and S. Martello, Optimal Scheduling of Tasks on Identical Parallel Processors, ORSA Journal on Computing **7**, 191–200 (1995)

[12] S.P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. Mathematical Programming **91**, 11–31 (2001)

[13] S.P. Fekete and J. Schepers. A Combinatorial Characterization of Higher-Dimensional Packing. Mathematics of Operations Research **29** 353–368 (2004)

[14] S.P. Fekete, J. Schepers and J. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. Operations Research (to appear).

[15] A. Fiat and G. Woeginger, editors. Online Algorithms — The State of the Art. Lecture Notes in Computer Science **1442** Springer, Heidelberg (1998)

[16] M. C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Academic Press (1980)

[17] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. European Journal of Operational Research, **83** 39–56 (1995)

[18] A. Helliesen. The Seat Reservation Problem - An Empirical Study. Master Thesis, Technical University of Denmark, October 2003.

[19] H. Kellerer, U. Pferschy and D. Pisinger. Knapsack Problems. Springer, Heidelberg (2004)

[20] Algorithmic Solutions Software GmbH LEDA 4.5 The LEDA User Manual Algorithmic Solutions Software GmbH, Germany,

[21] A. Lodi, S. Martello and D. Vigo. Recent Advances on Two-Dimensional Bin Packing Problems Discrete Applied Mathematics **123**, 379–396 (2002).

[22] S. Martello, D. Pisinger, D. Vigo. The Three-Dimensional Bin Packing Problem Operations Research, **48**, 256–267 (2000).

[23] S. Martello and D.Vigo. Exact Solution of the Two-Dimensional Finite Bin Packing Problem. Manage. Sci. **44** 388–399 (1998).

[24] S. Martello and P. Toth, Lower Bounds and Reduction Procedures for the Bin Packing Problem, Discrete Applied Mathematics **28**, 59–70 (1990).

[25] M. Pal and G. P. Bhattacharjee. A Sequential Algorithm for Finding a Maximum Weight k-Independent Set on Interval Graphs Intern. J. Computer Math. **60**, 205–214 (1996).

[26] D. Pisinger and M.M. Sigurd. Using Decomposition Techniques and Constraint Programming for Solving the Two-Dimensional Bin Packing Problem. INFORMS Journal on Computing, **19**, 16 pages (2007).

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

[28] M.Yannakakis, F.Gavril. The maximum k-colorable subgraph problem for chordal graphs Information Processing Letters **24** 133–137 (1987).