# Queue-constrained packing: a vehicle ferry case study

Christopher Bayliss
Christine S.M. Currie
Julia A. Bennell
Antonio Martinez-Sykora
University of Southampton, Mathematical Sciences
Salisbury Road
Highfield
Southampton, SO17 1BJ, UK

July 15, 2020

**Abstract**

We consider the problem of loading vehicles onto a ferry. The order in which vehicles arrive at the terminal can have a significant impact on the efficiency of the packing on the ferry as it may not be possible to place a vehicle in an optimal location if it is not at the front of one of the dockside queues at the right point in the loading process. As the arrival order of vehicles is stochastic, we model the loading process as a two-stage stochastic optimization problem where the objective is to reduce penalties incurred by failing to pack booked vehicles. The first stage consists of optimizing the yard policy for allocating vehicles to dockside queues while the second stage solves the packing problem for a realisation of the arrival process using the yard policy determined in stage one. A novel stage-wise iterative metaheuristic is introduced, which alternates between packing optimization for each of a training set of scenarios whilst fixing the yard policy and optimizing the yard policy whilst fixing the packing solutions. We introduce two novel packing encoders for the second stage packing problem. Termed Sequential Block Packing Encode (SOPE) and General Packing Encoder (GPE), the arrangements they produce are designed to be efficient and easy to implement for loading staff. Results show that the number of yard queues available is critical to the efficiency of the packing on board the ferry.

**Key words:** Packing, logistics, stochastic optimization

## 1 Introduction

We consider the problem of loading vehicles onto a ferry, where the arrival order of vehicles is stochastic and vehicles are allocated to one of a fixed number of parallel queues on the dockside prior to being loaded onto the ferry through a single ramp entrance. Vehicles vary in size from small motorbikes to large freight vehicles, leading to a very high variety of packing arrangements. While tickets are bought in advance, providing the ferry operator with information about the number of vehicles of each type booked, the order of arrival is not known. The queues at the terminal constrain the loading process and the derived packing problem in the ferry, as only vehicles that are at the front of a queue can be loaded next. A consequence of the queue constraints is that the packing problem may be feasible for one order of vehicles in the yard queues but infeasible for a different order of the same vehicles. This means that booked vehicles may be left off the ferry, incurring penalties. Practitioners in the industry have reported frequent instances of this problem, especially at busy times of the year such as school holidays. In this paper we aim to minimise penalties by jointly identifying the most suitable rules for arranging vehicles in the yard (a yard policy) and optimising the packing of vehicles on the ferry, constrained by their position in the queues.

The *accidental overbooking* of vehicles, which is the focus of our work, is very different from the *deliberate overbooking* of vehicles that takes place in much of the transport industry (e.g. Chapter 4 of Talluri and Ryzin (2004)), designed to allow for the possibility of cancellations and no-shows. Our focus is instead on minimizing the penalties incurred when vehicles cannot be fitted onto the ferry during the loading process, by improving packing efficiency and the yard policy. We would anticipate the results of the work also having a long-term impact on a company by improving their estimates of the difficulty of packing different sets of vehicles and hence allowing them to place better constraints on the numbers of tickets sold.

The queue constrained packing problem, for the case of a single deck vehicle ferry, can be characterised as a two-dimensional knapsack non-guillotine cutting problem (Beasley, 1985a) with precedence constraints on the order in which items are cut (Fekete et al., 2006), and loading constraints similar to the unloading constraints considered by da Silveira et al. (2011). A pair of orthogonal non-guillotine cuts corresponds to parking a vehicle in a position that is either next to a wall or another vehicle, i.e., an efficient corner position. The item order constraints emerge from the fixed orders of queues in the yard, because the vehicles in each queue are loaded onto the ferry in queue order. If vehicles $i$ and $j$ are in the same yard queue and vehicle $i$ arrived before vehicle $j$, vehicle $i$ must be loaded onto the ferry before vehicle $j$. On the other hand, if vehicles $i$ and $j$ are in different yard queues, then there are no precedence constraints regarding the loading times of vehicles $i$ and $j$, since any vehicle at the front of a queue can be loaded next. This means that the number of available yard queues influences the degree to which the queue constraints constrain the loading problem.

Furthermore, we have the requirement that it must be possible for vehicles to drive unobstructed into their assigned position starting from the ferry entrance, referred to as loading constraints. In this work, this constraint is satisfied if it is possible for vehicles to reach their assigned position on the ferry by following an arbitrary sequence of orthogonal movements which do not overlap with any of the previously loaded vehicles.

The item orders are uncertain due to the stochastic nature of vehicle arrival times at the yard. For the simplest case of a yard with a single queue, Figure 1 shows how a packing problem can become infeasible with item placement precedence constraints and particular orderings of items. In arrival scenario 1 (Figure 1 top), we see that the optimal packing solution is possible. However, in arrival scenario 2 (Figure 1 bottom), we see that the precedence constraints lead to the 5th arriving vehicle being left off the ferry, which is because the larger 3rd and 4th vehicles have to be loaded first, and they have the effect of blocking the entrance. For the case of multiple yard queues and uncertain arrival orders it is possible to optimise the policy used to allocate vehicles to queues to facilitate efficient packing, the problem that is addressed in this work.

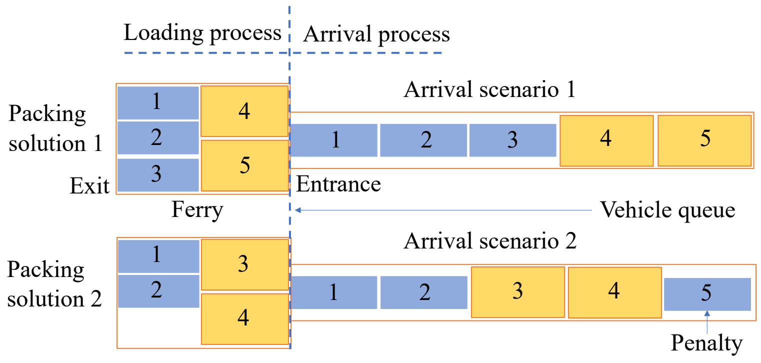


Figure 1: Small example illustrating packing problem infeasibility caused by vehicle arrival order.

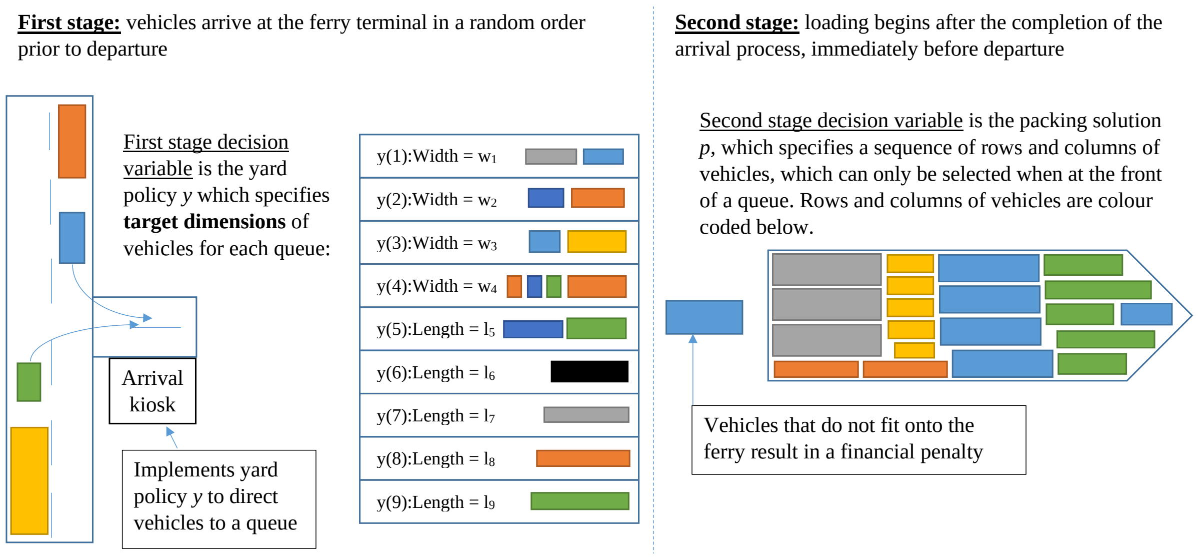

First stage: vehicles arrive at the ferry terminal in a random order prior to departure
First stage decision variable is the yard policy y which specifies target dimensions of vehicles for each queue:
y(1):Width = w1
y(2):Width = w2
y(3):Width = w3
y(4):Width = w4
y(5):Length = l5
y(6):Length = l6
y(7):Length = l7
y(8):Length = l8
y(9):Length = l9
Arrival kiosk
Implements yard policy y to direct vehicles to a queue
Second stage: loading begins after the completion of the arrival process, immediately before departure
Second stage decision variable is the packing solution p, which specifies a sequence of rows and columns of vehicles, which can only be selected when at the front of a queue. Rows and columns of vehicles are colour coded below.
Vehicles that do not fit onto the ferry result in a financial penalty

Figure 2: Problem diagram: first stage queueing process and second stage ferry packing process.

We formulate the loading process as a two-stage stochastic optimization problem as depicted in Figure 2.

The first stage decision problem is that of selecting a yard policy, which defines the yard queue that arriving vehicles are allocated to prior to loading onto the ferry. The yard policy is chosen to minimise penalties incurred due to leaving vehicles off the ferry. Since the arrival order of vehicles is uncertain the evaluation of candidate yard policies requires solving an uncertain packing problem. The second stage recourse problem is to solve a deterministic packing problem for a realisation of the random arrival process, with vehicles allocated to yard queues using the yard policy from the first stage. Given the large number of potential arrival orders for a typical vehicle ferry, with hundreds of vehicles and more than ten different vehicle types, we optimize the yard policy for a training set of random arrival scenarios $S$. In the remainder of the paper, we will use scenario to refer to an arrival scenario, i.e, a given order of the vehicles arriving at the terminal. While the overall objective is to minimize penalties through failing to pack vehicles, we consider two objective functions for the metaheuristic: minimising the expected penalties incurred in all of these scenarios and a worst-case measure that we describe below. The size of $S$ affects the total run time of the algorithm and the breadth of examples considered.

To solve the first stage problem, that of determining a yard policy which facilitates efficient packing under uncertain vehicle arrival orders, we propose a novel stage-wise iterative metaheuristic that alternates between packing optimization for each scenario in the training set whilst fixing the yard policy, and optimizing the yard policy whilst fixing the encoded packing instructions, again for each scenario in the training set. A solution to the first stage problem is a yard policy. The objective measure of a yard policy is the cost of penalties for the vehicles left off the ferry in the training set whilst using that yard policy to allocate arriving vehicles to queues. A *vehicle mix* is defined as the counts of vehicles of each type that fit onto the ferry in a given scenario. Following on from this, a robust or worst case measure of the penalties is obtained from the subset of booked vehicles which always fit onto the ferry regardless of the arrival order, which we refer to as the *vehicle mix intersection*. A vehicle mix intersection is defined as the number of vehicles of each type that can be successfully loaded onto the ferry in each of a set of different scenarios. This can be found by determining the minimum quantities of each vehicle type that are loaded onto the ferry in each of a set of scenarios. An upper bound on the penalties that we incur is the sum of the penalties incurred for vehicles that are booked but not within this vehicle mix intersection. See Section 3 for further details of this objective function. We find that minimizing this upper bound provides a better overall solution to the two-stage problem than minimizing the expected penalties over the training set. This method is addressed in detail in Section 5.

For the second stage ferry packing problem we develop a novel packing encoder termed the Sequential Block Packing Encoder (SOPE). Vehicles are not restricted to lanes on the ferry. Instead, SOPE is a 2-d rectangle packing approach that is based on packing rows and columns of vehicles at a time, referred to as (horizontal and vertical) blocks, where a vehicle currently at the front of a yard queue is allocated to the next block until the block is complete. This approach addresses all of the practical vehicle ferry loading constraints as well as the queue constraints. The orientations of blocks and the target sizes of vehicles within each block are encoded as integer vectors (see example presented in Figure 3), which can be optimized for different scenarios. We also introduce a variation of SOPE, which we term the General Packing Encoder or GPE, that allows for lateral and longitudinal compacting of rectangles and is designed to recover wasted space due to vehicles being placed within blocks when either their length or width may be smaller than the relevant block dimension. This effect is achieved by placing vehicles in the nearest efficient corner positions of the current available space, so as to maximize the remaining usable space following their placement. SOPE and GPE have the practical benefit that the generated packing instructions are intuitive and easy to implement by loading staff. We also note that our approach can be used to efficiently solve any knapsack packing problems, even when there are no queue or loading constraints. In which case these can be treated as artificial constraints which decompose the packing problem in a way that aids efficient packing. Section 7 describes a relaxed packing formulation that is used to obtain a lower bound and determine the quality of packing solutions.

As far as we are aware this is the first article to address the problem of rectangle packing in which the availability of items to pack at each point in the process is uncertain (in this case due to uncertainties in the arrival order). We term this the *queue-constrained 2-d rectangle packing knapsack problem under arrival order uncertainty*. Although we focus on the application to vehicle ferries, two-stage optimization problems with uncertain inputs are common in many other areas. Our stage stage-wise iterative metaheuristic provides a general framework for solving such problems.

Another approach to minimizing the penalties would be to identify the vehicle layout that minimizes the penalties whilst ignoring queue constraints and arrival order uncertainty and to set up yard queue policies so as to retain the feasibility of that vehicle layout. We do not consider this approach here because, even without uncertainty, the queue constraints alone can preclude the implementation of an optimal vehicle layout; for example, if all of the large vehicles need to be parked first but arrive last, the number of yard queues may not be sufficient to allow all of these vehicles to be parked at the front of the queues.

The remainder of this paper is structured as follows. Section 2 reviews relevant literature before we go on to describe the mathematical model for the two-stage problem in Section 3. In Section 4 we describe SOPE, the novel packing methodology which addresses each of the practical constraints of the loading problem. The stage-wise iterative metaheuristic used to derive robust yard queue policies in stage one of the two-stage optimization is described in Section 6. Section 7 evaluates the SOPE and GPE packing methodologies against lower bounds. The experimental results in Section 8 that are designed to evaluate the overall methodology, show the effect of the number of parallel yard queues on packing efficiency and also how solutions derived using the proposed worst case objective function compare with those derived from an expected value objective function. Finally, we conclude in Section 9 and discuss possibilities for future work.

## 2 Related literature

### 2.1 Cutting and Packing

Without any of the dockside constraints, under the typology of cutting and packing problems presented by Wäscher et al. (2007), the packing problem we consider is a Two-Dimensional Rectangle Single Knapsack Packing Problem (2D-RSKP). A survey of cutting and packing problems can be found in Lodi et al. (2002). Regarding exact procedures for cutting and packing problems of this nature, Carvalho (2002) defined a MILP formulation and Pisinger and Sigurd (2005) proposed a branch and price algorithm. Approximation algorithms can be found in Kang and Park (2003). Most recently, the genetic algorithm approach proposed by Goncalves and Resende (2013) obtains the best published results for this problem.

The packing methodology that we introduce is based on applying sequential orthogonal guillotine-cuts to a 2-D container to produce 1-D bins (or regions) in which rows or columns of vehicles are packed. There is a large class of cutting and packing problems that have guillotine-cut constraints (Lodi et al., 2002). Typically these relate to applications where small items are cut out of larger rectangles using guillotine-cuts that are parallel to one of the sides of the larger rectangle; textile, glass and paper cutting being the common examples (Bekrar and Kacem, 2009). Guillotine-cut constraints are also common in strip packing problems where a predefined set of items is cut from a roll of stock using the least total length. A frequent classification of guillotine-cut problems is by the number of stages. In an N-stage guillotine-cut problem, guillotine-cuts are applied iteratively to rectangles resulting from prior stages of cuts. Mrad et al. (2013) introduce an optimal integer programming formulation for the case where there are at most 2-stages. Typically, guillotine-cut solutions are represented as trees (Fleszar (2013) provides instructive diagrams) with nodes for cut rectangles and arcs representing orthogonal cuts. Another distinction lies in whether or not a guillotine cutting problem is constrained. Constrained refers to the case where the set of items to be cut is predefined, whereas the number of items of each type is a free variable in the unconstrained case. Morabito and Arenales (1996) consider staged and constrained two-dimension guillotine cutting problems using an AND/OR graph approach.

Our proposed approach has similarities to other packing problems with guillotine cut constraints in that vehicles are grouped into subsets consisting of either a line of cars arranged front to back, very much like a lane, or a line of cars arranged side by side. These blocks are arranged within the deckspace. A key difference between our method and the guillotine-cut methods is that when we cut out a block from the remaining space, the cuts do not need to extend across the whole length or width of the deck.

The packing problem considered in this work features item order and loading constraints, due to the requirement that vehicles must be able to drive unobstructed into their assigned positions. da Silveira et al. (2011) address the two dimensional knapsack packing problem with unloading constraints for rectangle containers where one side is the entrance and exit. Items are feasible for unloading if no other items block them from moving in a horizontal motion directly towards the exit. They propose approximation algorithms for several special cases with their algorithms being particularly applicable to vehicle routing problems with loading constraints (Gendreau et al., 2007; Iori et al., 2007; Guimarans et al., 2018). In contrast, in this work loading is feasible if items can be placed using an arbitrary sequence of horizontal and vertical motions starting from the entrance. Guillotine-cut constraints enable a highly efficient solution representation for the ferry loading problem. The use of guillotine-cut constraints often facilitates the application of optimization approaches such as dynamic programming and column generation (Cintra et al., 2008) but due to the complicating feature of uncertain arrival orders, we instead combine metaheuristics with a packing methodology that draws on ideas from guillotine-cut formulations Lodi et al. (1999); Hopper and Turton (2001); Alvarez-Valdes et al. (2005); Wei et al. (2011). In particular, we implement simulated annealing, also used by Leung et al. (2012) for similar packing problems.

## 2.2 Optimization Under Uncertainty

We model the ferry loading problem under arrival order uncertainty as a two-stage stochastic optimization problem and solve it via a scenario sampling approach. This has similarities to the Sample Average Approximation Algorithm (SAA) described in Kim et al. (2015). The approach allows us to solve the problem by directly modelling the vehicle arrival process and subsequent loading problems over a training set of arrival scenarios. Al-Khamis and M'Hallah (2011) is the closest existing work in that it models the parallel-machine scheduling problem as a two-stage stochastic problem in which the machine capacity is scheduled in stage one, prior to receiving information on the due dates of jobs. As with the method we describe here, the second-stage solves a deterministic optimization problem for the observed realisation of the first-stage uncertainties (due dates). The authors use SAA (Kim et al., 2015), incorporating ranking and selection to solve the problem. The use of a robust approach for solving the problem has not previously been considered.

The packing problem we consider here can also be described as an online knapsack problem, in which 2-D rectangles (vehicles) are being packed within a larger container and arrive at random times prior to a deadline (Papastavrou et al., 1996). Navarra and Pinotti (2017) provide a formal introduction to the online knapsack problem with unknown capacity in which a preference ordering of the small items has to be determined before the capacity of the knapsack is revealed. Once the capacity of the knapsack has been revealed the items are added according to the pre-determined order until reaching the first item that does not fit in the knapsack. All remaining items are discarded and no profit is received for them. Navarra and Pinotti (2017) cite mobile data transmission tasks as a practical application of this formulation, in which the duration of a phone call is uncertain and this duration represents the uncertain knapsack capacity. Data transmission tasks are then fitted into the call time as they can be performed in parallel with the call itself. Using a pre-determined preference ordering is attractive as it allows the policies developed to be fair and satisfy other measures such as improved energy efficiency. Similar issues are encountered in runway sequencing (Balakrishnan and Chandran, 2010; De Maere et al., 2017).

Dean et al. (2008) and Levin and Vainer (2013) consider the benefits of adaptive policies over static policies for knapsack problems. While these have been shown to be effective in other applications, the complexity of the 2-D packing problem makes them less attractive here.

The sale of tickets for vehicles can be formulated as a stochastic knapsack problem with the objective of maximising the revenue from ticket sales, where demands for tickets are uncertain. In previous work we solved this via dynamic pricing algorithms (Martinez-Sykora et al., 2017; Bayliss et al., 2016, 2019a). The problem considered here takes as input the list of vehicles booked on the ferry once sales are closed and solves the yard policy problem.

## 3 Problem Formulation

Since our problem consists of an arrival process followed by a ferry loading process, we formulate the problem as a two-stage stochastic optimization problem, where the first stage determines a yard policy, denoted $y$, which directs vehicles to different yard queues based on their physical dimensions. A yard policy constitutes a set of target vehicle dimensions for each yard queue. An arriving vehicle is allocated to the queue whose target dimensions most closely match its actual dimensions using an encoder described in Section 4. The second stage recourse problem aims to optimize the packing arrangement for a realisation of the arrival process to which the yard policy $y$ has been applied during the arrival process. In what follows we define a *packing solution* to be the encoding of the instructions for generating a packing arrangement, where the packing solution is stored in the form of an integer vector (see example presented in Figure 3). A *packing arrangement* refers to a precise layout of vehicles on a ferry and is generated from the packing solution using one of the *packing encoders*, SOPE or GPE, which will be described in more detail in Section 4.

The objective in stage one is to select a yard policy $y^*$ that minimizes penalties incurred when vehicles cannot be fitted onto the ferry under uncertain vehicle arrival order,

$$y^* = \arg\min_{y \in Y} \left\{ \sum_{s \in \mathscr{S}} \sigma_s \sum_{j \in J} \mu_{j,s} \rho_j \right\}, \tag{1}$$

where the set of all possible yard policies is denoted $Y$, the set of all possible scenarios is $\mathscr{S}$, $\sigma_s$ is the probability that scenario $s$ occurs, the set of vehicle types is $J$, $\mu_{j,s}$ is the number of vehicles of type $j$ that cannot be loaded in scenario $s$ and $\rho_j$ is the penalty for failing to load vehicles of type $j$. We evaluate $\mu_{j,s}$ using $\mu_{j,s} = b_j - K_{s,j}$, where $b_j$ is the number of vehicles of type $j$ booked on to the ferry and $K_{s,j}$ is the number of vehicles of type $j$ successfully loaded onto the ferry in scenario $s$.

The main structural constraints on which the proposed packing methodology is based are listed as follows:

- Vehicles are allocated to a yard queue on arrival at the terminal. Once loading begins vehicles can only be loaded when they are at the front of a queue.
- Yard queues are all ordered in increasing arrival time. This means that vehicles cannot change position in a queue once they have been allocated to it.
- Vehicles can only be parked in positions on the ferry that are reachable from the single entrance at the time they are loaded.

The stage one problem of identifying a yard policy is solved when the final set of booked vehicles is known but before vehicles begin to arrive at the terminal. For tractability reasons, in this work we evaluate yard policies based on a sample of scenarios referred to as the *training set* and is denoted by $S$. This allows us to explicitly solve the packing problems that arise in each training set scenario. The solutions developed for the scenarios in the training set are then tested over a different and larger *test set* of scenarios. Section 8 reports average penalties incurred over the test sets for a range of problem instances and training set sizes.

We consider two forms for the objective function used by the metaheuristic. In the first, we aim to find the yard policy that minimizes the expected value of penalties calculated over all of the scenarios in the training set $S$. As $S$ is an approximation of $\mathscr{S}$, this is an approximation to Equation 1, which we denote EXP in Section 8.

An alternative objective function, which is designed to generate yard policies that have good worst-case performance, is to instead minimize the penalties from vehicles that do not fit onto the ferry in at least one scenario in the training set. This helps to reduce the impact of over fitting when $|S|$ is small. As discussed in Section 1, to do this we determine the vehicle mix intersection, a subset of the booked vehicles that can be loaded onto the ferry in each and every scenario, and find the penalties incurred by the booked vehicles not included in this subset. A vehicle mix intersection is calculated by finding the maximum number of vehicles of each type that can definitely be packed in each and every scenario in the training set. Given a yard policy and the sets of possible packing solutions for each scenario, the calculation of the best vehicle mix intersection requires the selection of a packing solution for each scenario such that the vehicle mix intersection is maximised. This sub optimisation problem can be solved via a branch and bound algorithm. Section 5.1 presents the branch and bound algorithm that is used for this task. The objective of minimising the penalties of vehicles not included in the vehicle mix intersection can be expressed as

$$y^* = \arg\min_{y \in Y} \left\{ \sum_{j \in J} \rho_j \max_{s \in S} \mu_{j,s} \right\}, \tag{2}$$

which will tend to generate yard policies that guarantee that a particular set of vehicles will fit onto the ferry in each and every scenario. In contrast, minimising expected penalties can lead to yard policies that leave different sets of vehicles off the ferry in different vehicle scenarios. In Section 8, we denote this vehicle mix intersection based objective function VMI and we demonstrate that this second objective function performs better on average than minimising the expected penalties.

In Objective (2) the underlying assumption is that the $K_{s,j}$ values are based on a given packing solution for each scenario $s \in S$. In this work a stage-wise iterative metaheuristic is used to search for good packing solutions for each scenario for a given yard policy, see Section 6. To allow for these possibilities we use the notation $H_{p,s,j}$, in place of $K_{s,j}$, to denote the number of vehicles of type $j$ loaded in scenario $s$ when using packing solution $p$. Similarly we use $\nu_{p,s,j}$ to denote the number of vehicles of type $j$ not loaded in scenario $s$ when using packing solution $p$. The stage-wise iterative metaheuristic alternates between searching for the best set of packing solutions $P$ for a given yard policy $y$ and searching for the best yard policy $y$ for a given set of encoded packing solutions $P$.

## 4 Sequential Block Packing Encoder

This section introduces the Sequential Block Packing Encoder (SOPE), a packing methodology that directly addresses all of the practical constraints of the ferry loading problem and provides a two-layered integer vector solution encoding for yard policies and packing solutions. This encoder approach allows us to employ an iterative stage-wise metaheuristic algorithm to solve the problem efficiently by alternating between yard policy improvement while fixing the packing solutions and then packing solution improvement while fixing the yard policy.

In SOPE, candidate yard policies and packing solutions are encoded as vectors of integers. Yard policy vectors have an element for each yard queue, which specifies a target dimension, width or length, of the vehicles to allocate to that queue. The candidate packing solutions are interpreted as instructions for the sequence of blocks of vehicles to load onto the ferry. Each element specifies the orientation of a block of vehicles to load next, row or column, and the target sizes of the vehicles in the block. The target dimensions for rows of vehicles are their lengths whilst the target dimensions for columns of vehicles are their widths. Here, SOPE is used within a stage-wise iterative metaheuristic algorithm to search the space of yard policies and packing solutions over a set of scenarios, but it can be used on its own as an efficient tool for rectangle packing with constraints on the packing order.

## 4.1 Solution Encoding

In the SOPE approach vehicles are arranged on the ferry by adding a sequence of blocks one at a time to the ferry, where each block constitutes either a row or a column of vehicles. This step is equivalent to combining a set of individual vehicles into a new rectangular entity, composed of the vehicles arranged either in a line side-to-side (a row) or a line front-to-back (a column). Allocating vehicles to blocks gives a smaller number of larger items to be packed. The procedure continues in a sequential manner until either no vehicle remains to be packed or no vehicle fits into the final remaining space.

Each integer in a packing solution vector defines one of three cut types: a bottom horizontal cut, a left vertical cut, or a right vertical cut. No top horizontal cuts are used as this would be equivalent to blocking the ferry entrance, and this is how SOPE ensures that the vehicles can drive unobstructed into their assigned rows or columns. Each integer in the packing solution encodes whether the block of vehicles will be a row or a column, whether the column will be on the left or right of the deck, and also the size of vehicles to include in the block. Creating rows and columns of vehicles of similar dimensions ensures that wasted space is minimised within each cut. Look up tables are defined for both the yard policy and packing solutions for mapping integers to physical instructions. The look up tables are both two dimensional with one dimension for orientation and another for target dimension. In each case the number of orientations is fixed but the number of possible values for the target dimension is an arbitrary choice. A good initial choice is to set the possible target dimensions to those of a set of standard vehicle types.

In order to better demonstrate the encoding, we include an example solution. The example look up tables for encoding the yard queue policy and packing solutions are given in Tables 1 and 2, while Figure 3 illustrates a solution generated using the SOPE encoding. The example yard queue policy $[2,5,3]$ together with Table 1 specifies that the first yard queue is set aside for vehicles with a medium width, the second for vehicles with a large length and the third for medium length vehicles. The reason why the target dimension of a queue specifies a vehicle width or a vehicle length, and not both, is that the aim is to generate efficient rows or columns of vehicles, a process that is aided by storing vehicles with similar lengths or widths together in a yard queue. Vehicles with similar lengths can be used to generate efficient rows of vehicles, whilst vehicles of similar width can be used generate efficient columns of vehicles on the ferry. The encoding of yard policies includes the specification of whether the target dimension for a queue is a vehicle width or a vehicle length. In Table 1, values 0, 2 and 4 specify the target dimension as a vehicle width, while values 1, 3 and 5 specify the target dimension as a vehicle length.

The left hand panel of Figure 3, gives a representation of the terminal. We apply policy 2 to the top queue (first queue) in this diagram; policy 5 to the middle queue (second queue) and policy 3 to the final queue. The colours represent the vehicle types, and the number is the order in which the vehicles are loaded onto the ferry. In this example, the first six vehicles loaded correspond to the first four vehicles in the first queue and the first two vehicles in the second queue. This builds the first row inside the ferry. It is worth highlighting that the precise order in which the vehicles have arrived at the terminal is not shown, but we can observe that the red vehicle was assigned to the second queue because the first queue was already full. Note that in this example the width of the blue and red vehicles is the same.

| | | Quantiles | | |
|---|---|---|---|---|
| Target dimension | | Small | Medium | Large |
| | Width | 0 | 2 | 4 |
| | Length | 1 | 3 | 5 |

Table 1: Yard queue allocation policy look up table.

| | | Quantiles | | |
|---|---|---|---|---|
| Cut type | | Small | Medium | Large |
| | Bottom row | 0 | 3 | 6 |
| | Left column | 1 | 4 | 7 |
| | Right column | 2 | 5 | 8 |

Table 2: Packing solution encoding look up table.

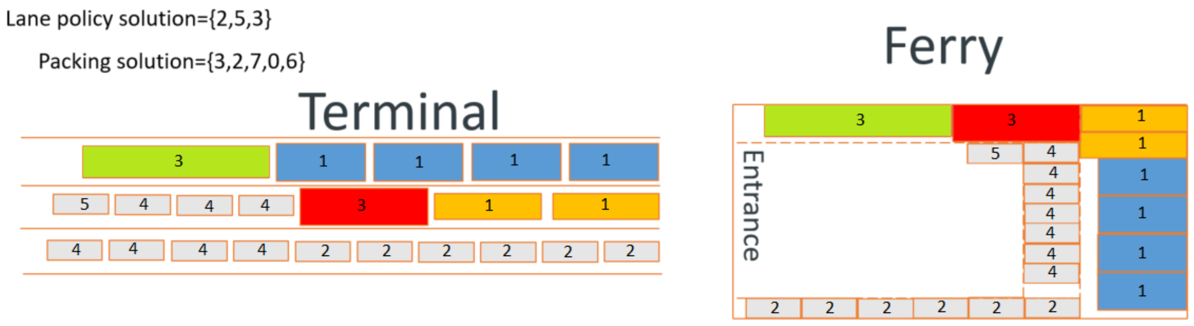


Figure 3: Illustration of how SOPE addresses the main structural constraints of the vehicle ferry loading problem.

To implement this example yard policy, vehicles are allocated to the yard queue that has the closest match to their dimension. In addition to minimizing the difference between the target dimension of the queue and a vehicle, consideration is also given to the number of different vehicle types already in the yard queue. To account for the latter we seek to maximize the concentration of the given vehicle type in the yard queue that it is allocated to. This approach helps to keep identical vehicle types together when vehicles are a close match for a number of different yard queues. More formally, upon arrival at the ferry terminal a vehicle type $j$ is allocated to the terminal queue $l$ that minimises:

$$G\left(1-\left(\frac{\left(vCounts_{l,j}\right)\beta_j}{Q_l+\beta_j}\right)\right)+\frac{|targetLength_l-\beta_j|}{\max\left(\beta\right)}, \tag{3}$$

where $vCounts_{l,j}$ is the number of vehicle type $j$ already allocated to yard queue $l$; $\beta_j$ is the length of vehicle type $j$; $Q_l$ is the current length of yard queue $l$; $targetDim_l$ is the target dimension of yard queue $l$ and $G$ is the weight given to the vehicle counts term relative to that given to the target dimension term. Equation (3) assumes that yard queue $l$ is set aside for vehicles of length $targetLength_l$. If the queue instead has a target width, $targetWidth_l$, $targetLength_l$ is replaced by $targetWidth_l$ and $\beta_j$ is replaced by $\alpha_j$.

The second term of Equation (3) measures the difference between the target dimension of the yard queue and the actual dimension of the vehicle. For the case where there is a tie between two queues, the first term breaks ties by giving a better score to the queue that already has the largest number of vehicles of the same type $j$ as the arriving vehicle. This approach to breaking ties increases the level of vehicle presorting performed in the arrival queueing process. For example, consider a yard with two queues, which both have the target dimension: “vehicles of length 5 metres”, queue 1 already has a 5 metre long vehicle in it, queue 2 is empty and another vehicle of length 5 metres has just arrived at the terminal. In this case, assuming $G=1$, Equation 3 gives queue 1 a score of 0 and queue 2 a score of 0.5, meaning that the new vehicle will be allocated to queue 1.

Figure 3 together with Table 2 illustrate how packing solutions are encoded. At each stage vehicles are selected from those currently at the front of the queues. As an example, the solution $[3,2,7,0,6]$ is interpreted as follows, with reference to Table 2. First, a bottom row of medium length vehicles are loaded onto the ferry. Second, a right column of small width vehicles are loaded, followed by a left column of large width vehicles, and then a row of small width vehicles. The final vehicle is loaded according to the instruction: a row of (relatively) large length vehicles.

A *cut type* defines the orientation of a cut while the target-size of a block defines the size of vehicles that should be placed within it. All cuts are straight-line guillotine cuts that dissect the remaining space into two parts: a vehicle block and an area that constitutes the remaining space in the next iteration of the procedure.

Using the standard convention that the cumulative distribution function is written as $F(x)$, where $x$ is the quantity of interest (in our case vehicle lengths or widths), we define the quantile corresponding to probability $p$ as $x=F^{-1}(p)$. In what follows we use a *quantile function*, which returns $F^{-1}(p)$.

If the cut types are integer codified and there are $c_0$ of them and the target size quantiles are similarly codified and there are $q_0$ of them then the look-up-table Table 2 can be replaced with Equations 4 and 5, which map an element of a solution, denoted $e$, to a cut type and vehicle size instruction. These equations play the roles of Tables 1 and 2 within the implementation of the packing encoders introduced in this section. Here, quantile refers to the codified target width or length of vehicles to include in the block.

$$cut\ type = modulo\left(e,c_0\right) \tag{4}$$

$$quantile = \left\lfloor\frac{e}{c_0}\right\rfloor \tag{5}$$

The number of distinct packing instructions is $c_0 q_0$. The choice of the number of quantiles $q_0$ is an arbitrary one, but a good starting point, which we use here, is to set this equal to the total number of vehicle types. Due to the analogous nature of the encoding of yard policies and packing solutions, i.e., target orientation and target size, yard policies use the same equations except that $c_0$ and $q_0$ are replaced with $c_1$ and $q_1$ respectively.

Figure 3 illustrates how the vehicle arrangement achieved by implementing the example packing solution respects the order of vehicles in the yard queues. In particular, the cut numbers are non-decreasing in each queue.

The advantages of the SOPE approach are as follows: 1) it provides a solution encoding that naturally respects the main structural constraints of the vehicle ferry loading problem; 2) the yard policy and packing solutions have an intuitive meaning and provide easy to follow instructions for yard staff and loaders; 3) in contrast to current practice based on trying to load vehicles within marked lanes on the vehicle deck of the ferry, the SOPE approach exploits the space efficiency benefits of 2-d packing arrangements.

We represent the yard policy and SOPE packing encoder presented in this section as follows:

$$q \leftarrow g(y, s) \tag{6}$$

for the process of generating queue orders, denoted by $q$, for a given yard policy $y$ and scenario $s$ and

$$v \leftarrow h^{SOPE}(p, q) \tag{7}$$

for the process of packing the ferry given queue orders $q$ and a packing solution $p$. Here, $v$ is a vector where $v_j$ denotes the number vehicle of type $j$ successfully packed onto the ferry.

## 4.2 General Packing Encoder: SOPE Relaxation

A downside of the SOPE algorithm is that it is not possible to recover wasted space within the blocks. This wasted space is due to vehicles within a block having different widths (columns) or lengths (rows). We devise a General Packing Encoder (GPE) that can be seen as a relaxation of SOPE and is designed to resolve this issue. The GPE algorithm follows the SOPE procedure as before but when calculating the remaining space and the available corner positions, it uses the actual positions of vehicles rather than the artificial boundaries of the blocks.

In GPE the block instructions: bottom row, left column and right column are replaced with bottom-left-like (BL packing heuristic) instructions: bottom-left (BL); left-bottom (LB); and right-bottom (RB). The quantile instruction is also interpreted in a slightly different way to that of the SOPE methodology. Instead of the quantile instruction defining the position of a cut, it is used only to select the target dimension of the vehicles that will be loaded to form the next (possibly staggered) block of vehicles. The vehicle at the front of a yard queue that is the closest match to this target dimension is loaded next.

In comparison to the SOPE methodology, GPE is more computationally demanding due to the requirement of keeping track of the available corner positions after each vehicle is placed. To do this, inner-fit polygons are calculated after each vehicle placement. Due to the staggered parking arrangements that occur in the GPE approach, a modified definition of a block of vehicles is required. Rows are constructed with the requirement that each consecutively placed vehicle is placed adjacent to the previous vehicle and as close to the ferry exit (bottom) as possible, continuing until no more vehicles fit in the row. The final touch in row construction is to shift the final vehicle away from the previous vehicle and towards the nearest side. This approach helps to reduce the creation of unusable gaps. Columns are constructed in an analogous but perpendicular manner, although without the equivalent final touch adjustment used in row construction.

In order to identify the efficient corner positions that are used in row and column construction, from all of the possible corners, the equation $(distance\ from\ bottom + 0.1 * distance\ from\ left)$ is minimized to identify the BL position, whilst the LB position is defined as that which minimizes $(0.1 * distance\ from\ bottom + distance\ from\ left)$. Similar equations are used to identify BR and RB positions when they are required. Additionally in the GPE methodology a fourth positional instruction is used, that of placing a single vehicle in the bottom-right position. This feature vastly increases the number of possible packing arrangements that can be implemented.

Figure 4 illustrates how the GPE algorithm reduces the occurrence of wasted space within blocks. The numbers on the vehicles in the diagram show the order of the blocks, and the yard queues illustrate how the queue constraints are respected by the requirement that block numbers are non-decreasing in each individual queue, -1 indicates that a queued vehicle did not fit on the ferry. The numbers above each queue indicate the target dimension of vehicles for those queues and whether that dimension refers to the vehicle width or length. The solutions depicted in Figure 4 were derived using the stage-wise iterative metaheuristic algorithm that will be introduced in Section 6.

Analogous to the representation of SOPE given in Equation 7, we represent the packing GPE encoder as

$$v \leftarrow h^{GPE}(p, q). \tag{8}$$

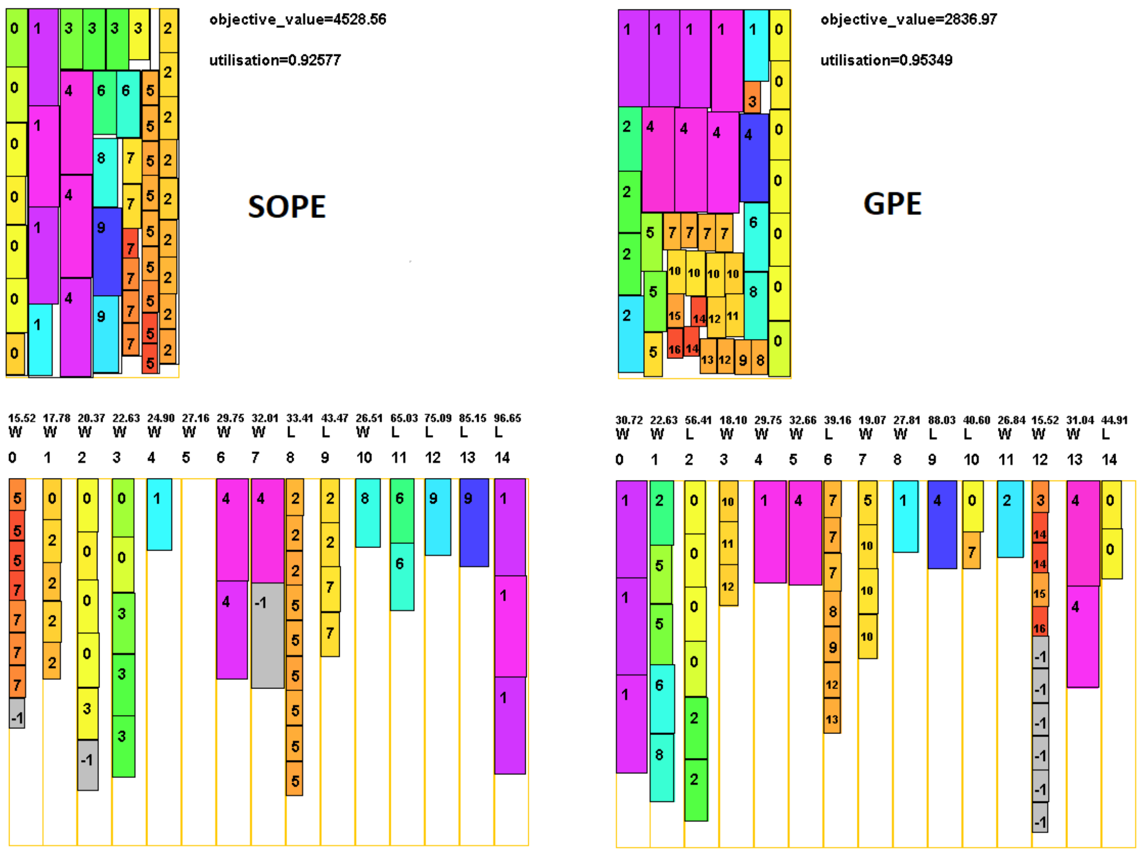


Figure 4: Illustration of how GPE reduces wasted space.

## 5 Evaluating Candidate Yard Policies and Packing Solutions

This section presents a mathematical formulation for the evaluation of candidate yard policies $y$ and sets of packing solutions $P$ that is used within the stage-wise iterative metaheuristic algorithm presented in Section 6.

Equations 9, 10 and 11 apply to both objective functions while Equation 13 is applicable for the expected value objective function and Equation 12 applies the vehicle mix intersection objective function. The evaluation of the objective value of candidate yard polices $y$ and packing solutions $P$ is a combinatorial optimisation problem that requires the identification of the best packing solution $p \in P$ for each scenario $s \in S$ such that the vehicle mix intersection is maximised or the expected penalty is minimised. For the former case a small branch and bound algorithm is used (Section 5.1), while for the latter the expected penalty is minimised by selecting the packing solutions with the lowest penalties for each scenario in the training set $S$. We set the number of candidate packing solutions to $|S|$, allowing each scenario its own packing solution.

We write the objective of minimizing the penalties associated with the numbers of vehicles of each type that are not able to be loaded, $m_j$, as

$$\min \sum_{j \in J} m_j \rho_j \tag{9}$$

and impose the constraint that

$$\sum_{p=1}^{|P|} x_{p,s} = 1, \forall s \in S, \tag{10}$$

where $x_{p,s}$ is a binary decision variable denoting whether or not the packing solution $p \in P$ is used for scenario $s \in S$. This means that each scenario must be allocated one packing solution and allows us to assign any candidate packing solution in $P$ to any scenario in $S$.

We define $v_{p,s}$ to be a vector with each element indicating the number of vehicles of a particular type that do not fit onto the ferry for a packing solution $p$ in scenario $s$. Here,

$$v_{p,s} \leftarrow h\left(p, g\left(y, s\right)\right), \forall s \in S, \forall p \in P, \tag{11}$$

which are obtained from the SOPE or GPE packing methodology introduced in Section 4. We package this set of equations as the function *YardPolicyEval(y,P,ν,S)* in what follows.

The values for $m_j$ depend on whether we are considering the expected penalties (EXP) or vehicle mix intersection (VMI) objective function according to the following equations:

$$m_j^{VMI} = \max_{s \in S} \nu_{p,s,j}, \ \forall j \in J \tag{12}$$

and

$$m_j^{EXP} = \frac{\sum_{s \in S} \nu_{p,s,j}}{|S|}, \ \forall j \in J, \tag{13}$$

where $m_j^{VMI}$ is the number of vehicles of type $j$ not in the vehicle mix intersection and $m_j^{EXP}$ is the average number of vehicles of type $j$ that cannot be fitted onto the ferry for training set $S$.

For the case of the expected penalties objective function (EXP) the $x_{p,s}$ values are found by selecting the packing solution that minimizes the financial penalty for each scenario $s \in S$. When using the vehicle mix intersection objective function a branch and bound algorithm is used to find the packing solution for each scenario. This is described in more detail in Section 5.1.

We also define *PackingSolutionEval(y,P* $\setminus \{p_t\}, p'_t$*,ν,S)*, which is used to evaluate the effect of replacing packing solution $p_t$ with an alternative packing solution $p'_t$ on the total penalties. *PackingSolutionEval(y,P* $\setminus \{p_t\}, p'_t$*,ν,S)* is the same as the *YardPolicyEval(y,P,ν,S)* formulation with the added constraint that the packing solution $p'_t$ is assigned to at least one scenario. To ensure that the alternative packing solution $p'_t$ is used, we add a final constraint (14) to the *YardPolicyEval(y,P,ν,S)* formulation:

$$\sum_{s \in S} x_{t,s} \geq 1. \tag{14}$$

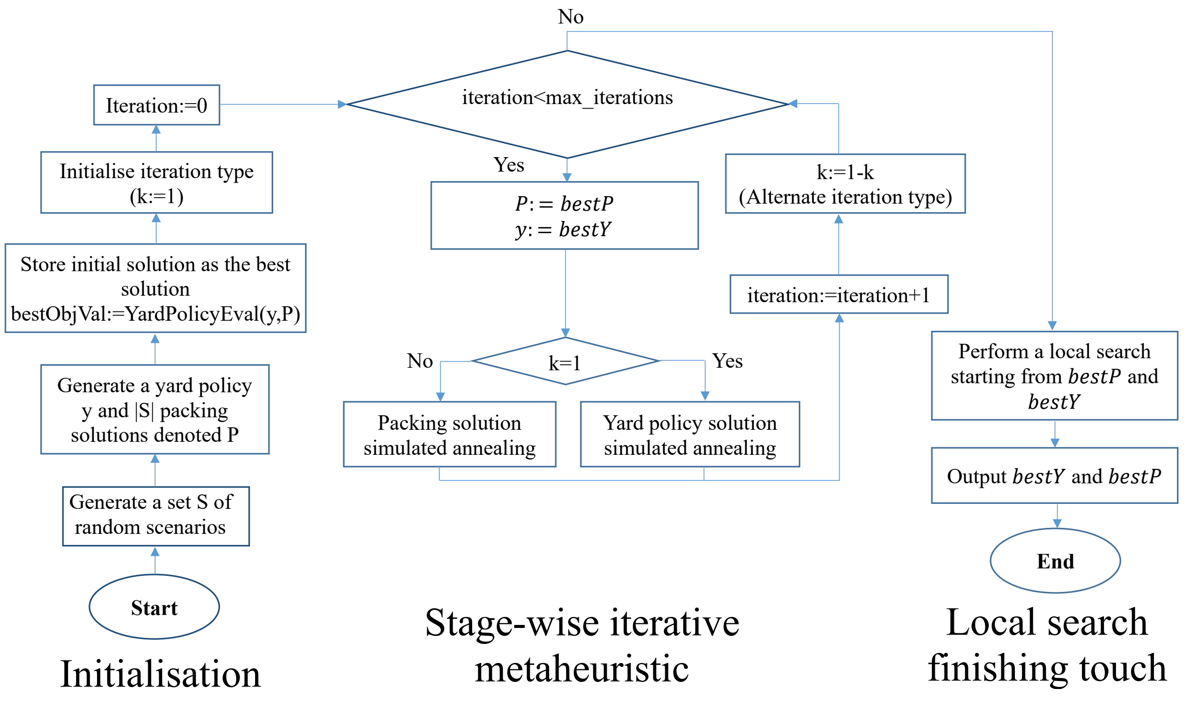
The function *PackingSolutionEval(y,P* $\setminus \{p_t\}, p'_t$*,ν,S)* is used in packing iterations of the stage-wise iterative metaheuristic algorithm that is introduced in Section 6 for solving the first stage yard policy problem.

## 5.1 Branch and Bound Algorithm for identifying combinations of packing solutions which maximise the vehicle mix intersection

For the case of the vehicle mix intersection objective function (2), evaluating a candidate solution involves selecting a packing solution from those in the current packing solution set $P$ for each scenario in the training set such that Objective (2) is minimised. This is achieved via an enumeration algorithm based on branch and bound logic, which proceeds as follows. Firstly an initial allocation of packing solutions is generated by selecting the packing solutions with the lowest penalties for each scenario in the training set. The sum of the penalties of the vehicles outside the vehicle mix intersection of this combination of packing solutions gives an initial upper bound on the penalties. Without considering all possible packing solutions, this can only be an estimate of the upper bound. The upper bound is used to eliminate packing solutions which have penalties that exceed the bound in at least one scenario, as this indicates that they cannot improve the vehicle mix intersection.

We build an enumeration tree in which the levels correspond to scenarios and the branches to packing solutions. As packing solutions are eliminated, their branches are removed. The tree is explored in a depth-first manner, with the branches at each new level sorted in increasing individual penalties. This approach means that the first solution generated by the procedure will be the solution found by a greedy heuristic, which sets the upper bound on penalties as low as possible early in the process. Doing so enables the bounding of branches as soon as possible. For each new branch, we calculate the vehicle mix intersection of the partial solution and the penalties associated with the vehicles not inside it. This allows us to immediately eliminate any branches that result in penalties exceeding the upper bound, improving the efficiency of the process. Every time the depth first search reaches the last scenario it indicates that a new improved lower bound has been found. At this point, packing solutions are ruled out for scenarios in which their corresponding penalty exceeds the new lower bound. The search continues iteratively by exploring the next branch of the parent node. When no more branches exist at a particular level, the search backtracks through the levels until a level is found that has unexplored branches. When all branches have been explored for all levels the solution is returned.

This approach exploits the problem structure to rule out possible solutions as fast as possible without having to explicitly explore them.



No
Iteration:=0
iteration<max_iterations
Yes
Initialise iteration type (k:=1)
P: = bestP
y: = bestY
k:=1-k (Alternate iteration type)
Store initial solution as the best solution bestObjVal:=YardPolicyEval(y,P)
iteration:=iteration+1
No
k=1
Yes
Perform a local search starting from bestP and bestY
Generate a yard policy y and |S| packing solutions denoted P
Packing solution simulated annealing
Yard policy solution simulated annealing
Output bestY and bestP
Generate a set S of random scenarios
End
Start
Initialisation
Stage-wise iterative metaheuristic
Local search finishing touch

Figure 5: Flowchart of the stage-wise iterative metaheuristic for yard policy optimization.

## 6 Stage-Wise Iterative Metaheuristic

The stage-wise iterative metaheuristic is used to identify a yard policy $y$ to use for the arrival process on departure day, such that the fitness function, *YardPolicyEval(y,P,ν,S,w)*, is minimised. The stage-wise iterative metaheuristic alternates between yard policy improvement, whilst fixing the set of candidate packing solutions, and improving the packing solutions whilst fixing the yard policy. This approach allows us to solve the problem efficiently by focusing on improving the decision made in a single stage of the problem at a time. In the first iteration, simulated annealing (Kirkpatrick et al., 1983) is used to improve the yard policy $y$. In the second iteration, simulated annealing is used to improve the packing solutions $P$. After this, the stage-wise iterative metaheuristic alternates between these two types of simulated annealing algorithm. We define the function $SAAcceptanceCriteria(newSol,$ $newObjVal, currentSol, currentObjVal, bestSol, bestObjVal, temperature)$ to denote the simulated annealing acceptance criterion. This function updates the current and best solutions, and their corresponding objective values, if a new solution is accepted. Figure 5 illustrates the stage-wise iterative metaheuristic for the optimization of the yard policy.

Every time simulated annealing is used to improve the yard policy $yardIterations$ iterations are performed. The temperature parameter in iteration $i$ is given by $temperature = t0\left(1 - \frac{i}{yardIterations}\right) bestOverallObjVal$ such that the $temperature$ decreases linearly with the simulated annealing iteration number $i$. Here, $t0$ is the initial temperature parameter. The temperature has a magnitude that is dependent upon the penalty arising from vehicles that cannot be packed in the best solution observed so far. The adaptive temperature scheme encourages a more aggressive search in the early iterations of the simulated annealing algorithm and convergence in the latter stages. In each iteration a neighbouring solution to that of the current yard policy ($y$) is generated using a mutation operator. The mutation operator for yard policy solutions is to select a random yard queue and to change the integer corresponding to that queue to a different value. Both the choice of the queue and the new integer value are sampled uniformly from all available options. The new candidate yard policy is evaluated using the $YardPolicyEval(z', P, \nu, S, B)$ fitness function. The simulated annealing acceptance criterion is then applied to see if the current and best solutions should be updated.

Every time simulated annealing is used to improve the packing solutions $packingIterations$ iterations are performed. The temperature parameter is the same as that used for the yard policy simulated annealing algorithm. Five local search neighbourhoods are considered for generating neighbouring solutions, and neighbourhood $j, j = 1,\dots,5$ has probability $\psi_j$ of selection in each iteration. The details of the five local search neighbourhoods are as follows.

1. Randomly mutate a single element of the initial candidate packing solution to any other different look up table value between 1 and $c_0 q_0$ (see Section 4.1).
2. Similar to the first approach but ensures that the mutation only changes the cut orientation encoded by an element of the solution.
3. Modify only the quantile value (vehicle size) encoded by a random element of the solution.
4. Swap the position of two elements of the initial solution.
5. The final approach is slightly more involved as it generates a completely new solution using a randomised constructive heuristic. This last approach is called random reasonable and proceeds by sequentially generating cuts. For each cut an orientation is randomly selected (uniform probabilities) and the target vehicle size (quantile value) is based on the dimension of a vehicle chosen at random from those remaining to be packed. The cut is then filled from the remaining vehicles, selecting those with the closest match target dimension while ignoring queue orders. This approach is based on the idea of generating a set of cuts so that there is just the right amount of space for the vehicles remaining to be packed.

Full details of the simulated annealing algorithm for improving packing solutions are provided in the accompanying appendix and a Java implementation of our approach is available at `https://github.com/chris8471155/QueueConstrainedPacking`.

The general framework of the proposed stage-wise iterative metaheuristic algorithm is also applicable to other two-stage stochastic optimisation problems. This could be achieved by replacing the first and second stage metaheuristic algorithms with algorithms relevant to the new problem and replacing the solution representations $y$ and $P$.

## 6.1 Finishing touch local search

Once the specified number of iterations have been performed, the best solution is subject to a finishing touch local search. This algorithm uses the same approach as that outlined in Figure 5, except that the simulated annealing iterations are replaced with a local search based on the same neighbourhood structures, i.e., full neighbourhood exploration where only solutions which are improvements upon the best solution are accepted. This approach ensures that the final solution is a local optimum. For the case of the packing solution local search, only the single element mutation and swap neighbourhoods are considered. The quantile and cut type neighbourhoods are part of the single element mutation neighbourhood, whilst the *random reasonable* neighbourhood is a restart mechanism. As a result, these three neighbourhoods are not useful when looking for a local optimum close to a given solution.

In generating results for Section 7, we identified the $\psi$ and $t0$ values that gave the best results for each of the 300 scenarios considered. The particular values for $\psi$ and $t0$, for each test instance are available online (Bayliss et al., 2019b). In this work it was found that $max\_iterations = 30$ and $yardIterations = packingIterations = 500$ was an effective choice.

## 7 Lower bounds on wasted space

In this section we describe an algorithm termed WASTELB for evaluating a lower bound on the wasted space (equivalently an upper bound on space utilisation) for any packing problem in which a set of smaller rectangles is being packed inside a larger rectangular container. In other work, lower bounds on waste for this special case of a two-dimensional rectangle knapsack packing problem are relatively few. Beasley (1985b) proposed a linear relaxation allowing fractions of items to be packed and a knapsack relaxation whereby only the total volume of items is required to be feasible. Young-Gun and Kang (2002) provide an upper bound on profit for the unconstrained problem, where the numbers of items of each type are not fixed. Our approach adapts the upper bound approach of Young-Gun and Kang (2002) for the constrained case to calculate a lower bound on wasted space. Other related work on bounds include Fekete and Schepers (2004) who provide a general framework for lower bounds on utilisation (upper bounds on waste) for packing problems. For other types of packing problems which are more amenable to exact approaches, such as strip packing and bin packing, bounds are more frequently available, such as in Khanafer et al. (2010); Martello and Vigo (1998); Yu et al. (2017). We leave a comparison of packing bounds as an open topic and note that the proposed bound is sufficient for proving a relatively small optimality gap for the proposed packing algorithms.

The WASTELB approach is formalised in Algorithm 1 and is based on identifying vehicle blocks (rows or columns of vehicles) that are the most efficient in the horizontal or vertical direction, respectively. We assume that the wasted length left after these 1-D efficient blocks have been packed extends along the perpendicular length of the ferry, thus providing a lower bound on the wasted area. Another simpler bound, that can be stronger than that of Algorithm 1, is to find the set of vehicles whose areas sum as closely as possible to the size of the ferry without exceeding it. The difference between the ferry size and the sum of the areas of those vehicles can act as a further (most often weaker) lower bound on the space that will always be wasted in the ferry.

**Algorithm 1** WASTELB algorithm

```
1: Inputs: Vehicle mix vector $M$, vehicle widths and lengths ($\beta$, $\alpha$), Ferry dimensions $W$ and $L$.
2: Find the width $\chi$ of the most efficient horizontal pattern of vehicles whose sum of widths is closest to W,
   where $n_j$ is the number of vehicles of type $j$ in that pattern.
3: $\chi = \max_{n_j \in \mathbf{N}\ \forall j \in J} \left\{ \sum_{j \in J} n_j \beta_j \middle| \sum_{j \in J} n_j \beta_j \leq W, n_j \leq M_j\ \forall j \in J \right\}$
4: Find the length $\theta$ of the most efficient vertical pattern of vehicles whose sum of lengths is closest to L, where
   $n_j$ is the number of vehicles of type $j$ in that pattern.
5: $\theta = \max_{n_j \in \mathbf{N}\ \forall j \in J} \left\{ \sum_{j \in J} n_j \alpha_j \middle| \sum_{j \in J} n_j \alpha_j \leq L, n_j \leq M_j\ \forall j \in J \right\}$
6: Calculate lower bound waste contribution and upper bound utilisation
7: $wasteLB = (W\theta + L\chi) - (\theta\chi)$
8: $utilUB = \frac{WL - wasteLB}{WL}$
9: Outputs: $wasteLB$, $utilUB$
```

We evaluate our methods against lower bounds calculated by Algorithm 1 on 300 test instances, where each instance corresponds to a set of booked vehicles. The test instances can be split into three equal-sized classes, with the 100 instances in each class ranging from one vehicle type in the first instance to 100 vehicle types in the final instance. The instance number, $m = 1, \ldots, 100$, dictates the number of vehicle types that are sampled from in that instance to form the vehicle mix. Vehicle types are related as described below.

- Class 1 instances: the lengths and widths of the vehicles both increase between one vehicle type and the next. This class can be described as having a nested structure similar to a set of Russian dolls.
- Class 2 instances: vehicle widths increase while their length decrease between one vehicle type and the next. This means that there is no nested structure among the vehicle types.
- Class 3 instances: the lengths and widths of the booked vehicles are sampled uniformly and independently. For each vehicle, a width is sampled from $modulo(m, 10)$ alternative values for the width and a length is sampled from $\lceil \frac{m}{10} \rceil$ alternative values for the length. This means that class 3 has a wide range of instances for vehicle dimensions, which also have some nesting of vehicle types.

All of the instance are based on vehicles with widths uniformly distributed in the range $[1.6, 3.4]$ metres, lengths uniformly distributed in the range $[3, 11]$ metres. The single vehicle type instances that mark the first instance of each class use the mid-points of these intervals. The ferry is a rectangle with length 37.21 metres and width 17.42 metres. The 300 problem instances are available online at Bayliss et al. (2019b). Each instance was generated by sampling the vehicle dimension distributions defined above until immediately after the first time the area of vehicles exceeded the space available on the ferry.

Figure 6 shows the ferry utilisation rates achieved by SOPE and GPE alongside the estimates of the upper bounds for utilisation derived from WASTELB and the bottom-left decreasing packing heuristic. The figure shows that both SOPE and GPE outperform the bottom-left heuristic and that the GPE method outperforms SOPE. In fact this ordering is only broken for trivial instances involving just one vehicle type, which mark the beginning of each class of problem instance, and for several non-trivial instances in class 3, where SOPE marginally outperforms GPE. The upper bound utilisations are good enough to show that in some non-trivial instances, optimal solutions were obtained by the SOPE and GPE algorithms. Optimal solutions were obtained for test instances in class 3 with only a small number of alternative widths and/or lengths for vehicles. In such instances the upper bound formulation was able to find relatively strong bounds.

Figure 7 shows the distribution of upper bound estimates for the optimality gaps for GPE over the 300 test instances and suggests that all of the solutions were within 5% of the true optimal solution with an average gap of 2.73%. The results show that GPE outperforms SOPE and so we implement GPE in the numerical experiments in Section 8.

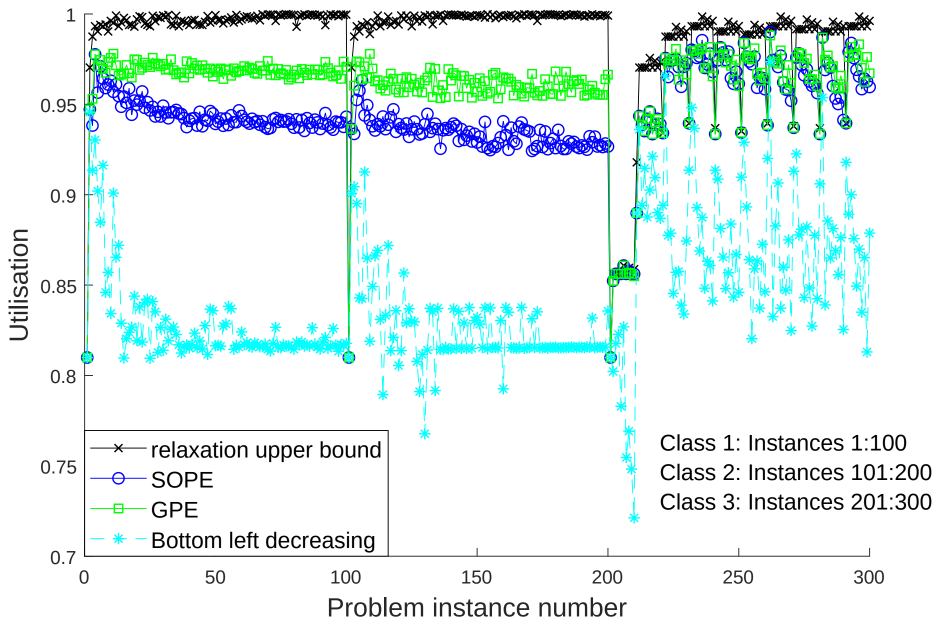


Figure 6: Ferry utilisation values for 300 test instances for the SOPE-based packing methodologies compared to an upper bound derived from WASTELB and the utilisations achieved by the bottom-left decreasing packing heuristic.

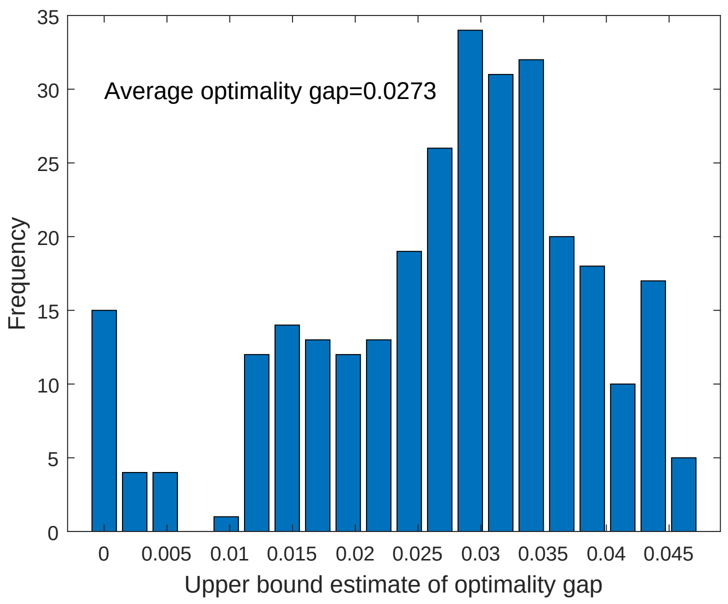


Figure 7: Frequency plot of upper bound optimality gap estimates for the GPE packing methodology.

## 8 Experimental Results

This section provides experimental results which illustrate the effect of the number of parallel vehicle queues on packing efficiency and relative performance of the VMI and EXP objective functions. The results reported in this section are based on testing the yard policies generated using the training set of scenarios, in a larger and different test set of scenarios. The test sets consist of 1000 scenarios, different from those used in the training sets but within the same three classes of problem instance described in Section 7. Full details are available online (Bayliss et al., 2019b). Each instance defines a ferry size and a set of booked vehicles, each with specified sizes. The yard queue dimensions for each instance are defined by the equation $yardQueueLength = \max\left(maxVehicleLength, \frac{3 * totalAreaOfVehicles}{numberOfQueues * yardQueueWidth}\right)$, where $yardQueueWidth$ is equal to the widest vehicle in the given test instance, and $totalAreaOfVehicles$ is equal to the sum of the areas of the booked vehicles. This approach ensures that all booked vehicles fit into the yard prior to the beginning of the loading process. The instances used for the experimental results presented in this section are defined by their class number and instance number within that class. For example, instance 1-65 denotes instance 65 in class 1. The penalty for failing to load a vehicle is equal to the area occupied by that vehicle.

## 8.1 The Effect of the Number of Yard Queues for Different Yard Policy Approaches

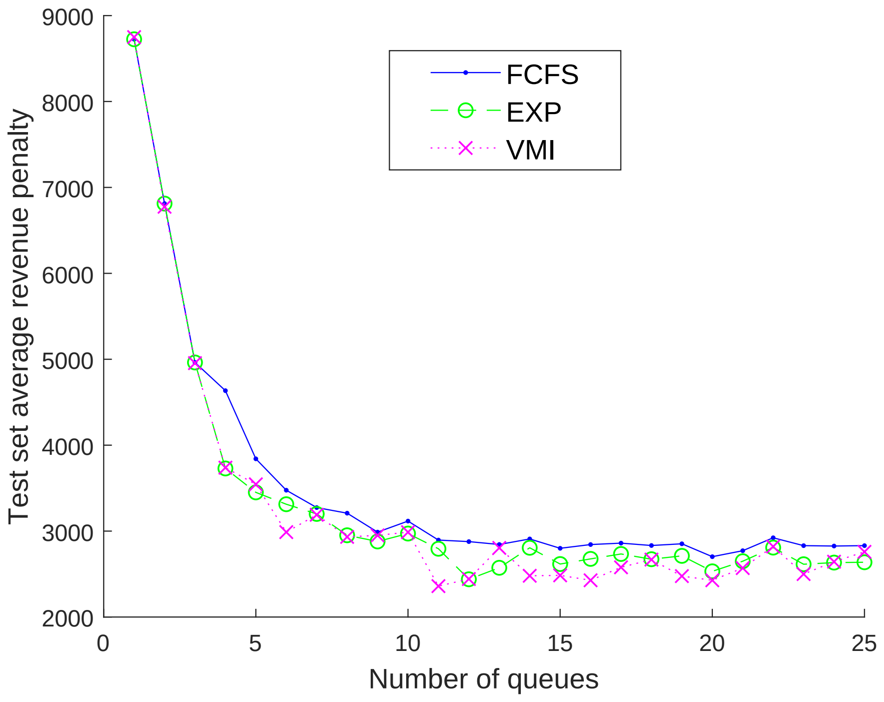


Figure 8: The effect of the number of yard queues on average test set revenue penalties.

Figure 8 shows the effect that the number of yard queues and the yard policy approach have on the average test set penalties, illustrating the intuitive result that fewer yard queues will constrain the ferry packing problem more and lead to poorer packing efficiency and therefore higher average revenue penalties. In this example, increasing the number of yard queues beyond 11 or 12 provides no noticeable benefit in terms of reduced average revenue penalties. We consider yard layouts with between 1 and 25 yard queues and consider three different yard policy approaches: FCFS; EXP ($|S| = 15$); and VMI ($|S| = 15$). The first come first served heuristic (FCFS) yard policy approach is included as a heuristic benchmark for comparison with the proposed EXP and VMI approaches. The set of possible target vehicle dimensions consists of the set of widths and lengths of the booked vehicles in the given problem instance. The FCFS yard policy consists of assigning an even distribution of target vehicle dimensions across the available yard lanes. Upon arrival vehicles are added to the yard lane with the closest associated target dimension. The FCFS yard policy is based on trying to apply the maximum amount of sorting of vehicles during the arrival process. Figure 8 suggests that the FCFS yard policy approach leads to the highest average revenue penalties for all numbers of yard queues. The VMI approach leads to the lowest average penalties overall, although the EXP approach appears to perform better in a few cases (where there are 9, 13 and 25 yard queues).

## 8.2 Comparison of Objective Functions

The results in Figure 9 show the average penalties incurred in a test set consisting of 1000 scenarios, where these scenarios are different to those used in the training sets. These provide a comparison of the VMI objective function, with the EXP objective function.

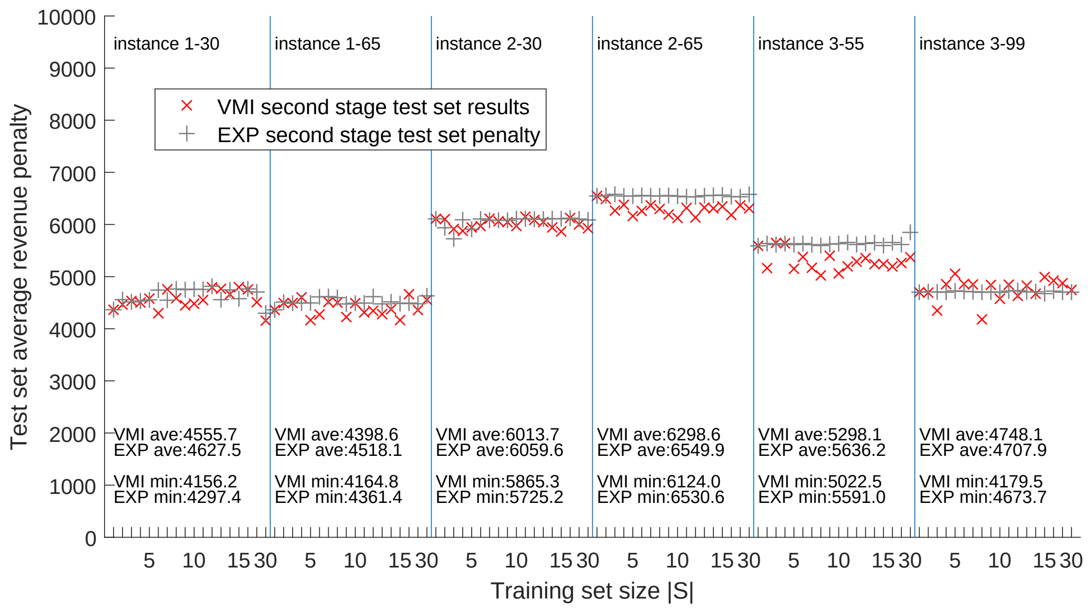


Figure 9: Comparison of the total penalties and the penalties incurred on departure day due to failing to load vehicles achieved with yard policies derived using expected penalty and VMI objective functions.

Figure 9 compares the performance of the VMI and EXP objective functions in six different instances: 2 instances from each class (instances 30 and 65 from class 1; instances 30 and 65 from class 2; and instances 55 and 99 from class 3). Full details of these instances are available in Bayliss et al. (2019b). For each instance the size of the training set is varied between a minimum of 1 and a maximum of 30 scenarios. Figure 9 shows that for five out of the six instances tested, the VMI objective function obtains the lowest average and minimum values for the average test set penalty. In addition, Figure 9 shows that the training set size appears to have little effect on the penalties observed in the test set. The explanation for this is that the average penalty values displayed in Figure 9 are those attained after solving the packing recourse problem based on the yard queue orders generated by the yard policy found in the first stage problem. The packing recourse problem allows for a large amount of recoverability, and as a result reduces the observable difference between different yard policy approaches. During the training phase for the case of the VMI objective function, the finishing touch local search step found improvements in 9 out of 108 instances in the training set, with a 5.83% average improvement in those 9 cases. For the case of the EXP objective function, the finishing touch local search step found improvements in 69 out of 108 instances, with a 2.85% average improvement in those 69 cases.

In Figure 10 we investigate the importance of the size of the training set by comparing the average penalties observed in the test set for the VMI and EXP objective functions attained for training set sizes between 1 and 25 before. We show the average penalties both before and after solving the second stage packing recourse problem. The average test set penalties before recourse are those obtained using the packing solution in $P$ (the best set of packing solutions for the training set identified in stage 1) that result in the lowest penalty for each test set scenario. Figure 10 also displays the average test set penalties after solving the packing recourse problem for the cases of the VMI and EXP first stage objective functions. From Figures 9 and 10 we conclude that the objective function used to solve the first stage yard policy problem is more important than the size of the training set used when solving the first stage problem. It may be the case that training set sizes of hundreds or thousands of scenarios might improve results, however our experimental results suggest that the potential additional benefit to be quite limited given that the objective function used in the first stage, the number of queues and the flexibility of packing recourse are the dominant factors affecting solution quality, as shown in Figures 8, 9 and 10. The apparent insensitivity of the average test set penalties to the size of the training set may be attributed to the concept referred to as coincidental coverage, in which a small number of training scenarios can lead to solutions of equal quality to those obtained using larger training sets because a single scenario encompasses important information relevant to all scenarios.

Figure 11 shows that the first stage solution times for both the VMI and EXP objective functions increase sharply as the training set size increases. Fitting a quadratic curve to the results for the VMI objective solution times suggests an $R^2$ of 0.977, and similar results are obtained for the EXP objective solution time results. This suggests that solution times are likely to be quadratic in the size of the training set size. This makes sense given

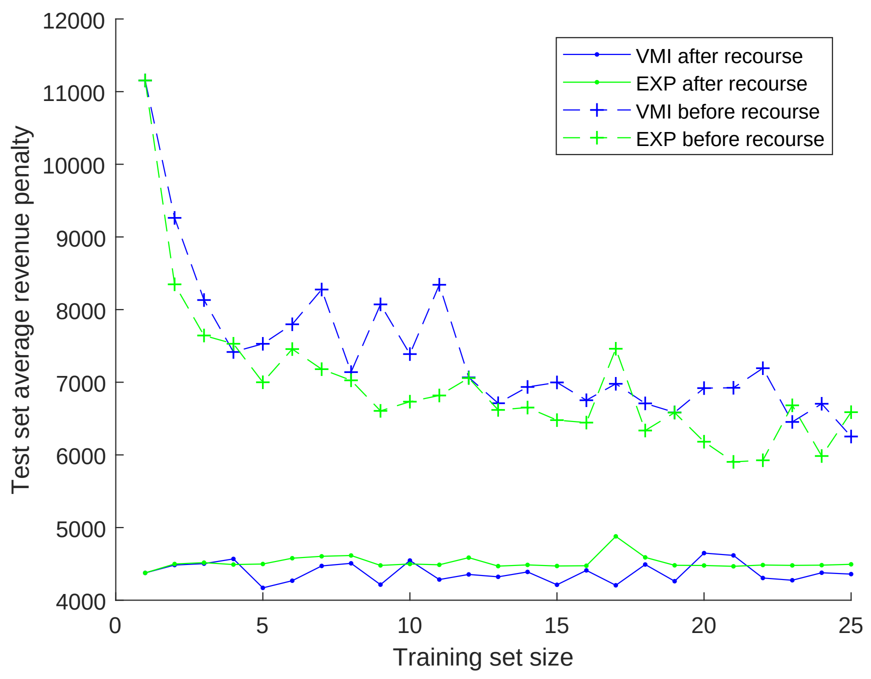


Figure 10: Comparison of the average test set penalties before and after solving the second stage packing recourse problem.

that in each packing iteration of the proposed stage-wise iterative metaheuristic, $|S|$ packing solutions are tested in each of $|S|$ scenarios.

## 9 Conclusion

The article describes a methodology with two key components: a stage-wise iterative metaheuristic that jointly optimizes the order of dockside queues and the packing of vehicles onto the deck of a ferry; and a computational method that translates the results of this optimization routine into a practical arrangement of vehicles.

The key methodological contributions of the paper, which both have the potential to be used in other application areas, include the development of SOPE and GPE for the packing problem and the two-stage iterative metaheuristic. SOPE and GPE are efficient heuristics used for finding good solutions to 2-D rectangle packing problems when the packing order of the rectangles is queue-constrained. The two-stage iterative metaheuristic works to jointly optimize two combinatorial optimization problems in the presence of uncertainty. Of particular note is the novel packing method that is flexible enough to allow for uncertain arrival orders, queue-constrained packing and vehicle manoeuvrability. Simulation testing showed that solutions obtained using the VMI objective function in the metaheuristic perform better on average than those found using the EXP expected value objective function.

As we show in the results, increasing the number of yard queues improves the efficiency of the packing but even when the number of yard queues equals the number of vehicle types, the queue constraints can still result in inefficiencies in the packing, underlining the need for a method such as ours for jointly optimizing the yard policies and the packing. In addition to minimising penalties incurred due to leaving vehicles off the ferry, the program developed can also be used as a training tool for new loaders and dockside staff, in conjunction with the simulation model developed in previous work Bayliss et al. (2019a). The ferry company we work with, which we believe to be typical of other ferry operators, do not currently use software to aid packing decisions but in man versus machine experiments, the SOPE or GPE packing methodologies produce more efficient packing arrangements in a fraction of the time required by a human operator. This shows its potential utility as a decision support tool in the ferry industry. The proposed approaches can be used to generate yard policies tailored to the set of booked vehicles by solving the first stage problem. In addition, the SOPE and GPE packing encoders can be used to solve the packing recourse problem once the queue orders are known, which is possible because for a single scenario with fixed queue orders the packing problem is solved within seconds.

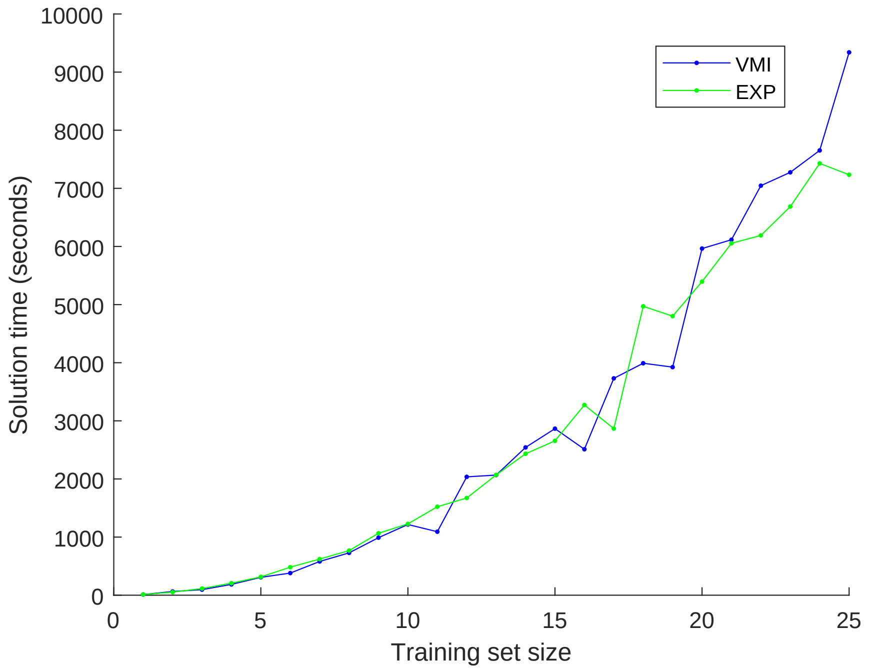


Figure 11: The effect of training set size on the time required to solve the first yard policy problem for the VMI and EXP objective functions.

This work suggests several avenues for future research. Of most interest is to develop an efficient packing method when there are multiple pick up and drop off destinations. This could be a ferry service that operates between multiple ports on one route or alternatively to a more general delivery problem where items need to be packed to allow them to be unloaded at several different locations. Additionally, for the case of a single queue, where the order of the queue represents a vehicle delivery order, the proposed algorithm also provides solutions for the problem of packing a vehicle that is constrained by its delivery route. Future work could then include an integration of a vehicle routing algorithm and either of the SOPE or GPE packing encoders within a stage-wise iterative metaheristic framework.

## ACKNOWLEDGEMENTS

This work was funded by the EPSRC under grant number EP/N006461/1. We are also grateful to Alexander Strong, Lee Hudson and Pete Smith at Red Funnel, UK who provided us with useful practical advice when designing this method.

## REFERENCES

Al-Khamis, T. and M'Hallah, R. (2011). A two-stage stochastic programming model for the parallel machine scheduling problem with machine capacity. Computers and Operations Research, 38:1747–1759.

Alvarez-Valdes, R., Parreno, F., and Tamarit, J. M. (2005). A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. The Journal of the Operational Research Society, 56(4):414–425.

Balakrishnan, H. and Chandran, B. G. (2010). Algorithms for scheduling runway operations under constrained position shifting. Operations Research, 58(6):1650–1665.

Bayliss, C., Bennell, J. M., Currie, C. S., Martinez-Sykora, A., and So, M.-C. (2016). A simheuristic approach to the vehicle ferry revenue management problem. In Roeder, T. M. K., Frazier, P. I., Szechtman, R., Zhou, E., Huschka, T., and Chick, S. E., editors, Proceedings of the 2016 Winter Simulation Conference, pages 2335–2346.

Bayliss, C., Currie, C. S., Bennell, J. M., and Martinez-Sykora, A. (2019a). Dynamic pricing for vehicle ferries: using packing and simulation to optimize revenues. European Journal of Operational Research, 273:288–304.

Bayliss, C., Currie, C. S., Bennell, J. M., and Martinez-Sykora, A. (2019b). Test instance data for sections 7 and 8. https://eprints.soton.ac.uk/435288/. last accessed April 2020, https://doi.org/10.5258/SOTON/D1130.

Beasley, J. (1985a). An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research, 33(1):49–64.

Beasley, J. E. (1985b). Bounds for two-dimensional cutting. The Journal of the Operational Research Society, 36(1):71–74.

Bekrar, A. and Kacem, I. (2009). An exact method for the 2d guillotine strip packing problem. Advances in Operations Research.

Carvalho, J. M. V. D. (2002). LP models for bin packing and cutting stock problems. European Journal of Operational Research, 141:253–273.

Cintra, G., Miyazawa, F., Wakabayashi, Y., and Xavier, E. (2008). Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. European Journal of Operational Research, 191:61–85.

da Silveira, J. L. M., Xavier, E. C., and Miyazawa, F. K. (2011). Two dimensional knapsack with unloading constraints. Electronic Notes in Discrete Mathematics, 37:267–272.

De Maere, G., Atkin, J. A. D., and Burke, E. K. (2017). Pruning rules for optimal runway sequencing. Transportation Science. articles in advance.

Dean, B. C., Goemans, M. X., and Vondrak, J. (2008). Approximationg the stochastic knapsack problem: The benefit of adaptivity. Mathematics of Operations Research, 4:945–964.

Fekete, S. P., Khler, E., and Teich, J. (2006). Higherdimensional packing with order constraints. SIAM Journal on Discrete Mathematics, 20(4):1056–1078.

Fekete, S. P. and Schepers, J. (2004). A general framework for bounds for higher-dimensional orthogonal packing problems. Mathematical Methods of Operations Research, 60:311–329.

Fleszar, K. (2013). Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. Computers and Operations Research, 40:463–474.

Gendreau, M., M. Iori, G. L., and Martello, S. (2007). A tabu search heuristic for the vehicle routing problem with two-dimensional loading. Networks, 51:4–18.

Goncalves, J. F. and Resende, M. G. (2013). A biased random key genetic algorithm for 2D and 3D bin packing problems. Int. J. Production Economics, 145:500–510.

Guimarans, D., Dominguez, O., Panadero, J., and Juan, A. A. (2018). A simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times. Simulation Modelling Practice and Theory, 89:1–14.

Hopper, E. and Turton, B. C. H. (2001). A review of the application of meta-heuristic algorithms to 2d strip packing problems. Artificial Intelligence Review, 16:257–300.

Iori, M., S-Gonzalez, J.-J., and Vigo, D. (2007). An exact approach for the vehicle routing problem with two-dimensional loading constraints. Transportation Science, 41:253–264.

Kang, J. and Park, S. (2003). Algorithms for the variable sized bin packing problem. Eur. J. Oper. Res., 147:365–372.

Khanafer, A., Clautiaux, F., and Talbi, E.-G. (2010). New lower bounds for bin packing problems with conflicts. European Journal of Operational Research, 206:281–288.

Kim, S., Pasupathy, R., and Henderson, S. G. (2015). A Guide to Sample Average Approximation, chapter 8. Handbook of Simulation Optimization. Springer, New York.

Kirkpatrick, S., Gelatt, C. D., Jr, and Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598):671–680.

Leung, S. C. H., Zhang, D., Zhou, C., and Wu, T. (2012). A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. Computers and Operations Research, 39:64–73.

Levin, A. and Vainer, A. (2013). The benefit of adaptivity in the stochastic knapsack problem with dependence on the state of nature. Discrete Optimization, 10:147–154.

Lodi, A., Martello, S., and Monaci, M. (2002). Two dimensional packing problems a survey. European Journal of Operational Research, 141:241–252.

Lodi, A., Martello, S., and Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing, 11(4):345–357.

Martello, S. and Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. Management Science, 44(3):388–399.

Martinez-Sykora, A., Currie, C. S., So, M.-C., Bayliss, C., and Bennell, J. M. (2017). Optimizing pricing and packing of variable sized cargo. Under review at Transportation Science.

Morabito, R. and Arenales, M. N. (1996). Staged and constrained two-dimensional guillotine cutting problems: An and/or-graph approach. European Journal of Operational Research, 94:548–560.

Mrad, M., Meftahi, I., and Haouari, M. (2013). A branch-and-price algorithm for the two-stage guillotine cutting stock problem. The Journal of the Operational Research Society, 64(5):629–637.

Navarra, A. and Pinotti, C. M. (2017). Online knapsack of unknown capacity: How to optimize energy consumption in smartphones. Theoretical Computer Science, 697:98–109.

Papastavrou, J. D., Rajagopalan, S., and Kleywegt, A. J. (1996). The dynamic and stochastic knapsack problem with deadlines. Management Science, 42(12):1706–1718.

Pisinger, D. and Sigurd, M. (2005). The two-dimensional bin packing problem with variable bin sizes and costs. Discrete Optimization, 2:154–167.

Talluri, K. and Ryzin, G. V. (2004). The Theory and Practice of Revenue Management. Springer Science, New York, USA.

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. European Journal of Operational Research, 183, 109-1130.

Wei, L., Oon, W.-C., Zhu, W., and Lim, A. (2011). A skyline heuristic for the 2d rectangular packing and strip packing problems. European Journal of Operational Research, 215(2):337–346.

Young-Gun, G. and Kang, M.-K. (2002). A new upper bound for unconstrained two-dimensional cutting and packing. The Journal of the Operational Research Society, 53(5):587–591.

Yu, G., Mao, Y., and Xiao, J. (2017). New upper bounds for online strip packing. Discrete Optimization, 23:20–32.

## A Details of the packing solution simulated annealing algorithm

The packing solution simulated annealing algorithms is now described.

**Algorithm 2** Packing solution simulated annealing algorithm

```
1:  Input: Set of currently accepted candidate packing solutions $C$, set of current best packing solutions $P$, current best yard policy $y$, set of scenarios $S$, $temperature$, an objective function and a packing encoder (SOPE or GPE), $packingIterations$ the number of iterations of the packing solution simulated annealing algorithm, $i$ current iteration of the packing solution simulated annealing algorithm.
2:  $C$ : current set of packing solutions, $C \leftarrow P$ if $i = 1$
3:  $C' \leftarrow \emptyset$ (the set of alternative packing solutions generated in this packing solution simulated annealing iteration)
4:  for $d \in \{1..|C|\}$ do
5:      $temperature = t0\left(1 - \frac{i}{packingIterations}\right) bestObj_d$
6:      Select neighbouring solution type $\pi$ randomly according to the probability distribution $\psi$
7:      Generate the neighbouring solution $c'_d \leftarrow neighbour(\pi, c_d)$
8:      $C' \leftarrow C' \cup \{c'_d\}$
9:      Implement packing solution $c'_d$ in each scenario $s \in S$ to obtain $H_{c'_d,s,j}$ and $Penalty_d\left(y, c'_d\right)$ the penalty of packing solution $c'_d$ in the $d^{th}$ scenario.
10:     $objVal = PackingSolutionEval\left(y, P \setminus \{p_d\}, c'_d, v, S\right)$
11:     Check if a new best overall solution has been found
12:     if $objVal < bestOverallObjVal$ then
13:         $p_d \leftarrow c'_d$
14:         $c_d \leftarrow c'_d$
15:         $bestOverallObjVal \leftarrow objVal$
16:     else
17:         Check if the packing solution $c'_d$ should replace $c_d$ as the current candidate packing solution dedicated to the $d^{th}$ scenario
18:         $weightedObjVal_d = \omega \times objVal + (1-\omega) Penalty_d\left(y, c'_d\right)$
19:         $SAAcceptanceCriteria\left(c'_d, weightedObjVal_d, c_d, currentObj_d, -, bestObj_d, temperature\right)$
20:     end if
21: end for
22: Check if the full current set of alternative packing solutions $C'$ leads to a best overall solution
23: $objVal = YardPolicyEval(y, C', v, S, B)$
24: if $objVal > bestOverallObjVal$ then
25:     $P \leftarrow C'$
26:     $C \leftarrow C'$
27:     $bestOverallObjVal \leftarrow objVal$
28: end if
29: $subIt \leftarrow subIt + 1$
30: Output: Current accepted candidate packing solution set $C$ and current packing solution set $P$.
```

Algorithm 2 corresponds to one iteration of to the *packing solution simulated annealing* block of Figure 5. The method carries out *packingIterations* iterations before switching back to the yard policy simulated annealing algorithm and resetting $i = 1$. The algorithm aims to find good sets of packing solutions that improve the objective function value over all scenarios. For the case in which the expected value objective function is used, this can be achieved by optimising the packing solution for each separate scenario. However, for the case of the VMI objective function, the overall objective value may be better even if individual sailings are packed less efficiently.

Algorithm 2 begins, for the case where $i = 1$, by initialising the current set of packing solutions $C$ to the best overall set of packing solutions $P$. It then considers each of the current packing solutions in turn. For each, the first step taken is to update the temperature parameter of the simulated annealing acceptance criterion. In line 6, a neighbouring solution $c'_d$ of the current packing solution under consideration $c_d$ is generated. This solution $c'_d$ is generated by mutating $c_d$ in one of five ways, where the mutation type is selected randomly from the probability distribution $\psi$.

Following the generation of $c'_d$, it is implemented for the training set of scenarios $S$, using either SOPE or GPE, to generate the number of vehicles of each type that can be loaded in each scenario (the $H_{c'_d}$ values). This information is used in the *PackingSolutionEval* fitness function that was described in Section 5, which finds the best overall objective value of the set of current best packing solutions $P$ with the constraint that $c'_d$ is used in at least one scenario. If the inclusion of $c'_d$ in $P$ leads to a new overall best solution, the best overall solution is updated (lines 10-13). Following this, on line 15, a weighted sum objective value ($weightedObjVal$) is calculated for $c'_d$, which gives a weight of $\omega$ ($\in [0,1]$, 0.5 in this case) to $objVal$ that was calculated on line 8, and a weight of $(1-\omega)$ to the penalties incurred when the packing solution $c'_d$ is used in scenario $d$, i.e., the value of the candidate solution $c'_d$ when devoted to the $d^{th}$ of $|S|$ scenarios. This approach is designed to encourage good packing solutions for each individual training set scenario even if they do not immediately improve the overall objective value. The simulated annealing acceptance criterion is then applied to see if $c'_d$ should replace $c_d$ in the current set of packing solutions $C$. Having considered the refinement of each of member of $C$ in lines 3-17, the *YardPolicyEval* fitness function is computed (line 19) for $C'$ to see if a new overall best solution results from replacing $P$ with $C'$ (lines 20-23).