European Journal of Operational Research 252 (2016) 1031–1039

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Innovative applications of O.R.

Container relocation problem with time windows for container departure

Dusan Ku*, Tiru S. Arthanari

Department of Information Systems and Operations Management, University of Auckland Business School, Private Bag 92019, Auckland 1421, New Zealand

ARTICLE INFO

Article history:
Received 11 November 2014
Accepted 25 January 2016
Available online 2 February 2016

Keywords:
Logistics
Container relocation problem
Truck appointment system
Stochastic dynamic programming
Abstraction heuristics

ABSTRACT

The blocks relocation problem is a classic combinatorial optimisation problem that occurs in daily operations for facilities that use block stacking systems. In the block stacking method, blocks can be stored on top of each other in order to utilise the limited surface of a storage area. When there is a predetermined pickup order among the blocks, this stacking method inevitably leads to the reshuffling moves for blocks stored above the target block and the minimisation of such unproductive reshuffling moves is of a primary concern to industry practitioners. A container terminal is a typical place where this problem arises, thus the problem being also referred to as the container relocation problem. In this study, we consider departure time windows for containers, which are usually revealed by the truck appointment system in port container terminals. We propose a stochastic dynamic programming model to calculate the minimum expected number of reshuffles for a stack of containers which all have departure time windows. The model is solved with a search-based algorithm in a tree search space, and an abstraction heuristic is proposed to improve the time performance. To overcome the computational limitation of exact methods, we develop a heuristic called the expected reshuffling index (ERI) and evaluate its performance.

1. Introduction

Ever since the first containers were introduced around 1950–60s, container handling techniques and strategies have always been key factors in measuring the efficiency of terminal operation. Especially, the dramatic growth in the volume of container traffic in more recent years has posed a serious challenge for terminal operators because the available surface for managing such traffic has remained virtually unchanged. Due to this space limitation, most container terminals as opposed to some terminals using wheeled terminal operation adopt block stacking systems, in which containers are stacked on top of each other using handling equipment such as rubber-tired gantry cranes (RTGCs), rail-mounted gantry cranes (RMGCs) or straddle carriers (SCs). An analytical comparison of SCs, RTGCs and RMGCs is presented in Chu and Huang (2005), which show that the handling capacities increase from SC over RTGC to RMGC. Also, optimisation strategies for handling containers have been studied widely to foster the efficiency of container terminal operation. Comprehensive surveys on optimisation strategies for handling containers can be found in Stahlbock and Voß (2008) and Steenken, Voß, and Stahlbock (2004).

In a marine container terminal, import containers are unloaded from ships and stored in a storage yard until their owners pick them up by dispatching trucks. The container dwell time, or the duration during which a container stays in the storage yard, usually takes from hours to days and often depends on free storage days of each terminal. In Ports of Auckland (POAL), for instance, the monthly average dwell time for import containers was about 2 days in 2013, ranging from 1.8 days to 2.6 days (Ports of Auckland eNews (2013)) and the standard deviation of the dwell time during a specific month was around 1.7 days (POAL internal data, June 2013). While the arrival times of import containers are predictable by the ship arrival time and the ship operation schedule, the departure times of them are subject to high variability due to the unpredictability of trucks' arrival times. It is typical that hauliers send some pre-information about trucks' visits to the terminal before their trucks arrive at the terminal. A truck appointment system (TAS) is one such system that includes the information about the time window for the arrival of the trucks.

1.1. The truck appointment system in container terminals

The main purpose of the TAS in container terminals is to alleviate the imbalance of peaks and troughs of truck arrival times and be related to solving terminal gate congestion as well as reducing the environmental impact of long-waiting trucks.

* Corresponding author. Tel: +64 99237154; fax: +64 93737430.
E-mail addresses: d.ku@auckland.ac.nz (D. Ku), t.arthanari@auckland.ac.nz (T.S. Arthanari).

http://dx.doi.org/10.1016/j.ejor.2016.01.055
0377-2217/

By restricting the number of truck visits in each time slot, the TAS can reduce the number of trucks during peak hours, thus improving the overall turnaround time of external trucks. This study suggests that the TAS can also lend itself to improving the reshuffling efficiency for pickup containers. Hongkong International Terminal (HIT) appears to be the first terminal that implemented such an appointment system around 1997. The current appointment system implemented in HIT is a straightforward quota system in which the number of appointments is allocated in every 30-minute time slot (Murty et al. (2005b)). The effective size of each time slot varies by terminals. In New Zealand, for example, POAL has introduced an appointment system called *Vehicle Booking System* around 2007 and has allocated one hour for each time slot.

The most optimistic scenario with the TAS is that all trucks arrive at the terminal during the appointed time slots, but in reality it is hard to live up to this optimism due to several reasons. Each port may have different context for the reasons. In a survey on the hauliers in the California region, which had passed a bill in 2002 to enforce marine port terminals to take measures to reduce truck emissions and in response several ports established truck appointment systems, Giuliano and O'Brien (2007) identified the reasons as follows: freeway congestion, delays on the customer end, delays at marine terminals and others. A study done by Morais and Lord (2006) confirmed that the implementation of an appointment system at west coast terminals in the US could be effective in reducing truck idling/queuing at west coast terminals. However, they pointed out that the impacts were dependent on the factors that were producing congestion. In HIT where most external trucks bring export containers from mainland China, the drivers could not provide reliable arrival times due to the fact that the drivers could not control the border crossing time. Hence, HIT required only trucks coming to the terminal for pickups to make appointments and only for the terminal peak time of 8 AM to 5 PM (Murty et al. (2005b)). In 2013, POAL achieved trucks' on-time ratio for the appointment to be around 55–60 percent, which was considerably higher than the previous years (Ports of Auckland eNews (2013)). This level of improvements could be made after years of implementation experience and continuous improvements such as with incentives, e.g. faster processing, for on-time trucks.

In the existing literature on the TAS, only a little addresses yard handling issues in a container terminal. Huynh and Walton (2008) focused on finding the appropriate level of capping for the allowable number of trucks given the input parameters such as the target average truck turn time and the number of yard equipment available at each yard zone and each time window. An ad hoc heuristic was used to solve their formulation. Jones and Michael Walton (2002) studied how the information about the departure times of import containers could be used to manage the container stack. They presented three methods of stacking import containers – noninformed strategy, informed strategy and random strategy – and developed an event-based simulation model that captures the interactions among various subsystems. They assumed that the port must have some method for gathering information about the departure time, e.g. the expected dwell time. Namboothiri and Erera (2008) addressed the TAS from a drayage company's perspective and studied how the TAS affected the management of a fleet of trucks providing container pickup and delivery services to a port. Zhao and Goodchild (2010) assumed that trucks' arrival times could be obtained after import containers were stored in the yard. They experimented with simulation programs to analyse the rehandling reductions under a variety of information quality regarding the truck arrival sequence and observed that significant reductions in rehandles could be obtained with small improvements in terminal information regarding truck arrivals. van Asperen, Borgman, and Dekker (2013) used a discrete-event simulation model to evaluate the impact of a TAS on the performance of online container stacking rules studied in their prior research (Borgman, van Asperen, & Dekker (2010)). Zehendner and Feillet (2014) proposed a mixed integer programming model to determine the number of appointments to offer with regard to the overall workload and the available handling capacity. Their model aims at minimising overall delays at the terminal, and discrete event simulation validates the results obtained by the optimisation model in a stochastic environment.

1.2. The container relocation problem

Storage yard in a container terminal consists of a number of stacks where containers are stored temporarily before they are claimed by external trucks or loaded onto a ship. The stack in this paper refers to a two-dimensional stock space comprised of multiple horizontal columns (or piles) and vertical tiers. Once containers are stacked in a stack, they can be accessed only from above. If there are containers on top of the target container, the above containers should be relocated to different columns, resulting in unproductive moves. According to Caserta, Schwarze, and Voß (2011), three types of post-stacking problems have been identified, namely, (*i*) the remarshalling problem, (*ii*) the premarshalling problem and (*iii*) the relocation problem. In the literature, the last problem is referred to as the *container relocation (reshuffling, rehandling) problem* (CRP) or the *blocks relocation problem* (BRP), which is the main discussion of this paper. Researchers note that container rehandles are a major source of inefficiency in most terminals when retrieving containers from yards. Most research that addresses the issue of container rehandling has the objective of minimising the number of reshuffles or, alternatively, the reshuffling time.

The CRP can be formally defined as follows: given a pickup order for containers stacked in a given stack layout, the objective is to clear all containers from the stack with the least number of reshuffling moves. A general version of this problem is known as the blocks world planning (BWP), the goal of which is to build any feasible stacking configuration of blocks. The complexity of the BWP has been proven NP-hard by Gupta and Nau (1992) and the complexity of the CRP has recently been proven NP-hard by Caserta, Schwarze, and Voß (2012).

The CRP has first been formulated by Kim and Hong (2006) with a dynamic programming (DP) model. Since this work, several variants to the CRP appear in the literature as noted in Ku and Arthanari (2016). By the assumption of container arrivals during the retrieval process, the *static* and *dynamic* (Borjian, Manshadi, Barnhart, and Jaillet (2013); Akyüz and Lee (2014); Wan, Liu, and Tsai (2009)) versions of the CRP are studied. Also, by the allowance of cleaning (or premarshalling) moves, the *restricted* and *unrestricted* versions of the CRP are studied. One latest trend of approaching this problem has a focus on investigating the informed searches and look-ahead methods in the unrestricted variant (Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2014); Jin, Zhu, and Lim (2015); Petering and Hussein (2013); Zhu, Qin, Lim, and Zhang (2012)). We also note that Ünlüyurt and Aydın (2012) study the same problem as Kim and Hong (2006) but extend it by considering the distance travelled by the crane.

Aside from the formalism proposed by Kim and Hong (2006), the model of the rehandling problem is influenced by the stack type (export or import). For export container stacks, Kim, Park, and Ryu (2000) derived an optimal stacking strategy for export containers with weight group information. Murty, Liu, Wan, and Linn (2005a) developed a system for determining a storage position of an arriving container to minimise the number of reshuffles measured by the reshuffle index (RI). Kang, Ryu, and Kim (2006) studied the rehandling problem for export containers and proposed a simulated annealing method to derive a good stacking strategy

for containers with uncertain weight information. Based on this, they introduced classifiers for weight grouping to evaluate the derived strategies. Yang and Kim (2006) discussed the storage demand unit, which can be stored together and be retrieved in any order, in finding the stacking policy to minimise the total number of reshuffles.

For import container stacks, Watanabe (1991) proposed an accessibility index as an indication of rehandling occurrences and applied it to both the SC system and the transfer crane system to estimate the expected number of reshuffles for a given storage bay. de Castillo and Daganzo (1993) proposed two basic strategies for storing import containers, the first being based on the expected number of moves per container and the second being based on segregating containers according to their arrival times. Kim (1997) proposed analytic evaluations to estimate the expected number of reshuffles to retrieve all containers in a bay. Sauri and Martin (2011) developed a mathematical model based on probabilistic distribution functions for container dwell time to evaluate the stacking strategies by the number of reshuffling moves generated.

The present paper considers only import container stacks and follows the static version for container arrivals. However, it differs from other studies by assuming randomness for container departures among containers booked in the same time window. We term this problem class the *CRP with Time Windows* or *CRPTW*. The arrival times of trucks assigned for pickup containers are usually unpredictable in container terminals that do not use appointment systems, but our model assumes that the time windows for trucks' visits are available by a TAS. Under this setting, there is a clear precedence relationship for pickup among containers in different time windows, but the precedence relationship among containers in the same time window is unclear. The objective of this problem is (*i*) to find the optimal decision for reducing the rehandles of pickup containers given the time windows usually informed by truck appointments and (*ii*) to calculate the minimum expected number of rehandles to clear containers from the stack.

The remainder of this paper is organised as follows: in the next section, the problem is formulated into a stochastic dynamic programming (SDP) model; in Section 3, the abstraction heuristics are proposed to improve the time performance of the tree search; in Section 4, a heuristic rule called the expected reshuffle index (ERI) is described; in Section 5, we report the results of the computational experiments with the methods discussed in this paper; in the last section, we conclude our findings and suggest future research directions.

2. Development of the stochastic model

To focus on the stochastic nature of our model, we confine our model to the static and restricted version, and the following two assumptions are generic to this variant of the CRP: (*i*) No container is arriving in the stack during the retrieval process; (*ii*) Containers are reshuffled if and only if any container below them is to be retrieved.

To include the components of the TAS, we also make the following assumptions in the model: (*iii*) Each container in the stack is booked to a time window, which then is mapped to a pickup sequence in an ascending order by the hour; (*iv*) Trucks associated with pickup containers in the stack arrive at the terminal within their appointed time windows.

When a group of containers are assigned to the same time window, they are all mapped to the same sequence, in which case the precedence relationship among them is unknown. We assume that there is an equal chance for any departure among containers with the same sequence. This stochastic scenario is different from the scenario modelled in Kim and Hong (2006) where a group of containers with the same sequence can be retrieved in any order. Below we introduce the notations for our SDP model.

- N: the total number of containers in the initial stack
- a^k: the action taken for the removal of the kth container
- S^k: the state of the stack after k containers are removed from the stack
- c_k: the container to be removed during action $a^{k+1}(k = 0, \ldots, N-1)$. It is a random variable
- $p_k(c_k)$: the probability of removing c_k given the state S^k
- $\pi(S^k, c_k)$: the state transformation function that observes container c_k to be removed from the state S^k
- $S^{k'}$: the state transformed by $\pi(S^k, c_k)$, i.e. $S^{k'} = \pi(S^k, c_k)$
- $r(a^k|S^{(k-1)'})$: the number of rehandles that occur during action a^k given the state $S^{(k-1)'}$
- $f(S^k)$: the expected minimum total number of rehandles to remove the remaining containers from the state S^k.

The model can be formulated as a recursive function as in Eq. (1). c_k is a random variable because which state the current state will be transformed into is uncertain before c_k is observed by external factors, e.g. truck arrivals. From Eq. (1), a general recursive function can be derived as in Eq. (2).

$$\begin{aligned} f(S^0) &= E[\min_{a^1}[r(a^1) + f(S^1)]|\pi, S^0] \\ &= \sum_{c_0} p_0(c_0) \min_{a^1}[r(a^1|S^{0'}) + f(S^1)] \end{aligned}$$

$$\text{where } S^{0'} = \pi(S^0, c_0) \text{ and } S^0 \xrightarrow{S^{0'}, a^1} S^1 \tag{1}$$

$$\begin{aligned} &f(S^{k-1}) = \sum_{c_{k-1}} p_{k-1}(c_{k-1}) \min_{a^k}[r(a^k|S^{(k-1)'}) + f(S^k)] \\ &f(S^N) = 0 \\ &\text{where } S^{(k-1)'} = \pi(S^{k-1}, c_{k-1}) \text{ for } k = 1, \ldots, N \\ &\text{and } S^{k-1} \xrightarrow{S^{(k-1)'}, a^k} S^k \text{ for } k = 1, \ldots, N \end{aligned} \tag{2}$$

The previous state and the action taken there are independent of future random variables, i.e. S^{k-1} and a^k are independent of $c_k, c_{k+1}, \ldots, c_{N-1}$. Also, the optimal action in the given state is independent of future actions, i.e. S^{k-1} and a^k are independent of $a^{k+1}, a^{k+2}, \ldots, a^N$. Thus, we can transform the recursive function of Eq. (2) into a general function as in Eq. (3). Its derivation may be inferred from the model by Zhang, Chen, Shi, and Zheng (2010), which is a corrected version of the model by Kim et al. (2000).

$$\begin{aligned} f(S^0) = &\sum_{c_0} \min_{a^1} \sum_{c_1} \min_{a^2} \cdots \sum_{c_{N-1}} \min_{a^N} \\ &\times \left[\prod_{k=1}^{N} p_{k-1}(c_{k-1}) \sum_{k=1}^{N} r(a^k|S^{(k-1)'}) \right] \\ f(S^N) = 0 \end{aligned} \tag{3}$$

3. Search in decision trees

3.1. Constructing a decision tree

The solution procedure of a DP model can often be modelled as a tree search in a state space starting from some given initial state with rules describing how to transform one state into another. They will be applied over and over again to eventually satisfy some goal condition. In the context of the CRP, each node in the search tree corresponds to a stack layout (see Fig. 1). The root node represents an initial state of the layout and the leaf nodes the empty layout, i.e. the goal state.

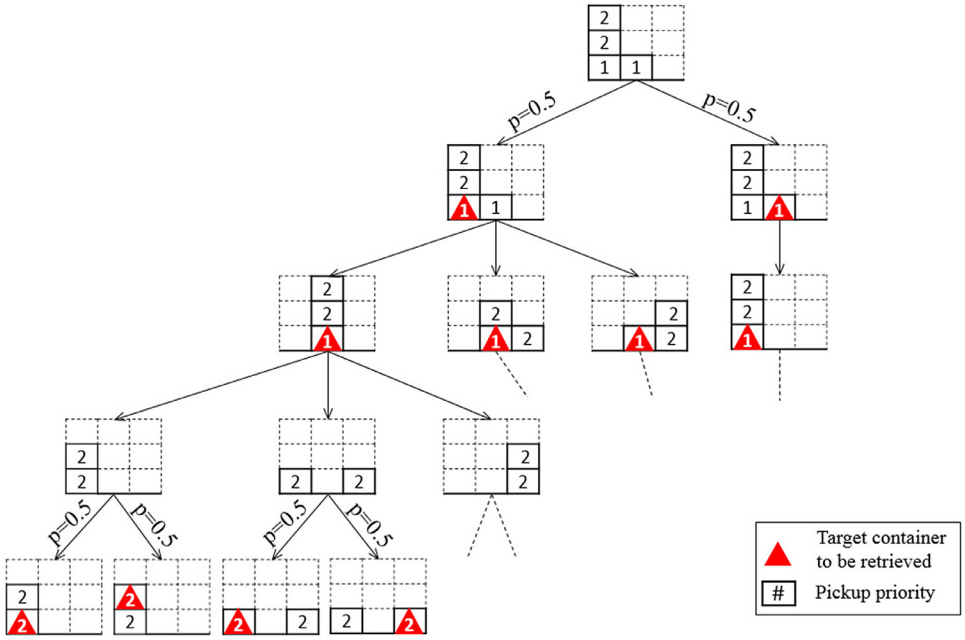

Fig. 1. A sample tree structure expressed in stack layouts.

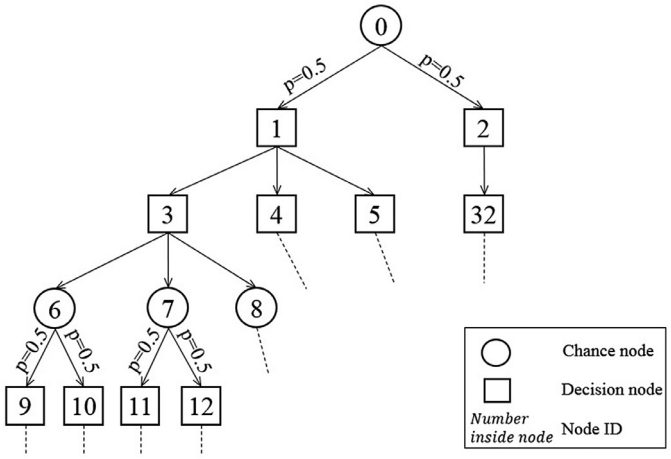

Fig. 2. A sample tree structure expressed in chance and decision nodes.

A decision tree in a stochastic DP model consists of chance and decision nodes. A *chance node* is a tree structure expressing the stochasticity while a *decision node* expresses possible decisions. A circle is normally used to define a chance node while a square is used for a decision node. From a decision node, descendant nodes (or *successors*) are reachable by an action, i.e. one or a series of moves; from a chance node, descendant nodes are branched out by a random variable, which observes a container to be the next target for pickup among containers assigned to the earliest time window (see Fig. 2).

3.2. Search-based algorithm

Let n be any node between the initial node and the goal during the search. Each node in the decision tree has the cost function, denoted $f(n)$, which combines the path cost from the root node to node n and the estimated cost of the cheapest path from n to the goal as in Eq. (4). In the context of CRPTW, $f(n)$ is the sum of the number of reshuffles incurred thus far, denoted $g(n)$, and the minimum expected number of future reshuffles, denoted $h(n)$.

$$f(n) = g(n) + h(n) \tag{4}$$

In essence, the objective of the CRPTW is to predict $f(n)$, or $h(n)$, at the root node. At any node n, getting $g(n)$ is trivial given the decision tree, but predicting $h(n)$ may be difficult in the remaining depth of the node n. Let there be m successors from any node n and each successor from the node n be denoted n_k $(k = 1, 2, \ldots, m)$. Also, let $cost(n, n_k)$ be the number of reshuffles by the action taken to transform the node n into its successor n_k. If the node n is a decision node, we find Eq. (5); if the node n is a chance node, we find Eq. (6), assuming the equal chance of the node n being transformed into any node n_k.

$$h(n) = \min[cost(n, n_k) + h(h_k)], \quad k = 1, 2, \ldots, m \quad \text{where } n \text{ is a decision node.} \tag{5}$$

$$h(n) = \frac{1}{m}\sum_{k=1}^{m} h(n_k) \quad \text{where } n \text{ is a chance node.} \tag{6}$$

The decision tree in Fig. 2 is an illustrative mapping of layouts in Fig. 1. We cannot calculate $h(n)$ for any node n until the search traversal reaches any leaf node. Due to the exponential growth in search space, we use the depth-first search (DFS) with back tracking to save memory consumption. In the DFS, the initial traversal of nodes in Fig. 2 is $0 \rightarrow 1 \rightarrow 3 \rightarrow 6 \rightarrow 9$ until we reach the first leaf node. After that, we find $h(n)$ for nodes in the order of $9 \rightarrow 10 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$. Likewise, a similar procedure will take place down the node 2. As a numerical example of Fig. 1 and 2, we present formulae for getting $h(n)$ in Table 1.

During the back tracking from the leaf nodes, we calculate the values for $h(n)$ as follows:

- $h(6) = 0.5(1 + 0) = 0.5$ because $h(9) = 1$ and $h(10) = 0$.
- It is trivial that $h(7) = 0$. Also, we note that the patterns for node 6 and node 8 are the same, therefore $h(8) = h(6) = 0.5$.
- $h(3) = 2 + min(0.5, 0, 0.5) = 2$. It also confirms that the optimal decision at node 3 is to move to node 7.
- $h(4) = 1$.

Table 1
Formulae of the expected number of rehandles at nodes in Fig. 1 and 2.

$h(n)$	Node Type	Formula
$h(0)$	Chance	$0.5 \cdot [h(1) + h(2)]$
$h(1)$	Decision	$2 + min[h(3), h(4), h(5)]$
$h(3)$	Decision	$2 + min[h(6), h(7), h(8)]$
$h(6)$	Chance	$0.5 \cdot [h(9) + h(10)]$

- $h(5) = 0.5$ because the patterns for node 5 and node 6 are the same if we remove container 1 from the layout at node 5.
- $h(1) = 2 + min(2, 1, 0.5) = 2.5$. It also confirms that the optimal decision at node 1 is to move to node 5.
- $h(0) = 0.5[2.5 + h(2)]$.

To get $h(0)$, we need to calculate $h(2)$ by traversing the decision tree down the node 2. By comparing the patterns of node 32 (the child of node 2) and node 3, however, we note that $h(2)$ should be the same as $h(3)$. Therefore, instead of computing $h(2)$, we find the value for $h(2)$ by mapping the value of $h(3)$, i.e. $h(2) = 2$. Hence, $h(0) = 0.5(2.5 + 2) = 2.25$, which is the optimal solution we are seeking.

The reasoning behind the reduction of the search space here is the abstraction of states and the caching for nodes whose $h(n)$ values have been calculated. In the next section we elaborate the abstraction heuristic applicable to the CRPTW.

3.3. Abstraction heuristic

The CRPTW does not allow the pruning by conventional branch-and-bound (B&B) methods when a search tree contains chance nodes. In order to estimate the expected future cost for an ancestor chance node, every descendant produced from the ancestor chance node should be fully explored for computing the future cost of each descendant. Without this step, no upper bound solution is available for using a B&B method. Also, the estimation of a lower bound at any node n is too limited when there is any chance node down the solution path. Since the essence of partial search methods is to avoid full enumeration, the limitation of not allowing the use of a B&B method makes the tree search very challenging especially when the growth of the search space is exponential in the depth of the tree, the phenomenon being called the *curse of dimensionality* in nearly all practical DP or SDP applications.

In Fig. 1, we observe that some layouts are similarly repeated in the different paths of the search tree, e.g. node 3 ≈ node 32 and node 6 ≈ node 8. This observation motivates us to improve the DFS by caching the solutions for visited nodes, which will let us avoid the re-computation along the visited nodes. Because the required amount of memory for caching will be enormous in the depth of the search, we consider a more memory-efficient caching strategy. One approach to this is by caching part of the search tree to avoid node regenerations (Miura & Ishida (1999)) or by using a transposition table to detect and avoid duplicate paths in graph search (Reinefeld & Marsland (1994)). In our caching strategy, we can reduce the cache size by projecting the original states to the corresponding abstract states. This abstraction method to reduce the search space for the CRP is first studied by the earlier work of the authors (Ku & Arthanari (2016)).

The projection follows two steps: (*i*) remove all empty columns from the stack; (*ii*), reorder the columns in an ascending order of the pickup priorities from the bottom tier to the top tier. For this projection to be reasonably applicable, we need to make additional assumption:

- The cost for handling a container, such as the travelling cost or time, is indifferent among different columns.

After removing the empty column(s), the layout in Fig. 3a may be represented as [(1, 2, 0), (1, 2, 4), (2, 3, 0), (1, 0, 0), (2, 4, 0)] in a 2-D array. This array can be reordered to [(1, 0, 0), (1, 2, 0), (1, 2, 4), (2, 3, 0), (2, 4, 0)], the graphical representation of which is illustrated in Fig. 3d. Likewise, layouts in Fig. 3b and c can be projected into the same abstract layout as of Fig. 3d.

We cache the abstract states only after their $h(n)$ values have been computed. As aforementioned, $h(n)$ values are computed with the back tracking from the leaf nodes. In the example of Fig. 1 and 2, $h(6)$ will be computed and cached after $h(n)$ values for node 9 and node 10 are computed and cached; $h(3)$ will be computed and cached after $h(n)$ values for node 6, node 7 and node 8 are computed and cached (there is only one abstract state for both node 6 and node 8); $h(1)$ will be computed and cached after $h(n)$ values for node 3, node 4 and node 5 are computed and cached (but the children of node 4 and node 5 do not require the re-computation, but their solutions can be looked up from the cache); $h(0)$ will be computed after $h(n)$ values for node 1 and node 2 are computed and cached (but the child of node 2 does not require the re-computation, but its solution can be looked up from the cache).

Although the abstraction heuristic can improve the time performance several times that of the DFS with no caching, the exact methods presented in this paper still do not show reasonable time performance for a practical use. In the following section, we introduce index-based heuristics that can be used for the domain of the CRPTW.

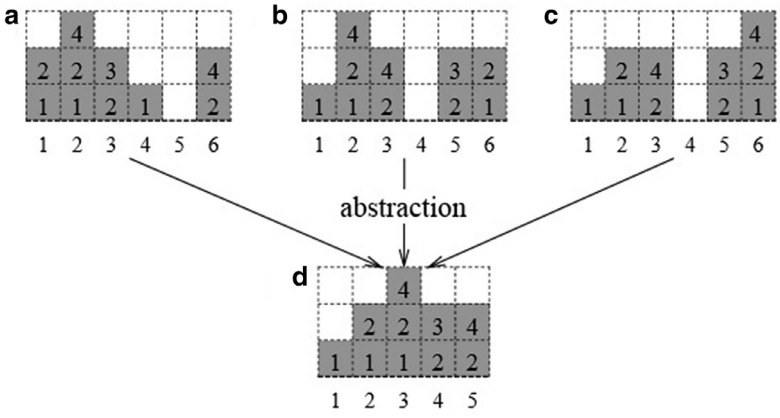

Fig. 3. Abstraction of stack layouts.

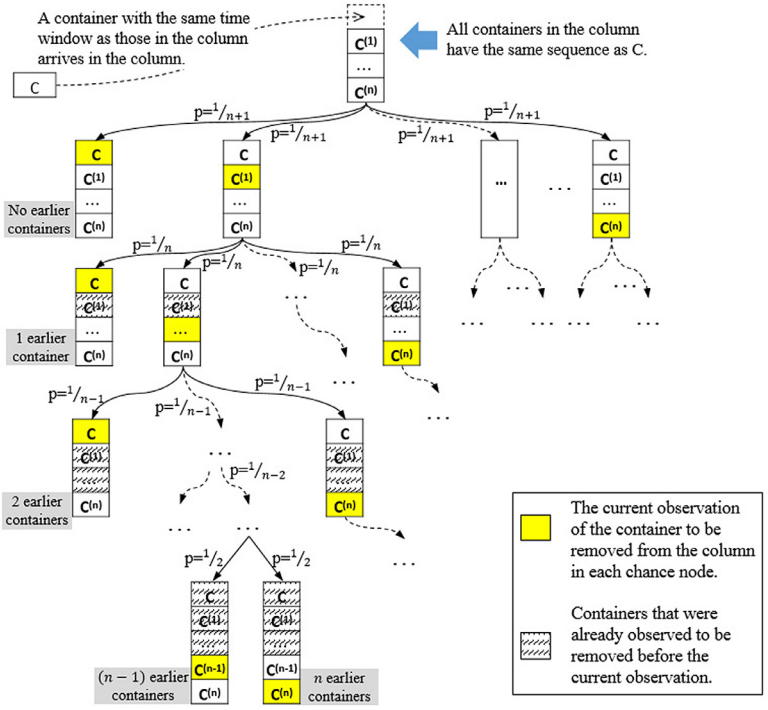

Fig. 4. Decision tree for the ERI calculation.

4. The expected reshuffle index (ERI) heuristic

In this section, we propose an index-based heuristic for choosing the target column for each container above the target container to be reshuffled in. The RI is the index indicating the number of containers in the column that departs earlier than the container being reshuffled to the column. It was first defined by Murty et al. (2005a). When a container above the target container should be reshuffled during the search of the CRP, the RI heuristic chooses the column to be reshuffled to by computing the lowest RI, and the tie is broken towards the taller column and arbitrarily afterwards. However, the RI heuristic cannot calculate the index of a column in the CRPTW where there exist containers, the precedence relationship among which is unknown.

As an application of the RI heuristic to the CRPTW, the ERI is defined as the expected number of containers that depart earlier than the container being reshuffled to the column. The ERI heuristic chooses the column with the lowest ERI as the target column for the container being reshuffled. In the computational experiment, the solution quality of the selection by the ERI is compared with that of the random selection. Below we introduce the notations for the calculation of the ERI.

- n: number of the containers in the column where their time windows are the same as that of the container being reshuffled to
- $p_n(k)$: the probability of having k number of earlier departing containers below the container being reshuffled to
- $f_n(k)$: the marginal contribution of having k number of earlier departing containers to the ERI
- f_n: the marginal ERI of the column.

Consider a column containing n containers with the same time window as the container being reshuffled. If the container being reshuffled is to be stacked in the column, the ERI for the column remains to be computed. With the arrival of such container, we now have $n+1$ containers with the same time window. Assume that every such container is equally likely to be the earliest departing container among them. Constructing a decision tree, the root at level 0 produces $n+1$ successors as in Fig. 4.

From left to right at level 1, each immediate successor of the root observes one container of the same sequence as the earliest departure in the order of top to bottom. That is, the far left successor observes the container at the top as the earliest departure while the far right successor observes the container at the bottom as the earliest departure. The far left successor does not branch out any successors because in terms of the ERI, there is no container departing earlier than the one that just arrived in the column. The remaining successors at level 1 branch out n successors each. Let's traverse down the second from the left successor at level 1. We branch out n successors, the first successor of which clearly has one earlier departing container. The remaining successors branch out $n-1$ successors at level 2. Again down the second branch at level 2 as in Fig. 4, we find the first successor of clearly having two earlier departing containers, thus having no further successors from there. By constructing a decision tree like this, we find the following rule to calculate the ERI.

Let k be the number of the containers in the column whose time windows are the same as the time window of the container arriving in the column, but which are later observed to depart earlier than it. The probability of reaching each leaf node having k earlier departing container(s) is $1/(n+1)\cdot {}_nP_k$, and the number of such leaf nodes is ${}_nP_k$. Since $p_n(k)$ is the product of these two terms and $f_n(k)$ is the product of k and $p_n(k)$, we find Eq. (7) and Eq. (8), respectively.

$$p_n(k) = \frac{1}{n+1} \tag{7}$$

$$f_n(k) = \frac{k}{n+1} \tag{8}$$

Lastly, f_n is the sum of $f_n(k)$ where k ranges from 1 to n. Hence, we get Eq. (9), which provides us with a simple heuristic to determine the then target column to be reshuffled to. Thus, the ERI for a column will be the sum of f_n for the column and the number of containers whose departure time window is earlier than that of

Table 2
Test scenarios with randomly generated instances.

Col.	Tier	3		4		5		6	
	Fill rate	50 percent	67 percent	50 percent	67 percent	50 percent	67 percent	50 percent	67 percent
5	No. blocks	8	10	10	13	13	17	15	20
	Test (exact)	✓		✓		✓			
	Test (heuristic)	✓	✓	✓	✓	✓	✓	✓	✓
6	No. blocks	9	12	12	16	15	20	18	24
	Test (exact)	✓		✓					
	Test (heuristic)	✓	✓	✓	✓	✓	✓	✓	✓
7	No. blocks	11	14	14	19	18	23	21	28
	Test (exact)	✓		✓					
	Test (heuristic)	✓	✓	✓	✓	✓	✓	✓	✓
8	No. blocks	12	16	16	21	20	27	24	32
	Test (exact)	✓		✓					
	Test (heuristic)	✓	✓	✓	✓	✓	✓	✓	✓
9	No. blocks	14	18	18	24	23	30	27	36
	Test (exact)	✓							
	Test (heuristic)	✓	✓	✓	✓	✓	✓	✓	✓
10	No. blocks	15	20	20	27	25	34	30	40
	Test (exact)	✓							
	Test (heuristic)	✓	✓	✓	✓	✓	✓	✓	✓

• (exact) : DFS in the search tree • (heuristic) : ERI heuristic

the container arriving in the column.

$$f_n = \sum_{k=1}^{n} f_n(k) = \sum_{k=1}^{n} \frac{k}{n+1} = \frac{1}{n+1}\frac{n(n+1)}{2} = \frac{n}{2} \tag{9}$$

5. Computational experiment

Numerical experiments have been performed to evaluate the performance of the abstraction heuristic and the ERI heuristic proposed in this paper. We compare the time performance for the exact methods (the DFS and the abstraction heuristic), and the solution quality for the ERI heuristic and the random selection heuristic. Instead of ranking the index for each column as in the ERI heuristic, the target column to be reshuffled to is chosen randomly in the random selection heuristic. These tests are performed using Java programming language code developed by the first author. The experiments are performed on a desktop with Intel i5-2500 Quad 3.3 gigahertz CPU, 8 gigabyte of RAM, and 64-bit Windows 7 OS.

For our experiment, we impose the ratio of distinct time windows (the number of blocks divided by the number of time windows) be around 50 percent. It is because a too high ratio of this generates instances similar to those applicable to the (deterministic) CRP, and a too low ratio may not provide us with a meaningful interpretation for our model configuration. For the comparison between the exact methods, we focus on small-scale instances. The fill rate (the number of blocks divided by the capacity) is around 50 percent. For the comparison between the ERI heuristic and the random selection heuristic, we have prepared instances, in which columns range from five to ten and tiers range from three to six. There are two groups for fill rates, 50 percent and 67 percent.

Table 2 summarises test scenarios for our computational experiments. Each test scenario consists of thirty instances, which are randomly generated with the aforementioned settings. The test instances used in our experiments are available from http://crp-timewindow.blogspot.com , therefore interested readers may refer to them as their benchmarking sets.

The experiments for the exact methods have been run only once per instance because the expected outcome will always be the same in every run. For the ERI heuristic and the random selection heuristic, we have run random simulation experiments, repeating 5, 000 times per instance. The reason for such simulation experiments for the heuristics is because the revelation of the earliest departing container from any chance node is stochastic. It also means that the heuristic rules apply only to decision nodes. We take the average number of reshuffling moves as the performance indicator for the heuristics, and the time performance is not reported here because the computational time for the heuristics simulation is marginal only by from a few seconds to less than one minute.

Table 3 compares the time performance between the exact methods. With the time limit of eight hours' run per instance, we could not find the optimal solutions from some instances in larger stacks. The comparison in the data shows that the time performance of the abstraction heuristic clearly outperforms the DFS by several times (referring to DFS/abstr. column in Table 3). Consider that in the abstraction heuristic, the computation time at each node is longer than that of the DFS due to the additional processing time required for the abstraction. Hence, it is conclusive that the reduction in the re-computation for the similar nodes visited before far outweighs the processing time for the abstraction on those nodes.

Table 4 compares the solution quality in terms of the average number of reshuffling moves between the ERI heuristic and the random selection heuristic. We find that the reduction in the average number of reshuffles in the ERI heuristic over the random selection heuristic is significant by 30–40 percent under the stack configurations of the test instances. From Fig. 5, we see that the performance increase by the ERI over the random selection is greater in stack configurations with more columns. The fewer the average number of reshuffles in the ERI compared with that in the random selection, the lower the ratio is.

Lastly, when we compare optimal solutions in Table 3 with the average numbers of reshuffles in Table 4, we find that the average numbers of reshuffles by the ERI are very close to the optimal solutions by the exact solution. We are aware that there are relatively a small number of stack configurations that can be solved optimally within the time limit. Thus, we may extend the time limit to provide enough evidence for the judgment on the acceptability of the ERI heuristic. However, from the stack size of 8×4 or bigger with the fill rate being 50 percent, the exact methods frequently encounter into instances, the number of explored nodes in which exceeds billions of nodes. With an enormous number of states to evaluate for those instances, we draw our conjecture from the available set of benchmarking data, stating that the performance of the ERI is reasonable and is applicable to stacks of any practical size.

Table 3
Comparison between the DFS and the abstraction heuristic.

Col.	Tier	No. blocks	Opt. sol.	Time (second) DFS	Time (second) Abstr.	DFS / abstr.	Nodes in abstraction Explored	Nodes in abstraction Revisited
5	3	8	1.7	0.04	0.02	–	297.5	191.0
	4	10	3.11	6.74	2.58	2.61	214,700.2	115,024
	5	13	5.32	∞	2483.30	–	2.49e+08	1.16e+08
6	3	9	1.74	0.041	0.013	3.10	788.4	191.0
	4	12	3.68	659.17	139.08	4.74	13,877,193	115,024
7	3	11	2.88	1.46	0.33	4.42	27,193.2	15,998.4
	4	14	4.16	867.98	207.62	4.18	19,404,986	9,007,958
8	3	12	2.31	1.42	0.33	4.3	24,700.9	15,802.5
	4	16	–	∞	∞	–	–	–
9	3	14	3.00	198.27	32.24	6.15	2,781,404	1,325,161
10	3	15	3.19	526.23	58.85	8.94	4,876,795	2,569,675

• Fill rate of the instances experimented = 50 percent

Table 4
Comparison between ERI heur. and random selection heur.

Col.	Tier	Fill rate (50 percent) OPT	Avg reshufl. ERI	Avg reshufl. Rand.	Perform. (ERI/rand)	Fill rate (67 percent) Avg Reshufl. ERI	Avg Reshufl. Rand.	Perform. (ERI/rand)
5	3	1.7	1.71	2.34	0.73	3.10	4.16	0.75
	4	3.11	3.20	4.62	0.69	5.80	8.22	0.71
	5	5.32	5.58	8.00	0.70	11.15	15.28	0.73
	6	–	8.29	12.35	0.67	16.14	22.93	0.70
6	3	1.74	1.75	2.43	0.72	3.92	5.53	0.71
	4	3.68	3.75	5.59	0.67	7.68	11.28	0.68
	5	–	6.18	9.54	0.65	10.97	16.42	0.67
	6	–	8.77	13.53	0.65	16.65	25.03	0.67
7	3	2.88	2.89	4.11	0.7	4.01	6.05	0.66
	4	4.16	4.02	6.03	0.67	7.68	11.61	0.66
	5	–	7.18	11.27	0.64	12.64	19.65	0.64
	6	–	10.98	17.69	0.62	19.49	30.79	0.63
8	3	2.31	2.32	3.23	0.72	4.7	6.94	0.68
	4	–	4.88	7.49	0.65	8.5	13.25	0.64
	5	–	8.27	13.45	0.61	14.44	23.12	0.62
	6	–	11.61	19.29	0.6	21.72	34.63	0.63
9	3	3.00	3	4.54	0.66	5.19	7.8	0.67
	4	–	5.8	9.29	0.62	9.92	16	0.62
	5	–	9.36	15.57	0.6	16.97	27.39	0.62
	6	–	12.09	20.93	0.58	22.73	38.14	0.6
10	3	3.19	3.2	4.75	0.67	5.3	7.86	0.67
	4	–	6.33	10.41	0.61	10.5	16.91	0.62
	5	–	9.8	16.63	0.59	17.23	29.03	0.59
	6	–	13.51	23.41	0.58	25.58	44.07	0.58

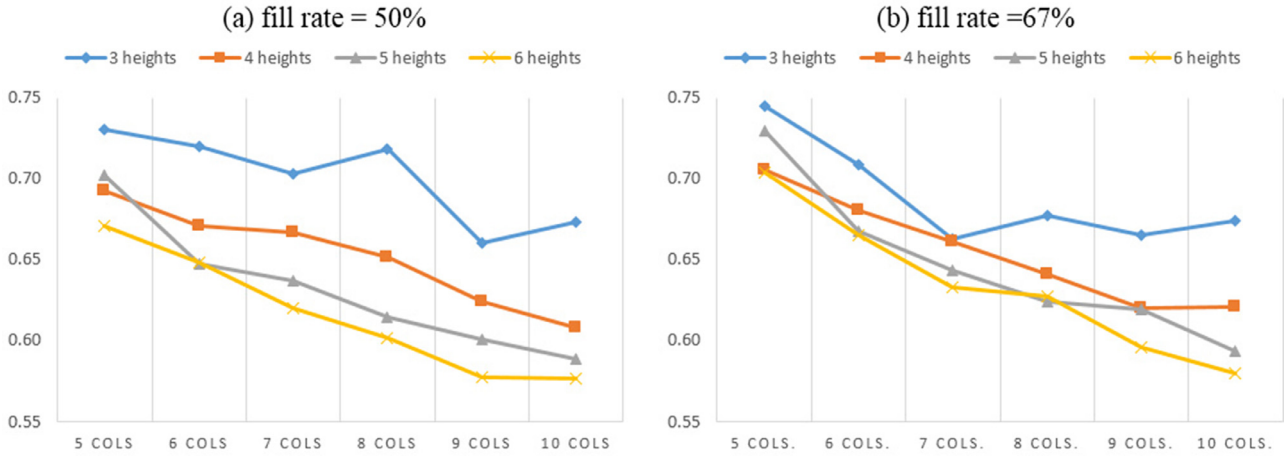

Fig. 5. Reshuffling ratio of the ERI over the random selection.

6. Conclusion

In this paper, we have considered departure time windows for the CRP, which is a classic combinatorial optimisation problem in container terminals. Hence, we term this problem the CRPTW, or the CRP with time windows. We formulate an SDP model and use a search-based algorithm (the DFS) to solve the model optimally. As an improvement for the search efficiency, we introduce the abstraction heuristic, i.e. the caching method by the abstraction of states in the search tree. To overcome the computational time restriction for the exact methods, we propose the ERI heuristic, which chooses the target column at each stage of the container reshuffling by ranking the candidate columns, and benchmark its average solution quality in comparison to that of the random selection method.

Salient points from the computational experiments are as follows: First, among the two exact methods discussed in this paper, the abstraction heuristic is at least several times faster than the DFS. This indicates that the reduction of the search space by the abstraction far outweighs the additional processing time required for the abstraction on each computed node. Second, the solution quality of the ERI heuristic is much better than that of the random selection method. We see that around 30–40 percent reduction in the average number of reshuffles can be achieved by the ERI heuristic as opposed to the random selection method. Also by analysing the relationship with the stack size, we find that there is a trend of further reduction as the number of columns increases. Lastly, we find that the average numbers of reshuffles by the ERI heuristic are very close to the optimal solutions by the exact solution. Our conjecture from this result is that the ERI is a reasonably acceptable heuristic for the CRPTW.

We suggest this way of predicting the expected number of reshuffles is useful to a real-time terminal system with the TAS. The ERI heuristic calculates the expected number of reshuffles fast and with near-optimal solutions. When the terminal system is able to make use of this information (the expected number of reshuffles per stack) in real-time, their stacking policy can be enhanced further. For instance, when choosing a stack to store import containers, the terminal systems will be able to consider the expected number of reshuffles per stack, thus be able to avoid stacks having more expected number of reshuffles.

For future research, developing more advanced heuristics to compare with the heuristics presented in this paper is desirable. We also note that the exact solution provided in this paper is very limited with respect to the size of the solvable instances. Although the problem falls into a difficult category, how to enlarge the exact solution approach could be requiring future research as well.

Acknowledgements

Support for this work was provided by the University of Auckland Doctoral Scholarship (Code No 43). The authors would like to thank Mr. Antony de Pont, Manager of Gate Operations and Documentation at Ports of Auckland, who shared with us essential information for the *Vehicle Booking System* implemented at Ports of Auckland. We also thank anonymous reviewers and the editor for their suggestions that led to the improvements of this paper.

References

Akyüz, M. H., & Lee, C.-Y. (2014). A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics, 61*(2), 101–118.

van Asperen, E., Borgman, B., & Dekker, R. (2013). Evaluating impact of truck announcements on container stacking efficiency. *Flexible Services and Manufacturing Journal, 25*(4), 543–556.

Borgman, B., van Asperen, E., & Dekker, R. (2010). Online rules for container stacking. *OR spectrum, 32*(3), 687–716.

Borjian, S., Manshadi, V. H., Barnhart, C., & Jaillet, P. (2013). Dynamic stochastic optimization of relocations in container terminals. *Working Paper*. MIT.

Caserta, M., Schwarze, S., & Voß, S. (2011). Container rehandling at maritime container terminals. In *Handbook of terminal planning* (pp. 247–269). Springer.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research, 219*(1), 96–104.

de Castillo, B., & Daganzo, C. F. (1993). Handling strategies for import containers at marine terminals. *Transportation Research Part B: Methodological, 27*(2), 151–166.

Chu, C.-Y., & Huang, W.-C. (2005). Determining container terminal capacity on the basis of an adopted yard handling system. *Transport Reviews, 25*(2), 181–199.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, J. M. (2014). A domain-specific knowledge-based heuristic for the blocks relocation problem. *Advanced Engineering Informatics, 28*(4), 327–343.

Giuliano, G., & O'Brien, T. (2007). Reducing port-related truck emissions: The terminal gate appointment system at the ports of los angeles and long beach. *Transportation Research Part D: Transport and Environment, 12*(7), 460–473.

Gupta, N., & Nau, D. S. (1992). On the complexity of blocks-world planning. *Artificial Intelligence, 56*(2), 223–254.

Huynh, N., & Walton, C. M. (2008). Robust scheduling of truck arrivals at marine container terminals. *Journal of Transportation Engineering, 134*(8), 347–353.

Jin, B., Zhu, W., & Lim, A. (2015). Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research, 240*(3), 837–847.

Jones, E. G., & Michael Walton, C. (2002). Managing containers in a marine terminal: Assessing information needs. *Transportation Research Record: Journal of the Transportation Research Board, 1782*(1), 92–99.

Kang, J., Ryu, K. R., & Kim, K. H. (2006). Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent Manufacturing, 17*(4), 399–410.

Kim, K. H. (1997). Evaluation of the number of rehandles in container yards. *Computers & Industrial Engineering, 32*(4), 701–711.

Kim, K. H., & Hong, G.-P. (2006). A heuristic rule for relocating blocks. *Computers & Operations Research, 33*(4), 940–954.

Kim, K. H., Park, Y. M., & Ryu, K.-R. (2000). Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research, 124*(1), 89–101.

Ku, D., & Arthanari, T. (2016). On the abstraction method for the container relocation problem. *Computers & Operations Research, 68*, 110–122.

Miura, T., & Ishida, T. (1999). Stochastic node caching for memory-bounded search. *Systems and Computers in Japan, 30*(2), 57–65.

Morais, P., & Lord, E. (2006). Terminal appointment system study. *Technical report*. Transportation Development Centre of Transport Canada.

Murty, K. G., Liu, J., Wan, Y.-w., & Linn, R. (2005a). A decision support system for operations in a container terminal. *Decision Support Systems, 39*(3), 309–332.

Murty, K. G., Wan, Y.-w., Liu, J., Tseng, M. M., Leung, E., Lai, K.-K., & Chiu, H. W. (2005b). Hongkong international terminals gains elastic capacity using a data-intensive decision-support system. *Interfaces, 35*(1), 61–75.

Namboothiri, R., & Erera, A. L. (2008). Planning local container drayage operations given a port access appointment system. *Transportation Research Part E: Logistics and Transportation Review, 44*(2), 185–202.

Petering, M. E., & Hussein, M. I. (2013). A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research, 231*(1), 120–130.

Ports of Auckland eNews (2013). Truck service statistics. *Technical report*. Ports of Auckland.

Reinefeld, A., & Marsland, T. A. (1994). Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16*(7), 701–710.

Sauri, S., & Martin, E. (2011). Space allocating strategies for improving import yard performance at marine terminals. *Transportation Research Part E: Logistics and Transportation Review, 47*(6), 1038–1057.

Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: a literature update. *Or Spectrum, 30*(1), 1–52.

Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operation and operations research-a classification and literature review. *OR spectrum, 26*(1), 3–49.

Ünlüyurt, T., & Aydın, C. (2012). Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation, 46*(4), 378–393.

Wan, Y.-w., Liu, J., & Tsai, P.-C. (2009). The assignment of storage locations to containers for a container stack. *Naval Research Logistics (NRL), 56*(8), 699–713.

Watanabe, I. (1991). Characteristics and analysis method of efficiencies of container terminal: an approach to the optimal loading/unloading method. *Container Age, 3*, 36–47.

Yang, J. H., & Kim, K. H. (2006). A grouped storage method for minimizing relocations in block stacking systems. *Journal of Intelligent Manufacturing, 17*(4), 453–463.

Zehendner, E., & Feillet, D. (2014). Benefits of a truck appointment system on the service quality of inland transport modes at a multimodal container terminal. *European Journal of Operational Research, 235*(2), 461–469.

Zhang, C., Chen, W., Shi, L., & Zheng, L. (2010). A note on deriving decision rules to locate export containers in container yards. *European Journal of Operational Research, 205*(2), 483–485.

Zhao, W., & Goodchild, A. V. (2010). The impact of truck arrival information on container terminal rehandling. *Transportation Research Part E: Logistics and Transportation Review, 46*(3), 327–343.

Zhu, W., Qin, H., Lim, A., & Zhang, H. (2012). Iterative deepening a$*$ algorithms for the container relocation problem. *Automation Science and Engineering, IEEE Transactions on, 9*(4), 710–722.