European Journal of Operational Research 235 (2014) 431–447

Contents lists available at ScienceDirect

European Journal of Operational Research

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

An exact method for scheduling a yard crane

Amir Hossein Gharehgozli [a], Yugang Yu [b,*], René de Koster [a], Jan Tijmen Udding [c]

[a] Rotterdam School of Management, Erasmus University, Rotterdam, The Netherlands
[b] School of Management, University of Science and Technology of China, Hefei, China
[c] Mechanical Engineering, Systems Engineering, Eindhoven University of Technology, The Netherlands

ARTICLE INFO

Article history:
Received 29 October 2012
Accepted 24 September 2013
Available online 15 October 2013

Keywords:
Container storage and retrieval
Sequencing
Multiple I/O points
Travel time
Traveling salesman problem

ABSTRACT

This paper studies an operational problem arising at a container terminal, consisting of scheduling a yard crane to carry out a set of container storage and retrieval requests in a single container block. The objective is to minimize the total travel time of the crane to carry out all requests. The block has multiple input and output (I/O) points located at both the seaside and the landside. The crane must move retrieval containers from the block to the I/O points, and must move storage containers from the I/O points to the block. The problem is modeled as a continuous time integer programming model and the complexity is proven. We use intrinsic properties of the problem to propose a two-phase solution method to optimally solve the problem. In the first phase, we develop a merging algorithm which tries to patch subtours of an optimal solution of an assignment problem relaxation of the problem and obtain a complete crane tour without adding extra travel time to the optimal objective value of the relaxed problem. The algorithm requires common I/O points to patch subtours. This is efficient and often results in obtaining an optimal solution of the problem. If an optimal solution has not been obtained, the solution of the first phase is embedded in the second phase where a branch-and-bound algorithm is used to find an optimal solution. The numerical results show that the proposed method can quickly obtain an optimal solution of the problem. Compared to the random and Nearest Neighbor heuristics, the total travel time is on average reduced by more than 30% and 14%, respectively. We also validate the solution method at a terminal.

1. Introduction

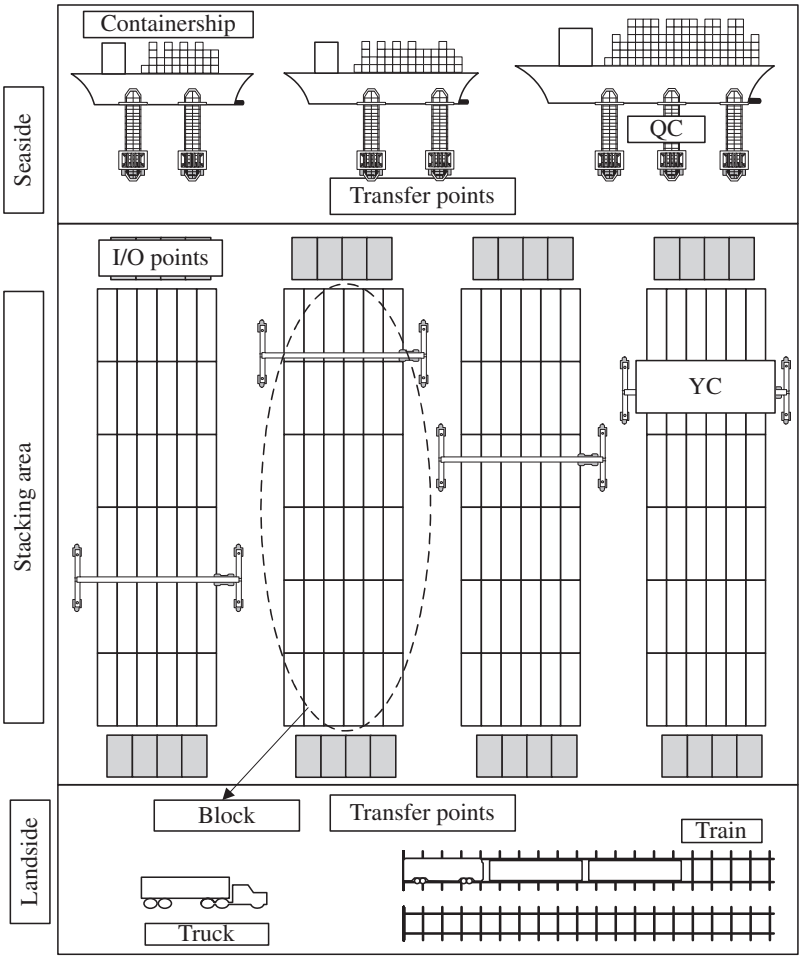
Containerized transportation has become an essential part of world trade during the past decades (Kim & Kim, 1999; Steenken, Voß, & Stahlbock, 2004). Between 1990 and 2015, the total number of full containers shipped internationally is expected to grow from 28.7 million 20-feet equivalent units (TEU) to 177.6 million TEU (United Nations: ESCAP, 2007). All these containers pass through different container terminals all around the globe. As a result, container terminals daily deal with a large number of containers (Drewry, 2011; Kim & Kim, 1999). The terminals in the Port of Rotterdam, for example, handled approximately 11 million TEU in 2010 (Port of Rotterdam Authority, 2010).

Fig. 1a shows a container terminal with several blocks of containers in the stacking area where yard cranes (YCs) stack and retrieve containers. Other terminal layouts are studied by Wiese, Suhl, and Kliewer (2010). Each YC can handle containers in a single block consisting of multiple rows, tiers, and bays as shown in Fig. 1b. The YC can move containers along the bays and rows of the block simultaneously. A number of input/output (I/O) points are located at the seaside and landside of the block. When a container is to be retrieved, the YC picks it up from its location in the block and drops it at an I/O point at the container's destination side of the block. In case of a storage request, the YC picks up a container from its I/O point and stores it in a location in the block.

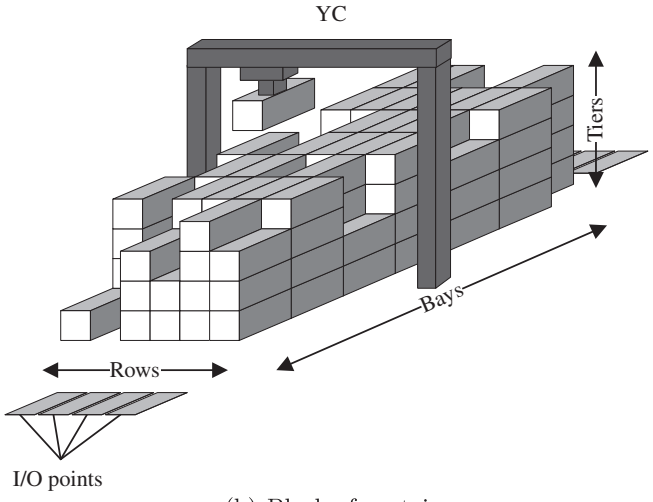
Handling a large number of containers affects the performance of a container terminal. For the container terminal operator, minimizing the makespan of vessels is the most important performance criterion (see, for example, Böse, 2011; De Koster, Balk, & Van Nus, 2009; Linn & Zhang, 2003), as shipping lines are the terminal's paying customers. In order to minimize the berthing time of a ship, container terminal operators assign multiple quay cranes (QCs) to load and unload each ship, and do not allow them to fall idle, if possible. Furthermore, over the years, the speed of QCs has improved substantially. However, the performance of QCs also depends on the performance of YCs. The speed of a YC, calculated as the number of handled containers per hour, is approximately one-third of a modern QC (Murty, 2007). As a result, in order to fully utilize QCs, containers to be loaded or unloaded by each QC must be distributed over several blocks. Furthermore, while retrieving and stacking containers of ships, each YC should also serve the landside. At the landside, trains and trucks supply the stack with containers and also demand services from the YCs.

* Corresponding author. Address: School of Management, University of Science and Technology of China, 96 Jinzhai, Hefei, Anhui, China. Tel.: +86 551 63606187.
E-mail addresses: AGharehgozli@rsm.nl (A.H. Gharehgozli), ygyu@ustc.edu.cn (Y. Yu), RKoster@rsm.nl (R. de Koster), jtu@checkmade.nl.

0377-2217/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2013.09.038



(a) Top view of a container yard



(b) Block of containers

Fig. 1. Schematic representation of a container yard layout.

Therefore, at every moment, each YC has to deal with a set of containers of the same relative priority waiting to be stacked or retrieved. If the makespan of YCs is not minimized, not only the berthing times of ships but also the waiting times of trucks and trains increase. Cost-based objective functions do not satisfy this purpose, as all costs for delays of individual containers are artificial. In addition, individual containers have quite a bit of flexibility in due time. In practice, a block sequencing policy is implemented in which containers are grouped and requests are performed block by block (see the case study in Section 4.2). Within each block, containers of about the same priority can be carried out in any order with the objective of minimizing YC makespan.

We therefore consider the problem of sequencing a given set of requests to be stacked or retrieved by a YC in a single block. The objective is to minimize the travel time of the YC to carry out the requests. We formulate this problem with multiple I/O points as a special case of an asymmetric traveling salesman problem (ATSP), in which the YC must visit an I/O point to carry out every request. We prove that the problem complexity stems from the number of requests and the number of I/O points. We propose a two-phase solution method which can quickly solve the problem. In the first phase of the solution method, we develop a new merging algorithm to patch subtours of an optimal solution of the assignment problem (AP) relaxation of the problem without adding extra travel time. In this phase, we first search for two arcs from every two different subtours visiting a common I/O point, and swap the destinations of them to merge the subtours. Next, based on the fact that in some arcs, the YC has multiple I/O point options with the same travel time to visit, we can create further opportunities to merge more subtours. We show that the first phase runs in polynomial time, and often finds an optimal solution. Otherwise, a branch-and-bound (B&B) algorithm is used in the second phase to find an optimal solution of the problem. In this phase, the merging algorithm is again used in each node of the B&B tree to save computation time.

The numerical experiments show that the two-phase solution method is quite efficient. For instances up to 200 requests, an optimal solution can be obtained in less than a second. Furthermore, we show that the travel time reduction between the optimal and random sequence is around 30%, on average. The reduction is around 14%, in case the Nearest Neighbor (NN) heuristic is used to find the sequence. Reducing the total travel time of the YC helps not only to improve the performance of the terminal but also to decrease the negative environmental impacts of container handling operations. Container terminal nowadays hugely invest in developing new technologies to reduce fuel consumption and YC emissions. The average fuel consumption of traditional manual YCs ranges between 20 and 30 liters of diesel per hour. In peak hours, the amount can exceed 100 liters per hour. The result is more than 350 tons of CO2 emissions per year (Zrnić & Vujičić, 2012). Reducing the travel time by 30% or 14% can result in substantially less fuel consumption, and consequently less emissions. Finally, the results show that the two-phase solution method can obtain significantly better results than CPLEX truncated after five hours.

The scheduling of stacking cranes at a container terminal has been studied for some time already. An important body of research concerns QCs that must be synchronized with berthing operations (see Bierwirth and Meisel (2010) for a comprehensive literature review). On the other hand, yard crane scheduling has not been well studied compared to QC scheduling (see the recent literature reviews, for example, Gharehgozli, Laporte, Yu, & De Koster, 2013; Stahlbock & Voß, 2008). In general, scheduling a yard crane to carry out storage and retrieval requests can be modeled as an ATSP or one of its special cases such as the rural postman problem, or Chinese postman problem, depending on the properties of the problem (Vis & Roodbergen, 2009). In the following, we present a brief review on the ATSP, and then we review related applications in sequencing storage and retrieval requests.

Simply stated, the ATSP is a combinatorial problem in which from a given list of locations with given pairwise travel times, a tour must be determined passing every location exactly once in such a way that the total travel time is minimized (Lawler, Lenstra, Rinnooy Kan, & Shmoys, 1985). In general, the ATSP is proven to be $\mathcal{NP}$-hard (see, for example, Karp, 1972). Several exact and heuristic algorithms have been proposed to solve the ATSP. Among all exact algorithms, a large body of research focuses on the B&B algorithm based on the subtour elimination approach proposed by Carpaneto and Toth (1980) and later improved by Carpaneto, Dell'Amico, and Toth (1995) and Miller and Pekny (1991). This is closely related to the second phase of our solution method. The core idea of the B&B algorithm is the AP relaxation. In each node of the B&B tree, an AP with some extra constraints is solved. Constraints exclude a number of arcs and require some others to appear in the AP solution so that some subtours can be avoided. Carpaneto et al. (1995) and Miller and Pekny (1991) also propose two merging algorithms to improve the computation time. In these algorithms, the reduced costs of the arcs are used to merge subtours, and they are thus time consuming. Therefore, Carpaneto et al. (1995) for example use the merging algorithm in every node if the number of zero-reduced cost arcs is more than a threshold. In addition, the objective value may increase after merging subtours. These issues make their algorithms different than the one developed in this paper, in a sense that our algorithm uses swappable arcs with one or multiple common visiting I/O points to quickly patch subtours without extra travel time.

In the literature available on sequencing container storage and retrieval requests, many authors deal with problems where either storage or retrieval requests are considered (see, for example, Kim & Kim, 1999; Narasimhan & Palekar, 2002; Ng, 2005), or where multiple YCs carry out the requests (see, for example, Cheung, Li, & Lin, 2002; Dorndorf & Schneider, 2010; Li, Wu, Petering, Goh, & de Souza, 2009; Vis & Carlo, 2010; Zhang, Wan, Liu, & Linn, 2002). In most of these papers discrete time models are used to formulate the problem and the solution approaches do not apply here. The first problem type is more limited than our problem and the second is more general and heuristic algorithms are usually proposed to solve it. Apart from this literature, reviewing the papers on scheduling a single crane to carry out storage and retrieval requests of containers at a container terminal (Vis & Roodbergen, 2009) or of unit loads in a warehouse (De Koster & Van der Poort, 1998; Ratliff & Rosenthal, 1983; Van den Berg & Gademann, 1999) reveals that specific properties of each individual problem determine whether a polynomial solution method can be developed. These properties are: (1) the number of I/O points and (2) the number of rows in which containers are stored or retrieved. The literature can therefore be categorized based on these properties as follows.

In case of a single I/O point, polynomial solution methods can be developed (see Burkard, Deineko, Van Dal, Van der Veen, & Woeginger (1998) for solvable TSP cases). In this case, all subtours of an optimal solution of an AP relaxation to the ATSP can be merged as a complete Hamiltonian tour, since the crane returns to the same I/O point for every request. Having returned to the I/O point, the crane can select any storage request. It can also select a retrieval request in case in the optimal AP solution that retrieval request is sequenced after another retrieval request. Therefore, all subtours can be merged without adding extra travel time.

Van den Berg and Gademann (1999) discuss a problem with both an output point where retrievals are dropped off and an input point where storages are picked up. They assume that storage requests are executed in a first-come-first-served order. The problem is to find which retrieval requests must be interleaved after each storage. This allows a formulation as a transportation problem using a bipartite graph. They prove that each feasible solution of the transportation problem corresponds to a retrieval and storage sequence of the main problem. Their problem is similar to the situation with a single I/O point, since the I/O is fixed for each request.

Vis and Roodbergen (2009) consider a problem with a single block where the block corresponds to multiple rows separated by aisles. Each row has one I/O point at each end which serves that row. A single straddle carrier stores or retrieves all containers in the rows, where each container passes an appropriate I/O point of its row. The straddle carrier must exit the current row at the

end in order to travel from one row to the other one, which is time-consuming. Therefore, it usually finishes all requests in that row before traveling to another. Based on this specific characteristic of straddle carriers, they can decompose the problem with multiple rows into several problems with one row and two I/O points and solve them separately. They propose a solution method based on dynamic programming to determine the shortest path for the straddle carrier crossing multiple rows. The dynamic programming allows multiple visits of a single row. It uses an optimal storage and retrieval request sequence in each row. Using an optimal AP solution to sequence the requests in a row with two I/O points results in maximum two subtours. They prove that in order to obtain the optimal solution, these subtours can be merged by enumerating and exchanging arcs in $O(N)$ time because the travel time matrix has the properties of a Monge matrix in the case of a single row and two I/O points (see Gilmore & Gomory, 1964). Vis and Carlo (2010) also use the same method to find a lower bound for their container sequencing problem with a single block and two YCs. To obtain the lower bound, they collapse all rows of the block together as a single row and assume that a single YC carries out all requests. Considering two YCs to carry out the requests makes the problem complex so that they resort to a metaheuristic to solve the problem.

Vis and Roodbergen (2009) extend the models proposed by Ratliff and Rosenthal (1983) and De Koster and Van der Poort (1998) for routing an orderpicker in a warehouse. Ratliff and Rosenthal (1983) consider a warehouse with parallel aisles (comparable to a container terminal with parallel rows), a central I/O point and no storage request, whereas De Koster and Van der Poort (1998) consider the same situation but the warehouse can have an I/O point at the end of each aisle. Dynamic programming is used in both studies to decompose and solve the problems.

Different from the previous line of research, in this paper, a continuous time integer programming model is proposed to stack and retrieve containers in a block with multiple I/O points and rows densely located together. Since the YC can simultaneously move across the rows and does not return to the row ends to switch rows, the problem is more complex. Compared to the current YC position, many locations in neighbor rows have identical travel times. In order to obtain the optimal solution, all rows and I/O points must be considered at the same time in the solution method. Decomposing the block into multiple single-row blocks for adopting the methods proposed by Vis and Roodbergen (2009), De Koster and Van der Poort (1998), and Ratliff and Rosenthal (1983) is not possible, since returning to an I/O point located at either end of each row is essential in the proposed network models and solution methods. Extra arcs are necessary in the network to model switching rows without returning to the I/O points. Even if the block is divided into multiple single-row blocks, the optimal solution of each row cannot be obtained using the enumerative method. The reason is that containers are not dropped off or picked up at dedicated I/O points at the end of the row. Some requests are connected to other I/O points, which means that by solving an AP, the number of subtours is not necessarily two but can be more.

The rest of this paper is organized as follows. In Section 2, we describe the technical aspects of the problem and present the mathematical model. In Section 3, the solution method is developed. Section 4 presents computational experiments and Section 5 contains the conclusions.

2. Problem description and model

This section describes the research problem, introduces notations, and then formulates the problem.

2.1. Problem description

We study how to sequence N container storage and retrieval requests in a single block of containers consisting of X rows, Y bays, Z tiers, and M I/O points. Let I/O_m, $m = 1,\ldots,S$, be the location of the mth I/O point at the seaside whereas I/O_m, $m = S+1,\ldots,M$, be the location of the mth I/O point at the landside. A single YC stacks n containers from the I/O points to given locations in the block and retrieves $N-n$ containers from the block to the I/O points. A storage container is a container that must be stacked in the block and a retrieval container is a container that must be retrieved from the block. The objective is to minimize the total travel time of the YC. From a given starting location, the YC can carry out the requests in any sequence in order to minimize total travel time. We denote by $0'$ and $0''$, respectively, the starting and ending locations of the YC. These locations can be anywhere in the block. The starting location is given, but the ending location depends on whether the request carried out last by the YC is a storage or a retrieval (see Table 1). The YC can only carry a single container at a time and containers cannot be dropped off at temporary locations.

We denote by $\mathcal{V} = \{1,\ldots,N,0',0''\}$ the set of all storage and retrieval locations including the YC starting and ending locations. Note that every request corresponds to a unique location in the block and a unique container. Thus, for ease of notation, we use their notations interchangeably as follows. Container i of storage request i is currently located at an I/O point and is waiting to be stored in location i in the block, $i = 1,\ldots,n$. Container j of retrieval request j is already stacked in location j in the block, $j = n+1,\ldots,N$, and is waiting to be delivered to an I/O point either at the seaside or landside, depending on whether they should be transported by truck/train, or by ship, respectively. Due to the abundance of internal transport vehicles (like automated guided vehicles (AGVs) and chassis) to pick up containers at the seaside and landside, retrieved containers can be delivered to any I/O point at the landside or the seaside.

Locations as well as destination sides of retrieval containers and I/O points of storage containers are known; in practice, they are determined at a higher hierarchical planning level. This level focuses on providing quick accessibility of containers for future handling (De Castillo & Daganzo, 1993; Kim, Park, & Ryu, 2000; Wan, Liu, & Tsai, 2009) and speeding up the loading and unloading process of a ship (Kim & Kim, 1999; Vis & Roodbergen, 2009). In addition, storage and retrieval locations do not coincide. These assumptions are in line with the literature (Li et al., 2009, 2012; Vis & Roodbergen, 2009; Vis & Carlo, 2010), and are also reasonable from a practical point of view. In practice, N is much smaller than the block size, and the block utilization is often fairly low. Considering these situations, while making the storage assignment decision, terminal operators separate retrieval and storage piles and avoid stacking a container on the top of another one that has to be retrieved in the same time period, as rehandling is time consuming. Storage containers are piled up based on type (i.e., weight, destination port, or size), strongly limiting the number of options. Finally, we assume that containers do not have explicit precedence constraints. In the introduction, it is already explained that container terminals usually use a block sequencing approach to carry out requests. Blocks are carried out one by one, but containers within each block are not prioritized and can be carried out in any order (see the case study in Section 4.2). Once there are more than two requests, we have a potential to shorten the travel time by optimization. If new requests arrive over time, they can be added to the list. Since our algorithm is fast, the algorithm can be called repeatedly to provide the optimal solution instantaneously.

Fig. 2 shows a top view of a small block ($X = 7$, $Y = 10$, and $Z = 4$) of containers with three storages and four retrievals. Locations of storages, retrievals and I/O points are highlighted with different

Table 1
Computation of pairwise travel times (t_{ij}).

Arc (i, j)[a]	Computation of t_{ij}		I/O
(storage location i, retrieval location j)	t_{ij}		None
(storage location i, storage location j)	$t_{i,\ I/O(j)} + t_{I/O(j),\ j}$		$I/O(j)$[b]
(storage location i, 0″)	0		None
(retrieval location i, retrieval location j)	$\min_{m=1,\ldots,S} \{t_{i,I/O_m} + t_{I/O_m j}\}$ (or $m = S+1,\ldots,M$)	If container i must be delivered to the seaside (or landside)	$\mathcal{P}$[c]
(retrieval location i, storage location j)	(1) $t_{i,\ I/O(j)} + t_{I/O(j),\ j}$	If container i must be delivered to the same side where container j is already located	$I/O(j)$
	(2) $\min_{m=1,\ldots,S} \{t_{i,I/O_m} + t_{I/O_m I/O(j)} + t_{I/O(j),j}\}$ (or $m = S+1,\ldots,M$)	If container i must be delivered to the seaside (or landside) and container j is already located at the opposite side	$\mathcal{P}$ and $I/O(j)$[d]
(retrieval location i, 0″)	$\min_{m=1,\ldots,S} \{t_{i,I/O_m}\}$ (or $m = S+1,\ldots,M$)	If container i must be delivered to the seaside (or landside)	$\mathcal{P}$
(0′, retrieval location j)	$t_{0',j}$		None
(0′, storage location j)	$t_{0',I/O(j)} + t_{I/O(j),j}$		$I/O(j)$
Other cases	∞		None

[a] In arc (i, j), the YC moves from location i to location j ($i \neq j$).
[b] Let $I/O(j)$ be the I/O point where storage container j, $j = 1,\ldots,n$, is located.
[c] Let $\mathcal{P}$ be the set of I/O points that the YC can deliver retrieval container i to either one and travel to another location with the minimum travel time.
[d] In this case, two I/O points must be visited: $I/O(j)$ and an I/O point in $\mathcal{P}$.

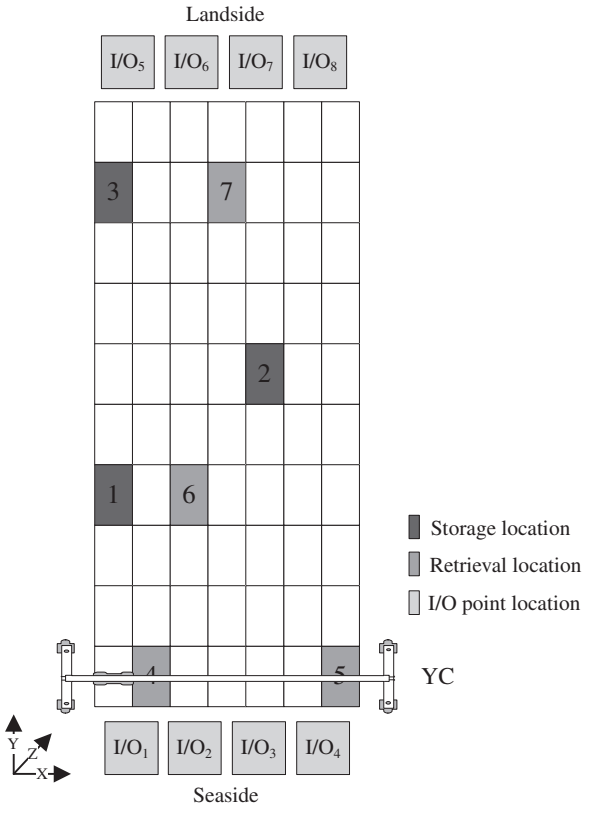


Fig. 2. A top view of a block with storage and retrieval locations.

colors. The origin of the Cartesian coordinate system is indicated at the bottom left corner (the tiers are counted from the top). The first storage container must be picked up at I/O_1, the second at I/O_1, and the third at I/O_7. On the other hand, retrieval container 4 must be moved to one of the I/O points at the seaside and containers 5, 6, and 7 must be moved to one of the I/O points at the landside.

In order to mathematically formulate the problem and calculate the total travel time of the YC, the pairwise travel times between the storage and retrieval locations in the block must be calculated. If the YC travels directly from location i, (x_i, y_i, z_i), to location j, (x_j, y_j, z_j), the travel time of the YC can be calculated as follows:

$$t_{ij} = \max\{|x_i - x_j|, |y_i - y_j|\} + |z_i| + |z_j|, \quad (1)$$

where (x_i, y_i, z_i) and (x_j, y_j, z_j) are the Cartesian coordinations of locations i and j, with $0 \leqslant x_i,\ x_j \leqslant T_x$, $0 \leqslant y_i,\ y_j \leqslant T_y$, and $0 \leqslant z_i,\ z_j \leqslant T_z$. Here, T_x, T_y, and T_z, are the furthest travel times of locations in the block in X, Y, and Z directions, respectively. The first term at the right-hand side of Eq. (1) emphasizes that the YC can move in the X and Y directions simultaneously. The vertical dimension travel time includes extra necessary processing times such as the time required to lock the spreader to the container or the time required to change the yard chassis or AGVs.

In order to calculate the travel times between storage and retrieval locations, we additionally must consider that the YC needs to travel through I/O points to pick up or deliver containers. For example, to move from storage location i to storage location j, the YC needs to travel first to the I/O point where container j is located and then move container j to location j. Table 1 summarizes how to calculate the travel times between different locations. It divides all feasible moves into three categories depending on whether the YC starts from a storage location, a retrieval location, or the starting location. All other moves (i.e., to move from a location to the starting location) are not feasible. Therefore, the travel times are forced to be infinite, as shown in the last row of the table, so that they do not appear in the solution. Furthermore, without loss of generality, we assume that the YC stops at location i, if storage request i is the last one. We also assume that the YC delivers the container j to an I/O point and stops there, if retrieval request j is the last request. Finally, in our problem, the travel time of the YC satisfies the triangle inequality, although this is not necessary in the ATSP.

The mathematical formulation of the problem as an ATSP requires a unique travel time between every two locations. However, in an arc in which the YC travels from a retrieval location to another location, the YC is allowed to deliver the retrieval container to any of the multiple I/O point options. Each possible I/O point results in a different travel time. The YC selects the I/O point which gives the minimum travel time between the retrieval location and the immediate next storage or retrieval location, and we define it as minimum pairwise travel time for each pair of locations. Note that multiple I/O points can result in the minimum travel time, and each can be selected to retrieve the container. Later in the solution method section, we explain how we should select an I/O point from multiple options.

2.2. Mathematical model

The problem of sequencing storage and retrieval requests is mathematically stated as the following integer programming model. Let x_{ij} be the binary decision variable which equals 1 if and only if location j is visited immediately after location i, $i,j \in \mathcal{V}, i \neq j$.

The objective function is to minimize the total travel time of the YC:

$$\text{minimize} \quad Z = \sum_{i \in \mathcal{V}\setminus\{j,0''\}} \sum_{j \in \mathcal{V}\setminus\{0'\}} t_{ij} x_{ij}. \tag{2}$$

The model has the following constraints.

Visiting all locations and carrying out all requests: Each location must be entered and exited exactly once. Of course, the starting location has no ingoing arc and the ending location has no outgoing arc:

$$\sum_{i \in \mathcal{V}\setminus\{j,0''\}} x_{ij} = 1, \quad \forall j \in \mathcal{V} \setminus \{0'\}, \tag{3}$$

$$\sum_{j \in \mathcal{V}\setminus\{i,0'\}} x_{ij} = 1, \quad \forall i \in \mathcal{V} \setminus \{0''\}. \tag{4}$$

Subtour elimination constraints: The YC must avoid traveling subtours. Therefore, for any subset of locations $\mathcal{O} \subseteq \mathcal{V} \setminus \{0', 0''\}$, $\mathcal{O} \neq \emptyset$ where $|\mathcal{O}|$ is the size of $\mathcal{O}$, we have the following constraints:

$$\sum_{i,j \in \mathcal{O}} x_{ij} \leqslant |\mathcal{O}| - 1. \tag{5}$$

All the variables are binary:

$$x_{ij} \in \{0,1\}, \quad \forall i,j \in \mathcal{V}. \tag{6}$$

The problem is defined as an ATSP (since the YC may not travel in both directions or the distances might be different), consisting of assignment constraints plus subtour elimination constraints. Therefore, every feasible solution of the YC scheduling problem corresponds to a feasible AP solution since by relaxing the subtour elimination constraints the remaining problem is an AP. This implies that the AP provides a lower bound for the problem (see also Bellmore & Nemhauser, 1968). Theorem 1 shows the complexity of a special case of the problem considered in this paper.

Theorem 1. *The YC scheduling problem with a prescribed I/O point for each retrieval request is $\mathcal{NP}$-hard.*

The proof can be found in Appendix A. In Section 3, we develop a solution method to quickly solve the problem to optimality.

3. Solution method

A relaxation of the problem is an AP that can be efficiently solved in $O(N^3)$ steps where N is the number of requests (Kuhn, 1955). An optimal AP solution often contains subtours since the subtour elimination constraints (Eq. (5)) may not be satisfied. In order to merge subtours to obtain an optimal solution of our problem, we first develop some unique properties of the problem. Using these properties, an algorithm is developed in Section 3.1 to merge subtours without adding extra travel time. If all subtours have been merged into a single tour, an optimal solution is found. Otherwise, some subtours have not been merged, and a B&B algorithm is developed in Section 3.2 to merge all remaining subtours to obtain an optimal solution of our problem.

3.1. Merging algorithm: using swappable arcs

In Section 3.1.1, we first explain how to patch subtours of an optimal AP solution. We merge all subtours in multiple steps by selecting two of them in each step and swapping arcs between them. We show that no extra travel time is added to the solution if the YC visits an I/O point twice while traveling from the origins to the destinations of the two arcs. Then, in Section 3.1.2, we introduce arcs in which the YC can visit multiple I/O point options while traveling from the origins to the destinations of the arcs. These arcs create opportunities to merge more subtours. Finally, in Section 3.1.3, we present the merging algorithm. We show that the objective value does not change after this merging.

3.1.1. Merging subtours based on an optimal AP solution

In this section, we find some properties of the problem to merge subtours of the AP solution without adding extra travel time.

Definition 1. *The visiting I/O point of an arc* is the I/O point that the YC needs to travel through either to pick up or to deliver a container in order to travel the arc connecting two requests.

In order to visualize the visiting I/O point of an arc, we need to distinguish carrying and non-carrying moves. In case of a retrieval request, the YC starts with a non-carrying move from a storage location (or an I/O point) to the retrieval location, and then continues with a carrying move from the retrieval location to an I/O point. In case of a storage request, the YC starts with a carrying move from an I/O point to the storage location, and then continues with a non-carrying move from the storage location to an I/O point or retrieval location. Note that the travel time from or to such I/O points is explicitly included in Table 1.

Fig. 3b shows carrying and non-carrying moves of the arcs of an optimal AP solution presented in Fig. 3a. For clarity, in Fig. 3b, only the first and second subtours are considered. For example, storage request 1 is connected to I/O_1 by a carrying move and to retrieval request 4 by a non-carrying move. Furthermore, retrieval request 4 is connected to I/O_1 by a carrying move and to storage request 1 by a non-carrying move. As a result, I/O_1 is the visiting I/O point of arc $(4,1)$.

If (i, j) and (k, l) are two arcs of two subtours ($x_{ij} = x_{kl} = 1$), the two subtours can be merged by setting $x_{ij} = x_{kl} = 0$ and $x_{il} = x_{kj} = 1$. Theorem 2 states that if arcs are swappable, as defined in the following, the total travel time remains the same. Otherwise, it may increase.

Definition 2. *Swappable arcs* are two arcs in two different subtours that have a common visiting I/O point.

Theorem 2. *If two subtours contain a pair of swappable arcs, they can be merged without adding extra travel time.*

The proof can be found in Appendix B.

In the optimal AP solution presented in Fig. 3a, (4, 1) and (0′, 2) define a pair of swappable arcs because both arcs visit I/O_1 (follow the carrying moves in Fig. 3b). Therefore, based on Theorem 2, replacing them with (4, 2) and (0′, 1) results in merging subtours 1 and 2 without adding extra travel time. Merged subtour 1 + 2 is shown in Fig. 3c: $0' \overset{x_{0'1}=1}{\rightarrow} 1 \overset{x_{14}=1}{\rightarrow} 4 \overset{x_{42}=1}{\rightarrow} 2 \overset{x_{25}=1}{\rightarrow} 5 \overset{x_{50''}=1}{\rightarrow} 0''$.

3.1.2. Merging subtours using replaceable I/O points

In the previous section, we used the following type of swappable arcs in Theorem 2 to patch subtours of an optimal AP solution.

Definition 3. *An arc with a single irreplaceable visiting I/O point* is an arc in which the YC has a unique visiting I/O point option with the minimum travel time to visit and travel from the origin to the destination of the arc.

The assumption that all arcs have a single irreplaceable visiting I/O point results in missing some merging opportunities. As the last column of Table 1 shows, in some arcs, the YC may be able to visit

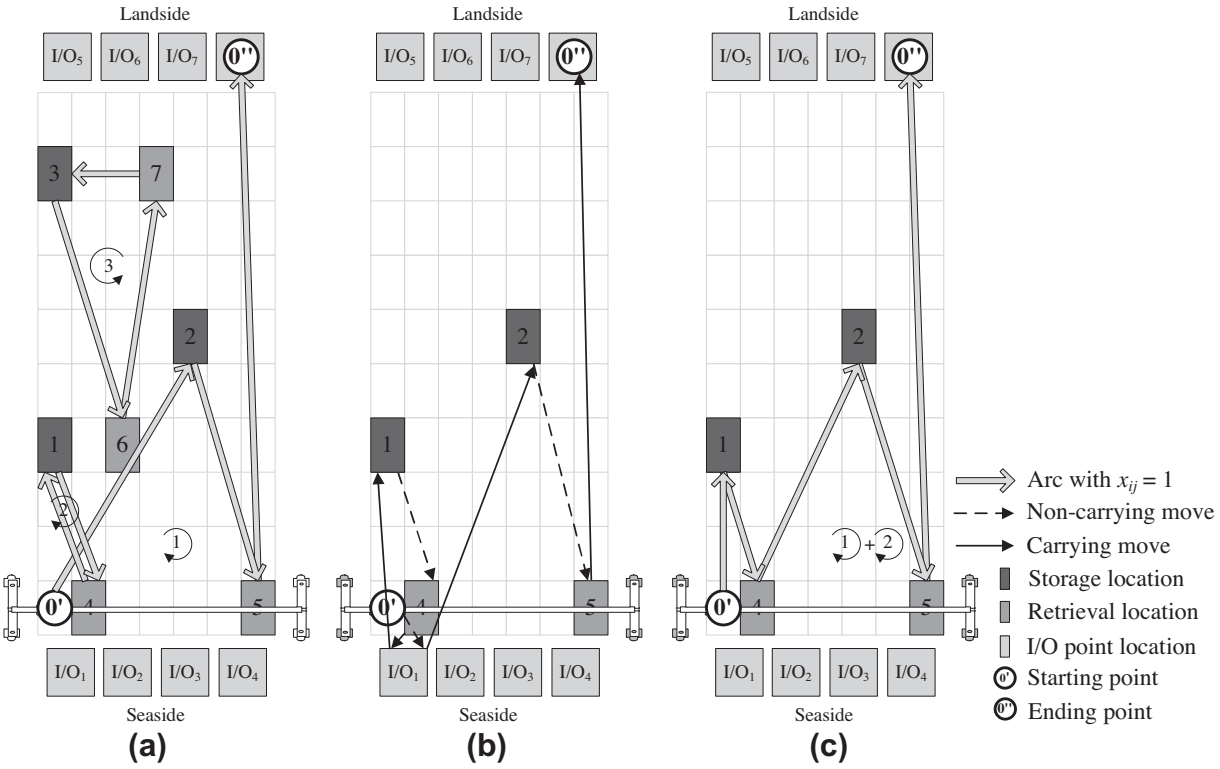


Fig. 3. Merging subtours using swappable arcs. *Note:* (a) An optimal AP solution consisting of $0' \overset{x_{02}=1}{\rightarrow} 2 \overset{x_{25}=1}{\rightarrow} 5 \overset{x_{50''}=1}{\rightarrow} 0''$; *subtour 2*: $1 \overset{x_{14}=1}{\rightarrow} 4 \overset{x_{41}=1}{\rightarrow} 1$; *subtour 3*: $3 \overset{x_{36}=1}{\rightarrow} 6 \overset{x_{67}=1}{\rightarrow} 7 \overset{x_{73}=1}{\rightarrow} 3$. (b) Carrying and non-carrying moves of subtours 1 and 2, (c) Merging subtours 1 and 2 using the swappable arcs (4, 1) and (0′, 2).

one of multiple I/O points with the same travel time. Arcs without any I/O point such as moving from a storage location to a retrieval location are not a matter of attention here because swappable arcs need to have a common visiting I/O points. Therefore, we can introduce the following type of arcs.

Definition 4. *An arc with replaceable visiting I/O points* is an arc in which the YC has multiple visiting I/O point options with the minimum travel time to visit and travel from the origin to the destination of the arc.

Based on the last column of Table 1, an arc with replaceable visiting I/O points can be realized if the request corresponding to the location at the origin of an arc is a retrieval request. The retrieved container must be delivered to an I/O point which results in the minimum travel time for the YC in order to travel from the origin to the destination of the arc. In some cases, multiple I/O points may give the same minimum travel time one of which can be chosen. Arcs with replaceable visiting I/O points create extra opportunities to merge subtours.

During the merging algorithm, we check each arc to find out the number of visiting I/O points. It is clear from Table 1 that arc (i, j) has a single irreplaceable visiting I/O point in these two cases (the other arcs may have replaceable or irreplaceable I/O points): (1) requests i and j are both storage requests or (2) request i is a retrieval request, request j is a storage request. In these arcs, the single irreplaceable visiting I/O point is the I/O point where storage container j is located. As an example, arc (4, 1) in Fig. 3a shows an arc with a single irreplaceable visiting I/O point, as I/O_1 is the only I/O point that can be visited (see Fig. 3b). In Fig. 4a where subtours 1 + 2 and 3 are shown, arc (5, 0″) is an arc with four replaceable visiting I/O points, I/O_m, $m = 5, 6, 7, 8$, that give the same minimum travel time, $t_{50''}$. Arc (6, 7) is also an arc with replaceable visiting I/O points, I/O_m, $m = 6, 7$. In Fig. 4b, carrying and non-carrying moves of arcs (5, 0″) and (6, 7) are shown in black and gray, respectively.

Based on Theorem 2, if two subtours contain a pair of swappable arcs of which at least one has an irreplaceable visiting I/O point, the subtours can be merged without extra travel time by using that single common visiting I/O point. In a more general case, if two subtours contain a pair of swappable arcs, both with replaceable visiting I/O points, the subtours can be merged without extra travel time using any of the common visiting I/O points. An example is presented in Fig. 4. Swappable arcs of subtours 1 + 2 and 3 are (6, 7) and (5, 0″), as shown in Fig. 4a. The replaceable visiting I/O points of these arcs are shown in Fig. 4b. Common visiting I/O points, I/O_m, $m = 6, 7$, can be used to merge subtours 1 + 2 and 3. A complete tour is shown in Fig. 4c. In this solution, I/O_6 is for example selected to deliver the retrieved container 6.

3.1.3. Merging algorithm steps

Having considered different properties of the problem, the merging algorithm can be presented. In the algorithm, merging subtours will be carried out by using the following arcs consecutively: (1) swappable arcs with a single irreplaceable visiting I/O point, (2) swappable arcs with replaceable visiting I/O points, and (3) swappable arcs of which one has a single irreplaceable visiting I/O point and the other has replaceable visiting I/O points. The reason behind this sequence is as follows:

In case both swappable arcs have a single irreplaceable visiting I/O point, merging two subtours results in a merged subtour in which the same common visiting I/O point will be visited by the new arcs replacing the swappable arcs. This is the ideal case because the subtours are merged without omitting any I/O point. In the other two cases, only common visiting I/O points remain. Other visiting I/O points are omitted and we lose the opportunity to use them for further merging, if necessary. In the second case, arcs replacing the swappable arcs can visit all common visiting I/O points, whereas in the third case, they can visit only a single irreplaceable common visiting I/O point. In other words, in the second

case, all common visiting I/O points can be used to patch the merged subtour to other subtours, whereas in the third case, only a single common visiting I/O point can be used for further merging. Therefore, the second case has priority in subtour merging over the third case.

In Algorithm 1, $\mathcal{U}_m$ and $\mathcal{U}'_m$ are the sets of all arcs in an optimal AP solution with irreplaceable and replaceable visiting I/O points respectively, which visit I/O_m, $m = 1,\ldots,M$.

Algorithm 1. Merging algorithm

Require: An optimal AP solution;
Ensure: An optimal solution of the problem with the same optimal AP objective value and a complete Hamiltonian tour, or an optimal AP solution with the same optimal AP objective value and a few subtours;
1: **procedure** MERGE
2: classify and assign arcs of the AP optimal solution to the sets of arcs with irreplaceable and replaceable visiting I/O points, $\mathcal{U}_m$ and $\mathcal{U}'_m$, $m = 1,\ldots,M$ using Definitions 3 and 4;
3: **for** $m = 1,\ldots,M$ **do**
4: **if** $|\mathcal{U}_m| > 1$ **then**
5: subtours can be merged two by two using the common I/O point, I/O_m;
6: update $\mathcal{U}_m$ and $\mathcal{U}'_m$, $m = 1,\ldots,M$;
7: **end If**
8: **end for**
9: repeat steps 3–8 for $\mathcal{U}'_m$. If $|\mathcal{U}'_m| > 1$, $m = 1,\ldots,M$;
10: repeat step 3–8 for $\mathcal{U}_m \cup \mathcal{U}'_m$. If $|\mathcal{U}_m \cup \mathcal{U}'_m| > 1$, $m = 1,\ldots,M$;
11: **if** the number of subtours is 1 **then**
12: an optimal solution of the problem is obtained; output the solution;
13: **else**
14: an optimal solution is not found because it does not satisfy Eq. (5). The B&B algorithm presented in Section 3.2 will be called to find an optimal solution;
15: **end if**
16: **end procedure**

The merging algorithm runs in polynomial time. The complexity of the algorithm comes from identifying each type of arc. In each step, it needs to go through all the arcs and decide whether they have replaceable or irreplaceable I/O points.

Theorem 3. *If the output of the merging algorithm is a complete tour, an optimal solution of the problem is obtained.*

The proof is straightforward as we solve an AP and patch subtours without adding any extra travel time (see Theorem 2). Since every feasible solution of the YC scheduling problem corresponds to a feasible AP solution, the AP provides a lower bound for the problem. Therefore, if the merging algorithm patches all subtours of an optimal AP solution as a complete tour, then the optimal solution of our problem is obtained.

The merging algorithm does not change any of the inputs of the problem and it does not add or remove any constraints (see Sections 3.1.1–3.1.3). In fact, it is a method to change an optimal AP solution to another optimal AP solution (possibly satisfying the subtour elimination constraints) by exchanging some of the arcs.

3.2. Branch-and-bound algorithm

If, upon completion of the merging algorithm, the solution still contains at least two subtours, a further step is needed. This section introduces a modified ATSP B&B algorithm to merge all subtours as a complete Hamiltonian tour with the minimum total travel time.

The classic ATSP B&B algorithm is based on the AP relaxation and subtour elimination. It has been shown that an optimal ATSP solution can be obtained by starting from an optimal AP solution with multiple subtours in Node 0 of the B&B tree, exhaustively branching on all the arcs in a chosen subtour, solving a new AP in each of the resulting nodes, and repeating the same procedure until all nodes are fathomed (see the overview by Laporte (1992), the first implementation of the algorithm by Eastman (1958), or the recent ones by Carpaneto and Toth (1980) and Carpaneto et al. (1995)).

To solve our YC scheduling problem, formulated as an ATSP, we start from an optimal AP solution (the solution of the first phase of the algorithm), and extend the B&B algorithm by using our merging algorithm to patch subtours of the optimal AP solution in each node. The AP usually has multiple optimal solutions. Carpaneto et al. (1995) explain that any of them may be used to bound the node. They further mention that the optimal solution with the minimum number of subtours generally leads to the minimum computation effort later on in the B&B tree. Therefore, they propose a heuristic algorithm to merge subtours in each node and find another optimal AP solution with the minimum or possibly no subtours (see the discussion in the introduction). Our merging algorithm is a new method to patch subtours of an optimal AP solution to obtain another optimal AP solution with the same optimal value but fewer or no subtours. Note that in the first phase of our algorithm we also solve an AP and try to merge all subtours. Therefore, the solution (with multiple subtours) can serve as the input in node 0. In fact, we separate the first phase because it provides an optimal solution in almost all instances (see the computational experiments). In the following, it is explained in detail how the B&B algorithm proceeds until the optimal solution is obtained.

3.2.1. Basic branch-and-bound algorithm

Step 1. Initialization

The output of the merging algorithm is an optimal AP solution with $|\mathcal{W}|$ subtours, where $\mathcal{W}$ is the set of subtours. The output serves as the input of the B&B algorithm in node 0. In addition, determine the best solution found so far X^*, with the objective value Z^* (which serves as an upper bound for the B&B algorithm) by using a heuristic algorithm such as the random or NN algorithm.

Step 2. First branching

The B&B algorithm continues by selecting a subtour $w \in \mathcal{W}$ containing the minimum number of arcs to branch node 0 (Carpaneto & Toth, 1980). The branches divide the solution space into complementary solution subspaces such that w can be removed. This is carried out by sequentially excluding a next arc of the (now broken) subtour w and including (keeping) the preceding arcs. The excluding arcs must disappear in the next solution and including arcs must appear. Arcs can be excluded and included by assigning 0 and 1 to their corresponding variables, respectively. Assume subtour w can in general be represented as $[1] \overset{x_{[1][2]}=1}{\rightarrow} [2] \rightarrow \cdots \rightarrow [K] \overset{x_{[K][1]}=1}{\rightarrow} [1]$, where $[i]$ is the ith location, $i = 1,\ldots,K$, that will be visited by the YC in this subtour. Subtour w results in K child nodes. As shown in Eqs. (8) and (7), in each node a set $\mathcal{E}_j$ of arcs is excluded and a set $\mathcal{I}_j$ of arcs is included in the solution for some j, where $j = 1,\ldots,K$. More specifically, in child node j, we exclude arc $([j],[j+1])$. In other words, $\mathcal{E}_j = \{([j],[j+1])\}$, and we have to set $x_{[j][j+1]}$ to zero. Furthermore, all the arcs before this arc in subtour w have to appear in the solution of child node j. This means that $\mathcal{I}_j = \{([1],[2]),\ldots,([j-2],[j-1])\}$, and associated variables, $x_{[1][2]},\ldots,x_{[j-2][j-1]}$, have to be set to one.

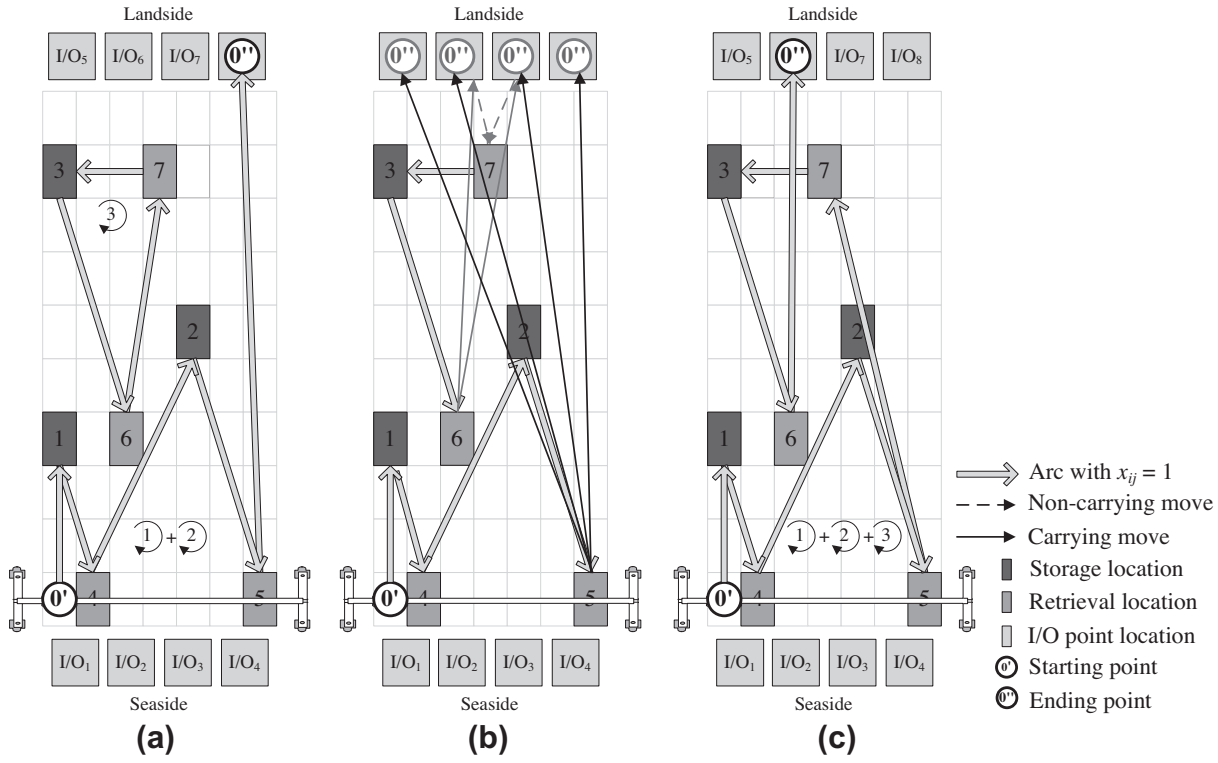


Fig. 4. Merging subtours using swappable arcs with replaceable visiting I/O points. *Note:* (a) Subtour 1 + 2 and subtour 3, (b) carrying and non-carrying moves of arcs (5, 0″) and (6, 7) with replaceable visiting I/O points in subtours 1 + 2 and 3, (c) merging subtours 1 + 2 and 3 by replacing $x_{50''} = x_{67} = 1$ with $x_{57} = x_{60''} = 1$.

$$\mathcal{E}_j = \{([j],[j+1])\}, \quad j = 1,\ldots,K, \tag{7}$$

$$\mathcal{I}_j = \{([1],[2]),\ldots,([j-2],[j-1])\}, \quad j = 2,\ldots,K, \tag{8}$$

where $\mathcal{I}_1 = \emptyset$, and $[K+1] = 1$ in Eq. (7).

Later, in the proof of Theorem 4, we show that such a branching strategy results in an optimal solution. The subspaces are collectively exhaustive with respect to feasible ATSP tours. The branching rule divides the solution space into complimentary subspaces such that in none of them the same subtour (selected from the parent node) appears since one arc of that subtour is excluded in each child node.

Step 3. Fathoming and bounding

At each child node, $l = 1,\ldots,K$, we solve a modified AP, introduced by considering constraints forced by Eqs. (7) and (8) as well as other general constraints of the AP (Eqs. (2)–(4) and (6)).

Assume Z^l_{AP}, and X^l, $l = 1,\ldots,K$, are the optimal AP objective value and an optimal AP solution at node l. If $Z^l_{AP} \geqslant Z^*$, fathom the node. If $Z^l_{AP} < Z^*$ and the optimal solution to the modified AP at node l is a complete Hamiltonian tour, update $Z^* = Z^l_{AP}$, $X^* = X^l$ and fathom the node. Otherwise, if $Z^l_{AP} < Z^*$ and the optimal solution is not a complete tour, add l to the set of live nodes, $\mathcal{L}$ ($\mathcal{L} = \emptyset$ at node 0). The set of live nodes is the set of nodes that are not fathomed yet and must be considered.

Step 4. Further branching

Choose one of the live nodes from $\mathcal{L}$ to be evaluated next. In our implementation, this is carried out by using the depth-first-search (DFS) as the overall strategy and the best-first-search (BeFS) when a choice is to be made between nodes at the same level of the tree (see Clausen, 2003). In case several nodes can be chosen, choose one randomly. Furthermore, if $\mathcal{L}$ is empty, X^* is an optimal solution and Z^* is the optimal value.

Assume node $\hat{l} \in \mathcal{L}$ is selected. If $Z^{\hat{l}}_{AP} \geqslant Z^*$, omit this node from the set and choose another one; otherwise, find out which subtour must be split. We follow Carpaneto and Toth (1980), who propose to select the subtour with the minimum number of non-included arcs to be split. A non-included arc of a subtour is an arc that is not forced by $\mathcal{I}_{\hat{l}}$ to appear in the optimal AP solution of node $\hat{l}$. The subtour with the minimum number of non-included arcs generates the minimum number of child nodes from node $\hat{l}$. Assume $\hat{l}$, has Q subtours where $\mathcal{G}_q$ is the set of arcs in the qth subtour. As a result, subtour $\bar{q}$ will be chosen to be split if $M = |\mathcal{G}_{\bar{q}}| - |\mathcal{G}_{\bar{q}} \cap \mathcal{I}_{\hat{l}}| = \min_{q=1,\ldots,Q}\{|\mathcal{G}_q| - |\mathcal{G}_q \cap \mathcal{I}_{\hat{l}}|\}$, where $|\mathcal{G}_q \cap \mathcal{I}_{\hat{l}}|$ is the number of arcs showing up both in the qth subtour and in the set of included arcs of node $\hat{l}$.

In order to break subtour $\bar{q}$, we use the same branching rule explained in Eq. (8) and (7). However, instead of considering all arcs in subtour $\bar{q}$, we only consider the non-included arcs because we cannot include or exclude a previously-included arc. Consider that subtour $\bar{q}$ can be represented as $[1] \overset{x_{[1][2]}=1}{\rightarrow} [2] \rightarrow \cdots \rightarrow [K] \overset{x_{[K][1]}=1}{\rightarrow} [1]$ of which the set of non-included in total contains C arcs, $\{([1o],[1d]),\ldots,([Co],[Cd])\}$, where $[io]$ and $[id]$ are the origin and destination of the ith non-included arc, $i = 1,\ldots,C$, respectively. The sequence of arcs in the set of non-included arcs can be determined randomly. Node $\hat{l}$ results in C child nodes, each with the corresponding sets of excluded and included arcs as follows:

$$\mathcal{E}_{B+j} = \mathcal{E}_{\hat{l}} \cup \{([jo],[jd])\}, \quad j = 1,\ldots,C, \tag{9}$$

$$\mathcal{I}_{B+j} = \mathcal{I}_{\hat{l}} \cup \{([1o],[1d]),\ldots,([j-1,o],[j-1,d])\}, \quad j = 2,\ldots,C, \tag{10}$$

where $\mathcal{I}_{B+1} = \mathcal{I}_{\hat{l}} \cup \emptyset$ and B is the total number of nodes in the B&B tree before generating the new child nodes.

The same fathoming and bounding strategy discussed in step 3 for the child nodes of node 0 applies here. In each child node, an optimal solution of the modified AP and the best solution found

so far determine whether (1) the child node must be fathomed because the optimal AP solution is worse than the best solution found, or (2) the child node must be fathomed because a better solution is found, or (3) the child node must be added to $\mathcal{L}$. Next, based on the branching strategy, another node will be chosen and the algorithm will repeat until the set of live nodes is empty, $\mathcal{L} = \emptyset$.

Fig. 5 shows the steps of the B&B algorithm. Note that using the merging algorithm, we can merge all subtours of the AP relaxation of the problem presented in Fig. 2 as a single tour in Fig. 4c. However, assume that the problem has two more requests: retrieval request 9 and storage request 8. These requests cause an extra subtour which cannot be merged with the other ones because its associated arc do not have any common visiting I/O point with the arcs of other subtours. Therefore, the outcome of the merging algorithm would be the solution presented at node 0. The B&B algorithm chooses subtour 4 for branching because it has only two arcs whereas the other one has eight arcs.

As it is shown in Fig. 5, solving the modified APs at child nodes 1 and 2 results in solutions each consisting of 2 subtours, namely subtours 6 and 7 for $l = 1$, and subtours 8 and 9 for $l = 2$. However, $Z^2_{AP} = 281.46 \geqslant 280.02 = Z^* = Z_{NN}$, so node 2 must be fathomed. Note that X^* and Z^* are obtained by the NN heuristic. On the other hand, $Z^1_{AP} = 277.96 < 280.02 = Z^* = Z_{NN}$. Therefore, node 1 will be added to the set of live nodes and the B&B algorithm continues as shown in Fig. 6a. The B&B algorithm must evaluate four more nodes (seven nodes in total), nodes 3, 4, 5, and 6 which are the child nodes of node 1, to find an optimal solution.

3.2.2. Modified branch-and-bound algorithm

In Step 4, after selecting the next live node, if $Z^{\hat{l}}_{AP} < Z^*$, $\hat{l} \in \mathcal{L}$, the B&B algorithm chooses a subtour to be split and generates the

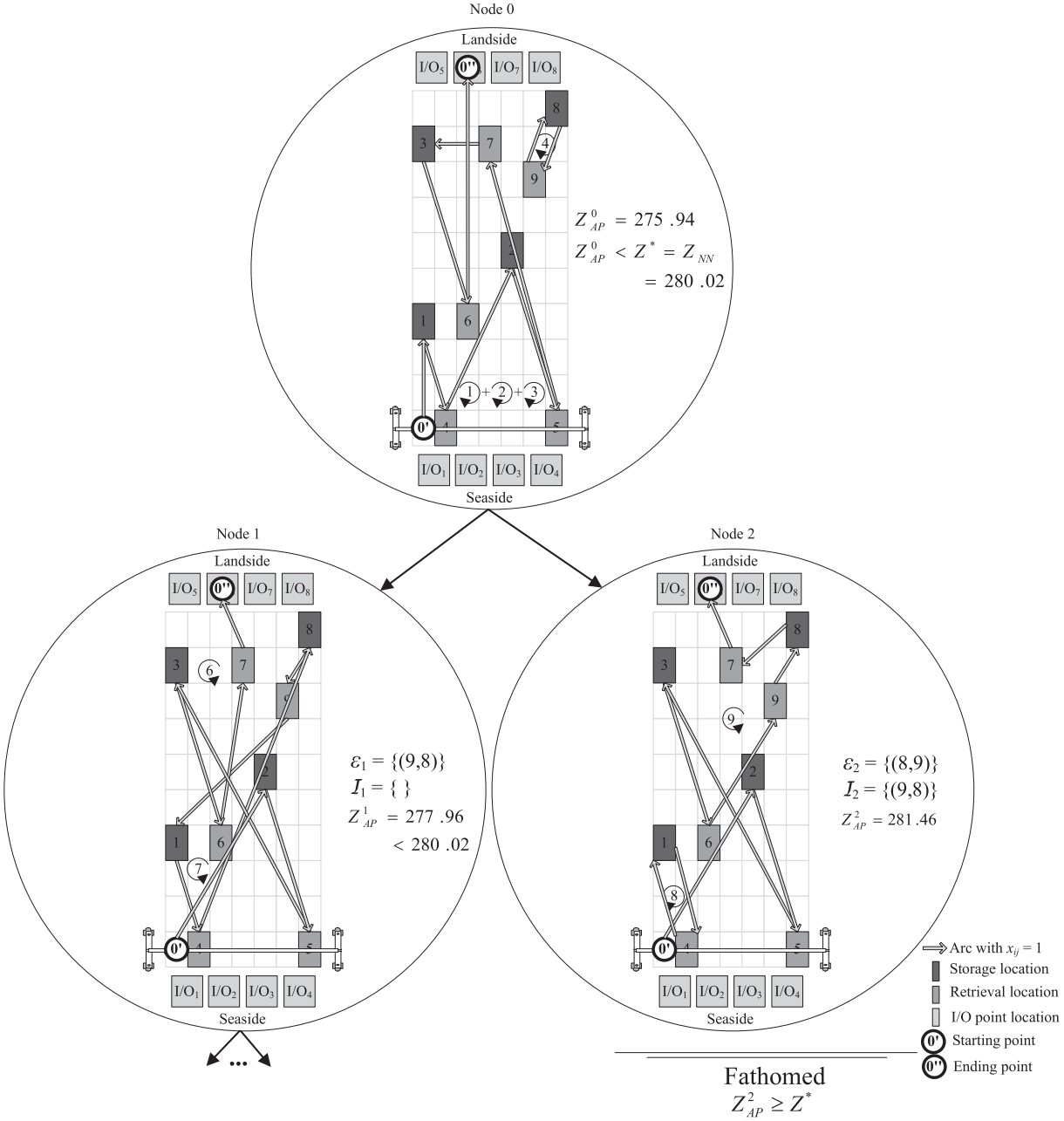


Fig. 5. The basic B&B algorithm.

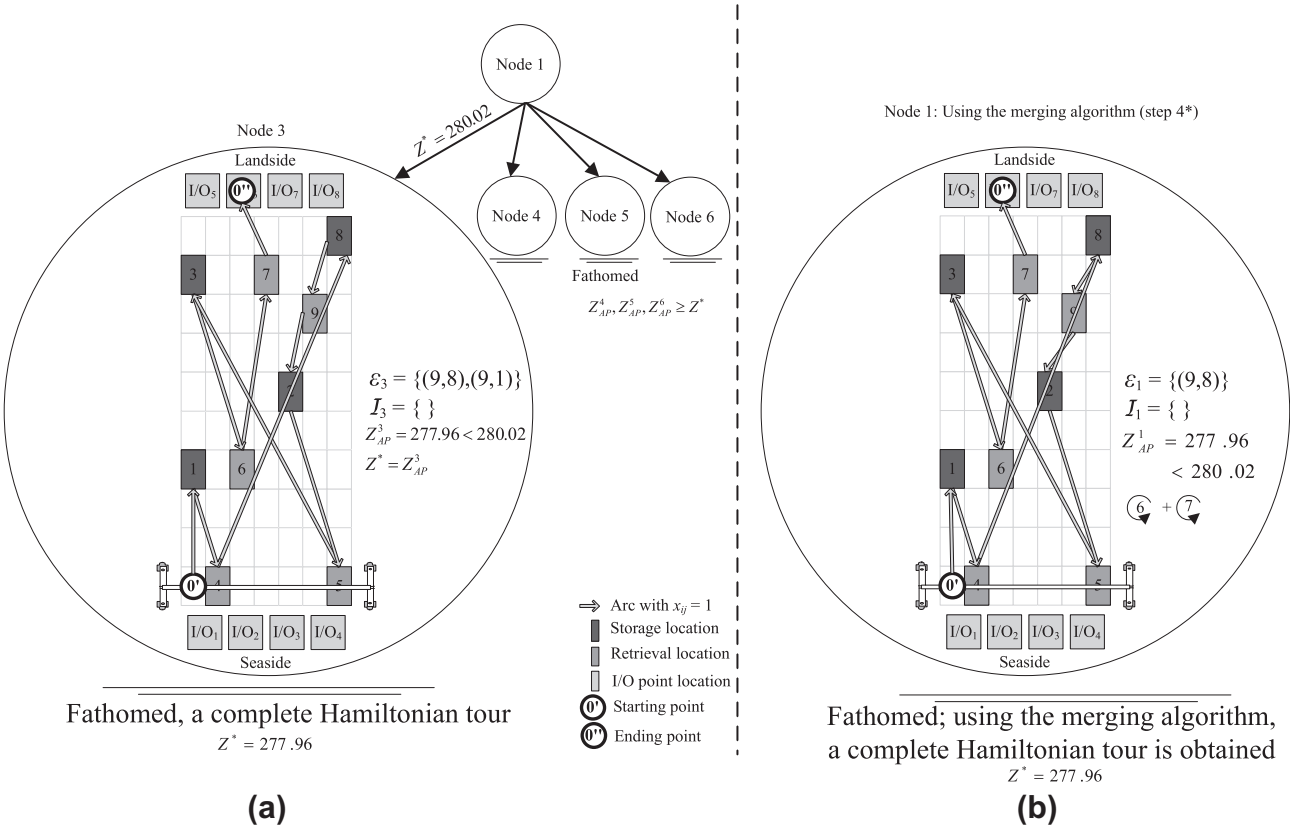


Fig. 6. (a) The child nodes of node 1 generated by the basic B&B algorithm and (b) fathoming node 1 by the modified B&B algorithm.

branches. However, in this section, before choosing a subtour and branching, we introduce an extra step (denoted by Step 4*) which tries to patch the subtours by using the merging algorithm. The smaller the number of subtours is, the fewer nodes will be produced afterwards in the B&B tree. Furthermore, we may also obtain a complete tour which fathoms the node and makes further branching unnecessary. As a result, the number of nodes of the B&B tree decreases significantly.

Step 4. Merging algorithm.* Merge subtours of the optimal AP solution in the selected live node using the merging algorithm. If the output of the merging algorithm is a complete Hamiltonian tour update $Z^* = Z^i_{AP}$, $X^* = X_i$, fathom the node and select another live node. Otherwise, choose a subtour and branch the node.

Fig. 6 is used to explain Step 4* of the algorithm. Fig. 6a shows that the basic B&B algorithm find an optimal solution in seven nodes. On the other hand, Fig. 6b shows that no extra branching is necessary if the modified B&B algorithm is used (3 nodes in total). As a result, computation time is reduced. This is due to the fact that, in Step 4*, using the merging algorithm, all subtours of node 1 are merged and therefore, it must be fathomed. Furthermore, in this specific case, X^1 is also an optimal solution with the optimal value Z^* = 277.96.

Theorem 4. *The two phase algorithm provides an optimal solution to the problem.*

The proof can be found in Appendix C.

4. Computational experiments

Multiple numerical experiments have been performed to investigate the effectiveness of the two-phase solution method consisting of the merging algorithm and the modified B&B algorithm, including validation at a real terminal. In the basic scenario, we consider a single block of containers with 30 bays, 10 rows, four tiers, and 10 I/O points. Furthermore, 100 requests are considered of which the percentage of retrieval requests, $\mathbb{P}$, is 50%. At large container terminals, the inbound and outbound flows of a container block are usually reasonably balanced. Other input data can be found in Table 2. The two-phase solution method is used to optimally sequence all the requests.

4.1. Evaluating the performance of the two-phase solution method

In this section, we evaluate the performance of the two-phase solution method. In Table 3, the results of the two-phase solution method are presented. In Table 4, our solution method is compared with the random and NN heuristics. In each scenario of the simulation study, 100 realizations of N requests with their storage and retrieval locations and corresponding I/O points are randomly generated by Monte Carlo simulation. The number of realizations

Table 2
Inputs of the simulation study.

Variable	Value
Gantry speed of the YC[a]	240 m/min
Trolley speed of the YC	60 m/min
Hoisting speed of the YC	72 m/min
Container length[b,c]	5.89 m
Container width	2.33 m
Container height	2.38 m

[a] No acceleration or deceleration time is considered.
[b] All containers are 20 feet long.
[c] No separating space between containers is considered.

Table 3
The results of the solution method.

X	Y	N	$\mathbb{P}$ (%)	M	Computation time (seconds)			Number of nodes			$\frac{Z^*_{AP}}{Z^*}$	Z^* (seconds)	ρ
					Ave	Max	Min	Ave	Max	Min			
10	30	**20**	50	10	0.0177	0.1353	0.0049	3.88	39	1	0.999923	976.4222	0.73
10	30	**50**	50	10	0.0397	0.171	0.0256	1.32	9	1	0.999995	2325.709	0.93
10	30	**100**	50	10	0.1321	0.3936	0.1072	1.04	5	1	1	4650.213	0.99
10	30	**150**	50	10	0.2973	0.3377	0.2397	1	1	1	1	6965.121	1
10	30	**200**	50	10	0.5462	0.7231	0.4062	1	1	1	1	9190.737	1
5	30	100	50	10	0.1295	0.1554	0.1038	1	1	1	1	4595.805	1
6	30	100	50	10	0.1315	0.4104	0.0999	1.04	5	1	1	4605.67	0.99
7	30	100	50	10	0.1419	0.6872	0.1034	1.16	9	1	1	4551.678	0.97
8	30	100	50	10	0.1353	0.413	0.1086	1.08	5	1	1	4609.479	0.98
9	30	100	50	10	0.1312	0.3934	0.0968	1.04	5	1	1	4612.963	0.99
10	**20**	100	50	10	0.112	0.165	0.0851	1	1	1	1	3788.589	1
10	**40**	100	50	10	0.1215	0.3179	0.094	1.04	5	1	1	5410.98	0.99
10	**50**	100	50	10	0.1272	0.4288	0.0931	1.08	5	1	1	6360.285	0.98
10	30	100	50	**2**	0.1237	0.1593	0.0878	1	1	1	1	4760.263	1
10	30	100	50	**8**	0.1301	0.1524	0.1123	1	1	1	1	4633.471	1
10	30	100	50	**14**	0.1415	0.4124	0.0923	1.2	5	1	1	4592.33	0.95
10	30	100	50	**20**	0.139	0.4238	0.1054	1.2	5	1	0.999998	4645.63	0.95
10	30	100	**0**	10	0.1147	0.3789	0.0978	1	1	1	1	6130.303	1
10	30	100	**20**	10	0.1401	0.248	0.1113	1	1	1	1	5330.784	1
10	30	100	**40**	10	0.1324	0.2342	0.1116	1	1	1	1	4672.246	1
10	30	100	**60**	10	0.1068	0.8279	0.0822	1.12	13	1	1	4800.1	0.99
10	30	100	**80**	10	0.0952	0.137	0.0809	1	1	1	1	5436.168	1
10	30	100	**100**	10	0.0942	0.1807	0.0781	1	1	1	1	6197.658	1
10	30	100	50	**50**	0.1976	1.1485	0.093	2.44	17	1	1	4595.972	0.75
10	30	100	50	**100**	0.2034	1.3404	0.0865	2.62	21	1	1	4624.975	0.79
10	30	100	50	**150**	0.3221	4.5623	0.084	4.88	83	1	1	4633.177	0.68
10	30	100	50	**200**	0.4459	3.8641	0.0761	7.02	75	1	0.999999	4620.406	0.57

Notes: We denote by Ave, Max, and Min the average, maximum and minimum computation time and number of nodes of the two-phase solution method. The table also presents the ratio of the objective value of the optimal AP solution (obtained from the original travel time matrix) to the objective value of the optimal solution (Z^*_{AP}/Z^*), the total travel time of the YC (Z^*), and the ratio of the realizations optimally solved in the first phase of the solution method (ρ). The other notations are explained in the text.

satisfies $N_{\text{realz.}} \geqslant \sigma^2 \left(\frac{(1+\varepsilon)\mathbb{Z}_{1-\alpha/2}}{\varepsilon\mu}\right)^2$ with a 90% confidence level ($\alpha = 10\%$), where σ^2 and μ are respectively the variance and mean of the objective values, $\mathbb{Z}_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of the normal distribution, and $\varepsilon = 5\%$ is the relative error (Law & Kelton, 1999). The study is performed on a Notebook with 2.40 gigahertz Intel® Core™ i5 processor, with 4.00 gigabytes of RAM and the programming language is MATLAB® 2010a.

Based on the results presented in Tables 3 and 4, the following insights can be obtained:

- The results in Table 3 show that the two-phase solution method can find the optimal solution of the problem in less than a second in all scenarios. The reason is that in majority of scenarios, the problem is completely solved in the first phase using the efficient merging algorithm, and the B&B algorithm is seldom recalled. This can be seen in the column presenting ρ, which is the ratio of the problems solved in the first phase of the solution method. When $\rho = 1$, the problem is completely solved in phase 1 and we do not need to enter phase 2. The same conclusion can be obtained from the columns showing the average, maximum and minimum number of nodes needed in each scenario. In most of the scenarios, the average number of nodes of the B&B tree is close to 1.
  The merging algorithm also performs well for large instances. The reason is that as the number of requests N increases, the number of arcs visiting each I/O point increases, and consequently the number of swappable arcs increases. Therefore, most of the subtours can be merged in the first phase and the second phase becomes unnecessary.
- The algorithm is robust. From Table 3 it is clear that for a fixed number of requests, the computation time and number of nodes of the two-phase solution method are quite insensitive to changes in problem inputs such as: number of bays, rows, requests and percentage of retrieval requests in different scenarios. The reason is that the performance of the solution method depends on the performance of the merging algorithm which mainly depends on the ratio of the number of requests to the number of I/O points. When the number of requests is 100 and the number of I/O points is less than 10, the ratio of the requests to I/O points is high enough (even in case of 10 I/O points, on average 10 containers are picked up or delivered at each I/O point) for the merging algorithm in order to be able to patch all subtours in the first phase of the algorithm. The computation time of the first phase of the solution method is about the same for all scenarios with a fixed number of requests.
- The last four rows of Table 3 investigate theoretical cases with 50 or more I/O points, in order to find out the effect of the number of I/O points on the computation time complexity. By increasing the number of I/O points the probability of merging subtours in the first phase decreases. As a result, ρ, the computation time and the number of nodes increases. However, in this problem, when the number of I/O points is 50, each I/O point is on average shared by 2 requests ($N = 100$). Therefore, it is still probable that the subtours can be merged in the first phase using swappable arcs with common visiting I/O points. Furthermore, we

Table 4
Comparing the random and NN heuristics with the optimal solution.

X	Y	N	$\mathbb{P}$ (%)	M	Z^*	Z_{rand}[a]	Z_{NN}[b]	G_{rand} (%)[c]	G_{NN} (%)[d]
10	30	**20**	50	10	976.42	1367.03	1113.1	28.57	12.28
10	30	**50**	50	10	2325.7	3416.29	2730.6	31.92	14.83
10	30	**100**	50	10	4650.2	6841.4	5406.32	32.03	13.99
10	30	**150**	50	10	6965.1	10269.3	8006.24	32.18	13.00
10	30	**200**	50	10	9190.7	13727	10692.1	33.05	14.04
5	30	100	50	10	4595.8	6750.53	5238.2	31.92	12.26
6	30	100	50	10	4605.7	6762.95	5288.25	31.90	12.91
7	30	100	50	10	4551.7	6765.15	5294.51	32.72	14.03
8	30	100	50	10	4609.5	6802.99	5264.34	32.24	12.44
9	30	100	50	10	4613	6836.42	5338.95	32.52	13.60
10	**20**	100	50	10	3788.6	5524	4423.2	31.42	14.35
10	**40**	100	50	10	5411	8763.42	6336.47	38.25	14.61
10	**50**	100	50	10	6360.3	9691.24	7421.24	34.37	14.30
10	30	100	50	**2**	4760.3	7081.74	5561.32	32.78	14.40
10	30	100	50	**8**	4633.5	6916.51	5386.99	33.01	13.99
10	30	100	50	**14**	4592.3	6867.52	5404.58	33.13	15.03
10	30	100	50	**20**	4645.6	6852.88	5394.12	32.21	13.88
10	30	100	**0**	10	6130.3	7308.65	6312.60	16.12	2.89
10	30	100	**20**	10	5330.8	7056.59	5850.07	24.46	8.88
10	30	100	**40**	10	4672.2	6899.69	5368.84	32.28	12.97
10	30	100	**60**	10	4800.1	6864.83	5485.70	30.08	12.50
10	30	100	**80**	10	5436.2	6916.22	5754.48	21.40	5.53
10	30	100	**100**	10	6197.7	7154.74	6219.10	13.38	0.34
10	30	100	50	**50**	4596	7168.31	5428.3	35.88	15.33
10	30	100	50	**100**	4625	6857.15	5363.7	32.55	13.77
10	30	100	50	**150**	4633.2	6868.57	5415	32.55	14.44
10	30	100	50	**200**	4620.4	6846.41	5380.8	32.51	14.13

[a] Z_{rand} is the average objective value obtained by the random heuristic over 100 runs.
[b] Z_{NN} is the average objective value obtained by the NN heuristic over 100 runs.
[c] $G_{rand}(\%) = (Z_{rand} - Z^*)/(Z_{rand}) \times 100$.
[d] $G_{NN}(\%) = (Z_{NN} - Z^*)/(Z_{NN}) \times 100$.

have 50 retrieval requests that can be delivered to any I/O point at their destination side. The flexibility created by these requests definitely helps to patch many subtours in the merging algorithm. Finally, the travel times in this problem are confined to the size of the block. As a result, the AP relaxation of the problem provides a good lower bound, as shown in column $\frac{Z_{AP}}{Z^*}$. Therefore, even if the merging algorithm cannot merge subtours using the common I/O points, the optimal solution can be found in a limited number of nodes.

- The optimal solution is significantly better than the solutions obtained by the NN and random heuristics. Based on the results presented in Table 4, when there is a balance between the number of storage and retrieval requests ($\mathbb{P} = 50\%$), the average objective value improvement is more than 30% in case of the random heuristic, and 14% in case of the NN heuristic. On the other hand, in case of an imbalance between the number of storage and retrieval requests (moving toward $\mathbb{P} = 0$ or 100%), both random and NN heuristics perform better. The reason is that in the optimal solution, more pure storage and retrieval requests must be individually executed by the YC, and there is less opportunity to sequence retrieval requests after storage requests for minimizing the empty travel time of the YC.

We also compare the results of 12 different random instances obtained by the two-phase solution method with those results obtained by CPLEX 12.2 coded in C++ using the Concert Technology framework and executed by a g++ compiler on a 2.40 gigahertz AMD Opteron™ Processor 250, with 8.00 gigabytes of RAM under the Linux operating system. The results presented in Table 5 confirm the complexity of the problem, as CPLEX cannot solve the problems in reasonable time. The computation in CPLEX was truncated after 18,000 seconds. Table 5 shows that there is a large gap between the feasible objective value obtained by CPLEX in 18,000 seconds and the optimal objective value (obtained in less than a second). By increasing the number of requests, the gap between the two values increases. To start CPLEX, we use the problem formulation presented in Eqs. (2)–(6).

4.2. Case study

In this section, we use real data, provided by the Ultimate project (see Ultimate, 2013), to evaluate the performance of our solution method at a hinterland terminal linked to the Port of Rotterdam. Table 6 shows an excerpt from the terminal database including requests with their locations and transport modes with which they arrive or leave the terminal in the next half hour. We use this list of requests to evaluate our method.

The terminal uses a score-based heuristic available in the terminal operating software to sort requests to be handled. The score points given to each container are determined by the due-time and the transport mode to deliver it or pick it up. Using this heuristic, many containers receive relatively the same total score. The heuristic can therefore categorize containers into multiple groups handled one by one based on their scores. Containers within each group can be sorted in any order, and are carried out using the NN heuristic.

All containers listed in Table 6 receive roughly the same score. Thus, the terminal uses the NN heuristic to sort them. We use our two-phase solution method to sort them. Our calculations

Table 5
Comparing the two-phase solution method and truncated CPLEX.

Inst.	X	Y	N	$\mathbb{P}$ (%)	M	CPLEX		Our method		G_{CPLEX}(%)
						CPU	Z	CPU	Z^*	
1	10	40	**10**	10	50	4019	699.10	0.00	699.10	0
2	10	40	**20**	10	50	18,000	1100.10	0.01	1100.10	0
3	10	40	**30**	10	50	18,000	1687.54	0.02	1612.81	4.43
4	10	40	**40**	10	50	18,000	2310.24	0.02	2262.71	2.06
5	10	40	**50**	10	50	18,000	2994.09	0.03	2751.34	8.11
6	10	40	**60**	10	50	18,000	3817.35	0.05	2978.48	21.98
7	10	40	**70**	10	50	18,000	4448.72	0.06	3769.85	15.26
8	10	40	**80**	10	50	18,000	NA	0.07	4442.05	NA
9	10	40	**90**	10	50	18,000	NA	0.09	4611.33	NA
10	10	40	**100**	10	50	18,000	NA	0.13	5122.75	NA
11	10	40	**150**	10	50	18,000	NA	0.20	7525.72	NA
12	10	40	**200**	10	50	18,000	NA	0.32	10413.57	NA

Notes: Let Z and Z^* be the objective value obtained by truncated CPLEX and the optimal objective value obtained by the two-phase solution method, respectively. Column CPU is the computation time in seconds. If the computation time of CPLEX is less than 18,000 seconds, the optimal value is obtained. The gap is calculated as: $G_{CPLEX}(\%) = ((Z - Z^*)/Z) \times 100$.

Table 6
A snapshot of the terminal database of storage and retrieval requests.

Container	ISO	Weight	Request	Modality	Travel info.	Position
APZU 438568 2	42GP	15,218	Retrieval	Train	121385-ELCH-O	14.08.03
PONU 133010 5	42GP	13,680	Retrieval	Train	121385-ELCH-O	07.02.02
UESU 426669 8	42GP	14,747	Retrieval	Train	121385-ELCH-O	17.06.02
CCLU 419140 0	42GP	14,650	Retrieval	Train	121385-ELCH-O	18.08.03
POCU 104202 3	42GP	2646	Retrieval	Train	121385-ELCH-O	14.04.02
IKEA 046846 3	20GP	2500	Storage	Barge	121399-ALVERNA-I	16.06.01
IKEA 101010 9	20GP	2500	Storage	Barge	121399-ALVERNA-I	22.10.02
IKEA 111111 0	20GP	2500	Storage	Barge	121399-ALVERNA-I	23.08.04
IKEA 135448 6	20GP	2500	Storage	Barge	121399-ALVERNA-I	17.07.03

show that the improvement over the NN objective value is 8.4%. The improvement depends on how many containers receive the same relative score. It can be large if many containers can be sequenced simultaneously. If there is a balance between the number of storage and retrieval requests at the seaside and landside, our solution method can improve the result significantly by performing double cycles.

At terminals, the list of container requests is updated in given time periods. We can run our solution method and find a new request sequence as soon as the list is updated. The ending location of the YC in the previous time period serves as the starting location of the YC in the current time period.

5. Conclusion and further research

We model and solve a difficult operational problem in which a set of container storage and retrieval requests must be sequenced. We minimize the total travel time of a single YC carrying out the requests in a single block of containers. An efficient YC scheduling operation can significantly affect the overall performance of the container terminal. We formulate the problem as an integer model, prove the problem complexity, and develop a two-phase solution method to obtain optimal solutions. The merging algorithm proposed in the first phase to patch subtours of an optimal AP solution works efficiently specially for large-scale problems. As the number of requests increases, the average number of requests per I/O point increases and therefore, the merging algorithm has more opportunity to patch subtours. As a result, in most of the realizations of the scenarios an optimal solution can be obtained in the first phase by optimally solving the AP and using the merging algorithm. If an optimal solution cannot be obtained in the first phase, a B&B algorithm in the second phase rapidly finds an optimal solution. The computation time of the solution method is low, i.e. it is less than a second when the number of requests is 200. The gaps between the optimal value and the objective values of the random and NN heuristics are on average more than 30% and 14%, respectively. Furthermore, the two-phase solution method significantly outperforms CPLEX for small-size instances. For instances with 80 requests, CPLEX cannot obtain a feasible solution after five hours. We also have tested the method on data from a hinterland terminal and found an improvement of 8.4%.

The model developed in this paper can be extended in several directions. We focus on scheduling isolated YC operations. Container handling operations at a terminal requires integrated coordination from QCs, vehicles, YCs, and gates at the seaside, stacking area, and landside. The terminal resources need to be coordinated effectively and the interactions among them need to be understood well. Furthermore, we consider that storage locations are given. However, one may consider combining the YC scheduling problem with the container allocation problem. Finally, we consider that all storage containers are available at the I/O points and a retrieval container can be dropped off at the same I/O point from where another container has to be stacked in the block. But safety regulations and space availability cause some limitations. Extra constraints are required to model reality.

Acknowledgements

This work was partially supported by the Ultimate project (funded by the Dutch Institute for Advanced Logistics (DINALOG)), Erasmus Smart Port Rotterdam, as well as the Thousand Young Scholar Program and the National Science Foundation of China under Grants 71225002, 71110107024, and 71121061. These supports are gratefully acknowledged. Thanks are due to the Editor and to the referees for their valuable comments. Finally, the help

of the annonymous reviewer, Leo Kroon, and Adriana Gabor for the proofs in this paper is acknowledged.

Appendix A. Proof of Theorem 1

The proof is via a reduction from the TSP with L^∞ norm (or maximum norm). It is well-known that this problem is $\mathcal{NP}$-hard, see Burkard et al. (1998) and Itai, Papadimitriou, and Swarcfiter (1982). The problem remains $\mathcal{NP}$-hard, if the objective function does not consider the distance between the last and the first point of the sequence. In other words, the problem remains $\mathcal{NP}$-hard if we have to visit each point once but do not need to return to the origin (this can be shown by adding a dummy-point and each ATSP can be transformed to a TSP with twice as many vertices). Obviously, the problem also remains $\mathcal{NP}$-hard, if the first point of the sequence is given.

Let I be an instance of the TSP with L^∞ norm. That is, I contains n points (x_i, y_i) in the (x, y)-plane. Without loss of generality, $x_i > 0$ and $y_i > 0$ for $i = 1,\ldots,n$. The travel time between two points (x_i, y_i) and (x_j, y_j) is measured as t_{ij}:=max $\{|x_i - x_j|, |y_i - y_j|\}$. Next, let M be a large number satisfying $M > \sum_i \sum_j t_{ij}$. Note that M is larger than the length of any TSP tour along the n points.

Now we create the following instance I' of the YC scheduling problem with prescribed retrieval I/O points. There are n I/O points. I/O point i is located at position $((i + 2)M, 0)$. We select the first I/O point to be located at position $(1 + 2)M$ on the X-axis, so that we can make sure that the travel time to and from the location of requests in the X-direction takes longer than the Y-direction. Moreover, there are n storage requests, where storage request i goes from I/O point i to point (x_i, y_i). There are n retrieval requests, where retrieval request i goes from point (x_i, y_i) to I/O point i. We assume that storage and retrieval locations coincide, which is actually not possible. However, the proof also works if they are located sufficiently close together. The initial location of the crane is (x_1, y_1).

Now we will prove the following statement: there exists a tour for the instance I of the TSP with length K if and only if there exists a tour for the instance I' of the YC scheduling problem with length $K + 2L$, where $L = \sum_i ((i + 2)M - x_i)$.

"If": Suppose there exists a tour for the instance I' of the YC scheduling problem with length $K + 2L$. According to the construction of I', storage container i goes from I/O point i to point (x_i, y_i), and retrieval container i goes from point (x_i, y_i) to I/O point i. These storage and retrieval movements account for a total length of $2L$ already. Note that the movements in the Y-direction do not cost any time, since they are shorter than the movements in the X-direction.

If retrieval request i would be followed directly by a different storage request than storage request i or by another retrieval request, then at least a length M would be added to the tour. This would make it impossible to end up with a tour of length $K + 2L$, since $M > K$. Thus retrieval request i is followed by storage request i. Thus the remaining part of the tour for the YC consists of a TSP tour of length K along the points (x_i, y_i).

"Only if": If I is a yes-instance of the TSP, then the above construction can be used to construct the requested schedule for the instance I' of the YC scheduling problem with makespan $K + 2L$.

Appendix B. Proof of Theorem 2

Consider two subtours resulting from an optimal AP solution with the swappable arcs (k, l) and (r, s) and the common visiting I/O point, I/O_m. By setting arcs $x_{kl} = x_{rs} = 0$ and $x_{ks} = x_{rl} = 1$, we can show that subtours are merged without extra travel time.

The objective function of the problem can be shown as follows:

$$\min Z = \sum_{i\in\mathcal{V}\setminus\{j,0'',r,k\}} \sum_{j\in\mathcal{V}\setminus\{0',l,s\}} t_{ij}x_{ij} + t_{kl}x_{kl} + t_{rs}x_{rs} + t_{ks}x_{ks} + t_{rl}x_{rl}. \quad (11)$$

Before merging, $x_{kl} = x_{rs} = 1$ and $x_{ks} = x_{rl} = 0$, so $Z_{\text{Before}} = \sum_{i\in\mathcal{V}\setminus\{j,0'',r,k\}} \sum_{j\in\mathcal{V}\setminus\{0',l,s\}} t_{ij}x_{ij} + t_{kl} + t_{rs}$. On the other hand, from Table 1, $t_{kl} = t_{k,I/O_m} + t_{I/O_m,l}$ and $t_{rs} = t_{r,I/O_m} + t_{I/O_m,s}$. Consequently, $Z_{\text{Before}} = \sum_{i\in\mathcal{V}\setminus\{j,0'',r,k\}} \sum_{j\in\mathcal{V}\setminus\{0',l,s\}} t_{ij}x_{ij} + t_{kl} + t_{rs} = \sum_{i\in\mathcal{V}\setminus\{j,0'',r,k\}} \sum_{j\in\mathcal{V}\setminus\{0',l,s\}} t_{ij}x_{ij} + t_{k,I/O_m} + t_{I/O_m,l} + t_{r,I/O_m} + t_{I/O_m,s}$.

After merging, $x_{kl} = x_{rs} = 0$ and $x_{ks} = x_{rl} = 1$, so $Z_{\text{After}} = \sum_{i\in\mathcal{V}\setminus\{j,0'',r,k\}} \sum_{j\in\mathcal{V}\setminus\{0',l,s\}} t_{ij}x_{ij} + t_{ks} + t_{rl}$. From Table 1, we have $t_{ks} = t_{k,I/O_m} + t_{I/O_m,s}$ and $t_{rl} = t_{r,I/O_m} + t_{I/O_m,l}$, therefore $Z_{\text{After}} = \sum_{i\in\mathcal{V}\setminus\{j,0'',r,k\}} \sum_{j\in\mathcal{V}\setminus\{0',l,s\}} t_{ij}x_{ij} + t_{k,I/O_m} + t_{I/O_m,s} + t_{r,I/O_m} + t_{I/O_m,l} = Z_{\text{Before}}$.

Appendix C. Proof of Theorem 4

Theorem 3 proves that if the output of the first phase of the algorithm is a complete tour, an optimal solution of the problem is obtained. Here, we prove that if the output is a solution with multiple subtours, then the branch and bound (B&B) algorithm obtains an optimal solution of the problem.

From the problem formulation, we know that a relaxation of the problem is an assignment problem (AP). The B&B algorithm starts from an optimal AP solution with multiple subtours and in every node modifies the solution by adding some constraints to eliminate subtours. Note that the output of the first phase is an optimal AP solution (with multiple subtours) since the merging algorithm only changes one optimal AP solution to another optimal AP solution. It does not modify the output, and it does not add or remove any constraints. The same argument also holds for other nodes of the B&B tree where we use the merging algorithm to patch subtours. It only speeds up the search (see, for example, Carpaneto et al., 1995 or Miller & Pekny (1991)). Now, we need to show that (1) the B&B algorithm branches the solution space into complimentary subspaces, (2) it searches through all these subspaces and does not fathom any node of which one of child nodes results in an optimal solution.

For branching, the B&B algorithm first needs to select one subtour to be broken. In this phase, any subtour can be selected, since if a subtour is not selected and it remains to appear in an optimal AP solution of the next child nodes, it can be selected later to be broken. Now, we prove that the branching scheme discussed in Section 3.2.1 divides the solution space into complementary subspaces and does not result in losing any part of the solution space. Assume that a subtour S with n arcs is selected to be broken. For simplicity, assume that $\mathcal{A} = \{x_1, x_2, \ldots, x_n\}$ is the set of arcs in subtour S. In order to omit subtour S from the solution, we can create n branches, where in child node i, arc x_i, $i = 1,2,\ldots,n$, is excluded. This branching is *exhaustive* but *not complimentary*, since the same optimal AP solution may be obtained in multiple child nodes. For example, it may be a solution in which none of the arcs of subtour S appear. To have an *exhaustive* and *complimentary* branching rule, in child node i, we need to exclude arc x_i and include all the other ones, which are, in turn, excluded in child nodes $1,2,\ldots,i-1$. The inclusion or exclusion of arcs is shown in the following matrix, where 0 indicates exclusion, 1 indicates inclusion, and 0/1 indicates a free arc (both combinations are investigated by the AP):

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{ccccc} \text{child node 1} & \text{child node 2} & \text{child node 3} & \ldots & \text{child node n} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & \ldots & 1 \\ 0/1 & 0 & 1 & \ldots & 1 \\ 0/1 & 0/1 & 0 & \ldots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0/1 & 0/1 & 0/1 & \ldots & 0 \end{pmatrix} \end{array}.$$

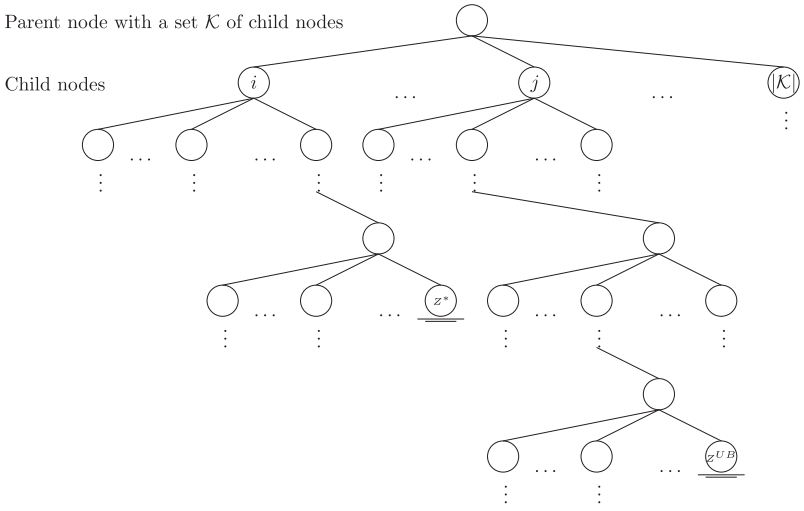


Fig. 7. The B&B tree.

In the matrix above, all combinations of inclusion and exclusion occur for every selection of different arcs. Therefore, the branching rule is collectively exhaustive. Furthermore, comparing every two neighboring columns shows that they differ in (at least) one arc which must be excluded from the AP solution of one node, and must be included in the AP solution of the other one. Therefore, the branching rule is also complimentary. Now, we have to prove that the B&B goes through all these nodes and does not fathom the node resulting in an optimal solution before obtaining that optimal solution.

Assume that a node results in a set $\mathcal{K}$ of child nodes (see Fig. 7). Furthermore, there is a child node $i \in \mathcal{K}$ with an optimal AP (with the including and excluding arcs constraints) solution value Z_i which eventually results in one of its child nodes in an optimal YC scheduling solution Z^*. Obviously, we have $Z_i \leqslant Z^*$, since by creating each child node extra including and excluding arcs constraints are added. We prove by contradiction that *node i is not fathomed before obtaining the optimal YC scheduling solution*. The B&B algorithm fathoms node i using two criteria:

Criterion (1): the solution in node i is a complete tour,
Criterion (2): the solution $Z_i \geqslant Z_{UB}$.

Assume that criterion (1) is satisfied so that node i can be fathomed. In this case, since we know that node i results in an optimal YC scheduling solution and an optimal AP solution in this node is already a complete tour, an optimal YC scheduling solution is obtained $Z^* = Z_i$, so the assertion is proved. Now, assume that criterion (2) is satisfied so that we can fathom node i. Assume that there is a child node $j \in \mathcal{K}$ one of whose child nodes has an optimal AP (with the including and excluding arcs constraints) solution which is a complete tour and provides Z_{UB}. Then, we have $Z_{UB} \leqslant Z_i$. Furthermore, we know $Z_i \leqslant Z^*$. It yields that $Z_{UB} \leqslant Z^*$, which cannot occur. This reasoning should be repeated for all the other nodes and since there are a finite number of nodes, Z_i will never be fathomed until the optimal solution is obtained.

References

Bellmore, M., & Nemhauser, G. L. (1968). The traveling salesman problem: A survey. *Operations Research, 16*(3), 538–558.

Bierwirth, C., & Meisel, F. (2010). A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research, 202*(3), 615–627.

Böse, J. W. (2011). *Handbook of terminal planning*. New York: Springer.

Burkard, R. E., Deineko, V. G., Van Dal, R., Van der Veen, J. A. A., & Woeginger, G. J. (1998). Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review, 40*(3), 496–546.

Carpaneto, G., Dell'Amico, M., & Toth, P. (1995). Exact solution of large-scale, asymmetric traveling salesman problems. *ACM Transaction on Mathematical Software, 21*(4), 394–409.

Carpaneto, G., & Toth, P. (1980). Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Science, 26*(7), 736–743.

Cheung, R. K., Li, C. L., & Lin, W. (2002). Interblock crane deployment in container terminals. *Transportation Science, 36*(1), 79–93.

Clausen, J. (2003). *Branch and bound algorithms – Principles and examples*. Department of Computer Science, University of Copenhagen.

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

De Koster, R., Balk, B. M., & Van Nus, W. T. I. (2009). On using dea for benchmarking container terminals. *International Journal of Operations and Production Management, 29*(11), 1140–1155.

De Koster, R., & Van der Poort, E. (1998). Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions. *IIE Transactions, 30*, 469–480.

Dorndorf, U., & Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum, 32*(3), 617–632.

Drewry (2011). *Container forecaster*. London: Drewery Publications.

Eastman, W. L. (1958). *Linear programming with pattern constraints*. Dissertation, Harvard University.

Gharehgozli, A. H., Laporte, G., Yu, Y., & De Koster, R. (2013). *Scheduling two yard cranes handling requests with precedences in a container block with multiple open locations*. Tech. rep., Rotterdam.

Gilmore, P. C., & Gomory, R. E. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research, 12*(5), 655–679.

Itai, A., Papadimitriou, C. H., & Swarcfiter, J. L. (1982). Hamilton paths in grid graphs. *SIAM Journal on Computing, 11*, 676–686.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of computer computations* (pp. 85–103). Plenum Press.

Kim, K. H., & Kim, K. Y. (1999). An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science, 33*(1), 17–33.

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.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly, 2*, 83–97.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research, 59*(2), 231–247.

Law, A. M., & Kelton, D. M. (1999). *Simulation modeling and analysis* (3rd ed.). Boston, Massachusetts: McGraw-Hill Higher Education.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The traveling salesman problem, a guided tour of combinatorial optimization*. Chichester, UK: John Wiley & Sons.

Li, W., Goh, M., Wu, Y., Petering, M. E. H., De Souza, R., & Wu, Y. C. (2012). A continuous time model for multiple yard crane scheduling with last minute job arrivals. *International Journal of Production Economics, 136*(2), 332–343.

Linn, R. J., & Zhang, C. Q. (2003). A heuristic for dynamic yard crane deployment in a container terminal. *IIE Transactions, 35*(2), 161–174.

Li, W., Wu, Y., Petering, M. E. H., Goh, M., & de Souza, R. (2009). Discrete time model and algorithms for container yard crane scheduling. *European Journal of Operational Research, 198*(1), 165–172.

Miller, D. L., & Pekny, J. F. (1991). Exact solution of large asymmetric traveling salesman problems. *Science, 251*(4995), 754–761.

Murty, K. G. (2007). Yard crane pools and optimum layouts for storage yards of container terminals. *Journal of Industrial and Systems Engineering, 1*, 190–199.

Narasimhan, A., & Palekar, U. S. (2002). Analysis and algorithms for the transtainer routing problem in container port operations. *Transportation Science, 36*(1), 63–78.

Ng, W. C. (2005). Crane scheduling in container yards with inter-crane interference. *European Journal of Operational Research, 164*(1), 64–78.

Port of Rotterdam Authority (Ed.). (2010). *Port statistics*. Port of Rotterdam Authority, Rotterdam.

Ratliff, H. D., & Rosenthal, A. S. (1983). Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Operations Research, 31*(3), 507–521.

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), 3–49.

Ultimate (2013). *Ultimate: Efficient multimodal*. <http://www.dinalog.nl/en/projects/r_d_projects/ultimate__efficient_multimodal/> Retrieved 03.01.13.

United Nations: ESCAP (2007). *Regional shipping and port development: Container traffic forecast 2007 update*. United Nations: Economic and Social Comission for Asia and the Pacific (ESCAP), New York.

Van den Berg, J. P., & Gademann, A. J. R. M. (1999). Optimal routing in an automated storage/retrieval system with dedicated storage. *IIE Transactions, 31*(5), 407.

Vis, I. F. A., & Carlo, H. J. (2010). Sequencing two cooperating automated stacking cranes in a container terminal. *Transportation Science, 44*(2), 169–182.

Vis, I. F. A., & Roodbergen, K. J. (2009). Scheduling of container storage and retrieval. *Operations Research, 57*(2), 456–467.

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.

Wiese, J., Suhl, L., & Kliewer, N. (2010). Mathematical models and solution methods for optimal container terminal yard layouts. *OR Spectrum, 32*(3), 427–452.

Zhang, C., Wan, Y. W., Liu, J., & Linn, R. J. (2002). Dynamic crane deployment in container storage yards. *Transportation Research Part B: Methodological, 36*(6), 537–555.

Zrnić, N., & Vujičić (2012). Evaluation of environmental benefits of CHE emerging technology by using LCA. In *Proceedings of IMHRC 2012*.