Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

The off-line group seat reservation problem

Tommy Clausen [a], Allan Nordlunde Hjorth [b], Morten Nielsen [b], David Pisinger [a,*]

[a] DTU Management Engineering, Produktionstorvet 424, DK-2800 Kgs. Lyngby, Denmark
[b] DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark

A R T I C L E I N F O

Article history:
Received 25 April 2007
Accepted 23 June 2010
Available online 30 June 2010

Keywords:
Two-dimensional packing problem
Bin packing
Seat reservation
Storehouse planning
Berth scheduling
Production planning

A B S T R A C T

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 special case 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 second is a special case of two-dimensional bin packing where all requests must be accommodated while trying to minimize the number of passenger cars needed. For both variants of the problem we present a number of bounds and develop exact algorithms. Computational results are presented for various instances based on realistic data, and 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. In the off-line version we assume 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 traveled. 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 Fig. 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 interchangeably.

The GSRP has numerous applications. In its direct formulation it reflects transportation of children going to a school camp. All requests are known in advance, and school classes may not be split for safety reasons. A similar problem appears when considering reservation of hotel rooms, where people wish to have adjacent rooms. A storehouse may have a number of shelves of fixed length. We know in advance which item will arrive at given date and how long time it will stay in the store house. Each item has a given length, and the sum of the item lengths at each shelf may not exceed the overall shelf capacity. In the well-studied berth scheduling problem [14,23] a number of vessels are planned to arrive to a quay. The arrival and departure time of each vessel is known as well as the length of the vessel. In the classical berth scheduling problem it is allowed to postpone the arrival of the vessels in order to accommodate all vessels. In our model, each vessel has an associated profit, and we are allowed to reject some vessels (which will choose a different quay for loading/unloading).

Finally, the GSRP appears when visualizing solutions to the finite capacity production planning problem [31]. In this problem a production plan is constructed subject to some limited resource. A solution to the problem describes when each job starts and finishes, and how much of the resource it makes use of. A graphical presentation of a solution should depict each task as a rectangle, where the x-dimension is the time, and the y-dimension is the capacity consumption. Only a fixed height is available, corresponding to the capacity in the GSRP.

In the on-line version of the GSRP, reservation requests arrive one by one, and should be assigned a group of seats immediately. Boyar and Medvedev [8] considered the single-customer version of the on-line problem, and showed that if all tickets have the same

* Corresponding author. Tel.: +45 45 25 45 55.
E-mail addresses: tomc@man.dtu.dk (T. Clausen), wolfie@diku.dk (A.N. Hjorth), mnielsen@diku.dk (M. Nielsen), pisinger@man.dtu.dk, pisinger@diku.dk (D. Pisinger).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.06.037

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

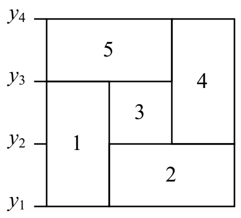


Fig. 1. The geometric interpretation of a group seat reservation problem instance. The train travels from station y_1 to station y_4. Five reservations are given in the table to the left and a packing of the train is shown to the right.

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 GSRP 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 [34]. Helliesen [22] 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 inspired by 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 if a subset of reservations can be feasibly packed within the train 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 stops at H stations, and contains W seats. Let $N = \{1,\ldots,n\}$ 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 $w_j \leqslant W$.

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

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

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

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

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 traveled for all chosen reservations. This problem is a special case 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. In the original 2DKP, the so-called guillotine constraint may be valid. It says that the layout can be recursively separated into the individual rectangles by cuts going between two opposite edges of the current block, parallel to the other edges. This constraint makes sense for technological cutting process and will not be considered here, see Fig. 1.

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 rectangle pairs which in some way share a station (y-coordinate). Finally, let $\ell_{ij} = 1$ iff request i is located left of j

$$\begin{aligned}
\max \quad & \sum_{j\in N} w_j h_j \delta_j && (1)\\
\text{s.t.} \quad & \sum_{j\in N_y} w_j \delta_j \leqslant W, \quad y \in Y, && (2)\\
& \delta_i + \delta_j - \ell_{ij} - \ell_{ji} \leqslant 1, \quad (i,j) \in E, && (3)\\
& x_i - x_j + W\ell_{ij} \leqslant W - w_i, \quad (i,j) \in E, && (4)\\
& 0 \leqslant x_j \leqslant W - w_j, \quad j \in N, && (5)\\
& \ell_{ij} \in \{0,1\}, \quad (i,j) \in E, && (6)\\
& \delta_j \in \{0,1\}, \quad j \in N, && (7)\\
& x_j \geqslant 0, \quad j \in N. && (8)
\end{aligned}$$

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. $\ell_{ij} = 1 \Rightarrow x_i + w_i \leqslant x_j$. Constraint (5) says that all requests should be placed inside the train. Constraints (6) and (7) say that the ℓ_{ij} and δ_j variables should be binary and (8) say that the reservation coordinate variables x_j should be non-negative.

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 special case 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 special case of 2DBPP, 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:

$$\begin{aligned}
\min \quad & v && (9)\\
\text{s.t.} \quad & \ell_{ij} + \ell_{ji} + p_{ij} + p_{ji} \geqslant 1, \quad (i,j) \in E, \ i < j, && (10)\\
& x_i - x_j + W\ell_{ij} \leqslant W - w_i, \quad (i,j) \in E, && (11)\\
& m_i - m_j + np_{ij} \leqslant n - 1, \quad (i,j) \in E, && (12)\\
& 0 \leqslant x_j \leqslant W - w_j, \quad j \in N, && (13)\\
& 0 \leqslant m_j \leqslant vj, \quad \in N, && (14)\\
& \ell_{ij}, p_{ij} \in \{0,1\}, \quad (i,j) \in E, && (15)\\
& m_j \in \mathbb{N}, \quad j \in N, && (16)\\
& x_j \geqslant 0, \quad j \in N. && (17)
\end{aligned}$$

Constraint (10) states that either requests i and j may not overlap, or they must be in different cars. Constraint (11) 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 (12) 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 (13) states that a request must be placed entirely inside a car, and constraint (14) bounds the number of cars used.

The formulation (9)–(16) 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 [21] studied various IP formulations, but were only able to solve instances of moderate size. Fekete et al. [19] 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 [9] presented an $\left(\frac{1}{3} - \epsilon\right)$-approximation algorithm for the 2DKP and developed four exact algorithms based on various enumeration schemes. Belov [5] consider a bounded 2DKP in which the placement of the rectangles should follow a two-stage guillotine packing pattern. Pisinger and Sigurd [32] used constraint programming to solve the general and guillotine-restricted version of the 2DKP. The algorithm follows a two-phase approach as in Fekete et al. [19] 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. Baldacci and Boschetti [2] and Clautiaux et al. [11] followed the same two-phase approach but used a different technique for testing feasibility of the orthogonal packing problem. Baldacci and Boschetti [2] presented a cutting-plane approach using a number of knapsack, dominance and incompatibility constraints. Clautiaux et al. [11] modeled the problem as a scheduling problem making it possible to use powerful constraint-based scheduling propagation techniques.

Berkey and Wang [4] 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 [15].

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 [28]). More recent bounds include Fekete and Schepers [17] and Boschetti and Mingozzi [6,7]. The latter also present a heuristic based upper bound. Clautiaux et al. [12], give a survey of dual feasible functions which are used for finding lower bounds to various cutting and packing problems.

Martello and Vigo [28] 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 et al. [27] to solve the three-dimensional bin packing problem.

Finally, Lodi et al. [26] provide 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)–(8). The optimal solution to this model gives us the upper bound $\mathcal{U}_1$. Notice that for any LP-optimal solution to (1)–(8) we may get an equivalent LP-solution by setting $x_i := 0$ and $\ell_{ij} := (W - w_i)/W$ for any i and j. In this way constraints (4) and (5) are always satisfied when (2) are trivially satisfied. Moreover (3) is satisfied, since with the above definition of ℓ_{ij} it reads

$$\delta_i + \frac{w_i}{W} + \delta_j + \frac{w_j}{W} \leqslant 3.$$

Using Lemma 1 from Appendix A gives the stated. What remains is the problem

$$\begin{aligned} \max \quad & \sum_{j \in N} w_j h_j \delta_j \\ \text{s.t.} \quad & \sum_{j \in N_y} w_j \delta_j \leqslant W, \quad y \in Y, \\ & 0 \leqslant \delta_j \leqslant 1, \quad j \in N. \end{aligned} \tag{18}$$

Now, suppose we introduce a variable δ_j^k for each single seat, such that $w_j\delta_j = \sum_{k=1}^{w_j} \delta_j^k$. This yields the formulation

$$\begin{aligned} \max \quad & \sum_{j \in N} \sum_{k=1}^{w_j} h_j \delta_j^k \\ \text{s.t.} \quad & \sum_{j \in N_y} \sum_{k=1}^{w_j} \delta_j^k \leqslant W, \quad y \in Y, \\ & 0 \leqslant \delta_j^k \leqslant 1, \quad j \in N, \; k = \{1, \ldots, w_j\}, \end{aligned} \tag{19}$$

which is equivalent to (18).

Let A be the constraint matrix of (19). 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, an optimal solution to (19) exists in which $\delta_j^k \in \{0, 1\}$. This means that (19), and thereby (18), is equivalent to splitting the reservations into single-seat reservations.

Denote by G the intersection graph of the columns 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 the time axis, 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 (19) as a weighted W-independent set problem. The weighted W-independent set problem considers a graph with a non-negative weight associated with each vertex. The problem is then to find W 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. [30]). This graph interpretation of the single-seat reservation problem is also noted in [8], although they noted it for the also equivalent W-colorable subgraph problem.

For the weighted W-independent set problem on interval graphs, Pal and Bhattacharjee [30] present an $O\left(Wm\sqrt{\log c}+\gamma\right)$ 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 (18), we can rewrite the running time using the notation of the GSR-KP. The number of independent sets 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. $c\leqslant\sum_{j=1}^{n}h_j$. The number of cliques in an interval graph is at most the number of vertices (see e.g. [20]), so γ is bounded by $m^2=\left(\sum_{j=1}^{n}w_j\right)^2$. Thus, the complexity is

$$O\left(W\cdot\sum_{j=1}^{n}w_j\cdot\sqrt{\log\sum_{j=1}^{n}h_j}+\left(\sum_{j=1}^{n}w_j\right)^2\right). \tag{20}$$

By noting that $\sum_{j=1}^{n}w_j\leqslant N\cdot W$ and $\sum_{j=1}^{n}h_j\leqslant N\cdot H$ expression (20) 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 W-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

$$\begin{aligned}\max\quad & \sum_{j\in N}w_jh_j\delta_j\\ \text{s.t.}\quad & \sum_{j\in N_y}w_j\delta_j\leqslant W,\quad y\in Y,\\ & \delta_j\in\{0,1\},\quad j\in N.\end{aligned}\tag{21}$$

The above problem is a multidimensional knapsack problem with $|Y|$ knapsack constraints, which is strongly $\mathcal{NP}$-hard to solve [24]. Let us denote the optimal value of (21) by $\mathcal{U}_3$. Clearly, $\mathcal{U}_1$ and $\mathcal{U}_2$ are the LP-relaxation of $\mathcal{U}_3$, and hence are dominated by $\mathcal{U}_3$.

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

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

We may now calculate an upper bound as

$$\mathcal{U}_4=\sum_{y\in Y}F_yH_y,$$

where H_y is the "height" of station y as defined in the beginning of Section 2. Calculating $\mathcal{U}_4$ is weakly $\mathcal{NP}$-hard as it contains $|Y|$ subset sum problems.

A similar subset sum problem is used by Clautiaux et al. [11] to prune the search tree by using subset sum problems to determine feasibility.

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 $\mathcal{U}_4$. That $\mathcal{U}_3$ dominates $\mathcal{U}_1$ and $\mathcal{U}_2$ is seen by comparing formulations (18) and (21). Clearly, $\mathcal{U}_1$ and $\mathcal{U}_2$ are the LP-relaxation of $\mathcal{U}_3$.

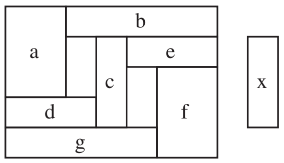
The bounds $\mathcal{U}_1$ and $\mathcal{U}_4$ split reservations in different ways: $\mathcal{U}_1$ splits into single-seat reservations and $\mathcal{U}_4$ splits into single-station reservations. Thus, $\mathcal{U}_1$ and $\mathcal{U}_4$ do not dominate each other. But since the problem is already 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 $\mathcal{U}_1=H\cdot W$. If the reservations are split into single-station reservations, each station can still only accommodate r_1 or r_2. Thus, $\mathcal{U}_4=H\cdot(\lfloor W/2\rfloor+1)$.

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 $\mathcal{U}_1=W\cdot(\lfloor H/2\rfloor+1)$. 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 $\mathcal{U}_4=W\cdot H$. □

As $\mathcal{U}_3$ dominates all the other bounds and is itself $\mathcal{NP}$-hard, it may be that $\mathcal{U}_3$ is simply a more simple formulation of the GSR-KP. This is not the case as is shown by the following example with seven seats, reservations *a*–*g*, and *x* as illustrated below. The packing of *a*–*g* represents an optimal solution. The bound $\mathcal{U}_3$ will split *x* into single-station reservations which may be placed next to *c*. Thus, $OPT<\mathcal{U}_3$ 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 $\mathcal{NP}$-complete 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 Sections 3.3–3.5.

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 Eqs. (3) and (10) 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 assignment of ℓ_{ij} variables is described in Section 3.5.

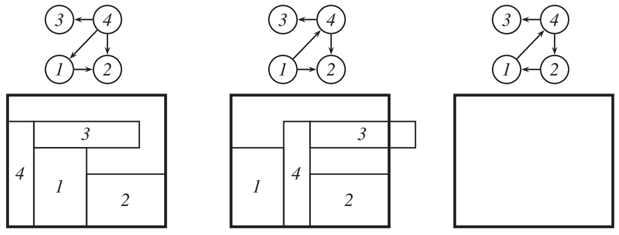
3.4. Feasibility of a packing

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 use a graph representation by Fekete and Schepers [18], and validation techniques by Pisinger and Sigurd [32]. The reason for choosing the framework by Fekete and Schepers [18] is that the additional constraints imposed by our problem give rise to some nice graph theoretical properties in the representation. However, instead of using interval graphs, whose complements are orientable, as in [18], we work already on directed graphs.

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 $p = \langle v_1,\dots,v_n\rangle$ in G, $\sum_{i\in p} w_i \leqslant W$.

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 three different graph representations and their corresponding packing (if legal).



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.

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, which is a special case of Theorem 1 [18]. A direct proof can be found in Appendix A.

Theorem 2. *The properties P1 and P2 are sufficient for describing a feasible packing.*

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

$$x_j = \begin{cases} \max\{x_i + w_i \mid (i,j) \in E\}, & \exists i \in N : (i,j) \in E, \\ 0, & \text{otherwise}. \end{cases} \tag{22}$$

These values are easily calculated by a longest path algorithm [1] which starts by topologically sorting G (and hence checking for cycles) and then running a label correcting algorithm. The time complexity is $\Theta(V + E)$. If for some j we have $x_j + w_j > W$, the packing is infeasible by P2.

3.5. Feasibility enumeration scheme

In the previous section we described how to test 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. By the assumption of leftmost placement (22) we consider the x_j variables to be uniquely determined by the corresponding ℓ_{ij} assignments. 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**.

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 compartments, 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 (9)–(16). It is easy to show (see [13] for details) that $\mathcal{L}_1 = 0$ will hold for any instance. This means that $\mathcal{L}_1$ will always say that the reservations can be packing using minimum one car.

A second lower bound $\mathcal{L}_2$ may be found as follows: For every station (y-coordinate), we 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

$$\begin{aligned}
\min \quad & \xi_y^2 = \sum_{i=1}^{n} \delta_i && (23)\\
\text{s.t.} \quad & \sum_{j\in N_y} w_j x_{ij} \leqslant W\delta_i, \quad i \in B, && (24)\\
& \sum_{i\in B} x_{ij} = 1, \quad j \in N_y, && (25)\\
& \delta_i \in \{0,1\}, \quad i \in B, && (26)\\
& x_{ij} \in \{0,1\}, \quad i \in B,\ j \in N_y. && (27)
\end{aligned}$$

We may now calculate the lower bound as

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

The model (23)–(27) 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 [29], Dell'Amico and Martello [16]

$$\xi_y^3 = \left|\left\{ j \in N_y : w_j > \frac{W}{2} \right\}\right| + \max_{1\leqslant p\leqslant \frac{W}{2}} \left\{ \left\lceil \frac{\sum_{j\in N_s(p)} w_j - \left(|N_\ell(p)|W - \sum_{j\in N_\ell(p)} w_j\right)}{W} \right\rceil, \left\lceil \frac{|N_s(p)| - \sum_{j\in N_\ell(p)} \left\lfloor \frac{W-w_j}{p} \right\rfloor}{\left\lfloor \frac{W}{p} \right\rfloor} \right\rceil \right\}, \qquad (28)$$

where

$$N_\ell(p) = \left\{ j \in N_y : W - p \geqslant w_j > \frac{W}{2} \right\}, \qquad (29)$$

$$N_s(p) = \left\{ j \in N_y : \frac{W}{2} \geqslant w_j \geqslant p \right\}. \qquad (30)$$

This leads to the third lower bound

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

which according to [16] 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 expand $\mathcal{L}_2$ to consider two stations $y, y' \in Y$ at the same time. This results in a new measure $\xi_{y,y'}^4$ which extends (23)–(27). 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

$$\begin{aligned}
\min \quad & \xi_{y,y'} = \sum_{i=1}^{n} \delta_i && (31)\\
\text{s.t.} \quad & \sum_{j\in N_y} w_j x_{ij} \leqslant W\delta_i, \quad i \in B, && (32)\\
& \sum_{j\in N_{y'}} w_j x'_{ij} \leqslant W\delta_i, \quad i \in B, && (33)\\
& x_{ij} = x'_{ij}, \quad j \in N_y \cap N_{y'}, && (34)\\
& \sum_{i\in B} x_{ij} = 1, \quad j \in N_y, && (35)\\
& \delta_i \in \{0,1\}, \quad i \in B, && (36)\\
& x_{ij}, x'_{ij} \in \{0,1\}, \quad i \in B,\ j \in N_y. && (37)
\end{aligned}$$

Here constraint (32) and (33) are the ordinary bin packing constraints for each of the stations y, y' while (34) demands that a request is assigned to the same car at both stations. Constraint (35) enforces that each request $j \in N_y$ must be assigned a car.

We may now calculate the lower bound as

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

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 [28].

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 and to one new bin. A bin is *open* if at least one rectangle has been assigned to it. If the number of open bins exceeds the upper bound we may backtrack. 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, that always chooses the lowest numbered bin first. This delays the opening of new bins, and thus postpones the parts of the solution space using a large number of bins to a point where they may hopefully be pruned.

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).

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 C and C++ using LEDA 4.5 [25]. All tests have been performed on an Intel Pentium 4 with 2 Giga Bytes of memory running at 3 Giga Hertz. For all 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.

Table 1
Overall comparison of the four bounds for GSR-KP. Average time (in seconds) and average gap in percent to the optimal solution and the standard deviations for the four upper bounds.

Bound		$\mathcal{U}_1$	$\mathcal{U}_2$	$\mathcal{U}_3$	$\mathcal{U}_4$
Time	Average	0.13	0.83	0.05	0.00
	Standard deviation	0.11	1.81	0.04	0.01
Gap	Average	10.92	10.92	0.00	19.23
	Standard deviation	7.72	7.72	0.00	8.32

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 have been created. We have created two types of instances. One type was inspired by existing instances taken from the literature of 2D packing. The other type was inspired by real-life problems. All instances can be downloaded from the last author's home page.[1]

The packing instances were created by modifying the 2D knapsack packing instances also considered by Caprara and Monaci [9]. A detailed description of the CGCUT instances can be found in [10]. The GCUT instances are described in [3] and the OKP instances are described in [19]. Finally the WANG instances are described in [33].

We needed to add a starting station for all the reservations, while making sure that the ending stations did not exceed the final station. 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.

The real-life instances were created by considering the normal usage pattern of regional train services in Denmark. Most train routes cover a mix of major cities and less populated areas. Typically, the utilization of the train is high at the city stations and decreasingly lower further from the city. At peak hours, the train will not be able to accommodate all reservations around the city stations, but is unlikely to be filled outside the city areas. The generated instances reflect this by having the requests grouped near a city in the middle of the route, and almost none near the route endpoints. A series of instances with a varying number of requests and group sizes were generated.

To evaluate the quality of $\mathcal{U}_1$ we used CPLEX 9.1 to solve the problem as a linear programming problem. $\mathcal{U}_3$ 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 implemented the upper bounds $\mathcal{U}_2$ and $\mathcal{U}_4$ described in Section 3.1.

Table 1 gives a summary of the overall quality of the bounds on the instances derived from packing. More detailed results can be found in [13]. $\mathcal{U}_3$ is clearly the tightest of the considered bounds, as it finds the optimal solution for all of the instances. The two bounds $\mathcal{U}_1$ and $\mathcal{U}_2$ solve the same relaxation by two different approaches, hence the bound values are the same. However, $\mathcal{U}_1$ is generally faster to calculate than $\mathcal{U}_2$.

As $\mathcal{U}_3$ is the tightest bound considered and the corresponding solution times are reasonable we will choose this bound as one of the bounds for the exact algorithm. It is also obvious that $\mathcal{U}_4$ is significantly faster to calculate so we choose this upper bound as the bound in a second version of the exact algorithm.

[1] http://www.diku.dk/hjemmesider/ansatte/pisinger/seatres_instances.zip.

Table 2
Comparison between the CPLEX IP-solver and the exact algorithm for the GSR-KP for the packing-inspired instances. Reported is the instance name, the train size, and number of reservations N, as well as average time usage (and number of instances solved within 30 minutes in parentheses) for each algorithm. The best result is shown in bold face.

Instance	Stations	Seats	N	CPLEX	$\mathcal{U}_3$	$\mathcal{U}_4$
CGCUT01	15	10	16	0.02(5)	0.01(5)	**0(5)**
CGCUT02	40	70	23	0.79(5)	**0.14(5)**	0.49(5)
CGCUT03	40	70	62	**1.99(5)**	2.09(5)	4.35(5)
GCUT01	250	250	10	**0(5)**	0.02(5)	0.01(5)
GCUT02	250	250	20	**0.04(5)**	0.13(5)	0.15(5)
GCUT03	250	250	30	**0.08(5)**	0.41(5)	0.59(5)
GCUT04	250	250	50	**0.41(5)**	1.73(5)	4.74(5)
GCUT05	500	500	10	**0.02(5)**	0.05(5)	0.05(5)
GCUT06	500	500	20	**0.04(5)**	0.28(5)	0.39(5)
GCUT07	500	500	30	**0.06(5)**	0.64(5)	0.79(5)
GCUT08	500	500	50	**0.4(5)**	3.06(5)	10.89(5)
GCUT09	1000	1000	10	**0.02(5)**	0.05(5)	0.08(5)
GCUT10	1000	1000	20	**0.06(5)**	0.3(5)	0.7(5)
GCUT11	1000	1000	30	**0.2(5)**	1.81(5)	3.49(5)
GCUT12	1000	1000	50	**0.33(5)**	4.69(5)	13.63(5)
GCUT13	3000	3000	32	**1.09(5)**	2.9(5)	74.31(5)
OKP01	100	100	50	6.11(5)	**1.16(5)**	4.27(5)
OKP02	100	100	30	0.31(5)	0.22(5)	**0.21(5)**
OKP03	100	100	30	**0.17(5)**	0.25(5)	0.24(5)
OKP04	100	100	61	5.62(5)	**1.9(5)**	6.7(5)
OKP05	100	100	97	9.42(5)	**6.59(5)**	623.79(5)
WANG20	70	40	42	**0.07(5)**	0.4(5)	0.3(5)
Average	484.77	485.91	35.14	1.24(110)	1.31(110)	34.1(110)

Due to the very high running times of $\mathcal{U}_2$ on the large instances we shall not consider it further, but use $\mathcal{U}_1$ for comparisons with the remaining bounds.

5.2. Results from the GSR-KP

The results from the exact algorithm for the GSR-KP are summarized in Tables 2 and 3. The algorithm is run twice for each instance, using the bounds $\mathcal{U}_3$ and $\mathcal{U}_4$. The results are compared to those of the CPLEX IP-solver. The running time has been limited to 30 minutes for each run.

For the packing instances, all algorithms could solve the instances within the time limit. Overall, the CPLEX solver performs slightly better than the algorithm using $\mathcal{U}_3$. However, the algorithm using $\mathcal{U}_3$ shows only a slight increase in computational time as the number of requests increase, whereas the CPLEX solver appears to be less stable in this regard.

The algorithm using $\mathcal{U}_4$ also shows good computational times for the smaller instances, but degrades significantly for the larger instances. This is because $\mathcal{U}_4$ is not very tight, so the solution space becomes too large, even though the bound is faster to compute.

The instances inspired by real-life problems are more difficult to solve, and seem to confirm the tendencies seen for the packing instances. Indeed, only the algorithm using $\mathcal{U}_3$ could solve all instances within the 30 minutes time limit. Interestingly, the algorithm using $\mathcal{U}_4$ was also able to solve more instances than CPLEX.

5.3. Lower bounds for the GSR-BPP

The test instances used in the GSR-BPP is a further modification of the instances 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 $0 < w_j \leqslant W$ we set

Table 3
Comparison between the CPLEX IP-solver and the exact algorithm for the GSR-KP for the real-life-inspired instances. Reported is the instance name, the train size, and number of reservations *N*, as well as average time usage (and number of instances solved within 30 minutes in parentheses) for each algorithm. A dash indicates that no instances were solved. The best result is shown in bold face.

Instance	Stations	Seats	N	CPLEX	$\mathcal{U}_3$	$\mathcal{U}_4$
G20N10_30	100	100	20	0.05(5)	**0.02(5)**	0.04(5)
G20N20_20	100	100	20	0.35(5)	**0.04(5)**	0.14(5)
G20N30_10	100	100	20	47.09(5)	**0.31(5)**	1.77(5)
G20U20_20	100	100	20	1.41(5)	**0.01(5)**	**0.01(5)**
G30N10_30	100	100	30	0.27(5)	**0.08(5)**	0.32(5)
G30N20_20	100	100	30	8.77(5)	**0.42(5)**	3.06(5)
G30N30_10	100	100	30	11.98(4)	**17.95(5)**	97.1(5)
G30U20_20	100	100	30	12.28(5)	**0.46(5)**	5.56(5)
G40N10_30	100	100	40	0.61(5)	**0.21(5)**	2.16(5)
G40N20_20	100	100	40	15.36(5)	**0.94(5)**	29.45(5)
G40N30_10	100	100	40	–	**23.77(5)**	770.97(5)
G40U20_20	100	100	40	15.34(4)	**10.33(5)**	77.06(5)
G50N10_30	100	100	50	2.9(5)	**0.85(5)**	19.89(5)
G50N20_20	100	100	50	104.48(5)	**1.98(5)**	64.81(5)
G50N30_10	100	100	50	1114.53(2)	**16.77(5)**	–
G50U20_20	100	100	50	318.15(2)	**12.67(5)**	722.35(1)
Average	100	100	35	110.24(67)	5.43(80)	119.65(71)

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

To avoid reservations for *0* seats, we set $w_j = W$ when $w_j \bmod W = 0$.

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. [29]). $\mathcal{L}_4$ 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 for the exact algorithm.

Using $\mathcal{L}_3$ we are able to solve the bound in only a few milliseconds. Moreover, in the cases where we have an optimal solution, $\mathcal{L}_3$ matched the upper bound in every packing 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 had 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 packing instances in less than 30 minutes. For the real-life instances this tendency is even more apparent, since only 8 or 9 of the 80 instances were solved by CPLEX.

The exact algorithm was also unable to solve all of the instances, but performed significantly better than the CPLEX IP-solver as can be seen in Tables 4 and 5. For the packing instances, the exact algorithm solved between 51 and 58 of the 110 instances to optimality within the 30 minutes time limit. For the 80 real-life instances, between 42 and 48 were solved. 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 most of 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

Table 4
Comparison between CPLEX IP-solver and the proposed branch and bound algorithm on the packing-inspired GSR-BPP instances. We report the instance name, the number of stations on the train route, the reservation count *N*, the average run time in seconds (and the number of instances solved within 30 minutes in parentheses) for each bin size. A dash indicates that no instances were solved. The best result for each bin size is shown in bold face.

Instance	Stations	N	Bin size 10		Bin size 20		Bin size 40	
			CPLEX	B & B	CPLEX	B & B	CPLEX	B & B
CGCUT01	15	16	0.19(3)	**0(5)**	0.31(5)	**0.04(5)**	0.06(5)	**0(5)**
CGCUT02	40	23	–	**148.52(2)**	–	**0.29(5)**	–	**1.21(4)**
CGCUT03	40	62	–	–	–	–	–	–
GCUT01	250	10	0.4(5)	**0(5)**	0.13(5)	**0(5)**	0.48(5)	**0(5)**
GCUT02	250	20	–	**11.37(5)**	–	**1.01(5)**	–	**1.04(5)**
GCUT03	250	30	–	**781.71(2)**	–	**923.31(1)**	–	**482.18(1)**
GCUT04	250	50	–	–	–	–	–	–
GCUT05	500	10	2.34(5)	**0(5)**	1.36(5)	**0(5)**	1.97(5)	**0(5)**
GCUT06	500	20	–	**0.14(5)**	–	**0.33(5)**	–	**9.95(3)**
GCUT07	500	30	–	**8.08(2)**	–	**20.96(2)**	–	**1003.49(1)**
GCUT08	500	50	–	–	–	–	–	–
GCUT09	1000	10	1.63(5)	**0(5)**	1.15(5)	**0(5)**	3.06(5)	**0(5)**
GCUT10	1000	20	–	**0.32(5)**	–	**21.97(5)**	–	**58.28(5)**
GCUT11	1000	30	–	**644.54(2)**	–	–	–	**976.63(4)**
GCUT12	1000	50	–	–	–	–	–	**228.71(1)**
GCUT13	3000	32	–	**81.88(2)**	–	**75.02(4)**	–	**0.98(3)**
OKP01	100	50	–	–	–	–	–	–
OKP02	100	30	–	**33.15(2)**	–	**75.6(4)**	–	**60.03(2)**
OKP03	100	30	–	**310.31(4)**	–	–	–	**4.79(5)**
OKP04	100	61	–	**0.69(1)**	–	–	–	**0.4(2)**
OKP05	100	97	–	–	–	–	–	–
WANG20	70	42	–	–	–	–	–	**4.33(2)**
Average	484.77	35.14	1.14(18)	134.71(52)	0.74(20)	93.21(51)	1.39(20)	166.59(58)

Table 5
Comparison between CPLEX IP-solver and the proposed branch and bound algorithm on the real-life-inspired GSR-BPP instances. We report the instance name, the number of stations on the train route, the reservation count *N*, the average run time in seconds (and the number of instances solved within 30 minutes in parentheses) for each bin size. A dash indicates that no instances were solved. The best result for each bin size is shown in bold face.

Instance	Stations	*N*	Bin size 10		Bin size 20		Bin size 40	
			CPLEX	B & B	CPLEX	B & B	CPLEX	B & B
G20N10_30	100	20	256.74(3)	**5.37(4)**	0.36(1)	**73.23(5)**	223.95(2)	**1.35(5)**
G20N20_20	100	20	–	**179.44(4)**	–	**119.81(5)**	–	**576.58(5)**
G20N30_10	100	20	–	**1.56(5)**	–	**11.12(5)**	–	**0.16(5)**
G20U20_20	100	20	485.27(5)	**33.76(5)**	232.59(5)	**0(5)**	14.24(5)	**0.55(5)**
G30N10_30	100	30	–	**3.26(5)**	–	**0.03(3)**	–	**4.06(3)**
G30N20_20	100	30	–	**263.17(3)**	–	**6.78(4)**	–	**252.99(1)**
G30N30_10	100	30	–	**2.16(2)**	–	**74.22(2)**	–	**110.77(3)**
G30U20_20	100	30	1721.43(1)	**25.03(5)**	45.29(1)	**0.09(5)**	1399.65(1)	**0.03(2)**
G40N10_30	100	40	–	**0.06(1)**	–	**0.06(2)**	–	**0.56(3)**
G40N20_20	100	40	–	**14.08(1)**	–	**740.75(2)**	–	**16.22(1)**
G40N30_10	100	40	**–**	**–**	–	**90.12(1)**	–	**0.63(2)**
G40U20_20	100	40	–	**116.75(3)**	365.68(1)	**2.54(2)**	–	**9.96(3)**
G50N10_30	100	50	–	**0.15(2)**	–	**0.13(2)**	–	**0.33(2)**
G50N20_20	100	50	–	**12.28(1)**	**–**	**–**	**–**	**–**
G50N30_10	100	50	–	**276.88(1)**	**–**	**–**	**–**	**–**
G50U20_20	100	50	–	**13.45(2)**	–	**5.18(5)**	–	**13.73(2)**
Average	100	35	821.15(9)	63.16(44)	160.98(8)	80.29(48)	545.95(8)	70.57(42)

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 traveled. A more general approach would be to represent the profit of a reservation as a separate parameter, as is the case in two-dimensional 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.

For the subproblem of determining the feasibility of a packing, the implemented solution is based on graph structure properties and the solution of longest path problems. It is not known if improvements can be made by considering more recent bounds on packing feasibility problems in the literature, e.g. those of Clautiaux el al. [12].

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 special case of the 2DKP and the GSR-BPP which is a special case of the 2DBPP. For each problem, a number of bounds have 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 have 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 four upper bounds, which have been compared theoretically and computationally. Of these, the tightest bound and the fastest bound have been used in an exact algorithm. For comparison, the instances were 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. 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 minutes 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.

Appendix A

Lemma 1. *$a + b + c + d \leqslant 3$ when $ab + cd \leqslant 1$ and $0 \leqslant a, b, c, d \leqslant 1$.*

Proof. This is easily checked with a quadratic solver. □

Proof of Theorem 2. 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 $x_m + w_m \leqslant x_{m+1}$. By transitivity this will ensure $x_i + w_i \leqslant x_j$, 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 $x_j + w_j \leqslant x_i$.

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

Since a packing without overlap exists for each of the three cases 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 $w_j \leqslant W$, $x_j > 0$. By assumption (22), 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 $x_j + w_j = \sum_{v \in p} w_v \leqslant W$ by property P2, which leads to the desired contradiction. Since j was chosen arbitrarily, this proves (2). □

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice Hall, New Jersey, 1993.
[2] R. Baldacci, M.A. Boschetti, A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem, European Journal of Operational Research 183 (2007) 1136–1149.
[3] J.E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting, Journal of the Operational Research Society 36 (1985) 297–306.
[4] J.O. Berkey, P.Y. Wang, Two dimensional finite bin packing algorithms, Journal of the Operational Research Society 38 (1987) 423–429.
[5] G. Belov, Problems, Models and Algorithms in One- and Two-Dimensional Cutting, Ph.D. Thesis, Technischen Universität, Dresden, 2003. <http://www.math.tu-dresden.de/~belov/publ/text030908_SUBMIT.pdf>.
[6] M.A. Boschetti, 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 (2003) 27–42.
[7] M.A. Boschetti, 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 (2003) 135–147.
[8] J. Boyar, P. Medvedev, The relative worst order ratio applied to seat reservation, in: T. Hagerup, J. Katajainen (Eds.), Proceedings of SWAT 2004, Lecture Notes in Computer Science, vol. 3111, Springer, Heidelberg, 2004.
[9] A. Caprara, M. Monaci, On the two-dimensional knapsack problem, Operations Research Letters 32 (2004) 5–14.
[10] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research 25 (1977) 30–44.
[11] F. Clautiaux, A. Jouglet, J. Carlier, A. Moukrim, A new constraint programming approach for the orthogonal packing problem, Computers and Operations Research 35 (2008) 944–959.
[12] F. Clautiaux, C. Alves, J. Valerio de Carvalho, A survey of dual feasible and superadditive functions, Annals of Operations Research (2008), doi:10.1007/s10479-008-0453-8.
[13] T. Clausen, A.N. Hjorth, M. Nielsen, D. Pisinger, The Off-Line Group Seat Reservation Problem, Technical Report, DIKU, University of Copenhagen, 2007.
[14] J.-F. Cordeau, M. Gaudioso, et al., Solving Berth Scheduling and Yard Management Problems at the Gioia Tauro Maritime Terminal, 2003. <citeseer.ist.psu.edu/cordeau03solving.html>.
[15] J. Csirik, G. Woeginger, On-line packing and covering problems, in: A. Fiat, G. Woeginger (Eds.), Online Algorithms, Lecture Notes in Computer Science, vol. 1442, Springer, Heidelberg, 1998, pp. 147–177.
[16] M. Dell'Amico, S. Martello, Optimal scheduling of tasks on identical parallel processors, ORSA Journal on Computing 7 (1995) 191–200.
[17] S.P. Fekete, J. Schepers, New classes of fast lower bounds for bin packing problems, Mathematical Programming 91 (2001) 11–31.
[18] S.P. Fekete, J. Schepers, A combinatorial characterization of higher-dimensional packing, Mathematics of Operations Research 29 (2004) 353–368.
[19] S.P. Fekete, J. Schepers, J. van der Veen, An exact algorithm for higher-dimensional orthogonal packing, Operations Research 55 (2007) 569–587.
[20] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980.
[21] E. Hadjiconstantinou, N. Christofides, An exact algorithm for general, orthogonal, two-dimensional knapsack problems, European Journal of Operational Research 83 (1995) 39–56.
[22] A. Helliesen, The Seat Reservation Problem – An Empirical Study, Master Thesis, Technical University of Denmark, October 2003.
[23] A. Imai, J.-T. Zhang, E. Nishimura, S. Papadimitriou, The berth allocation problem with service time and delay time objectives, Maritime Economics and Logistics 9 (2007) 269–290.
[24] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, Heidelberg, 2004.
[25] Algorithmic Solutions Software GmbH LEDA 4.5, The LEDA User Manual Algorithmic Solutions Software GmbH, Germany.
[26] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics 123 (2002) 379–396.
[27] S. Martello, D. Pisinger, D. Vigo, The three-dimensional bin packing problem, Operations Research 48 (2000) 256–267.
[28] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management Science 44 (1998) 388–399.
[29] S. Martello, P. Toth, Lower bounds and reduction procedures for the bin packing problem, Discrete Applied Mathematics 28 (1990) 59–70.
[30] M. Pal, G.P. Bhattacharjee, A sequential algorithm for finding a maximum weight k-independent set on interval graphs intern, Journal of Computer Mathematics 60 (1996) 205–214.
[31] P.C. Pandey, P. Yenradee, S. Archariyapruek, A finite capacity material requirements planning system, Production Planning and Control 11 (2000) 113–121.
[32] D. Pisinger, M.M. Sigurd, Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem, INFORMS Journal on Computing 19 (2007) 16.
[33] P.Y. Wang, Two algorithms for constrained two-dimensional cutting stock problems, Operations Research 31 (1983) 573–586.
[34] M. Yannakakis, F. Gavril, The maximum k-colorable subgraph problem for chordal graphs, Information Processing Letters 24 (1987) 133–137.