NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPTIMIZING CONTAINER MOVEMENTS USING ONE AND TWO AUTOMATED STACKING CRANES

by

Ioannis Zyngiridis

December 2005

Thesis Advisor:	Robert Dell
Second Reader:	Johannes O. Royset

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2005	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: Optimizing container movements using one and two Automated Stacking Cranes	5. FUNDING NUMBERS
6. AUTHOR(S) Ioannis Zyngiridis	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The number of containers shipped through ports has increased substantially in recent years and has stimulated research and development of ways to improve storage yard operations. The productivity of a port's storage yard depends, in part, on the cranes that are working in storage blocks. Each crane follows a route described primarily by the order to move each container that enters or leaves a block and the position to stack each container in the block. Each container that leaves (enters) the block must be unloaded (loaded) in a limited capacity transfer point before (after) a given time. This thesis is the first to develop Integer Linear Programs (ILPs) to prescribe routes for one and two equal sized Automated Stacking Cranes (ASCs) in a single block working with straddle carriers to load and unload containers from the transfer points. Using real world data, we construct test problems varying both the number of container bays (length) and excess capacity of each block. We find one ASC working alone over four hours requires up to 70% more time than two ASCs working together to accomplish the same required container movements. ILP solution time is typically only a few seconds.

14. SUBJECT TERMS Optimization, Integer Linear Program, Container Storage, Port Logistics, Port Operations, Crane Scheduler, Automated Stacking Crane	15. NUMBER OF PAGES 71
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

OPTIMIZING CONTAINER MOVEMENTS USING ONE AND TWO AUTOMATED STACKING CRANES

Ioannis Zyngiridis
Captain, Hellenic Army
B.S., Hellenic Military Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author: Ioannis Zyngiridis

Approved by: Robert Dell
Thesis Advisor

Johannes O. Royset
Second Reader

James N. Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The number of containers shipped through ports has increased substantially in recent years and has stimulated research and development of ways to improve storage yard operations. The productivity of a port's storage yard depends, in part, on the cranes that are working in storage blocks. Each crane follows a route described primarily by the order to move each container that enters or leaves a block and the position to stack each container in the block. Each container that leaves (enters) the block must be unloaded (loaded) in a limited capacity transfer point before (after) a given time. This thesis is the first to develop Integer Linear Programs (ILPs) to prescribe routes for one and two equal sized Automated Stacking Cranes (ASCs) in a single block working with straddle carriers to load and unload containers from the transfer points. Using real world data, we construct test problems varying both the number of container bays (length) and excess capacity of each block. We find one ASC working alone over four hours requires up to 70% more time than two ASCs working together to accomplish the same required container movements. ILP solution time is typically only a few seconds.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I extend my personal thanks to Erik Tiemroth, Mehmet Ayik and John Christman from Navis Llc for their expert guidance along the way and I appreciate the significant efforts of Robert Dell and Johannes Royset.

For her devotion, sacrifice, patience and overwhelming support to the very end, my wife, Evi.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The number of containers shipped through ports has increased substantially in recent years and has stimulated research and development of ways to improve storage yard operations. A port's storage yard productivity depends, in part, on the cranes that are working in storage blocks. Each crane follows a route described primarily by the order to move each container that enters or leaves a block and the position to stack each container in the block. Each container that leaves (enters) the block must be unloaded (loaded) in a limited capacity transfer point before (after) a given time. This thesis is the first to develop Integer Linear Programs (ILPs) to schedule routes for one and two equal sized Automated Stacking Cranes (ASCs) in a single block working with straddle carriers to load and unload containers from the transfer points.

When there is only one ASC, we schedule it in three connected steps. First, containers that are entering (imported) or leaving (exported) the block are scheduled. This is the primary step where a route is found that minimizes the total travel distance of the ASC. The ASC considers moves that coordinate the placement of imported and exported containers while giving priority to export container requirements. We assume that the ASC removes containers above an exported container for storage elsewhere (reshuffles) immediately before moving the exported container. During the first step, we only reserve time to reshuffle. In the second step we schedule the reshuffles. If there is still available time, we schedule (the third step) reshuffles that reduce future reshuffles (house-keeping).

For two ASCs working in a single block, we divide the block into two sub-areas: Water Side (WS) working area and Land Side (LS) working area. One ASC is responsible for the demands on the LS working area and the other for the demands on the WS working area. We use the same three steps described

above and initially assume that the ASCs can crossover. At the beginning of the third step, we adjust (if needed) the two ASCs' moves to ensure no collisions occur.

We investigate how different block characteristics influence the performance of one and two ASCs. Using real world data, we construct test problems varying both the number of container bays (length) and excess capacity of each block. We find one ASC working alone over four hours requires up to 70% more time than two ASCs working together to accomplish the same required container movements. The length of the block and the area fullness significantly affect the performance of one ASC. For two ASCs, only the length of the block influences performance. We discover in all cases we consider that the coordinated use of two ASCs significantly outperforms one ASC working alone. ILP solution time is typically only a few seconds for all cases.

I. INTRODUCTION

The number of containers shipped through ports has increased substantially in recent years and has stimulated research and development of ways to improve storage yard operations. The productivity of storage yards depends, in part, on the cranes that are working in storage blocks. Each crane follows a route described primarily by the order to move each container that enters or leaves the block and the position to stack each container in a block. This thesis develops Integer Linear Programs (ILPs) to prescribe routes for one and two Automated Stacking Cranes (ASCs) in a single block.

A. BACKGROUND

1. Overview of Container Transshipment

The handling of containers at Container Terminals (CTs) is becoming more demanding in ports worldwide. Foxcroft [2002] reports that an estimated 15 million containers were handled in 2002 around the world. Aston et al. [2005] report inbound containers into the U.S. will increase by 6.7% in 2005 and grew 50% over the last five years (2001-2005). Ioannou et al. [2002] forecasts that the U.S. container trade will experience an average annual growth of 7.8% through the year 2010.

The construction vessel industry is building ships with higher container capacity (Figure 1) to respond to the increase in container shipments [Henesey 2004]. This increase also causes an increased demand for port storage that influences port stacking policies and creates a demand for improved technical equipment for container logistics [Volk 2002 and Stenken, Vob and Stahlbock 2003].

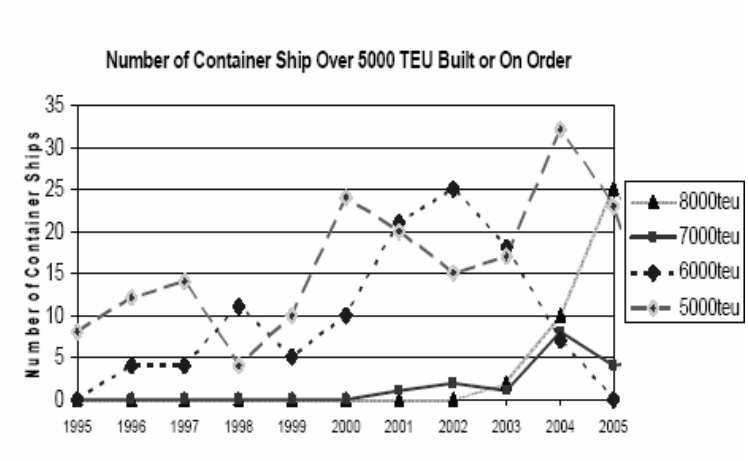


Figure 1. Number of Container Ships Built or on Order, 1995-2005. (From: Henesey [2004])

2. Container Description

For each container already in a block or scheduled to arrive, we know the following information:

a. Size (20 ft., 40 ft., and 45 ft.).

b. Identification number.

c. Content.

d. The name of the vessel.

e. Destination port.

f. Priority.

g. Status.

h. Future transfer point.

The content of the container is the type of material the container carries. Hazardous containers are usually stacked in special blocks or in special areas where they can be easily inspected. The name of the vessel and the destination port are given for any container that will be loaded onto a vessel in the future. This information helps to identify the best stacking location for a container. For example, consider two containers (A and B) that will be loaded on the same vessel. The container that has the most distant port destination (in this example, assume A) should be placed above container B to allow A to be easily stacked in a lower position when loaded on its future vessel. The priority of a container is

an estimate on roughly when this container will leave the block. The status indicator declares if the container is empty, dry or fridge.

3. Container Handling in a Port

When a vessel arrives in a port, a berth location is assigned to the ship and the procedure of unloading and loading containers into the vessel starts almost immediately. Henesey [2004] describes four main ordered steps that each container has to follow from the moment it arrives on a vessel until it exits a port either on a truck, train, or in another vessel. A typical overview of a CT is shown in Figure 2.

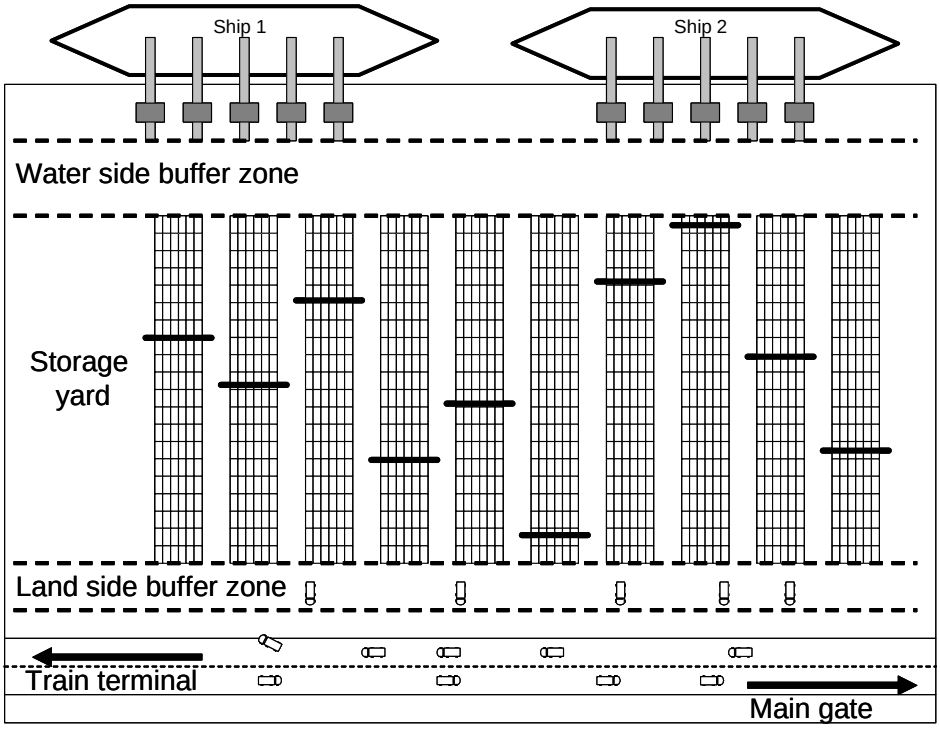
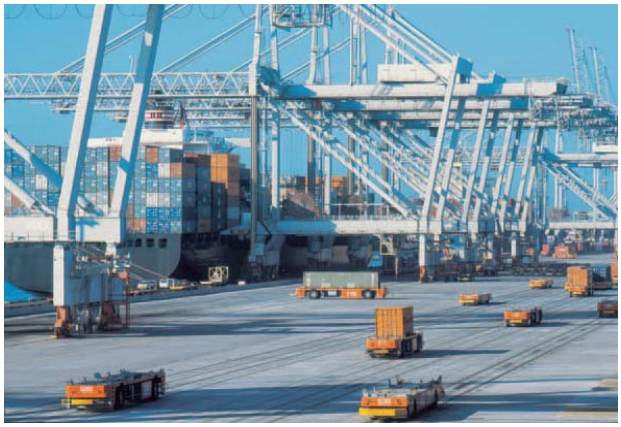


Figure 2. Operation Areas in a Sea Port Terminal.

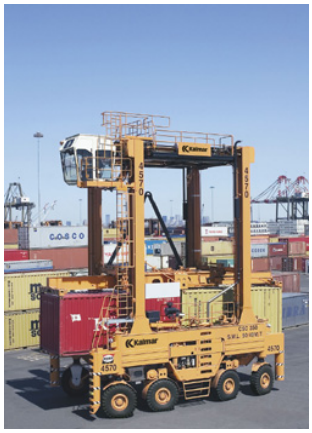
Quay cranes unload and load containers from vessels. The blocks in the storage yard are perpendicular to the shoreline, and the loading and unloading areas in the storage yard can be described as Water Side (WS) and Land Side (LS).

The first step is the ship-to-shore movement where a Quay crane lifts a container from a vessel and moves it to shore, where carriers are waiting to transport it to a stacking area. Figure 3a shows Quay cranes unloading a vessel and Automated Ground Vehicles (AGVs) transporting the containers to the storage yard.

The second step is the transportation of the container to a storage yard. In addition to AGVs, there are other transportation vehicles such as straddle carriers (Figure 3b). AGVs do not need a driver but they require the assistance of a crane to load or unload a container. A straddle carrier, unlike AGVs, can pick a container from the ground or unload it without the presence of a crane. The storage yard includes several blocks (Figure 2), where an AGV or straddle carrier unload each container to its assigned block.



a.



b.

Figure 3. Quay cranes, AGVs and Straddle Carrier.
a. Quay crane with AGVs transporting the containers (Gottwald Port Technology [2005]). **b.** Straddle Carrier (Kalmar Industries [2005])

The third step, the focus of this thesis, is the storage of containers in a block. Each block has a specified number of bays, rows, and tiers where containers are stored in stacks (Figure 4). Cranes in each block are responsible for the storage and transport of the container into and out of the block.

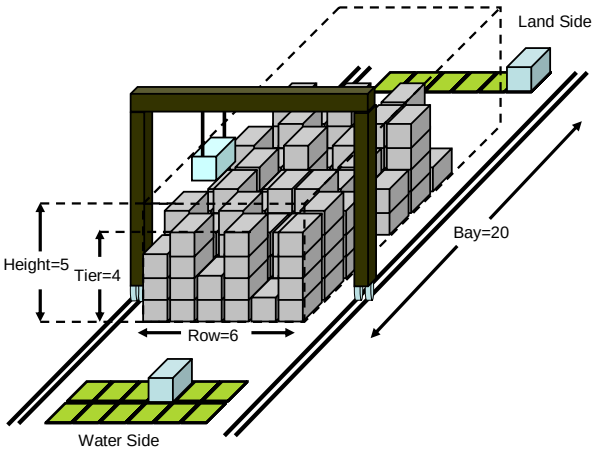


Figure 4. Graphical Representation of an ASC Working in a Block. The bay refers to the number of containers along the block, the row refers to the width of the block, and tier refers to the maximum number of containers in each.

To improve block operations, there are many ports that are using two cranes in each block. In most of these cases, the two cranes are of the same size and type and they cannot crossover. There are also cases where two cranes of different heights are working in the same block, giving more flexibility in the movements they can make.

There are three main types of cranes and each one has the ability to transport at most one container. The Rubber-Tired Gantry Crane (RTG) moves on rubber tires and has the flexibility to move between blocks (Figure 5).

Figure 5. Rubber-Tired Gantry Crane (RTG). (From: Kalmar Industries [2005])

The Rail-Mounted Gantry Crane (RMG) moves over railways only in a specific block and does not have the ability to move between different blocks (Figure 6).

Figure 6. Rail-Mounted Gantry Crane (RMG). (From: Kalmar Industries [2005])

An Automated Stacking Crane (ASC) also moves over rails in a specific block. It does not have the ability to move between blocks but is fully automated in all of its operations and works without the assistance of a driver (Figure 7).

Figure 7. Automated Stacking Crane (ASC). (From: Kalmar Industries [2005]) ASC operating at ECT Delta terminal in Rotterdam

The blocks in the storage yard are either parallel or perpendicular to the shoreline. Blocks parallel to the shoreline are most common at ports that are using RTGs. Blocks perpendicular to the shoreline as in Figure 2 are common when a RMG or ASC stacks the containers. In the latter case, due to the relative position between the blocks and the shoreline, the containers arrive to the block either from the Water Side (WS) or from the Land Side (LS).

Typically, an arriving container that will be loaded on a vessel is called an *inbound* container and a container that will be loaded on a truck is called an *outbound* container. A container that arrives in a block from the LS usually comes on a truck, either from the gate of the port or from a train terminal. The LS and WS buffer zones are the locations where the ASC loads and unloads containers. In this study, a container that the ASC stacks in the block is classified as an *imported* container and a container that is already stacked in the block that the ASC transports to a transfer point is referred to as an *exported* container.

The best position to stack a container depends on the container's characteristics and what containers are already in each of the stacks. It is found that ports do not adopt the same stacking policy, mostly because of the differences in the available area of the storage yards and the types of the cranes and other equipment that they use.

The fourth step is a delivery-receipt movement where an ASC picks a container from the stack and unloads it to the LS or WS buffer zone for a waiting truck or vessel. Straddle trucks work on the LS of the block loading containers on a truck (Figure 8). A straddle carrier transports a container from WS to the berth area of a vessel.

Figure 8. Straddle Trucks are Loading Containers on Trucks. (From: Ceres Paragon [2005])

B. OBJECTIVES

This thesis formulates and solves ILPs for scheduling one and two ASCs in a block. An optimal schedule for an ASC has to satisfy all the demand that appears on LS and WS without delays, while finding an optimal place to stack each imported container. The optimal placement must consider containers that are already in the stack as well as time limits. The primary objective is to satisfy all the exported container time commitments. There are LS and WS buffer zones where a carrier leaves a container to be stacked, so an ASC can more easily delay the transport of an imported container for a short time without hampering overall port operations. Finding efficient ASC schedules helps to increase the productivity of the ASC, increase the number of containers that can be imported and exported from the block, reduce the cost that a vessel or truck has to pay in a port, and finally, improve the flow of the containers in the port.

II. OTHER CRANE SCHEDULING PROBLEMS

Port operations have been an area of increased study over the last few years. Steenken et al. [2004] and Henesey [2004] present an integrated view of current research. They categorize each study according to its focus (ship to shore, transportation, stacking, delivery) and the type of analysis (operational, tactical, or strategic). Lin [2000] shows that a general crane scheduling problem is NP hard. Hence, crane scheduling optimization is expected to be computational expensive.

We have not found any published reports on scheduling container stacking with one or two ASCs and straddle carriers. Most related studies focus on scheduling RMGs or RTGs and AGVs. They approach the problem in ways that give useful background and insight.

An ASC schedule is a sequence of container moves, where each move specifies the exact position to stack each container. Lin [2000] proposes heuristic approaches for large-scale versions using RTGs. He deploys RTGs among different storage blocks, depending on the workload in each block, and suggests a network flow formulation with a piecewise-linear objective function that minimizes the unfinished workload that is transferred each time to the next time period. Similarly, Linn et al. [2003] explore the deployment of RTGs between blocks and constructs an ILP to determine the optimal crane allocation by minimizing the crane workload overflow. Zhang et al. [2002] formulate the same problem as an ILP where they minimize the workload at the end of each period, and apply Lagrangian relaxation to solve it.

Kozan and Preston [1999], without referencing to a specific crane, minimize the traveling time of containers from a stack to a vessel. They proposed an ILP for small-scale instances and a Genetic Algorithm solution for large-scale instances. They show that a fixed storage policy significantly reduces vessel load time compared with a random policy for various storage-area fullness. In the same study, they conclude that the transfer time of a fixed

number of containers increases exponentially when the number of yard cranes decreases. This result suggests the use of cranes that move among blocks such as RTGs.

Kim and Kim [1997] propose a routing algorithm for a single crane working in a block loading only exported containers out of the block. Kim and Kim [1999] propose a heuristic algorithm to solve the same problem. In both cases, the algorithm does not dynamically schedule imported and exported containers from the block, and a specific sequence for each of the exported containers is not determined.

The problem of scheduling two cranes that are working in the same block has not been widely studied. Stenken et al. [2004] provide a reference for an unpublished study (Eisenberg et al. [2003]) that examines the case of two RMGs that can crossover.

Navis, a company that develops software for scheduling port operations, offers SPARCS [2004] for heuristically scheduling one ASC. SPARCS is used in many ports around the world. Navis reports that it finds efficient stacking positions even for large-scale problems. SPARCS selects the stacking positions according to a penalty system that evaluates the characteristics of each stack, giving the flexibility to adopt different stacking policies according to a port's requirements. [Ayik 2005]

III. SCHEDULING ONE ASC

A. INTRODUCTION

We first consider only one ASC working in a single block. Given a predefined time window (typically 15 minutes) the ASC must transport all containers that are entering or leaving the block within the time window. An entering container is either in a transfer point at the beginning of the time window or arrives at a specific buffer zone during the time window. A container that the ASC must export is currently in the block and must be transported to a transfer point before a known time when a carrier will arrive. Any ACS delay in delivering exported containers causes delays to the straddle carriers and ultimately to the loading time of a vessel, truck or train.

Even with only a small number of containers there is a large number of possible sequences. The value of stacking a container at the top of a stack changes dynamically each time an ASC stacks, exports, or reshuffles a container from a stack.

A heuristic solution to ASC scheduling is to order the containers according to when they arrive or must be placed at a transfer point and then schedule the jobs in that order. The ASC selects the first job in the schedule and finds the shortest route to accomplish it. If the first job is to transport a container from the block to a transfer point, then it calculates the travel time and any time for *reshuffling* or *rehandling* (restacking containers that are over the exported container). If the first job is to transport an imported container, then it finds a position that it can stack the container, while satisfying time limits. If there are no imported or exported containers, the ASC performs what is known as *house-keeping* jobs where it restacks some of the containers in the block in order to save future reshuffling time.

This heuristic approach can produce results that are far from optimal. Figure 9 provides an example where the ASC starts in the WS and has two containers to export (A and B identified by grey color in the block). For simplicity,

we assume that containers A and B are on the top of their stacks. In the first case, the ASC first transports container A to the WS and then transports container B to the LS. Figure 9a shows the step-by-step ASC route. By transporting container B first (Figure 9b), the ASC accomplishes both jobs in a significantly shorter distance and time. The example in Figures 9c and 9d switches the destination for container A to the LS and again shows the importance of selecting the best order to transport the containers.

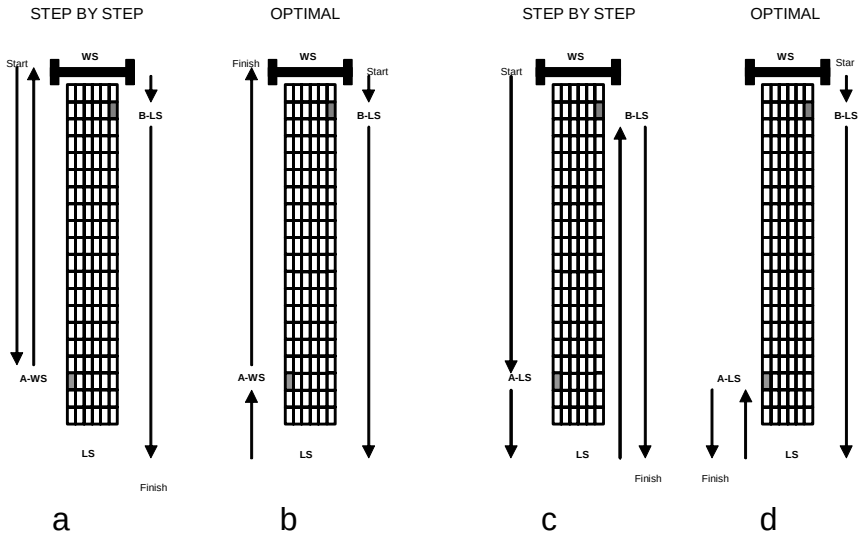


Figure 9. The Importance of Selecting the Best Sequence. The arrows indicate the route and the travel distance of the ASC. **a.** First case, container A before container B. **b.** B before A. **c.** Second case, A before B. **d.** B before A.

B. PROCEDURAL OVERVIEW

We schedule the ASC in three connected steps. First, containers that are entering or leaving the block are scheduled. This is the primary step where a route is found that minimizes the total travel distance and satisfies all the requirements to export containers. We assume that the ASC moves containers above an exported container (reshuffles) immediately before moving an exported container. During the first step, we only reserve time to reshuffle. In the second step, we schedule the reshuffles. The third step is the scheduling of house-keeping jobs.

To formulate the first step, we divide the total path of the ASC during the time window into *m* different moves. A move starts from one side of the block

and finishes on either side. Thus, there are four different routes that the ASC can travel during a specific move. It can start from the WS and finish on the LS, or from LS to WS, or from WS to WS, or from LS to WS.

We assume that exported containers have priority, so every ASC move includes an exported container as long as one is available. The ASC may combine the move (if there is available time) with importing a container and stacking it in the block. Giving the exported containers priority should help minimize the overall consequences of unexpected delays.

After finishing with all exported containers, we schedule the ASC to move any remaining imported containers until the end of the time window. Figure 10 explains the movement of the ASC in these three different cases. For simplicity of the example, a container that is already in the stack does not have other containers above it.

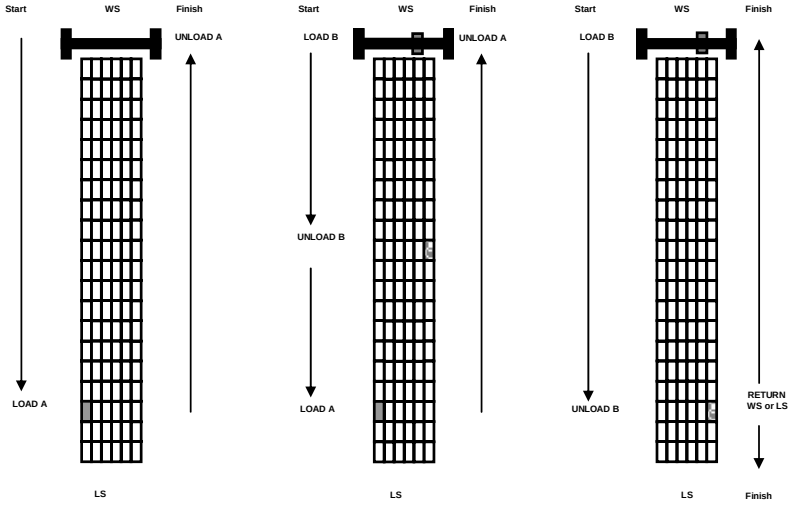


Figure 10. Three Different Types of Moves.
The arrows indicate the route and the travel distance of the ASC. **a.** The ASC picks only one exported container (A) during this move. **b.** The ASC transports an imported and an exported container during the same move. **c.** The ASC transports only an imported container during the move from the transfer point to the stack.

In the first step, an ILP finds the moves that the ASC makes and routes that it follows in order to transport the exported and imported containers. We

assume the ASC is able to transport all the exported containers within an allowable delay of being on time. To reduce ILP solve time, we do not schedule the reshuffle of any container above an exported container. We only reserve time to reshuffle. A high reserved time helps ensure sufficient time to stack them in the best position, but reduces the time for importing and exporting containers. A reserve time must be at least the sum of loading, unloading, and minimal ASC travel time.

In order to evaluate the relative merit of different positions and find the position for an ASC to place a specific container, we use a penalty system. We evaluate each stacking position in the block considering the specific container and each available stack of the block, using the following factors:

1. Height of stack - the number of containers in a stack can not exceed a height limit.
2. Size of containers - containers in each stack should be of equal size or stacked in a descending order.
3. Content – nonhazardous containers can not be mixed in the same stack with hazardous containers.
4. Status - containers with different status should be stacked in different stacks.
5. Priority - stacking a container over containers with higher priority should be avoided.
6. Buffer zone capacity - avoid leaving an imported container in the buffer zone for a long time.
7. Future saving time – we add bonus to a stack position that is close to a container's future transfer point.

In step 2, we find the best positions to stack containers above exported containers using the reserved time and the same penalty system from step 1.

In step 3, if there is time available, house-keeping jobs are assigned to the ASC. The jobs with the highest position value that satisfy time limitations are selected in order to decrease future reshuffling.

C. STEP 1 FORMULATION.

The step 1 formulation is an ILP. The objective function seeks to maximize the position values of imported containers. This objective function

encourages the ASC not to select short and fast moves that would result in a stacking with increased future travel time. A solution to this problem provides the moves that the ASC should make in order to transport all the exported containers before the time limits, and as many imported containers as possible in the best selected positions.

1. Indices

c, c´	container position in the block.	$c \in \{1,2,...,C\}$
i, i´	container from WS or LS.	$i \in \{1,2,...,I\}$
m, m´	ASC move.	$m \in \{1,2,...,M\}$
r	ASC route in one move (1 if it starts from WS and finishes on the LS , 2 from LS to WS, 3 from WS to WS , 4 from LS to LS).	
x, x´	bay.	$x \in \{1,2,...,X\}$
	(0 refers to WS and *X* refers to LS position).	
y, y´	row.	$y \in \{1,2,...,Y\}$
a	area of a block.	$a \in \{1,2,...,A\}$

2. Sets

E	positions of exported containers.
RF	available container *c* and route *r* combinations.

3. Scalars

height	maximum number of containers that can be stacked on top of one another (usually 4 or less).
idleTimeB	bonus for every minute that the ASC remains idle.
inCrPos	position of the ASC at the beginning (0 LS, 1 WS).
noImpPen	penalty for an imported container that is not stacked.
timeWindow	time period for the ASC's schedule [min].

timePerCont	time available to reshuffle and find a good location to stack a container that is above an exported container [min].

4. Parameters

$dest_c$	destination of container in position *c* (0 LS, 1 WS).
$expT_{c,r}$	travel time (reshuffling times are not included) to move a container in position *c* to its transfer point following route *r* [min].
$impT_{i,a,r}$	travel time to move an imported container *i* to a selected position in area *a* using route *r* [min].
$impExpT_{c,i,a,r}$	travel time (reshuffling times are not included) to move an imported container *i* to a selected position in area *a* and then move an exported container in position *c* to its transfer point using route *r* [min].
$inStackNum_{i,a}$	height (number of containers) in area *a* where imported container *i* could be stacked.
$limitT_c$	time the container in position *c* must be available at the LS or the WS [min].
$limMoveAbT_c$	time to remove and stack all the containers that are above an exported container in position *c* [min].
$maxValAr_{i,a}$	the value of stacking container *i* in area *a*.
$schedT_i$	the earliest imported container *i* is available [min].
xIn_c	initial *x*-position (number of bay) of container in position *c* at the beginning of the period.
$xStack_{i,a}$	*x*-position in area *a* that container *i* could be stacked.
yIn_c	initial y-position (row number) of container in position *c*.
$yStack_{i,a}$	*y*-position in area *a* where container *i* could be stacked.
zIn_c	the z-axis position for a container in position *c* (1 = bottom, 2 = second place from bottom, and so on).

5. Binary Variables

CR_m — 1 if the ASC ends move m at the WS and 0 if it ends at the LS. $\forall\ m$

$G_{i,a,m,r}$ — 1 if the ASC transports imported container i to area a during move m following route r. $\forall\ i,a,m,r$

$Q_{c,m,r}$ — 1 if the ASC transports exported container from position c during move m following route r (maybe another imported container i is transferred in the same move m). $\forall\ (c,r) \in RF, m \leq |E|$

$QG_{c,i,a,m,r}$ — 1 if the ASC transports exported container from position c and imports container i to area a. Both containers are going to be transported during move m by following route r; 0 otherwise. $\forall\ (c,r) \in RF, i, m \leq |E|$

$QQ_{c,m,r}$ — 1 if the ASC is going to transport only one exported container from position c during move m following route r (when the value is 1, it also means that no imported container i is moved). $\forall\ (c,r) \in RF, m \leq |E|$

6. Continuous Variables

DEL_m — idle time at the beginning of move m. When idle, the ASC remains either at the WS or the LS where it is at the end of move *m-1* [min]. $\forall\ m$

$SN_{i,a,m}$ — height (number of containers) in the area a stack at the end of move m where container i could be stacked. $\forall\ i,a,m$

$TOTALT_m$ — time at the end of move m [min]. $\forall\ m$

$TRAVELT_m$ — travel time during move m. It does not include the time that the ASC might remain idle in the beginning of move m [min]. $\forall\ m$

7. Objective Function

$$(1)\quad \max_{Q,G} \sum_{i,a,m,r} maxValAr_{i,a} \cdot G_{i,a,m,r} - noImpPen \cdot \left(I - \sum_{i,a,m,r} G_{i,a,m,r} \right) + idleTimeB \cdot \sum_{m} DEL_m$$

8. Constraints

(2) $$\sum_{(c,r)\in RF} Q_{c,m,r} = 1 \qquad \forall\ m \le |E|$$

(3) $$\sum_{m,r} Q_{c,m,r} = 1 \qquad \forall c \in E$$

(4) $$\sum_{i,a,r} G_{i,a,m,r} \le 1 \qquad \forall\ m$$

(5) $$\sum_{m,a,r} G_{i,a,m,r} \le 1 \qquad \forall\ i$$

(6) $$CR_m = \sum_{(c,r)\in RF} dest_c \cdot Q_{c,m,r} \qquad \forall\ m \le |E|$$

(7) $$TOTALT_m = TOTALT_{m-1} + DEL_m + TRAVELT_m + \sum_{(c,r)\in RF} limMovAbT_c \cdot Q_{c,m,r}$$

$$\forall\ \ m\ > 1$$

(8) $$TOTALT_1 = DEL_1 + TRAVELT_1 + \sum_{(c,r)\in RF} limMovAbT_c \cdot Q_{c,1,r}$$

(9) $$TOTALT_{m-1} + DEL_m \ge \sum_{i,a,r} schedT_i \cdot G_{i,a,m,r} \qquad \forall\ \ m\ > 1$$

(10) $$DEL_1 \ge \sum_{i,a,r} schedT_i \cdot G_{i,a,1,r}$$

(11)

(a) $$2 \cdot CR_{m-1} - 1 = \sum_{\substack{(c,r)\in RF,\\ r=1\ or\ r=3}} Q_{c,m,r} - \sum_{\substack{(c,r)\in RF,\\ r=2\ or\ r=4}} Q_{c,m,r} \qquad \forall m > 1 \text{ and } m \le |E|$$

(b) $$\sum_{\substack{(c,r)\in RF,\\ r=1\ or\ r=4}} Q_{c,m,r} \cdot dest_c = \sum_{\substack{(c,r)\in RF,\\ r=2\ or\ r=3}} Q_{c,m,r} \cdot (1 - dest_c) \qquad \forall\ m \le |E|$$

(c) $$\sum_{\substack{(c,r)\in RF,\\ r=2\ or\ r=4}} Q_{c,1,r} \cdot inCrPos = \sum_{\substack{(c,r)\in RF,\\ r=1\ or\ r=3}} Q_{c,1,r} \cdot (1 - inCrPos)$$

(12)

(a) $$\sum_{\substack{i,a,r \\ r=1 \text{ or } r=3}} G_{i,a,m,r} \le CR_{m-1} \qquad \forall m > |E| \text{ and } m > 1$$

(b) $$\sum_{\substack{i,a,r \\ r=1 \text{ or } r=4}} G_{i,a,m,r} \le 1 - CR_{m} \qquad \forall m > |E|$$

(c) $$\sum_{\substack{i,a,r \\ r=2 \text{ or } r=3}} G_{i,a,m,r} \le CR_{m} \qquad \forall m > |E||$$

(d) $$\sum_{\substack{i,a,r \\ r=2 \text{ or } r=4}} G_{i,a,m,r} \le 1 - CR_{m-1}$$

$$\forall m > |E| \text{ and } m > 1$$

(e) $$\sum_{\substack{i,a,r \\ r=2 \text{ or } r=4}} G_{i,a,1,r} \cdot inCrPos = \sum_{\substack{i,a,r \\ r=1 \text{ or } r=3}} G_{i,a,1,r} \cdot (1 - inCrPos) \quad \text{if } |E| = 0$$

(13) $$\sum_{i,a} G_{i,a,m,r} \le \sum_{c \in E} Q_{c,m,r} \qquad \forall m \le |E|, r$$

(14)

(a) $$QQ_{c,m,r} \ge Q_{c,m,r} - \sum_{i,a} G_{i,a,m,r} \qquad \forall (c,r) \in RF, m$$

(b) $$QQ_{c,m,r} \le 2 - \sum_{i,a} G_{i,a,m,r} - Q_{c,m,r} \qquad \forall (c,r) \in RF, m$$

(c) $$QQ_{c,m,r} \le 1 - \sum_{i,a} G_{i,a,m,r} \qquad \forall (c,r) \in RF, m$$

(d) $$QQ_{c,m,r} \le \sum_{i,a} G_{i,a,m,r} + Q_{c,m,r} \qquad \forall (c,r) \in RF, m$$

(15) $$QG_{c,i,a,m,r} \ge Q_{c,m,r} + G_{i,a,m,r} - 1 - \sum_{(cc,rr) \in RF} QQ_{cc,m,rr} \qquad \forall (c,r) \in RF, i, a, m$$

(16)

(a) $TRAVELT_m = \sum_{(c,r)\in RF, i,a} impExpT_{c,i,a,r} \cdot QG_{c,i,a,m,r} + \sum_{(c,r)\in RF} expT_{c,r} \cdot QQ_{c,m,r}$

$\forall\ m\ \leq |E|$

(b) $TRAVELT_m = \sum_{i,a,r} impT_{i,a,r} \cdot G_{i,a,m,r}$ $\forall m > |E|$

(17) $TOTALT_m \leq \sum_{(c,r)\in RF} limitT_c \cdot Q_{c,m,r}$ $\forall\ m\ \leq |E|$

(18) $TOTALT_m \leq timeWindow$ $\forall\ m$

(19) $\sum_{m'|m'\leq m,r} Q_{c,m',r} \geq \sum_{m'|m'\leq m,r} Q_{c',m',r}$ $\forall c,c' \in E, \forall m$

if $xIn_c = xIn_{c'}$ *and* $yIn_c = yIn_{c'}$ *and* $zIn_c > zIn_{c'}$ *and* $c \neq c'$

(20) $\sum_{m'|m'\leq m,r} Q_{c,m',r} \geq \sum_{m'|m'\leq m+1,r} G_{i,a,m',r}$ $\forall c \in E, i, a, m$

if $xStack_{i,a} = xIn_c$ *and* $yStack_{i,a} = yIn_c$

(21)

(a) $SN_{i,a,m} = SN_{i,a,m-1} + \sum_r G_{i',a,m,r}$ $\forall i,i',a,m$ *if* $m > 1$ *and* $xStack_{i,a} = xStack_{i',a}$ *and* $yStack_{i,a} = yStack_{i',a}$

(b) $SN_{i,a,1} = inStackNum_{i,a} + \sum_r G_{i',a,1,r}$ $\forall i,i',a$

if $xStack_{i,a} = xStack_{i',a}$

$and\ yStack_{i,a} = yStack_{i',a}\ and\ i \neq i'$

(22) $SN_{i,a,m} \leq height \qquad \forall i,a,m$

(23)

(a) $TOTALT_m, TRAVELT_m, DEL_m, SN_{i,a,m} \geq 0 \quad S_m\ URS \qquad \forall i,a,m$

(b) $Q_{c,m,r}, G_{i,a,m,r}, QQ_{c,m,r}, QG_{c,i,a,m,r}, CR_m$ binary variables $\forall c,i,a,m,r$

9. Objective Function and Constraint Description

The objective function expresses the total value for stacking imported containers as well as penalties for not placing imported containers. Constraint sets (2) and (3) require the ASC to transfer every exported container. Constraint sets (4) and (5) require the ASC to transfer at most one imported container per move. Constraint set (6) specifies the position of the ASC at the end of each move. Constraint sets (7) and (8) calculate the time at the end of move *m*. Constraint sets (9) and (10) ensure an imported container *i* is not moved before it arrives. Constraint sets (11) and (12) link binary variables $Q_{c,m,r}$, $G_{i,a,m,r}$ with CR_m and $dest_c$. Constraint set (13) links constraints between binary variables $Q_{c,m,r}$ and $G_{i,a,m,r}$. If a container in position *c* and container *i* are going to be moved in the same move *m*, then the ASC follows one route *r* during move *m*. The constraint refers to the initial moves where there are still exported containers to transport. Constraint sets (14) and (15) link constraints between binary variables $QQ_{c,m,r}$, $QG_{c,i,a,m,r}$, $G_{i,a,m,r}$, and $Q_{c,m,r,}$. Constraint set (16) calculates the travel time that the ASC needs to finish move *m*. Constraint set (17) balances time at the end of each move *m.* Constraint set (18) assigns time limits for each move *m*. Constraint set (19) assigns priority between two exported containers *c* and *c´* of the same stack. Constraint sets (20) to (22) specify limitations where an imported *i* container can be stacked. Constraint set (23) defines variable type.

10. Additional Data

There are substantial additional data needed to calculate some of the parameters for the previous formulation. We provide a description of these additional parameters as follows.

row	number of rows.
bay	number of bays.
$conflict_c$	parameter with value 1 if a container in position *c* is an exported container or a container over an exported container.
$contAb_c$	number of containers that are above a container in position *c* at the beginning of the current time period.
$content_c$	content of container in position *c* (0 empty, 1 dry, 2 fridge).
$content\_I_i$	content of imported container *i* (0 empty, 1 dry, 2 fridge).
contentPen	penalty for stacking a container over a container that has different content.
$contID_c$	the ID of the container in position *c.*
$contID\_I_i$	the ID of imported container *i.*
crAcc	acceleration of the ASC in the x-axis [meters/min^2].
$dest\_I_i$	future transfer point for imported container *i* (0 LS, 1 WS).
$impPosValue_{i,x,y}$	the value of stacking imported container *i* at the top of stack with coordinates *x* and *y*.
maxSpeed	maximum speed of the ASC in the x-axis [meters/min].
$moveT_{x,x'}$	time that the ASC needs to move from position *x* to position x' (along bay axis) [min].
overstackPen	penalty for stacking a container on a stack that already has the maximum height capacity.
$portDest_c$	the port destination for the container in position *c* (blank if it will not be loaded on a vessel).
$portDest\_I_i$	the port destination for imported container *i* (blank if it will not be loaded on a vessel).
portDestPen	penalty for stacking a container that has the same destination ship, but its destination port is later.
$priority_c$	priority of the container stacked in position *c*; the smaller the number, the higher the priority.
priorityPen	penalty for stacking a container over one with higher priority.
sameShipBonus	bonus for stacking a container on a stack where there is already a container that is going to be loaded on the same vessel.

savingTimeBonus	bonus for each minute saved from future transportation when an imported container is stacked close to the transfer point where it is going to be loaded.
$shipName_c$	the vessel for loading container in position *c* (blank if it will not be loaded on a vessel).
$shipName\_I_i$	the vessel for loading imported container *i* (blank if it will not be loaded on a vessel).
$side_i$	transfer point for imported container *i* (0 LS, 1 WS).
$size_c$	size of container in position *c* (20 - 40 - 45) [ft].
$size\_I_i$	size of the imported container *i* [ft].
sizePen	penalty for stacking a container over a container of larger size.
transferPen	penalty for stacking a container over a container that has a different future transfer point (WS or LS).
unwrapT	time the ASC needs to unwrap (unlock) a container [min].
wrapT	time the ASC needs to wrap (lock) a container [min].
$xStack_{i,a}$	x-position (number of bay) in area *a* where imported container *i* could be stacked.
$yStack_{i,a}$	y-position (number of row) in area *a* where imported container *i* could be stacked.

11. Data Preprocess

To improve solution time, we calculate the values of some data for step 1 before using it in the ILP (i.e, the travel time for each move *m*).

We divide the block into different zones to avoid some of the complexity caused by the large number of possible stacking positions, and because the value of each position dynamically changes after each move *m*. We select the best stacking position in each zone *a*, according to its stacking value, for each imported container *i* (Figure 11).

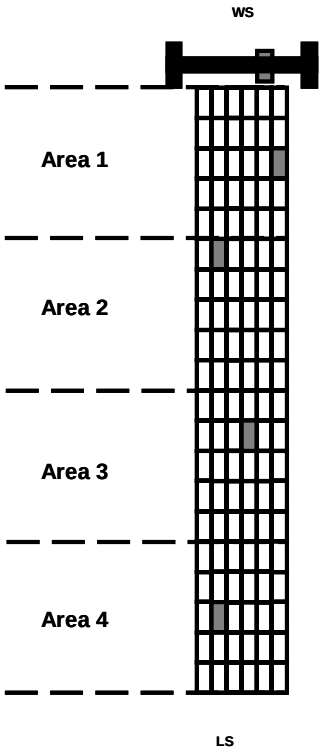


Figure 11. An Example of a Block with Four Different Areas
For each container that is going to be placed in the block during the time window, we use the best position from each area as a candidate for stacking.

The main parameters that determine the best position value for an imported container are the available time and the value of the stack. The larger the available time, the more time the ASC has to stack an imported container close to its future transfer point. Different stacking policies imply different penalty values.

The positions values for imported containers that the ASC transports during the time window are calculated without considering the exported containers. Constraints in the ILP restrict an imported container from moving to a position where an exported container is stacked before the exported container is transferred. Penalties also help avoid stacking a large container over a smaller container.

D. STEP 2 FORMULATION

In step 2, we find the optimal positions for the containers that are above exported containers. If the ILP in the first step provides an optimal (feasible)

solution, then the time to reshuffle these containers is known. These containers (in step 2) are reshuffled to new positions that have the highest value within the allowable time.

We solve a separate ILP, one for each move *m*, where an exported container has other containers above it. Preprocessing the data is necessary to reduce the solution time of the ILP. Reshuffling these containers in the same stack requires more time than removing these containers to other stacks, so it is not considered.

1. Sets

E_m	positions of exported containers during move *m*.
MC_m	containers in the same stack above an exported container that the ASC transports in move *m*.

2. Parameters

$conStackNum_{x,y,m}$	the number of containers that are in the *x* and *y* position stack at the beginning of move *m*.
$limMoveAbT_c$	time to remove and stack all the containers that are above an exported container in position *c* [min].
$moveAbT_{c,x,y}$	time that the ASC needs to move container in position *c* to a new *x* and *y* position.
$moveAbPosV_{c,x,y}$	the value of moving container in position *c* to a new *x* and *y* position.

3. Decision Variables

$QAB_{c,x,y,m}$	binary variable with value 1 if the ASC transports the container in position *c*, to the *x* and *y* position during move *m*.

4. Objective

$$(1) \quad \max_{QAB} \sum_{c \in MC_m, x, y} moveAbPosV_{c,x,y} \cdot QAB_{c,x,y,m} \qquad \forall m$$

5. Constraints

$$(2) \quad \sum_{x,y} QAB_{c,x,y,m} = \sum_{r} Q_{c',m,r} \qquad \forall\, m, c \in MC_m, c' \in E_m$$

(3) $$\sum_{c \in MC_m, x, y} moveAbT_{c,x,y} \cdot QAB_{c,x,y,m} \leq limMoveAbT_{c'} \cdot \sum_{r} Q_{c',m,r} \qquad \forall m, c' \in E_m$$

(4) $$\sum_{c \in MC_m} QAB_{c,x,y,m} + conStackNum_{x,y,m} \leq height \qquad \forall\ m, x, y$$

(5) $QAB_{c,x,y,m}$ binary variable $\forall\ \ c,\ x,\ y,\ m$

6. Objective Function and Constraint Description

The objective function expresses the total value of the positions where the ASC stacks the containers. Constraint set (2) ensures that the containers above an exported container in position *c* are moved in same move *m*. Constraint set (3) assigns the time limits. Constraint set (4) assigns the height limits in each stack, while constraint set (5) defines variable type.

7. Data Preprocess

In the data preprocess phase of step 2, we calculate the values of the parameters $moveCon_c$, $inConStackNum_{x,y}$, $moveAbT_{c,x,y}$ and $moveAbPosV_{c,x,y}$. This identifies the positions that must be moved. The calculation is dynamic, considers the containers that have been placed in the stack in the previous moves, and is made for each move *m*. If in the beginning of the next move *m+1* the ASC remains idle for some time, then we add this amount of time to the available time that the ASC has in move *m* to transport these containers.

E. STEP 3

If after steps 1 and 2 there is time available, we schedule house-keeping jobs. Because the purpose of the house-keeping jobs is to improve the block stacking, we use a penalty system to evaluate the best house-keeping jobs that can be done in each move *m* within the available time. After adjusting the time from steps 1 and 2, the ASC may be idle at the WS or LS at the beginning of some of the moves. If this idle time is higher than the time that the ASC needs to load and unload a container, then we evaluate all containers on the top of their stacks and each possible location that the ASC can place them. We select and

assign the best house-keeping job that has a position value over a specified threshold. This repeats until there is no available time to accomplish an improving house-keeping job.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SCHEDULING TWO ASCS

A. INTRODUCTION

For two ASCs working in a single block, we divide the block into two areas: WS working area and LS working area. One ASC is responsible for the demands on the LS working area and the other for the demands on the WS working area. The primary contrasts with scheduling one ASC are:

1. The ASCs cannot crossover.
2. The moves of the ASCs do not have the same duration.
3. There is a single route that each ASC can follow in each move. The WS (LS) ASC starts and ends each move at WS (LS).
4. The ASCs must maintain a security distance between them.
5. The value of placing a container in a stack changes dynamically at the end of each move of each ASC.

We use an example with three containers (A, B, and C) to help highlight some of the issues of using two ASCs (Figure 12). The destination of container C is LS so the LS ASC must be used. Containers A and B (above container C) can be reshuffled either by the WS ASC, the LS ASC, or both. If the LS ASC reshuffles containers A and B then the WS ASC must wait until the LS ASC leaves. A better approach would be to assign the reshuffling of containers A and B to the WS ASC. The LS ASC can use this time for other jobs. After the WS ASC finishes reshuffling, the LS ASC picks container C and transports it to LS transfer point.

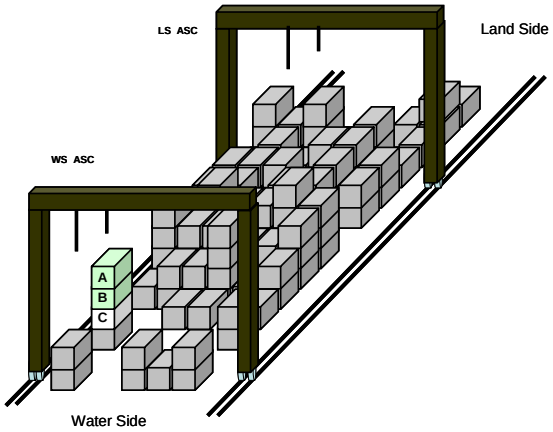


Figure 12. Two ASCs that Cannot Crossover Are Working in the Same Block Container C has to be transported to LS and containers A and B have to be reshuffled.

A reported approach for two ASCs is to use a buffer zone in the middle of the block (Figure 13). The ASC on the WS (LS) working area transports all containers stacked in the WS (LS) working area to the buffer zone that have to be moved to the LS (WS) transfer point (for example container B in Figure 13). This requires every such container to be loaded and unloaded twice.

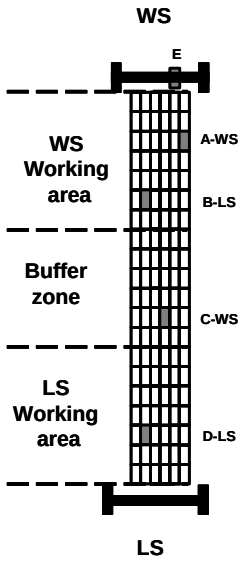


Figure 13. Two ASCs Working in a Block with Buffer Zone

We require a container that is to be delivered to the LS (WS) to be transported only by the LS (WS) ASC. This eliminates the use of a middle buffer zone.

We make the following assumptions:

1. Each imported container can be stacked only in the working area of the ASC that is responsible to pick it from the transfer point.
2. Each container above an exported container can be reshuffled in a position in the same working area.
3. The house-keeping jobs for each ASC are inside the limits of the working area of the ASC.
4. An ASC enters to the opposite working area only when it has to pick an exported container.

Under these assumptions, we may not always provide an optimal solution but the solution provided has (without considering reshuffles) the minimum number of handlings of a container from the time that it enters into the block until the time that it leaves.

B. PROCEDURAL OVERVIEW

Like the one ASC scheduling problem, we schedule the two ASCs in three steps: (1) we schedule containers that are entering or leaving the block, (2) we schedule containers that are above containers that are leaving the block and (3) we schedule house-keeping jobs.

To formulate and solve the first step problem, we divide the total path of each ASC during the time window into *m* different moves. For the ASC in the WS (LS) working area each move starts from the WS (LS) and finishes in the WS (LS).

In step 2, we find the best positions to stack containers above exported containers using the reserved time and evaluating the same penalty system of step 1.

In step 3, we identify conflicts that may occur from the solution of the previous steps and we add appropriate delays to each ASC that enters the opposite working area. If there is time where ASC remains idle on WS or LS, house-keeping jobs are assigned.

C. STEP 1 FORMULATION.

We formulate the first step as an ILP where we assume that the ASCs can crossover. This provides the order of moves that each ASC has to follow to satisfy all the demands of the imported and exported containers. The ASC in the WS (LS) working area is responsible for imported containers in the WS (LS) transfer point and for exported containers that have WS (LS) as a destination. An amount of time (not necessarily equal) is reserved for an ASC in each move in the following cases:

1. An ASC that is about to enter into the opposite working area is delayed until the other ASC finishes the current move.
2. When there are containers above an exported container that the ASC is responsible to reshuffle.
3. An ASC with its working area occupied by the opposite ASC is delayed until that ASC returns to its working area.

An ASC can enter into the opposite working area to pick an exported container only if the containers above it are already reshuffled by the other ASC. In case all exported containers for one ASC are in the opposite working area and they all have containers above them, the first move of this ASC should not include the transport of an exported container but the transport of an imported container or a reshuffling job. The description that follows includes only the elements that change from the one ASC formulation.

1. Indices

w working area (0 LS, 1 WS).

2. Sets

E_w positions of exported containers for the ASC in working area w (e.g., $c \in E_0$ in Figure 12)

I_w containers for the ASC in working area w.

V_w positions of exported containers with containers above them that must be reshuffled by an ASC in working area w before the ASC in the other area can transport the exported container (e.g., $c \in V_1$ in Figure 12).

3. Scalars

delayType1 time reserved for an ASC each time it enters into the opposite working area [min].

delayType2 time reserved for an ASC each time the other ASC enters into its working area [min].

$noImpPen_w$ penalty for an imported container in working area *w* that is not stacked.

4. Binary Variables

$G_{w,i,a,m}$ 1 if the ASC in working area *w* transports imported container *i* to area *a* during move *m*. $\forall\, w,i \in I_w,a,m$

$Q_{w,c,m}$ 1 if the ASC in working area *w* transports exported container from position *c* during move *m* (maybe another imported container *i* is transferred in the same move). $\forall\, w,c \in E_w,m$

$QG_{w,c,i,a,m}$ 1 if the ASC in working area *w* transports an exported container from position *c* and imports container *i* to area *a*. Both containers are going to be transported during move *m*. $\forall\, w,c \in E_w,i \in I_w,a,m$

$QQ_{w,c,m}$ 1 if the ASC in working area *w* transports only one exported container from position *c* during move *m* (when the value is 1 it also means that no imported container *i* is moved). $\forall\, w,c \in E_w,m$

$J_{w,c,m}$ 1 if the ASC in working area *w* transports the containers above a container in position *c* during move *m* (where the ASC in the opposite working area is going to transport the container in position *c*). $\forall\, w,c \in V_w,m$

5. Continuous Variables

$DEL_{w,m}$ idle time at the beginning of move *m* for the ASC in working area *w* [min]. $\forall\, m$

$SN_{w,i,a,m}$ height (number of containers) in the area *a* stack in working area *w* at the end of move *m* where container *i* could be placed. $\forall\, w,i \in I_w,a,m$

$TOTALT_{w,m}$ time for the ASC in working area *w* at the end of move *m* [min]. $\forall\, m$

$TRAVELT_{w,m}$ travel time for the ASC in working area *w* during move *m*. It does not include the time that the ASC might remain idle in the beginning of move *m* [min]. $\forall\, m$

6. Objective Function

(1) $$\max_{Q,G} \sum_{w,i,a,m} maxValAr_{i,a} \cdot \left(G_{w,i,a,m}\right) - noImpPen \cdot \left(I - \sum_{w,i,a,m} G_{w,i,a,m} \right) + idleTimeB \cdot \sum_{m} DEL_m$$

7. Constraints

(2) $$\sum_{c \in E_w} Q_{w,c,m} = 1 \qquad \forall\ w,m \ \ if \ \ m \le |E_w| \ \ and \ \ |V_{1-W}| < |E_w|$$

(3) $$\sum_{m|\ m \le |E_w|} Q_{w,c,m} = 1 \qquad \forall\ \ w,c \in E_w \ \ \ if \ \ \ |V_{1-w}| < |E_w|$$

(4) $$\sum_{c \in E_w} Q_{w,c,m} = 1 \qquad \forall\ \ w,m \ \ if \ \ 1 < m \le |E_w| + 1 \ \ \ and \ \ |V_{1-w}| = |E_w|$$

(5) $$\sum_{m\ |\ 1 < m \le\ |E_w| + 1} Q_{w,c,m} = 1 \qquad \forall\ \ w,c \in E_w \ \ \ if \ \ \ |V_{1-w}| = |E_w|$$

(6) $$\sum_{c \in V_w} J_{w,c,1} + \sum_{i \in I_W,\ a} G_{w,i,a,1} \ge 1 \qquad \forall\ \ w \ \ if \ \ \ |V_{1-w}| = |E_w|$$

(7) $$\sum_{i \in I_W, a} G_{w,i,a,m} \le 1 \qquad \forall\ \ \ w,m$$

(8) $$\sum_{m,a} G_{w,i,a,m} \le 1 \qquad \forall\ \ \ w,i \in I_W$$

(9) $$\sum_{m} J_{w,c,m} = 1 \qquad \forall\ \ w,c \in V_W$$

(10)

(a) $$TOTALT_{w,m} = TOTALT_{w,m-1} + DEL_{w,m} + TRAVELT_{w,m} \qquad \forall\ \ w,m > 1$$

(b) $$TOTALT_{w,1} = DEL_{w,1} + TRAVELT_{w,1} \qquad \forall\ \ \ w$$

(11)

(a) $$TOTALT_{w,m-1} + DEL_{w,m} \ge \sum_{i \in I_W,\ a} schedT_i \cdot G_{w,i,a,m} \qquad \forall\ \ w,m > 1$$

(b) $$DEL_{w,1} \ge \sum_{i \in I_W,\ a} schedT_i \cdot G_{w,i,a,1} \qquad \forall\ \ \ w$$

(12) $$\sum_{c \in V_w} J_{w,c,m} + \sum_{i \in I_W,\ a} G_{w,i,a,m} \ge \sum_{i \in I_W,\ a} G_{w,i,a,m+1} \qquad \forall\ \ w,m \ \ \ if \ \ \ |E_w| < m < M$$

(13)

(a) $$\sum_{c \in V_W} J_{w,c,m} \le 2 - \sum_{c \in E_W} Q_{w,c,m} - \sum_{i \in I_W, a} G_{w,i,a,m} \qquad \forall \; w,m$$

(b) $$\sum_{c \in V_W} J_{w,c,m} + \sum_{i \in I_W, a} G_{w,i,a,m} \le 1 \qquad \forall \; w,m$$

(14)

(a) $$QQ_{w,c,m} \ge Q_{w,c,m} - \sum_{i \in I_W, a} G_{w,i,a,m} \qquad \forall w,c \in E_w, m$$

(b) $$QQ_{w,c,m} \le 2 - \sum_{i \in I_W, a} G_{w,i,a,m} - Q_{w,c,m} \qquad \forall w,c \in E_w, m$$

(c) $$QQ_{w,c,m} \le 1 - \sum_{i \in I_W, a} G_{w,i,a,m} \qquad \forall w,c \in E_w, m$$

(d) $$QQ_{w,c,m} \le \sum_{i \in I_W, a} G_{w,i,a,m} + Q_{w,c,m} \qquad \forall w,c \in E_w, m$$

(15)

(a) $$QG_{w,c,i,a,m} \le Q_{w,c,m} \qquad \forall w,c \in E_w, i \in I_W, m$$

(b) $$QG_{w,c,i,a,m} \le G_{w,i,a,m} \qquad \forall w,c \in E_w, i \in I_W, m$$

(c) $$QG_{w,c,i,a,m} + 1 \ge Q_{w,c,m} + G_{w,i,a,m} \qquad \forall w,c \in E_w, i \in I_W, m$$

(16)

(a) $$TRAVELT_{w,m} = \sum_{c \in E_w, i \in I_W, a} impExpT_{c,i,a} \cdot QG_{w,c,i,a,m} + \sum_{c \in E_w} expT_c \cdot QQ_{w,c,m} +$$

$$+ \sum_{\substack{c \in V_w \\ or \\ c \in E_w}} limMovAbT_c \cdot \left(J_{w,c,m} + Q_{w,c,m} \right) + delayType1 \cdot \sum_{\substack{c \in E_w \; and \\ c \in V_{1-w}}} Q_{w,c,m}$$

$$\forall w,m \le |E_w| \quad if \; |V_{1-w}| < |E_w|$$

(b) $$TRAVELT_{w,m} = \sum_{c \in E_w, i \in I_W, a} impExpT_{c,i,a} \cdot QG_{w,c,i,a,m} + \sum_{c \in E_w} expT_c \cdot QQ_{w,c,m} +$$

$$+\sum_{\substack{c\in V_w \\ or \\ c\in E_w}} limMovAbT_c \cdot \left(J_{w,c,m} + Q_{w,c,m}\right) + delayType1 \cdot \sum_{\substack{c\in E_w \text{ and} \\ c\in V_{1-w}}} Q_{w,c,m}$$

$$\forall w,m \quad if \;\; 1 < m \le |E_w| + 1 \quad and \; |V_{1-W}| = |E_w|$$

(c) $$TRAVELT_{w,m} = \sum_{i\in I_W,a} impT_{i,a} \cdot G_{w,i,a,m} + \sum_{c\in V_w} limMovAbT_c \cdot J_{w,c,m}$$

$$\forall w,m > |E_w| \quad if \; |V_{1-w}| < |E_w|$$

(d) $$TRAVELT_{w,m} = \sum_{i\in I_W,a} impT_{i,a} \cdot G_{w,i,a,m} + \sum_{c\in V_w} limMovAbT_c \cdot J_{w,c,m}$$

$$\forall w,m \quad if \;\; m = 1 \;\; or \;\; m > |E_w| + 1 \quad and \; |V_{1-w}| = |E_w|$$

(17)

(a) $$0 \ge TOTALT_{w,m} - timeWindow \cdot \left(2 - Q_{1-w,c,1} - J_{w,c,m}\right)$$

$$\forall \quad w,c \;\in\; V_w,m$$

(b) $$TOTALT_{1-w,m-1} \ge TOTALT_{w,mm} - timeWindow \cdot \left(2 - Q_{1-w,c,m} - J_{w,c,mm}\right)$$

$$\forall w,c \;\in V_w,m > 1,mm$$

(18)

(a) $$TOTALT_{w,m} \le \sum_{c\in E_w} limitT_c \cdot Q_{w,c,m}$$

$$\forall w,m \le |E_w| \quad if \; |V_{1-w}| < |E_w|$$

(b) $$TOTALT_{w,m} \le \sum_{c\in E_w} limitT_c \cdot Q_{w,c,m}$$

$$\forall w,m \quad if \;\; 1 < m \le |E_w| + 1 \quad and \; |V_{1-w}| = |E_w|$$

(19) $$TOTALT_{w,m} \le timeWindow - \sum_{c\in V(w,c)} delayType2 \cdot J_{w,c,m} \qquad \forall \quad w,m$$

(20) $$\sum_{m'|m'\le m} Q_{w,c,m'} \ge \sum_{m'|m'\le m} Q_{w,c',m'} \qquad \forall w,c \in E_w,c' \in E_w,m$$

$$if \; xIn_c = xIn_{c'} \; and \;\; yIn_c = yIn_{c'} \;\; and$$

$zIn_c > zIn_{c'}$ *and* $c \neq c'$

(21) $\sum_{m'|m' \leq m} Q_{w,c,m'} \geq \sum_{m'|m' \leq m+1} G_{w,i,a,m'}$ $\forall w, c \in E_w, \forall i \in I_W, a, m$

if $xStack_{i,a} = xIn_c$ *and*

$yStack_{i,a} = yIn_c$

(22)

(a) $SN_{w,i,a,m} = SN_{w,i,a,m-1} + G_{w,i',a,m}$ $\forall w, i \in I_W, i' \in I_W, a, m$ *if* $m > 1$ *and*

$xStack_{i,a} = xStack_{i',a}$

and $yStack_{i,a} = yStack_{i',a}$

(b) $SN_{w,i,a,1} = inStackNum_{i,a} + G_{w,i',a,1}$ $\forall w, i \in I_W, i' \in I_W, a$

if $xStack_{i,a} = xStack_{i',a}$ *and*

$yStack_{i,a} = yStack_{i',a}$ *and* $i \neq i'$

(23) $SN_{w,i,a,m} \leq height$ $\forall w, i \in I_W, a, m$

(24)

(a) $TOTALT_{w,m}, TRAVELT_{w,m}, DEL_{w,m}, SN_{w,i,a,m} \geq 0, S_{w,m}$ *URS*

$\forall w, i \in I_W, a, m$

(b) $Q_{w,c,m}, G_{w,i,a,m}, QQ_{w,c,m}, QG_{w,c,i,a,m,}, J_{w,c,m}$ binary variables

$\forall w, c \in E_w, i \in I_W, a, m$

8. Objective Function and Constraint Description

The objective function expresses the total value for stacking imported containers and penalties for not placing imported containers. Constraint sets (2) to (5) define the moves where the ASC transfers all exported containers. Constraint set (6) requires the ASC not to transfer exported container in the first move if all of the exported containers are on the opposite working area and have containers above them. Constraint sets (7) and (8) require the ASC to transfer at most one imported container per move. Constraint set (9) ensures that the ASC reshuffles all containers that are above exported containers in its working area.

Constraint set (10) calculates the time at the end of move *m*. Constraint set (11) ensures an imported container *i* is not moved before it arrives. Constraint set (12) requires that each move includes at least one job. Constraint set (13) determines the possible combinations between jobs in each. Constraint sets (14) and (15) link constraints between binary variables $QQ_{w,c,m}$, $QG_{w,c,i,a,m}$, $G_{w,i,a,m}$ and $Q_{w,c,m,}$. Constraint set (16) calculates the travel time that the ASC needs to finish move *m*. Constraint set (17) ensures that the ASC will not enter to the opposite working area to pick an exported container before the opposite ASC reshuffles all the containers above it. Constraint set (18) balances time at the end of each move *m.* Constraint set (19) assign time limit constraints for each move *m*. Constraint set (20) assigns priority between two exported containers *c* and *c´* of the same stack. Constraint sets (21) to (23) specify limitations where an imported *i* container can be stacked. Constraint set (24) defines variable type.

B. STEP 2 FORMULATION

In the second step, we find the optimal positions for the containers that are above exported containers. An ASC reshuffles a container to a new position inside the limits of its working area. The description that follows includes only the elements that change from the one ASC formulation.

1. Sets

$E_{w,m}$ positions of exported containers in working area *w* during move *m*.

$MC_{w,m}$ positions of containers in the same stack and above an exported container that the ASC in working area *w* transports in move *m*.

2. Decision Variables

$QAB_{w,c,,x,y,m}$ binary variable with value 1 if the ASC in working area *w* transports the container in position *c* which is above an exported container, to a new *x* and *y* position during move *m*.

$QOP_{w,c,,x,y,m}$ binary variable with value 1 if the ASC in working area *w* transports the container in position *c* which is above an exported container that is going to be picked from the opposite ASC, to a new *x* and *y* position during move *m*.

3. Objective

(1) $\max_{QAB,\, QOP} \sum_{c \in MC_{w,m},\, x,\, y} moveAbPosV_{c,x,y} \cdot \left(QAB_{w,c,x,y,m} + QOP_{w,c,x,y,m}\right) \quad \forall\ w, m$

4. Constraints

(2)

(a) $\sum_{x,\, y} QAB_{w,c,x,y,m} = Q_{w,c',m} \quad \forall \quad w,m,c' \in E_{w,m} \ \ and \ \ c' \notin V_w, c \in MC_{w,m}$

(b) $\sum_{x,\, y} QOP_{w,c,x,y,m} = J_{w,c',m} \quad \forall \quad w,m,c' \in V_w, c \in MC_{w,m}$

(3)

(a) $\sum_{c \in MC_{w,m},x,y} moveAbT_{c,x,y} \cdot QAB_{w,c,x,y,m} \le limMoveAbT_{c'} \cdot Q_{w,c',m}$

$\forall \quad w,m,c' \in E_{w,m} \quad and \quad c' \notin V_w$

(b) $\sum_{c \in MC_{w,m},x,y} moveAbT_{c,x,y} \cdot QOP_{w,c,x,y,m} \le limMoveAbT_{c'} \cdot J_{w,c',m}$

$\forall \quad w,m,c' \in V_w$

(4) $\sum_{c \in MC_{w,m}} QAB_{w,c,x,y,m} + \sum_{c \in MC_{w,m}} QOP_{w,c,x,y,m} + conStackNum_{x,y,m} \le height$

$\forall \quad w,m,x,y$

(5) $QAB_{w,c,x,y,m}$, $QOP_{w,c,x,y,m}$ binary variable $\forall \quad w,\ c,\ x,\ y,m$

5. Objective Function and Constraint Description

The objective function expresses the total value of the positions where the two ASCs stack containers. Constraint set (2) ensures that the containers above an exported container in position *c* will be moved in same move *m*. Constraint set (3) assigns the time limits. Constraint set (4) assigns the height limits in each stack. Constraint set (5) defines the variables.

C. STEP 3

In step 3, the exact position for each ASC for every small time step is calculated in order to identify points of crossover. When a crossover occurs then the appropriate delay time (*delayType1* or *delayType2*) is added to create a feasible schedule. The amount of the delay that we add also must include a

security distance that we want the two ASCs to have between them at all times. After we achieve the elimination of the crossovers between the two ASCs, we assign house-keeping jobs during the idle times of each move for each ASC.

V. COMPUTATIONAL STUDY

A. OBJECTIVE

We test the performance of one ASC, measure it under different types of blocks, and compare it with the performance of two ASCs. We implement the ILPs and the preprocessing algorithms in GAMS [GAMS 2005] and solve the ILPs using CPLEX [ILOG 2003]. The step 1 ILP consists about 2,600 continuous variables, 2,400 binary variables and 5,500 constraints. The step 2 ILP consists about 6,000 continuous variables, 6,000 binary variables and 170 constraints.

We address the following questions:

1. How does the length of the block affect the performance of one and two ASCs?
2. How does the fullness of the block affect the performance of one and two ASCs?
3. When do two ASCs perform better than one ASC?
4. How fast are the two algorithms and how sensitive are they in scheduling different types of blocks?

B. DESCRIPTION

The test data for the study are from the port of Rotterdam and refer to a block with 60 bays. Each bay has six rows and a maximum height of four containers, a total of 1,440 available positions for container stacking. The number of containers that are initially stacked in the block is 318 (22.08% block area fullness). Based on the original block of 60 bays, we create two blocks for testing, one with length of 20 bays and 22.08% fullness (106 containers), and one with length of 20 bays and 66.24% fullness (318 containers).

In this study, we measure the performance of one or two ASCs during a time period of four hours. We assume we only know the containers in the current status of the block and the containers that are entering or leaving the block in the next fifteen minutes. For each imported container, we know when it arrives, and for every exported container, we know the latest it must be available at its transfer point. After the ASC finishes with all required containers or reaches fifteen minutes, we consider the next time window of fifteen minutes. Although

the containers that need to be exported or imported to the WS (vessels) are often known for more than fifteen minutes, truck arrivals at the gate for the LS of a block are more unpredictable, so fifteen minutes seems like a reasonable time limit.

At the end of the four-hour time period, imported containers might still be stacked in the transfer points, implying that the ASC could not transport all imported containers on time.

C. ASSUMPTIONS

We make some basic assumptions to have comparable results:

1. The ASCs have the same characteristics and capabilities.
2. The penalty values (stacking policy) are the same for both algorithms and remain constant during the four hours. In general, because there was no information about the future container schedule, we prefer a low stack policy where empty stacks are preferred locations for imported containers.
3. Imported and exported containers that need to be transported during the four hours are separated in 16 different batches. In the case of one ASC, if the ASC finishes all scheduled jobs (imported and exported) before the end of the fifteen minutes, then it starts working on the next batch of containers. In the case of two ASCs, the next scheduling starts when both of the ASCs finish their jobs.
4. No unexpected delays or ASCs breakdowns.
5. The imported containers and their arrival time remain constant in all cases.
6. The exported containers, their export time, and their placement remain constant in the blocks of 20 bays with different fullness. We use different exported containers stacked in different positions in the comparison between 20 and 60 bays blocks, where the total number of them remains constant in both cases.
7. If an ASC cannot transport an imported container during the time window of fifteen minutes, the container remains in the transfer point, and it can be imported in the next time window. There is a buffer zone capacity limit of five containers on each side. For each batch, an ASC transports all exported containers before the time demand (if possible) and all imported containers that are necessary to keep the buffer zone below the capacity limit.

8. If an ASC cannot transport an exported container during the time window of fifteen minutes, there is a delay in the following schedule until the demand for the exported containers is satisfied (possibly causing delay to the delivery of the next batch of exported containers).

D. RESULTS

1. Bay Size Factor

We measure the performance of one or two ASCs in two different blocks with bay size 20 and 60 and present the results in Table 1. The fullness of the blocks in both cases is 22%.

ASCs	Bay	Block fullness	Number imported	Number exported	Number reshuffles	Enter to opposite Working Area	Travel time (min)	Total time (min)
1	20	22%	57	53	38	-	95.3	243.3
1	60	22%	57	53	41	-	243.7	394.7
2	20	22%	57	53	38	11	81.5	154.5
2	60	22%	57	53	41	21	204.7	279.7

Table 1. Performance of One and Two ASCs in Blocks with Different Bays.

The results above show that the number of bays significantly affects the performance of one and two ASCs. As the total number of bays is tripled, the total time for the one ASC increases by a factor of 1.62, and for the two ASCs a factor of 1.81. The factor for two ASCs is higher because the number of times that an ASC enters into the opposite working area is almost double in the 60-bays block, and this causes additional traveling and idle time.

In comparing the performance between the one and two ASCs, for a 20-bay block, the one ASC needs 57.4% more time than the two ASCs to finish all the jobs. In the 60-bays block, the one ASC needs 41.1% additional time.

2. Number of Containers

Table 2 shows the performance of one or two ASCs in two different blocks with bay size 20 and block fullness 22% and 66%.

ASCs	Bay	Block fullness	Number imported	Number exported	Number reshuffles	Enter to opposite Working Area	Travel time (min)	Total time (min)
1	20	22%	57	53	38	-	95.3	243.3
1	20	66%	57	53	47	-	111.1	268.1
2	20	22%	57	53	38	11	81.5	154.5
2	20	66%	57	53	45	11	80.6	157.6

Table 2. Performance of One and Two ASCs in Blocks with Different Fullness.

The results above indicate that when the fullness increases from 22% to 66%, the total time for only one ASC increases 10.2%. In the case of two ASCs, the total time increases only 2%. The main reason for this result is the increase in the number of reshuffles for one and two ASCs when 66% full. The increase is less in the two ASC case because each ASC, on average, makes 5.5 reshuffles instead of 11 reshuffles for one ASC.

When 22% full, one ASC needs 54.5% more time than two ASCs to finish its schedule. When 66% full, one ASC needs 70.1% more time.

3. Two ASCs Route Graphs

The following graphs (Figures 14, 15 and 16) display the bay position of each ASC in every time step of 0.125 min (7.5 sec). The bay number 0 refers to WS position where the ASC either remains idle or loads or unloads containers at the WS transfer point. The bay number 21 (or 61) refers to LS position where the ASC either remains idle or loads or unloads containers at the LS transfer point.

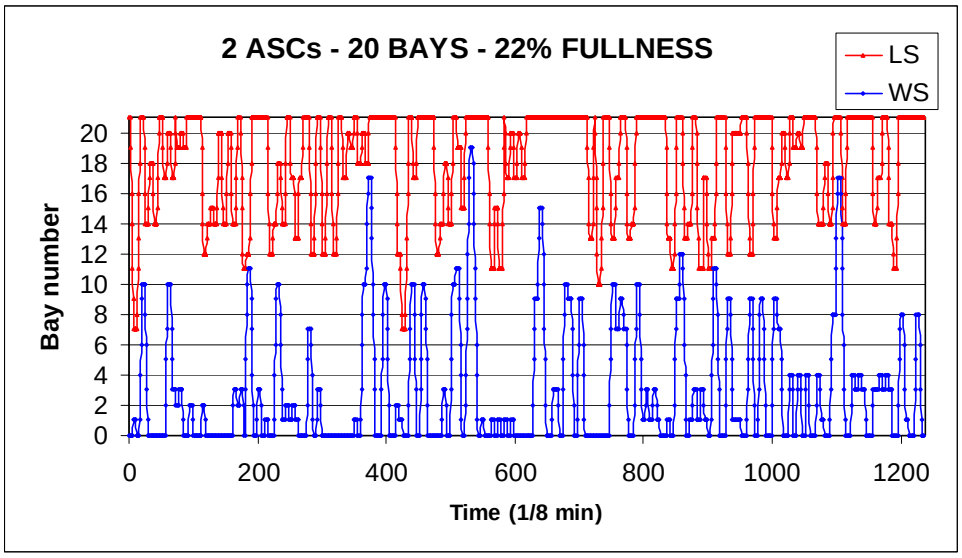


Figure 14. Graph of Two ASCs Performing in a Block with 20 Bays and 22% Fullness.

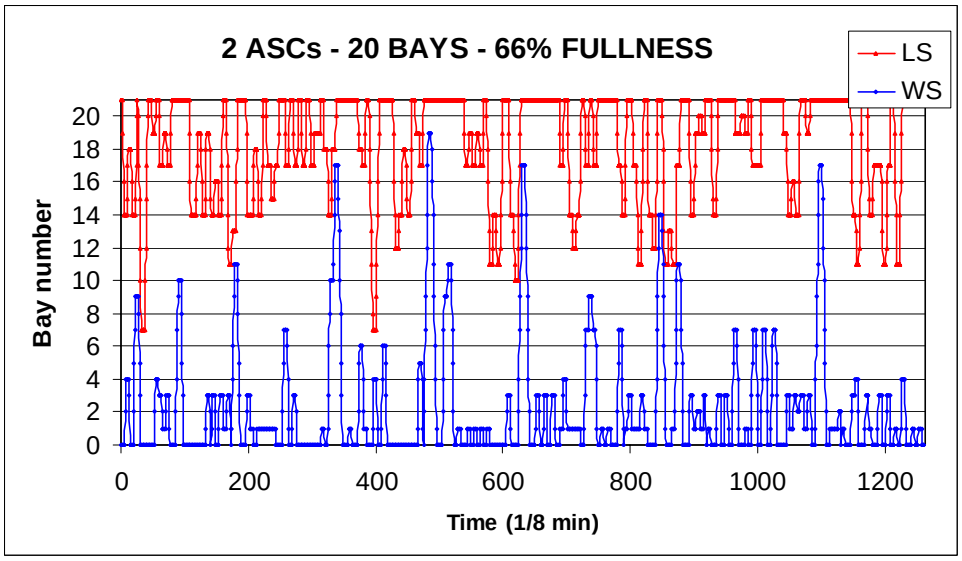


Figure 15. Graph of Two ASCs Performing in a Block with 20 Bays and 66% Fullness.

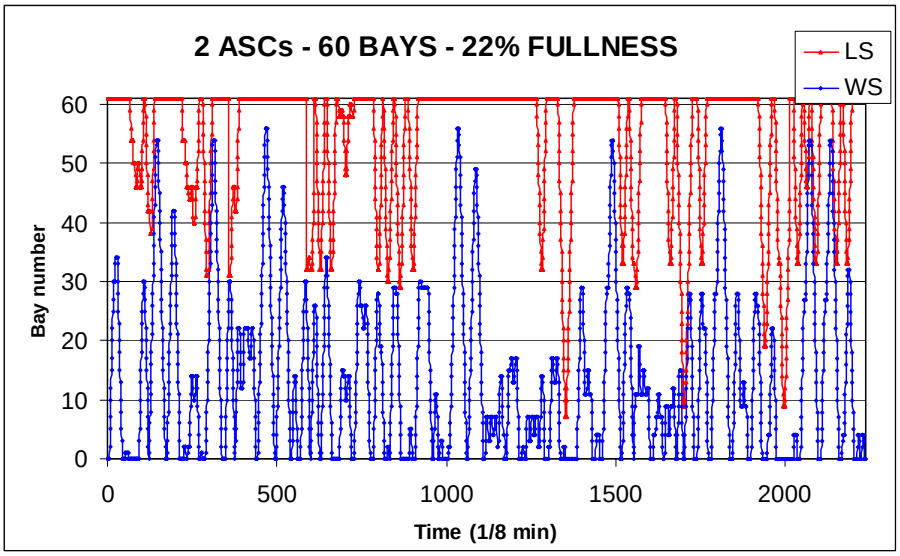


Figure 16. Graph of Two ASCs Performing in a Block with 60 Bays and 22% Fullness.

Apparently there are some long time periods where one of the two ASCs remains idle at its transfer point. The reasons for this are:

a) We have no information about the future workload and cannot identify house-keeping jobs that reduce future reshufflings.

b) An ASC finishes its assigned jobs earlier than the other ASC because it has less workload to satisfy. This ASC remains idle because we assume that the information of a new batch of container that needs scheduling in both sides of the block arrives when both ASCs finish their scheduled jobs.

c) A sufficient security distance is required when an ASC enters to the opposite working area.

d) An ASC is loading or unloading a container the transfer point.

4. Algorithm Running Time

We report (Table 3) the run times of the ILPs in different types of blocks.

						1st Step		2nd Step	
ASCs	Bay	Block fullness	Aver. total time (sec)	Min total time (sec)	Max total time (sec)	Aver. time per batch	Max time (sec)	Aver. time per batch	Max time (sec)
1	20	22%	10.50	4	20	<1	2	<1	2
1	60	22%	51.50	38	75	<1	2	<2	4
1	20	66%	13.81	4	29	<1	2	<1	2
2	20	22%	28.50	5	57	<1	2	<1	2
2	60	22%	156.31	40	499	<1	2	<2	4
2	20	66%	41.56	7	111	<1	2	<1	2

Table 3. Run Time of the Algorithms

The 1st and 2nd step columns describe the time CPLEX uses to solve the ILPs and not any preprocessing computation. The results show that the average time to solve the ILP in the first step is always less than one second and in the second step is always less than two seconds. The total time includes the time that GAMS needs to generate the ILP, preprocess the data, solve the problem and produce the output. This time increases as the number of bays increases mostly because the number of possible stacking positions in step 2 also increases and more calculations need to be made. This generation time could be significantly reduced by writing a custom generator that is compiled in place of the interpreted GAMS code.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

This thesis is the first to develop Integer Linear Programs (ILPs) to prescribe routes for one and two equal sized Automated Stacking Cranes (ASCs) in a single block working with straddle carriers to load and unload containers.

We compare the performance of one and two ASCs working in blocks with different characteristics during a four-hour period. Using real world data, we find that the length and the fullness of the block significantly affect the performance of one ASC. For two ASCs, only the length of the block influences performance. We also find that the performance of the two ASCs is dependent on the number of times that an ASC enters to the opposite working area. As a general conclusion, two ASCs outperform one ASC in all our test cases.

In addition to ASCs studied in this thesis, there are other carriers and yard cranes currently used for transporting and stacking containers. Future work should consider extending results from this thesis to blocks that use other equipment. Additionally, the characteristics of a block in the performance of one or two yard cranes could be further investigated to gain insight into various port stacking policies.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Aston, A., Arndt, M., and Zellner, W., 2005, “Choked Ports, Pricier Products,” http://www.businessweek.com/magazine/content/05_19/b3932126.htm - check of address: December 01, 2005.

Ayik, M., 2005, Personal conversation with Dr. M. Ayik, Advanced Applications Manager, May 2005, Navis Llc.

Ceres Paragon, 2005, http://www.ceresglobal.nl/resourcecenter/photos.asp - check of address: December 01, 2005.

Eisenberg, R., Stahlbock, R., Vob, S., and Steenken, D., 2003, “Sequencing and scheduling of movements in an automated container yard using double rail-mounted gantry cranes,” Working paper. University of Hamburg.

Foxcroft, A., 2002, “Balancing the Books,” *Containerisation*, Vol. 35, pp. 45-47.

GAMS, 2005, http://www.gams.com - check of address: December 01, 2005.

Gottwald Port Technologies, 2005, http://www.gottwald.com/start.htm - check of address: December 01, 2005.

Henesey, E., 2004, “Enhancing Container Terminal Performance: A Multi Agent Systems Approach,” PhD in Systems and Software Engineering, Blekinge Institute of Technology, Sweden, http://www.ipd.bth.se/lhe/Lic.pdf - check of address: December 01, 2005.

ILOG, 2003, CPLEX 9.0, ILOG Inc., Mountain View, California.

Ioannou, A., Kosmatopoulos, B., Vukadinovic, K., Liu, I., Pourmohammadi, H., and Dougherty, E., 2000, “Real time testing and verification of loading and unloading algorithms using Grid Rail (GR),” Final report for Center for Advanced Transportation Technologies. University of Southern California.

Kalmar Industries, 2005, http://www.kalmarind.com - check of address: December 01, 2005.

Kim, H. and Kim, Y., 1997, “A routing algorithm for a single transfer crane to load export containers onto containership,” *Computers & Industrial Engineering*, Vol. 33, pp. 673-676.

Kim, H. and Kim, Y., 1999, “An optimal routing algorithm for a transfer crane in port container terminals,” *Transportation Science*, Vol. 33, pp. 17-33.

Kozan, E. and Preston, P., 1999, “Genetic algorithm to schedule container transfers at multimodal terminals,” *International Transactions in Operational Research*, Vol. 6, pp. 311-329.

Lin, W., 2000, “On dynamic crane deployment in container terminals,” MS of philosophy in industrial engineering and engineering management, Hong Kong University of Science and Technology, http://www.isye.gatech.edu/linq/mthesis.pdf - check of address: December 01, 2005.

Linn, R., Liu, J., Wan, Y., Zhang, C., and Murty, K., 2003, “Rubber tired gantry crane deployment for container yard operation,” *Computers & Industrial Engineering*, Vol. 45, pp. 429-442.

SPARCS Terminal Operation System – User’s Manual, 2004, Version 3.4, Navis LLC.

Stenken, D., Vob S., and Stahlbock R., 2004, “Container terminal operation and operations research-a classification and literature review,” *OR Spectrum*, Vol. 26, pp. 3-49.

Volk B., 2002, “Growth factors in container shipping,” Australian Maritime College, http://maritimebusiness.amc.edu.au/papers/AMC3_GRO.pdf - check of address: December 01, 2005.

Zhang, C., Wan, Y., Liu, J., and Linn, R., 2002, “Dynamic crane deployment in container storage yards,” *Transportation Research*, Vol. B 36, pp. 537-555.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. NAVIS LLC
   Oakland, California

4. Associated Professor Robert Dell
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

5. Research Assistant Professor Johannes O. Royset
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

6. Captain Ioannis Zyngiridis
   Command and General Staff College
   Thessaloniki, Hellas