Research Article

Algorithms & Computational Technology

Journal of Algorithms & Computational Technology
Volume 15: 1–11
© The Author(s) 2021
DOI: 10.1177/1748302621994010
journals.sagepub.com/home/act
SAGE

Reshuffle minimisation to improve storage yard operations efficiency

Hammed Bisira and Abdellah Salhi

Abstract

There are many ways to measure the efficiency of the storage area management in container terminals. These include minimising the need for container reshuffle especially at the yard level. In this paper, we consider the container reshuffle problem for stacking and retrieving containers. The problem was represented as a binary integer programming model and solved exactly. However, the exact method was not able to return results for large instances. We therefore considered a heuristic approach. A number of heuristics were implemented and compared on static and dynamic reshuffle problems including four new heuristics introduced here. Since heuristics are known to be instance dependent, we proposed a compatibility test to evaluate how well they work when combined to solve a reshuffle problem. Computational results of our methods on realistic instances are reported to be competitive and satisfactory.

Keywords

Reshuffle, retrieval, container terminal, binary integer programming, heuristic

Received 16 August 2019; revised 14 January 2020; accepted 7 April 2020

Introduction

Containers are often stored temporarily in the yard between the time they arrive at the port either through vessels in the case of import and through external trucks in the case of export. In practice, containers for loading are placed in the export area and those unloaded from ships are placed in the import area.[1] Some ports stack export containers close to the quay and import containers close to the landside gate. Others have a dedicated area for marshalling containers just unloaded from or to be loaded onto vessels. Fast access to stored containers is a major concern in container terminals. Choosing an appropriate location for a container that is to be relocated is essential in reducing the subsequent reshuffles. There are many ways of tackling this problem.

When containers arrive there is need to have a storage strategy to stack them in order to make retrieval efficient. Container types are stored in different places (for instance 20-foot containers are stored separate from 40-foot containers). Where they are stored together, the 20-foot containers are stacked beneath the 40-foot because the latter have a higher priority when loading onto a vessel. The storage location could also be based on whether the container is full or empty, the destination of vessel and even their weight. After storage, containers can be reshuffled/pre-marshalled in advance of retrieval. The aim here is to change the initial layout of the block to a desired layout to facilitate retrieval.[2]

If the containers are placed exactly where they can quickly be accessed, the retrieval process will be efficient. However, this is hardly the case due to many reasons especially inaccurate information as to when the containers will be retrieved when storing them.[3] Other reasons include change in vessels arrival time due to delay, external trucks arriving late due to traffic and the vessels stowage plan amongst others. Since only containers at the top of a stack are accessible, there is usually need to reshuffle containers in order to retrieve the desired container beneath them. This process takes time and thus hampers the operation of

Department of Mathematical Sciences, University of Essex, Essex, UK

Corresponding author:
Hammed Bisira, Department of Mathematical Sciences, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK.
Email: hbisir@essex.ac.uk

yard cranes YCs which will delay the trucks and/or the vessels.

We are interested in retrieving all the containers in a bay at minimal reshuffle. However, decision on where to place a reshuffled container is not as easy as it looks because it affects subsequent retrievals. Even in a static case where there is no arrival of new containers while stacking, the problem is still dynamic due to the fact that the configuration of the bay changes each time there is a retrieval. In Kim and Hong[4] it was suggested that in order to minimise reshuffles, the storage location of incoming container should be well assigned and the location of a reshuffled container should also be determined. For the purpose YC drivers are given work orders in the form of a movement sequence. The movement sequence contains the order of container movements, instructing the driver which container to move, where and when to move them Lee and Chao.[5]

Containers can be classified using different attributes such as weight, port of destination, length, being inbound or outbound and full or empty. This determines where they are stored for easy retrieval. For instance, a container destined for a farther port has a higher priority when loading onto a vessel since it will be unloaded later than a container destined for a port that is nearer. Heavier containers are stored in higher tiers in the yard since they are loaded in lower parts of vessels and are thus retrieved earlier. Containers that are stored earlier are likely to need earlier retrieval but would be buried under later arriving containers. Hence, there is need for reshuffle.

Literature review

The literature on selecting a storage strategy includes Van Asperen et al.[6] which investigated the role truck arrival could have on the stacking policy of a container terminal. They concluded that it will be more beneficial to improve available information at the time of stacking than attempting to fix poor stacking decisions later. This was done using discrete-event simulation to evaluate expected departure time for an import container to schedule the pre-emptive remarshalling moves. It was a follow-up on the work of Dekker et al.[7] which examined various stacking strategies for an automated container terminal and Borgman et al.[8] where a study of the knowledge of departure and stacking further away or close to exits points were considered. It was discovered that the trade-off between where to stack and accepting more reshuffles leads to improvements over the benchmark. Simulation technique was deployed in Zhao and Goodchild[9] to investigate the truck arrival information and container rehandles in the import container retrieval process. They found that a complete arrival order is not required to significantly reduce rehandles. However, benefit can be obtained from information about truck arrival.

There are benefits in stacking higher and using a larger number of rows. A storage system was developed in Casey and Kozan[10] that simulates and optimises the movement and storage of containers within the terminal. A mathematical model was presented that minimises rehandling moves, while having total job times as the objective function. Using four heuristics and three meta-heuristics, they concluded that the model and optimisation should be used within a larger model. The focus was on optimising the movements of machines and ensures they arrive at their final destination on time.

The problem of assigning containers to storage spaces that minimizes the total expected number of relocations was investigated in Yang and Kim.[11] The paper addressed both dynamic and static location problems. The static model was solved using the Genetic Algorithm (GA) and the dynamic one by the minimum space waste rule, which was found to outperform GA. The problem of assigning locations to incoming containers and the need for reshuffle in a container terminal was examined in Wan et al.[12] Yard operations in some container terminals was investigated in Chen,[13] it was concluded that higher container stacking have a direct impact on the number of unproductive moves. They claimed that the major impact was on the delivery operation. The use of heuristics that use ϵ- optimal policies was proposed in Li et al.[14] to compute a specific allocation of empty containers between different ports. The problem was formulated as a multi-port containerisation model.

Decision trees was deployed in Kim et al.[15] from a set of optimal solutions to solve a dynamic programming model formulated to locate export containers considering weight. Simulated Annealing (SA) was used in Kang et al.[16] to derive the best stacking strategy for containers in a yard with uncertain weight information. Experiments showed that the strategy was able to reduce the number of re-handles compare to traditional same-weight group stacking strategy. They advised that more improvement can be obtained where the accuracy of the weight classification is done by machine learning. The integer programming model, a neighborhood search process and three functions were deployed in Lee and Chao[5] to minimise the number of container pre-marshalling and reduce re-handles. The pre-marshalling problem was modelled as an integer programming problem based on multi-commodity network flow in Lee and Hsu.[17] The optimization objective was to minimize the number of container movements during pre-marshalling. A heuristic called the tree search procedure was developed in Bortfeldt and Forster[18] for solving the container pre-marshalling problem. It is based on a natural classification of possible moves, making use of a lower bound and applying a branching scheme for move sequences rather than single moves. It proved effective on large real-world instances, according to the authors.

The problem of rearranging containers before they are shipped was examined in Kim and Bae[19] using dynamic programming. The move planning was solved by using a transportation model. However, it was computationally demanding. Hence, heuristics were advised. A two-step SA was proposed in Choe et al.[20] to investigate intra-block remarshalling plan that is free from rehandles during both the loading and remarshalling, considering twin Automatic Straddle Carriers (ASCs). The first step identifies the slot that minimises reshuffles and the second step schedules ASC that minimises the interference between the ASCs. Integer programming was used in Lee et al.[21] to solve the terminal allocation problem for vessels and yard allocation problem for transhipment container movements for a port that has multiple terminals. A two level heuristic approach was adopted to solve the integrated problem. Work on the relocation problem includes Jin et al.[2] which proposed an improved greedy look ahead heuristic for the Container Relocation Problem (CRP). The objective of the CRP is to find an optimal operation plan for the crane with the fewest number of container relocations. The method was found to be efficient especially for large scale problems.

Three heuristic methods; index based, binary IP and beam search were developed in Hakan Akyüz and Lee[1] to solve a binary integer programming model of the CRP. They concluded that the beam search heuristic outperformed the others. Emptying a stack without new arrivals was modelled as integer programming to derive an optimum reshuffle sequence. The problem was broken into parts and solved by four heuristics, IP-based, Lowest Slot (LS), Reshuffle Index (RI) and the Expected Number of Additional Relocations (ENAR). They claimed that their heuristic MRIP-Dk (MRIP that minimises the number of reshuffles in retrieving *k* containers plus estimated future reshuffles in stack after *k* containers are retrieved) outperformed other heuristics found in the literature. A tree search procedure was deployed in Forster and Bortfeldt[22] to solve CRP. The heuristic is based on natural classification of possible moves and used a branching scheme that moves sequences of promising single moves.

A meta-heuristic called the Corridor method was developed in Caserta et al.[23] to solve the CRP in stacking containers in a container terminal yard, pallets and boxes in a warehouse. The objective is to find the block location that minimises the number of movements that is required in the desired retrieval sequence. The imposition of exogenous constraints reduced the size of the problem and made use of constrained dynamic programming, a practical approach even for large instances. A three-phase heuristic was proposed in Lee and Lee[3] to optimise the work plan for a crane to retrieve containers from a yard according to a given order. They aimed at minimising the weighted sum of the number of container movements and the total cranes working time. The first phase generates an initial feasible movement sequence. The other two phases are iterative and terminate when a number of consecutive iterations cannot improve the current solution. This study is similar to that in Bortfeldt and Forster[18] on a tree search procedure for the pre-marshalling of containers.

The schedule of container movement by using an autonomous learning method was addressed in Hirashima.[24] This is based on a new learning model considering container groups and the Q-learning algorithm. The desired position of containers in a group is provided by an algorithm based on the Markov Decision Process (MDP). Using simulations, the proposed method was able to find solutions that had a smaller number of rehandles compared to conventional methods. This was a follow-up to previous works by Hirashima[25] where they discovered that the number of container arrangements increases exponentially with increase in the total count of containers. The desirable movements of containers was determined in Hirashima et al.[26] to reduce the total turnaround time of ships, using the Q-learning algorithm. Heuristics were deployed Expósito-Izquierdo et al.[27] to minimise the number of movements required to locate all containers. Features that consider the occupancy rate of the bays and the percentage with high priority are considered. An instance generator was also suggested for instances with varying degrees of difficulty.

After this exhaustive review, we believe that the heuristics we introduce here has the following advantages in particular over those described in Tang et al.[28]

1. Four new simple but still effective heuristics are introduced.
2. Compatibilty test between the different heuristics was undertaken.

Problem definition and modelling

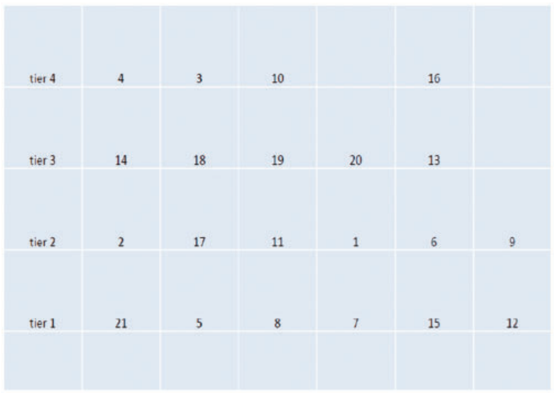
Due to the limited space in a container yard, containers are stacked on top of each other. However, the higher containers are stacked on top of each the higher the likelihood that there will be need for reshuffle. On the other hand, lower stacking is likely to require less reshuffle but takes more space in the yard. Therefore, there is trade off between the need to manage space and minimise reshuffle. The time lag between when these containers are stored and when they are retrieved results in improper location. The result is that containers that are needed earlier are underneath containers that are needed later. In order to have access to the desired container, we need to reshuffle the container (s) on top. Reshuffles are only done within bays for safety and operational reasons. When a crane picks a container with its hoist either for retrieval, storing or reshuffle, it only moves the containers vertically or horizontally while the frame of the crane is kept still Wan et al.[12]

The container reshuffle problem is made up of three basic decisions; Which container to move, when to move it and where to move it to. The configuration of a bay can be defined by the number of columns, the number of tiers, the number of containers and their position in the bay. Let S be the number of container in a bay with C columns and T tiers. The number of container S that should be in the bay for reshuffle to be feasible can be defined as follows;

$$S \leq CT - (T-1) \tag{1}$$

where CT is the number of containers that can be in the bay when it is full and $(T-1)$ is the number of empty slots to allow for reshuffle. Figure 1, shows a bay with six columns, four tiers and twenty-one containers with their positions.

Containers are ranked from 1 to S, indicating the priority order for retrieval. Where containers are not stored while retrieval is going on, we have a static state i.e. the number of containers only decreases. We have a dynamic state when containers are stored while retrieval is in process. Each time a container is retrieved, the configuration of the bay changes due to reshuffles. This dynamic nature of the problem even for a small instance, the number of states grows exponentially with the increase in the number of containers.



tier 4	4	3	10		16	
tier 3	14	18	19	20	13	
tier 2	2	17	11	1	6	9
tier 1	21	5	8	7	15	12

Figure 1. A typical bay configuration.

According to Carraro and de Castro,[29] let $\{x_1, x_2, x_3 \ldots, x_{s-1}\}$ be the stages (updated state of the bay) for retrieval of all containers. At each stage, the incremental reshuffles due to retrieval of container s is y_{si}.

$$y_{si} = \begin{cases} 1, & \left\{ \begin{array}{c} \text{if container } i \text{ is reshuffled} \\ \text{in retrieving container } s \end{array} \right\} \\ 0, & \text{otherwise} \end{cases}$$

It is important to note that s is the container to be retrieved and also the stage of the retrieval. The number of reshuffles necessary to retrieve container s in stage x_s can be defined as follows;

$$x_s = \sum_{i}^{S} y_{si} \tag{2}$$

The total number of reshuffles in retrieving all the containers is thus;

$$\sum_{s=1}^{S-1} \sum_{i=s+1}^{S} y_{si} \tag{3}$$

We are interested in determining the minimum reshuffle possible in retrieving containers from a given bay.

The model formulation

The first binary integer programming (BIP) model for a container reshuffle problem was presented in Wan et al.[12] The model introduced column-relationship variables that check whether a container to be retrieved is in the same column as the container reshuffled. Due to the long computational time needed to solve problems with large number of containers, the model was improved in Tang et al.[28] The improved model removed all the column-relationship variables and added reshuffle related constraints. This reduced the number of decision variables by $3S^2$.

The model we suggest here is a further improvement on the two previous ones found in Tang et al.[28] and Wan et al.[12] We have made the following modification in view of the limitations of these models. We observed that the last two constraints in the the two models force containers to keep the same location they had at the first stage of retrieval. We have removed them and introduced a bay configuration constraint to reflect the change arising from each container reshuffled. The two constraints are now replaced by a new one as shown in constraint (18). This constraint assigns binary values to x_{1ict} indicating the locations of the containers in the bay at the start of the first stage of retrieval. The modified model reduces the number of constraints by $(S-1)SCT$ thereby reducing the solution time. For the sake of continuity we use the same notations as in the two previous models.

Parameters.

Decision variables.

$$x_{sict} = \begin{cases} 1, & \text{if container } i \text{ is at tier } t \text{ of column } c \text{ at the beginning of stage } s. \\ 0, & \text{otherwise} \end{cases}$$

$$y_{si} = \begin{cases} 1, & \text{if container } i \text{ is reshuffled for retrieving container } s. \\ 0, & \text{otherwise} \end{cases}$$

$$w_{sij} = \begin{cases} 1, & \text{if container } i \text{ and } j \text{ are reshuffled during stage } s \text{ and } j \text{ container is at a higher tier than container } i \text{ before reshuffling container } i \text{ before reshuffling.} \\ 0, & \text{otherwise} \end{cases}$$

The model.

$$\min \sum_{s=1}^{S-1}\sum_{i=s+1}^{S} y_{si} \tag{4}$$

subject to:

$$\left(1-\sum_{t=1}^{T} x_{ssct}\right)T + y_{si} \geq \left(\sum_{t=1}^{T} t x_{sict} - \sum_{t=1}^{T} t x_{ssct}\right)/T \quad 1 \leq s < i \leq S, 1 \leq c \leq C \tag{5}$$

$$\left(\sum_{t=1}^{T} t x_{ssct} - \sum_{t=1}^{T} t x_{sict}\right)/T \leq 1 - y_{si} \quad 1 \leq s < i \leq S, 1 \leq c \leq C \tag{6}$$

$$\sum_{c=1}^{C}\sum_{t=1}^{T} x_{sict} = 1, 1 \leq s \leq i \leq S \tag{7}$$

$$\sum_{i=s}^{S} x_{sict} \leq 1, 1 \leq s \leq S, 1 \leq c \leq C, 1 \leq t \leq T \tag{8}$$

$$\sum_{i=s}^{S} x_{sict} \leq \sum_{i=s}^{S} x_{sic,t-1}, \quad 1 \leq s \leq S, 1 \leq c \leq C, 2 \leq t \leq T \tag{9}$$

$$\sum_{t=1}^{T} x_{s+1,ict} + \sum_{t=1}^{T} x_{ssct} \leq 2 - y_{si}, 1 \leq s < i \leq S, 1 \leq c \leq C \tag{10}$$

$$2 - y_{si} - y_{sj} + w_{sij} \geq \left(\sum_{c=1}^{C}\sum_{t=1}^{T} t x_{sjct} - \sum_{c=1}^{C}\sum_{t=1}^{T} t x_{sict}\right)/T \quad 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j \tag{11}$$

$$y_{si} + y_{sj} + w_{sij} \leq 3 + \left(\sum_{c=1}^{C}\sum_{t=1}^{T} t x_{sjct} - \sum_{c=1}^{C}\sum_{t=1}^{T} t x_{sict}\right)/T \quad 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j \tag{12}$$

$$w_{sij} \leq y_{si}, 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j \tag{13}$$

$$w_{sij} \leq y_{sj}, 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j \tag{14}$$

$$\sum_{t=1}^{T} t x_{s+1,ict} - \sum_{t=1}^{T} t x_{s+1,jct} \geq -T(1-w_{sij}) - T(1-y_{si}) - T(1-y_{sj}) - T\left(1 - \sum_{t=1}^{T} x_{s+1,ict}\right) \quad 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j, 1 \leq c \leq C \tag{15}$$

$$x_{s+1,ict} - x_{sict} \geq -y_{si}, 1 \leq s < i \leq S \quad 1 \leq c \leq C, 1 \leq t \leq T \tag{16}$$

$$x_{sict} - x_{s+1,ict} \geq -y_{si}, 1 \leq s < i \leq S \tag{17}$$

$$x_{1ict} = X_{ict}, 1 < i \leq S, 1 \leq c \leq C, 1 \leq t \leq T \tag{18}$$

$$y_{si} \in \{0,1\}, 1 \leq s < i \leq S \tag{19}$$

$$w_{sij} \in \{0,1\}, 1 \leq s < i \leq S, \quad 1 \leq s < j \leq S, i \neq j; \tag{20}$$

$$x_{sict} \in \{0,1\}, 1 \leq s \leq i \leq S, \quad 1 \leq c \leq C, 1 \leq t \leq T; \tag{21}$$

Constraints (5) and (6) determine the reshuffle variable y_{si}. $\sum_{t=1}^{T} t x_{sict}$ and $\sum_{t=1}^{T} t x_{ssct}$ represent the position of containers i and j in column c tier t, while $\sum_{t=1}^{T} x_{ssct}$ is one if container s is in column c tier t at stage s. If container i is above container s, the RHS is a value less than one and the expression in bracket in LHS is zero. This forces y_{si} to be one i.e., container i is reshuffled in retrieving container s. On the other hand, if container s is above container i in constraint

(6), there will be no need for reshuffle hence y_{si}=0 and the LHS of the equation is less than one. Constraints (7) ensure that each container occupies only one spot. At each stage of retrieval, each container i, $i \geq s$ can be traced to one slot. Constraints (8) implies not more than one container can be at a slot i.e., a slot is either empty or has a container. The fact that containers cannot float is defined in Constraints (9). This means, if in column c, there is a container in tier t>1 then, there must be a container below it.

Constraints (10) imply a reshuffled container cannot be in the same column after reshuffle. If container i is above container s and both are in the same column then, container i must be reshuffled i.e., y_{si} and $\sum_{t=1}^{T} x_{ssct}$ are both one. This forces $\sum_{t=1}^{T} x_{s+1,ict}$ to be zero i.e., container i cannot be in column c at the next stage $s+1$. Suppose container i, j and s are in the same column and container i and j are above container s. If container j is above container i before reshuffle therefore, the expression in bracket on RHS of constraints (11) and (12) will be a value less than one. Since y_{si} and y_{sj} are one, this forces w_{sij} to be one. In constraints (13) and (14), if either container i or j is not reshuffled or container j is not above container i then $w_{sij} = 0$.

Constraints (15) address the relative positions of two containers before and after both are reshuffled. Suppose container i, j and s are in the same column and container j is above container i before reshuffle. If containers i and j are reshuffled in retrieving container s then, y_{si}, y_{sj} and w_{sij} are all one. If both containers i and j are reshuffled to same new column therefore, container i will be above container j in the new column. Constraints (16) and (17) ensure that a container not reshuffled keep its position. If container i is not reshuffled at stage s therefore $y_{si} = 0$. This means $x_{s+1,ct} = x_{sict}$. If the container is reshuffled, it is stored in a position higher than those already in the column. Constraints (18) assign known values of x_{1ict} to slots in the initial configuration and this is updated at each stage of retrieval i.e., the lowest ranked container is retrieved at each stage. Constraints (19) to (21) are self explanatory.

Illustration

Consider a bay with seven containers, three columns and three tiers. The initial position is as shown in stage (a) of Figure 2 adapted from Carraro and de Castro.[29] The first reshuffle is done in stage (b) where container 7 moves from top of container 1 to top of container 2. At stage (c), container 1 is retrieved. Container 7 moves again from top of container 2 to the slot vacated by container 1; this is the second reshuffle at stage (d). At stage (e), container 2 is then retrieved. The third reshuffle is done in stage (f) where container 6 moves to top of container 7 and that is the last reshuffle. If container 6 had been moved to the top of container 4 that would have necessitated another reshuffle. This would have led to four reshuffles instead of three. Subsequently all the remaining containers can be retrieved without need for reshuffle.

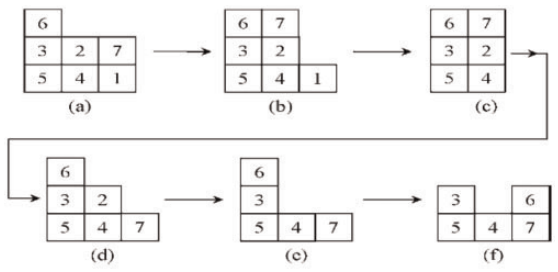


Figure 2. Reshuffle process.

A heuristic approach

Due to the computational complexity of the model, the exact approach is only able to solve small scale problems. This is at variance with real world situations where medium to large scale instances are encountered and fast results are required. There is thus need to seek an approximate approach solution. In this section, we review some heuristics found in literature and propose four new ones based on the least priority rule. They are referred to as LPH1, LPH2, LPH3 and LPH4.

Reshuffle index (RI) heuristic

This is based on the rule that a blocking container should be moved to a column having the least number of containers that have to be retrieved earlier than the blocking container as described in Murty.[30] Identify the blocking container and compare it to the minimum numbered container in each column that has empty space. Put the blocking container in a column that has the least number of container to be retrieved earlier than the blocking container. Where there is a tie, the container is put in the column with higher tier and by arbitrary choice for a further tie.

H1 and H2 heuristic

H1 and H2 are each based on two conditions as described in Tang et al.[28] The difference between the two heuristics is the decision rule for moving blocking container. The first condition which is common to both is to identify the blocking container referred to as k and compare it to the minimum numbered container referred to as n_c in each column c that has empty space. Put k in column c that satisfies $k < n_c$ and arbitrarily decide on which is the closest column where there is a tie. The total number of containers in the bay is given the value S. A column with all empty space n_c is given value $S+1$ thereby having the highest possible value in the bay and the blocking container will always move there, if it exists. Where there is no column that satisfies the above condition which means there is no column c where $k < n_c$, the second condition must be applied depending on the heuristic. H1 will put k in the column with the minimum RI. This is the total number of container that has a lower number than k in each column that has empty space. Where there is a tie, put k in the closest column. H2 on the other hand puts k in the column with the minimum BI. This is the column that has lower number of container that will block the minimum number container in a column if k is moved to that column. Where there is a tie, the decision on where to put the container is done arbitrarily.

Framework for RI, H1 and H2

The number of containers in the bay is initially numbered from 1 to S. The first container to be picked is container 1, once that is done, the remaining container is renumbered from 1 to S-1. At each stage, the container to pick is always container 1. M denotes the cumulative reshuffle at a stage and $M1$ is the number of reshuffles in retrieving a container. The reshuffle process for the three heuristics above as described in Tang et al.[28] can be summarised as follows;

The least priority heuristic (LPH1)

It is based on the fact that the highest numbered container should be retrieved last hence, it is given the least priority. The algorithm starts by inputting a matrix of dimension t x c representing the initial state of the bay with t being the number of tiers and c the number of columns. Empty slot is represented by zero which can be randomly generated or input manually. The minimum numbered container is always the one to be retrieved at each stage hence, this indicates the priority order i.e retrieval is done in ascending order. At each stage, the minimum numbered container refer to as the desired container s is identified and retrieved if there is no container blocking it. If there is a blocking container refer to as k, the container s has to be moved to another column.

The decision on where to put k is determined by calculating the sum of the reciprocal for containers in those columns where there is empty spaces. These sums will be compared, and the column that returns the lowest sum of reciprocal is chosen as the column where k should be moved to. Where a column has all slots empty, k is placed in this column. This is done for all k blocking the s until there is no container blocking it at which stage such container can now be retrieved. Since at each stage, the minimum numbered container is always the one to be retrieved the algorithm updates the configuration and the process is repeated until all containers are retrieved.

The least priority heuristic (LPH2)

The blocking container k is compared to the minimum numbered container n_c in each column having empty spaces. Put k in a column that satisfies the condition $k < n_c$ and put k in the closest column where more than one column satisfies the condition. Where no column satisfies the condition ie $k > n_c$ for those columns, apply the least priority principle.

The least priority heuristic (LPH3)

The blocking container k is compared to the minimum numbered container n_c in each column having empty spaces. Put k in a column that satisfies the condition $k < n_c$ by applying the least priority principle. Where no column satisfies the condition ie $k > n_c$ for those columns, apply the reshuffle index principle as in condition 2 of H1 heuristics.

The least priority heuristic (LPH4)

Apply the first condition in LPH3 where $k < n_c$ and the blocking index principle where $k > n_c$.

Framework for LPH

Computational experiments

Experiments were conducted to investigate the performance of the model on different problem instances randomly generated. The model is coded in GMPL and executed on an Intel Core i7-4790 3.60 GHz CPU with 16 GB RAM. For an exact solution, each of the problems has been solved using GLPSOL while the heuristic is coded in MATLAB R2018b. The results given in Tables 1 to 6 are for the static reshuffle scenario, i.e., emptying all container in the bay. 600 instances were randomly generated for 12 problem classes each having 50 cases. 1-hour was set as the benchmark for a solution to be generated for each instance of the problems.

Static case

Here we compare the results from the model solved with B&C and the heuristics for a small size problem defined as a bay with 50% utilisation capacity. We observe that they all return the same results for problem classes $(6-2-6)$ and $(6-3-8)$. However, as the problem size increases for $(6-4-11)$ and $(6-5-13)$ we begin to observe differences in results between the methods as the tier and number of container increase. The utilisation capacity is calculated as:

$$\frac{\text{Number of containers in the bay}}{\text{Total number of possible container reshuffles}} \tag{22}$$

In Table 1, we compare the model with original version of the heuristics for small size problems. While Table 2 shows comparison with the extended versions. The extended version of each heuristic as defined in Wan et al.[12] is obtained by considering each possible slot for a reshuffled container based on the original rule of the heuristic and the one that gives the minimum possible reshuffle to empty the bay is chosen. They are denoted by E for each of the heuristic. For problem class $(6-2-6)$ and $(6-3-8)$, all the methods return the same average reshuffle. This is due to the fact that the number of tiers is still reasonably small; hence, reshuffle can still be achieved with relative ease. When the size increases to $(6-4-11)$ and $(6-5-13)$ we observe a difference in the average reshuffle between the methods because of the increase in tiers and number of containers. Note that all extended heuristics performed better than the original heuristic; LPH4_E has the least performance. H1_E and LPH2_E have the best performance among the heuristics for problem class $(6-4-11)$. LPH2_E has the best performance for problem class $(6-5-13)$.

'- ': Out of time.

Medium size problems are considered in Tables 3 and 4. We compared the model with the heuristics for 80% utilisation capacity. H1_E, H2_E and LPH2_E performed better than other heuristics for problem classes $(6-3-13)$. While H1_E and LPH_E have the best performance for problem classes $(6-4-17)$ and $(6-5-21)$ respectively. The model was not able to return results for all the 50 instances for problem classes $(6-4-17)$ and $(6-5-21)$ within 1-hour.

In Tables 5 and 6, we compare all heuristics for large size problems defined as bay with 100% capacity. As expected, all the heuristics return the same average reshuffle for problem class $(6-2-11)$ due to the fact that all the 50 instances in the class have the same result. As we increase the problem class by increasing both the number of containers and the number of tiers, we begin to notice a difference in results. All the heuristics return the same results for some of the instances but there are instances where we observe differences depending on the configuration of the bay. HI_E has the best performance for problem class $(6-3-16)$ while LPH2_E has the best performance for problem classes $(6-4-21)$ and $(6-5-26)$.

Dynamic case

The true test of a reshuffle process is the application to a dynamic situation. In container terminal, reshuffle is done while both retrieval and storage is taking place. 50 instances were generated for four problem classes with 80% utilisation capacity. 1000 containers were retrieved and 650 containers stored for each problem class. The retrieval of containers is the same as static case except that an incoming container will affect the priority order based on the expected departure time of the new container. In view of this, every container in the bay will be re-assigned a new index each time there is an incoming container. The heuristic rule determines where the container will be stored similar to how reshuffles are treated. The RTG is assumed to be positioned close to the sixth column hence, the incoming container is stored from right to left of the bay. Note,

Table 1. Comparison between model and heuristics for small problems.

Classes	Model	H1	H2	RI	LPH1	LPH2	LPH3	LPH4
6 – 2 – 6	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
6 – 3 – 8	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
6 – 4 – 11	3.32	3.46	3.46	3.46	3.48	3.42	3.56	3.58
6 – 5 – 13	4.32	4.64	4.68	4.68	4.64	4.52	4.82	4.82

Table 2. Comparison between model and heuristics for small problems.

Classes	Model	H1_E	H2_E	RI_E	LPH1_E	LPH2_E	LPH3_E	LPH4_E
6 – 2 – 6	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
6 – 3 – 8	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
6 – 4 – 11	3.32	3.38	3.42	3.40	3.48	3.38	3.42	3.48
6 – 5 – 13	4.32	4.52	4.54	4.62	4.64	4.52	4.64	4.70

Table 3. Comparison between model and heuristics for medium problems.

Classes	Model	H1	H2	R1	LPH1	LPH2	LPH3	LPH4
6 – 2 – 9	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
6 – 3 – 13	3.98	4.16	4.14	4.18	4.12	4.10	4.16	4.16
6 – 4 – 17	–	6.94	7.02	6.98	7.0	6.80	7.20	7.30
6 – 5 – 21	–	13.4	13.28	13.34	13.30	12.92	14.02	13.94

Table 4. Comparison between model and heuristics for medium problems.

Classes	Model	H1_E	H2_E	R1_E	LPH1_E	LPH2_E	LPH3_E	LPH4_E
6 – 2 – 9	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
6 – 3 – 13	3.98	4.06	4.06	4.08	4.12	4.06	4.12	4.12
6 – 4 – 17	–	6.74	6.76	6.82	6.78	6.78	7.0	6.98
6 – 5 – 21	–	12.52	12.52	12.64	13.26	12.32	13.34	13.18

Table 5. Comparison between the heuristics for large problems.

Classes	H1	H2	R1	LPH1	LPH2	LPH3	LPH4
6 – 2 – 11	2.72	2.72	2.72	2.72	2.72	2.72	2.72
6 – 3 – 16	7.16	7.04	7.14	7.10	7.02	7.26	7.10
6 – 4 – 21	11.88	11.70	11.94	11.78	11.54	12.26	11.96
6 – 5 – 26	22.80	22.36	23.08	22.60	21.74	23.88	23.56

Table 6. Comparison between the heuristics for large problems.

Classes	H1_E	H2_E	R1_E	LPH1_E	LPH2_E	LPH3_E	LPH4_E
6 – 2 – 11	2.72	2.72	2.72	2.72	2.72	2.72	2.72
6 – 3 – 16	6.92	6.96	6.96	7.10	6.94	7.08	7.08
6 – 4 – 21	11.44	11.28	11.46	11.72	11.20	11.72	11.72
6 – 5 – 26	22.18	21.72	22.42	22.60	21.06	23.12	22.90

since the extended heuristics has been shown to be better than the original version, only the extended are treated here.

The average number of reshuffle for the 50 instances and the CPU time in seconds is shown in Table 7. It can be observed that LPH1 has the best performance for problem class (6 – 2 – 9) while H1 and LPH3 have the same result. LPH2 is the best performing heuristics for other problem classes.

Compatibility

Here we check how well the different heuristics work together when they are combined to solve a given problem. Since heuristics are known be instance dependent, it is not surprising to observe some heuristics performing well on some instance and very poorly on some others. In view of this, it is worthwhile investigating how well the heuristics perform individually compare to when they are combined to solve an instance of a problem. We have chosen the problem class (6 – 5 – 26) since it is the most difficult problem as seen in previous sections.

Table 8 show the average reshuffle when the heuristics are used individually as well as when they are combined to solve the problems. Entries along the main diagonal are results when each of the heuristics are used alone and other entries indicate pairing of the heuristics. Once again we observe LPH2 returning the best result either as a stand alone heuristics or when combined with other heuristics.

Conclusion

Container reshuffle is a potent way to measure the efficiency of operations in container ports. Basically, the fewer the reshuffles made to retrieve needed containers, the better since a reshuffle, when necessary, is a waste

Table 7. Dynamic.

	Average number of reshuffles Average CPU time(s)							
Heuristics	(6 – 2 – 9)		(6 – 3 – 13)		(6 – 4 – 17)		(6 – 5 – 21)	
H1	1.68	0.0272	8.02	0.0346	13.28	0.0478	19.26	0.0446
H2	1.70	0.0289	8.40	0.0346	13.84	0.0401	20.86	0.0472
RI	1.70	0.0280	8.22	0.0369	13.28	0.0440	18.74	0.0490
LPH1	1.66	0.0297	7.16	0.0345	13.10	0.0415	20.10	0.0461
LPH2	1.70	0.0304	6.88	0.0342	11.76	0.0386	17.18	0.0443
LPH3	1.68	0.0287	8.32	0.0338	14.14	0.0414	21.94	0.0477
LPH4	1.70	0.0259	8.46	0.0337	15.38	0.0436	22.94	0.0499

Table 8. Compatibility.

Heuristics	H1	H2	RI	LPH1	LPH2	LPH3	LPH4
H1	21.98	21.66	21.73	22.25	21.19	22.42	22.34
H2	21.66	21.42	21.48	21.99	20.94	22.18	22.13
RI	21.73	21.48	21.58	22.01	21.00	22.28	22.22
LPH1	22.25	21.99	22.01	22.38	21.45	22.70	22.57
LPH2	21.19	20.94	21.00	21.45	20.44	21.77	21.72
LPH3	22.42	22.18	22.28	22.70	21.77	22.96	22.80
LPH4	22.34	22.13	22.22	22.57	21.72	22.80	22.74

of effort. The problem of minimising the number of reshuffles has therefore, been recognised for some time now and has been approached by many. Integer programming models of the binary type have been constructed for it and solved both exactly and approximately using heuristics and meta-heuristics. The issue is that these models are less than satisfactory for the following reasons.

1. The models are too big; they have too many unnecessary variables and/or constraints.
2. The heuristics use arbitrary rules to resolve issues of ties to determine storage position of reshuffled containers.

We have, thus addressed these issues by considering an alternative model to fit our requirements. This alternative model has been given and described extensively. Given the computational complexity of the problem (NP-hard, since it is represented as an ILP), papers relying on exact methods are not realistic. Those using heuristics are more realistic in that sense although some of the heuristics are crude. We have, therefore carried out experiments on realistic instances using the model as well as some of the prominent heuristics found in the literature and used on the reshuffle problem namely H1, H2 and RI. Moreover, we have designed four novel heuristics LPH1, LPH2, LPH3 and LPH4 which has been tested and compared with others on static and dynamic reshuffle problems. We also presented a compatibility test to check how well the heuristics work when combined to solve a problem. The results show that LPH2 is superior to other heuristics.

Acknowledgements

The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: We are grateful to ESRC for funding this work within the "Smart Data Analytics for Business and Local Government" project, Grant ES/L011859/1.

ORCID iD

Hammed Bisira https://orcid.org/0000-0001-6225-4565

References

1. Hakan Akyüz M and Lee CY. A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Res Logist* 2014; 61: 101–118.

2. Jin B, Zhu W and Lim A. Solving the container relocation problem by an improved greedy look-ahead heuristic. *Eur J Oper Res* 2015; 240: 837–847.
3. Lee Y and Lee YJ. A heuristic for retrieving containers from a yard. *Comput Oper Res* 2010; 37: 1139–1147.
4. Kim KH and Hong GP. A heuristic rule for relocating blocks. *Comput Oper Res* 2006; 33: 940–954.
5. Lee Y and Chao SL. A neighborhood search heuristic for pre-marshalling export containers. *Eur J Oper Res* 2009; 196: 468–475.
6. Van Asperen E, Borgman B and Dekker R. Evaluating impact of truck announcements on container stacking efficiency. *Flex Serv Manuf J* 2013; 25: 543–556.
7. Dekker R, Voogd P and van Asperen E. Advanced methods for container stacking. In: *Container terminals and cargo systems*. Berlin: Springer, 2007, pp.131–154.
8. Borgman B, van Asperen E and Dekker R. Online rules for container stacking. *Or Spectrum* 2010; 32: 687–716.
9. Zhao W and Goodchild AV. The impact of truck arrival information on container terminal rehandling. *Transport Res Part E: Logist Transport Rev* 2010; 46: 327–343.
10. Casey B and Kozan E. Optimising container storage processes at multimodal terminals. *J Oper Res Soc* 2012; 63: 1126–1142.
11. Yang JH and Kim KH. A grouped storage method for minimizing relocations in block stacking systems. *J Intell Manuf* 2006; 17: 453–463.
12. Wan Y, Liu J and Tsai PC. The assignment of storage locations to containers for a container stack. *Naval Research Logistics* 2009; 56: 699–713.
13. Chen T. Yard operations in the container terminal – a study in the unproductive moves. *Maritime Policy & Management* 1999; 26: 27–38.
14. Li JA, Leung SC, Wu Y, et al. Allocation of empty containers between multi-ports. *Eur J Oper Res* 2007; 182: 400–412.
15. Kim KH, Park YM and Ryu KR. Deriving decision rules to locate export containers in container yards. *Eur J Oper Res* 2000; 124: 89–101.
16. Kang J, Ryu KR and Kim KH. Deriving stacking strategies for export containers with uncertain weight information. *J Intell Manuf* 2006; 17: 399–410.
17. Lee Y and Hsu NY. An optimization model for the container pre-marshalling problem. *Comput Oper Res* 2007; 34: 3295–3313.
18. Bortfeldt A and Forster F. A tree search procedure for the container pre-marshalling problem. *Eur J Oper Res* 2012; 217: 531–540.
19. Kim KH and Bae JW. Re-marshaling export containers in port container terminals. *Comput Ind Eng* 1998; 35: 655–658.
20. Choe R, Park T, Oh MS, et al. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. *J Intell Manuf* 2011; 22: 201–217.
21. Lee DH, Jin JG and Chen JH. Terminal and yard allocation problem for a container transshipment hub with multiple terminals. *Transport Res Part E: Logist Transport Rev* 2012; 48: 516–528.
22. Forster F and Bortfeldt A. A tree search procedure for the container relocation problem. *Comput Oper Res* 2012; 39: 299–309.
23. Caserta M, Voß S and Sniedovich M. Applying the corridor method to a blocks relocation problem. *Or Spectrum* 2011; 33: 915–929.
24. Hirashima Y. A Q-learning system for container marshalling with group-based learning model at container yard terminals. In: *Proceedings of the international multiconference of engineers and computer scientists 2009 (IMECS 2009)*, volume 1. Princeton: Citeseer, 2009.
25. Hirashima Y (2008) A Q-learning system for container transfer scheduling based on shipping order at container terminals. *Int J Innovat Comput Inform Control* 4(3): 547–558.
26. Hirashima Y, Takeda K, Harada S, et al. A Q-learning for group-based plan of container transfer scheduling. *JSME Int J Ser C* 2006; 49: 473–479.
27. Expósito-Izquierdo C, Melián-Batista B and Moreno-Vega M. Pre-marshalling problem: heuristic solution method and instances generator. *Expert Syst Appl* 2012; 39: 8337–8349.
28. Tang L, Jiang W, Liu J, et al. Research into container reshuffling and stacking problems in container terminal yards. *IIE Trans* 2015; 47: 751–766.
29. Carraro LA and de Castro LN. A clonal selection algorithm to minimize reshuffling in container stacking operations. In: *2012 IEEE congress on evolutionary computation*. Piscataway, NJ: IEEE, 2012, pp.1–8.
30. Murty KG. Yard crane pools and optimum layouts for storage yards of container terminals. *J Ind Syst Eng* 2007; 1: 190–199.