2009-05

Optimizing Container Movements Using One and Two Automated Stacking Cranes

Royset, Johannes O.

JOURNAL OF INDUSTRIAL AND MANAGEMENT OPTIMIZATION Volume 5, Number 2, May 2009 pp. 285-302

http://hdl.handle.net/10945/48823

JOURNAL OF INDUSTRIAL AND
MANAGEMENT OPTIMIZATION
Volume 5, Number 2, May 2009

doi:10.3934/jimo.2009.5.285

pp. 285–302

OPTIMIZING CONTAINER MOVEMENTS USING ONE AND TWO AUTOMATED STACKING CRANES

ROBERT F. DELL AND JOHANNES O. ROYSET

Operations Research Department
Naval Postgraduate School
Monterey, CA 93943, USA

IOANNIS ZYNGIRIDIS

Command and General Staff College
Hellenic Army
Thessaloniki, Hellas

(Communicated by Irene Loiseau)

ABSTRACT. Productivity of a sea port depends, in part, on stacking cranes working in blocks of its storage yard. Each container leaving a block must be moved by a storage-yard crane to a buffer zone during a specific time window so it can reach its destination on time. Containers entering a block for storage must be moved out of the buffer zone sufficiently soon to avoid overflow. In this paper, we formulate integer linear programs to prescribe movements to transport and stack containers in storage yards using one and two equally-sized Automated Stacking Cranes (ASCs) working with straddle carriers. Using real world data, we construct test problems varying both the number of container bays and fullness of the block. We find that one ASC working alone requires up to 70% more time than two ASCs working together to accomplish the same container movements. Optimal solutions of the integer linear programs are typically obtained in only a few seconds.

1. **Introduction.** The overall efficiency of maritime container terminals depends on its storage yard operations. We formulate and solve integer linear programs to prescribe crane movements to transport and stack containers in storage yards. Container terminals vary based on their layout and the equipment they use to transport containers. Figure 1 portrays a container terminal for our purposes and its division into water side (WS), storage yard, and land side (LS). When a container vessel arrives at its berth, the ship-to-shore movement starts. Quay cranes lift containers from the vessel and move them to shore, where carriers transport them to the storage yard's WS buffer zones. This reverses when containers move from the storage yard's WS buffer zones to ships. Similar to the WS activity, carriers transport containers to and from the storage yard's LS buffer zones and the areas for truck or train loading and unloading. Carriers are either manned trucks, automated ground vehicles, or straddle carriers. Manned trucks and automated ground vehicles need cranes for loading and unloading, unlike straddle carriers that handle containers without crane assistance and thereby permit the use of buffer zones.

2000 *Mathematics Subject Classification.* Primary: 90C10; Secondary: 90B06.

Key words and phrases. maritime container terminal, storage yard operations, automated stacking crane scheduling, integer linear programming.

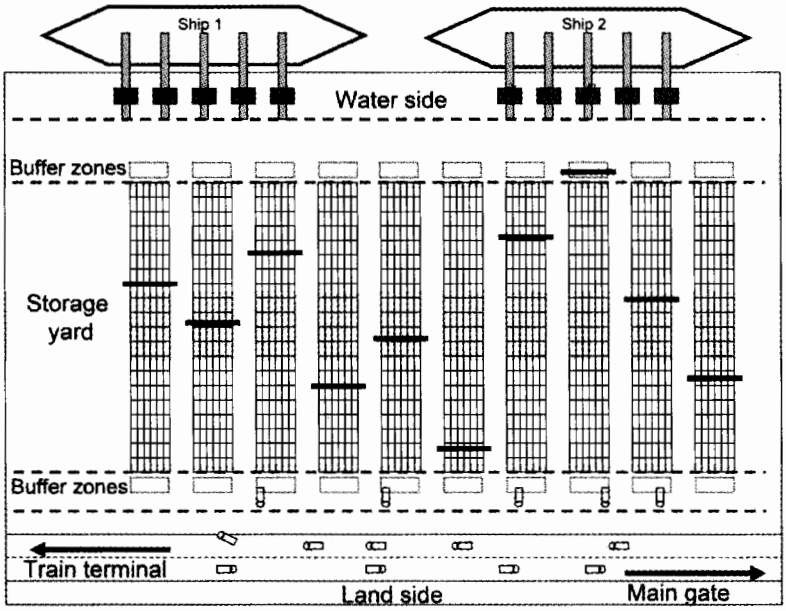

FIGURE 1. Crane moves containers between block and buffer zones

A storage yard consists of multiple blocks each with a specific number of bays, rows, and tiers where containers are stored in stacks (Figure 2) until they are either moved to the WS for loading onto a ship or moved to the LS for transportation inland. In each block, one or more storage yard cranes move containers between buffer zones and the block. There are three main types of storage yard cranes: rubber-tired gantry cranes that can move between blocks, rail-mounted gantry cranes that move on rail tracks in a specific block and rarely or never move between blocks, and Automated Stacking Cranes (ASCs) that move on rails in a specific block and are fully automated. Each of these cranes lifts one container at a time. To improve block operations, many ports employ two cranes in the same block. In most of these cases, the two cranes are of the same size and type and cannot crossover.

In this paper, we optimize the moves of ASCs working with straddle carriers. Our motivation to consider this combination comes from Navis, Llc [1], a port logistics software company, in response to one of their customers considering such equipment for future port operations. Ideally, the next crane, carrier, and container movements would be simultaneously optimized for an entire port while also accounting for future requirements (especially LS requirement). Such an optimization process would need to be quick enough for on-line calculations as requirements change and equipment completes their tasks. It would also need to accurately estimate future requirements. We consider both one and two ASCs working in a single block; just one piece of this overall optimization that we find can be solved quickly and provide needed ASC schedules. In the case of two ASCs, we assume that they are of the same type and cannot crossover. Because straddle carriers can pickup and drop off containers without assistance from cranes, ASCs do not exactly time their moves with straddle carriers. An ASC only needs to move containers out of its block no later than the scheduled pickup time by a straddle carrier. We assume containers arriving at and departing from a block are known in advance for a given time window and

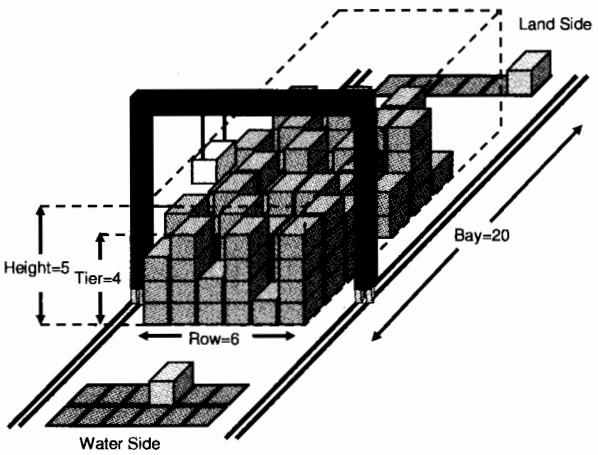

FIGURE 2. Overview of a maritime container terminal

information beyond that time is not available. Hence, it is reasonable to adopt a moving time-horizon defined by the time window and only indirectly account for the later (unknown) requirements. We also assume buffer zone capacity is unlimited because we anticipated only a moderate rate of containers arriving at and departing from a block.

We define a container that moves into the block from either the LS or WS as imported and a container that moves out of the block to a buffer zone as exported. These definitions differs from some authors' use of exported containers to mean containers being loaded on a ship and imported containers to mean containers designated for the LS.

Section 2 presents a brief overview of the recent literature. Sections 3 and 4 present our approach for one and two ASCs as well as our formulations for one ASC. Section 5 includes numerical studies. Section 6 has some conclusions.

2. **Literature review.** Recent surveys [5, 19] provide overviews of strategic and operational planning and analysis of maritime container terminals including studies of berth allocation, quay crane scheduling, WS and LS transportation, and storage yard operations. [14, 15, 22] discuss strategic allocation of storage yard cranes among blocks for rubber-tired gantry cranes, and in [21] for rail-mounted gantry cranes.

Several authors examine where to place containers arriving in a storage yard. In [11], the authors reserve space in one or more blocks for containers designated for a specific ship before the containers arrive at the terminal. They formulate the resulting optimization problem as an integer linear program and solve it using two heuristics. In [18], the authors optimize placement of containers designated for a ship with known departure time and solve the resulting optimization problem using a genetic algorithm. The placement problem is also considered in [20] where they demonstrate that, in addition to the marshalling area near the quay cranes and the main storage yard, a "rough pile" temporary-storage area is beneficial. In [10], they optimize the placement of individual containers designated for a ship over all possible locations reserved for that ship and obtain an optimal solution using

dynamic programming. Container reservations for a particular ship are computed using the approach in [11]. [16] provides simple, general rules for finding a "good" location for a container.

[8, 9, 10] address scheduling crane movements in a storage yard. The authors only consider crane moves necessitated by containers leaving a block for the WS, and not simultaneous export and import to and from the LS and WS buffer zones. They propose three solution techniques: a dynamic-programming based heuristic, a genetic algorithm, and a breath-first-search based heuristic. In [17], the authors consider containers leaving a block for the WS and present exact, branch-and-bound and heuristic algorithms for optimizing crane moves. In [7], the authors determine crane moves undertaken to place containers in their "ideal" position for loading a particular ship. Dynamic programming is used to match bays where the containers are currently located with bays where the containers should ideally be placed. The number of containers moved between various bays is determined by solving transportation problems. The solution of a traveling salesman problem with precedence constraints provides a minimum travel-time sequence of crane moves to achieve the required container moves.

In [4, 13], the authors approach crane optimization in storage yards as machine-job scheduling where container moves are jobs assigned to machines (cranes). They include side constraints to account for restrictions such as precedence relations (some containers must move before others) and spatial limitations in the case of multiple cranes working in the same block. They assume that the required jobs are known in advance, i.e., the exact placement of containers in the block is given. They do not optimize the placement of the containers.

To the authors' knowledge, there are no published results on scheduling one and two ASCs working in a block with straddle carriers, with simultaneous need for moving containers between the block and the WS and LS buffer zones, and with a goal of optimization both crane moves and container placement. We address this situation.

3. Mathematical formulation for one ASC.

An ASC working in a block must export containers to the LS and WS buffer zones in time for pickup by straddle carriers and, simultaneously, import containers designated for the block from the buffer zones. The placement of imported containers must account for containers already in the stack as well as available time for the current placement and subsequent tasks.

Containers designated for storage in a block (containers scheduled to leave a block) are typically known at least 15-20 minutes before they arrive (prior to their pickup by straddle carriers) in the buffer zones of that block [1]. More advanced warning is rarely possible due to uncertainty surrounding other activities at the terminal. Hence, we adopt the following moving time-horizon approach for scheduling an ASC working in a specific block:

(i): Determine a time horizon (typically 15 minutes).

(ii): Collect all relevant data about, and timing of, containers arriving at the block's LS and WS buffer zones and containers leaving the block during the time horizon.

(iii): Schedule moves of the ASC for the whole time horizon with the goal of satisfying demands given in (ii). If time allows, moves may also be scheduled to improve the organization of the block (house-keeping).

(iv): Execute the planned moves. Towards the end of the time horizon, go to (i) and repeat planning for the next time period.

We assume that the information collected in (ii) generates a (deterministic) schedule of arriving containers at its buffer zones, and a (deterministic) time-restricted list of containers to be removed from the block. In the following, we derive integer linear programs (ILPs) for determining the schedules in (iii). Even with few containers entering and leaving a block during a time horizon, there are a large number of possible moves for the ASC. In addition, the merit of placing a container at a particular location in the block changes each time the block's content and organization change. Consequently, the combinatorial nature of the ASC scheduling problem can preclude exact solutions in short computing times. Because we aim for on-line scheduling in a moving time-horizon and need quick solution times, we opt to impose a modest restriction on the problem by decomposing it into three steps. The steps are: (1) scheduling containers entering and leaving the block during the time horizon, (2) scheduling the relocation of containers stacked above leaving containers, and (3) scheduling reorganization of the block to improve the overall stacking.

We expect the decomposition to produce good results but overall performance depends on several characteristics of the block. Step 1 is the primary step in our decomposition as it prescribes the simultaneously scheduling of all entering and leaving container moves during the entire time horizon. Steps 2 and 3 provide repositioning of containers currently in the block with the goal of being able to improve future operations within the block. The importance of steps 2 and 3 is a function of how many containers are stacked above leaving containers (step 2) and how containers are currently positioned within the block (step 3). Indeed, if there are no containers on top of leaving containers and containers are currently in ideal locations for future operations, steps 2 and 3 would be unnecessary.

Step 1 consists of finding a path for the ASC that exports containers to their designated buffer zones in time for pickup during the time horizon and maximizes a user-defined objective function which rewards imports of containers. We impose an additional restriction on the scheduling problem by assuming that the ASC moves (reshuffles) containers stacked above a leaving container immediately before moving the leaving container. During step 1, we only reserve time for this reshuffle. In step 2, we schedule the reshuffles identified in step 1.

To formulate an ILP for Step 1, we divide the total path of the ASC during the time horizon into moves (referred to using the index m or m'). A move starts from one side of the block and finishes on either side. Thus, there are four different routes (indexed by r) 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 LS. A move may involve both importing a container and exporting a different container. Figure 3 displays some possible routes.

We assume that exported containers have priority over imported containers, so every ASC-move includes an exported container as long as one is available. This priority reduces the chance of having a crane failure or some other unexpected event causing delay before all containers have been exported. Note that a delay of exported containers propagates quickly to other parts of the container terminal. Whereas, containers arriving at the block's buffer zones can normally wait to be imported into the block without delaying other operations. Of course, arriving containers must eventually be imported to avoid overflow of the buffer zones. However,

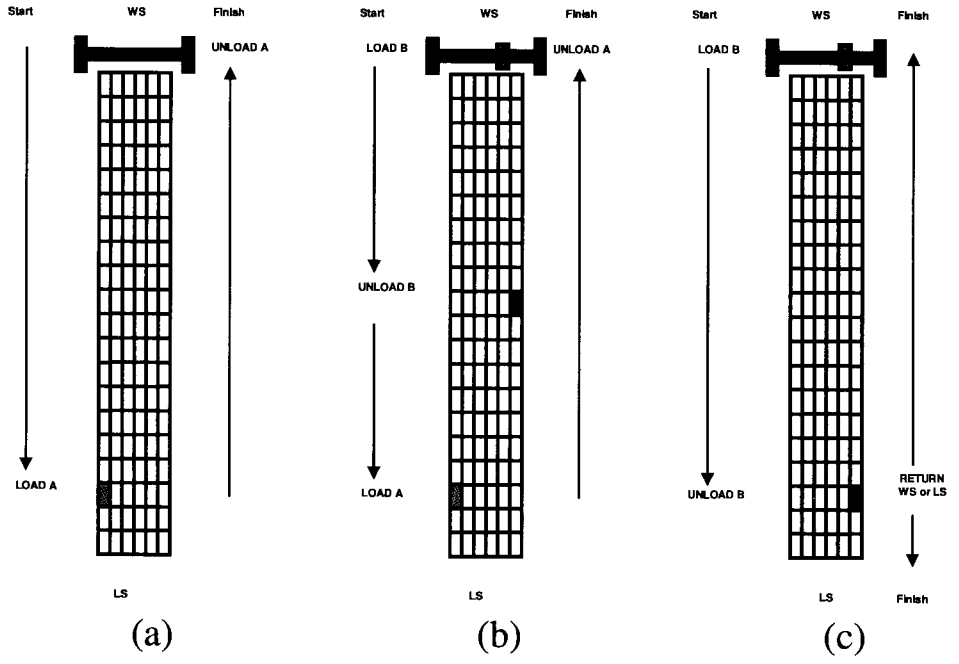

FIGURE 3. Different ASC routes. (a) ASC exports one container only. (b) ASC imports and exports. (c) ASC imports one container only

we do not explicitly model the buffer zone capacities. The model simply rewards the move of containers out of the buffer zones.

In order to evaluate the merit of placing a container at a given position in the block we use a value system based on the factors: height of stacks, size and content of containers (e.g., hazardous containers must be separated from others), container priority, expected time in storage yard, and final destination. We combine these factors to provide one numerical position value for each potential location for a considered container. We note that only positions on top of stacks are considered. Other positions require moving containers located above the considered positions, which is expected to be impractical. To reduce the number of imported container placement options for Step 1, we divide the block into areas and only consider the best position within each area for possible placement.

The time reserved for step 2 for each set of containers is decided before solving the step 1 ILP. An output of the step 1 ILP is the amount of time reserved for step 3. A parameter of the step 1 ILP controls the tradeoff between time spent positioning containers in step 1 and reserving time for step 3.

We now present the ILP for Step 1 in NPS standard format [2]. An optimal solution to this program prescribes moves resulting in all required containers being exported and as many imported containers as possible being placed in the block.

Integer Linear Program for Step 1 - One ASC

Indices

a	area of block, $a \in \{1, 2, ..., A\}$
e, e'	container to be exported, $e \in \{1, 2, ..., E\}$
i, i'	container to be imported, $i \in \{1, 2, ..., I\}$
m, m'	ASC move, $m \in \{1, 2, ..., M\}$
r	ASC route (1 if it moves from WS to LS, 2 from LS to WS, 3 from WS to WS, 4 from LS to LS). Figure 3 shows some examples.

Sets

$(e, e') \in ABOVE$	set of all container pairs (e, e') to be exported where container e is above container e' in a stack
$(e, r) \in RF$	available container e and route r combinations
$(e, i) \in SIE_a$	set of exported and imported container pairs (e, i) that can be stacked in the same place in area a
$i' \in SSI_i$	set of all imported containers that can be stacked in the same place as imported container i

Parameters

$aHeight_{i,a}$	height available at position with best position value in area a where container i can be placed
$arriveT_i$	arrival time of container i in buffer zone [min]
$dest_e$	destination of container e (0 LS, 1 WS)
$idleBonus$	bonus for every minute ASC remains idle
$inOutT_{e,i,a,r}$	additional time to import container i to the best position in area a when also moving container e to its buffer zone using route r (does not include time to move container e) [min]
$inT_{i,a,r}$	time to import container i to the best position in area a using route r [min]
$outT_{e,r}$	time (reshuffling not included) to move export container e to its buffer zone using route r [min]
$placeT_e$	time when container e must be in buffer zone [min]
$restackT_e$	time to restack containers above export container e [min]
$start$	position of ASC at the beginning (0 LS, 1 WS)
$timeLimit$	time horizon [min]
$valIn_{i,a}$	value of stacking container i in the best position in area a

Nonnegative Variables

$ENDT_m$	time at the end of move m [min], $\forall m$
$IDLE_m$	idle time [min] at the beginning of move m, $\forall m$
$TRAVT_m$	travel time during move m [min], $\forall m$

Binary Variables

$BOTH_{e,i,a,m,r}$	1 if container e is exported and container i is imported to the best position in area a during move m using route r; 0 otherwise, $\forall (e,r) \in RF, i, m \leq E$
$IN_{i,a,m,r}$	1 if container i is imported to area a during move m using route r, $\forall i, a, m, r$
$OUT_{e,m,r}$	1 if container e moves during move m using route r, $\forall (e,r) \in RF, m \leq E$
WS_m	1 if move m ends at WS and 0 if at LS, $\forall m$

Formulation

$$\max \sum_{i,a,m,r} valIn_{i,a} IN_{i,a,m,r} + idleBonus \sum_{m} IDLE_m \tag{1}$$

s.t.

$$\sum_{(e,r)\in RF} OUT_{e,m,r} = 1 \ \forall m \leq E \tag{2}$$

$$\sum_{m,r} OUT_{e,m,r} = 1 \ \forall e \tag{3}$$

$$WS_m = \sum_{(e,r)\in RF} dest_e OUT_{e,m,r} \ \forall m \leq E \tag{4}$$

$$\sum_{(e,r)\in RF, r\in\{2,4\}} startOUT_{e,m,r} = \sum_{(e,r)\in RF, r\in\{1,3\}} (1 - start) OUT_{e,m,r} \text{ for } m = 1 \tag{5}$$

$$2WS_{m-1} = \sum_{(e,r)\in RF, r\in\{1,3\}} OUT_{e,m,r} - \sum_{(e,r)\in RF, r\in\{2,4\}} OUT_{e,m,r} \ \forall 1 < m \leq E \tag{6}$$

$$\sum_{(e,r)\in RF, r\in\{1,4\}} dest_e OUT_{e,m,r} = \sum_{(e,r)\in RF, r\in\{2,3\}} (1 - dest_e) OUT_{e,m,r} \ \forall m \leq E \tag{7}$$

$$\sum_{i,a,r} IN_{i,a,m,r} \leq 1 \ \forall m \tag{8}$$

$$\sum_{m,a,r} IN_{i,a,m,r} \leq 1 \ \forall i \tag{9}$$

$$\sum_{r\in\{1,3\},i,a} IN_{i,a,m,r} \leq WS_{m-1} \ \forall m > E; \text{ if } E > 0 \tag{10}$$

$$\sum_{r\in\{2,4\},i,a} IN_{i,a,m,r} \leq 1 - WS_{m-1} \ \forall m > E; \text{ if } E > 0 \tag{11}$$

$$\sum_{r\in\{1,4\},i,a} IN_{i,a,m,r} \leq 1 - WS_m \ \forall m > E \tag{12}$$

$$\sum_{r\in\{2,3\},i,a} IN_{i,a,m,r} \leq WS_m \ \forall m > E \tag{13}$$

$$\sum_{r\in\{2,4\},i,a} startIN_{i,a,m,r} = \sum_{r\in\{1,3\},i,a} (1 - start) IN_{i,a,m,r} \text{ for } m = 1; \text{ if } E = 0 \tag{14}$$

$$\sum_{i,a} IN_{i,a,m,r} \leq \sum_{e} OUT_{e,m,r} \ \forall m \leq E, r \tag{15}$$

$$BOTH_{e,i,a,m,r} \leq OUT_{e,m,r} \ \forall (e,r) \in RF, i, a, m \leq E \tag{16}$$

$$BOTH_{e,i,a,m,r} \leq IN_{i,a,m,r} \ \forall (e,r) \in RF, i, a, m \leq E \tag{17}$$

$$BOTH_{e,i,a,m,r} \geq OUT_{e,m,r} + IN_{i,a,m,r} - 1 \ \forall (e,r) \in RF, i, a, m \leq E \tag{18}$$

$$ENDT_m = ENDT_{m-1}|_{m>1} + IDLE_m + TRAVT_m + \sum_{(e,r)\in RF} restackT_e OUT_{e,m,r} \ \forall m \tag{19}$$

$$ENDT_{m-1}|_{m>1} + IDLE_m \geq \sum_{i,a,r} arriveT_i IN_{i,a,m,r} \ \forall m \tag{20}$$

$$\begin{aligned} TRAVT_m \ = \ & \sum_{(e,r)\in RF,i,a,} inOutT_{e,i,a,r} BOTH_{e,i,a,m,r} \\ & + \sum_{(e,r)\in RF} outT_{e,r} OUT_{e,m,r} \ \forall m \leq E \end{aligned} \tag{21}$$

$$TRAVT_m = \sum_{i,a,r} inT_{i,a,r} IN_{i,a,m,r} \ \forall m > E \tag{22}$$

$$ENDT_m \leq \sum_{(e,r)\in RF} placeT_e OUT_{e,m,r} \ \forall m \leq E \tag{23}$$

$$ENDT_m \leq timeLimit \ \forall m \tag{24}$$

$$\sum_{r} OUT_{e',m,r} \leq \sum_{m' \leq m,r} OUT_{e,m',r} \ \forall (e,e') \in ABOVE, m \tag{25}$$

$$\sum_{r} IN_{i,a,m,r} \leq \sum_{m' \leq m-1,r} OUT_{e,m',r} \ \forall a, (e,i) \in SIE_a, m \tag{26}$$

$$\sum_{r} IN_{i,a,m,r} = 0 \ \forall a, (e,i) \in SIE_a, \text{for } m = 1 \tag{27}$$

$$\sum_{r,m,i' \in SSI_i} IN_{i',a,m,r} \leq aHeight_{i,a} \ \forall i, a \tag{28}$$

$$ENDT_m, TRAVT_m, IDLE_m \geq 0, \forall m$$

$$OUT_{e,m,r} \in \{0,1\}, \forall e, m, r, IN_{i,a,m,r} \in \{0,1\}, \forall i, a, m, r$$

$$BOTH_{e,i,a,m,r} \in \{0,1\}, \forall e, i, a, m, r, WS_m \in \{0,1\}, \forall m$$

The objective function expresses the total value of placed containers as well as a bonus for idle time. Hence, the model seeks to import as many containers as possible and to place them at their most favorable locations. Such placement indirectly accounts for tasks beyond the time horizon not explicitly scheduled in

the ILP. Quicker moves are preferred and the objective function includes a bonus for time-saving moves. The magnitude of the idle bonus is determined by the user. Constraint sets (2) and (3) ensure that all required containers are exported. We note that if all containers cannot be exported within the time horizon, i.e., the program becomes infeasible, the user must prioritize the containers and postpone those with lowest priority. Constraint set (4) specifies the position of the ASC at the end of moves. Constraint set (5-7) ensures correct starting position for each move. Constraint sets (8) and (9) ensure that at most one container is imported per move. Constraint sets (10-18) link various binary variables. Constraint set (19) calculates the time at the end of moves. Constraint set (20) ensures that imported containers are not moved before they arrive in the buffer zone. Constraint sets (21) and (22) calculate the travel time needed to finish a move. Constraint set (23) balances time at the end of moves. Constraint set (24) limits time for each move. Constraint set (20) prioritizes between two leaving containers in the same stack. Constraint sets (26-28) limit where an imported container can be stacked.

After obtaining an optimal solution from the Step-1 ILP, the need for reshuffling containers is known. In Step 2 of our procedure, we find the optimal positions for containers located above leaving containers. Obviously, these containers must be moved (reshuffled) to allow access to the leaving containers. A prescription from the Step-2 ILP places reshuffled containers in new positions with the highest value reachable within the allowable time. We calculate the value of potential positions as we calculate the position values in Step 1.

In Step 2 we solve a set of ILPs, one for each move in Step 1 with containers above a leaving container. We note that the ILPs of Step 2 are solved sequentially with data being updated after each ILP-solution to reflect changes to the block. Reshuffling containers and returning them to the same stack requires a significant amount of time and, hence, is not considered. All reshuffling consists of moving containers above a leaving container to other stacks. We now present the ILP for Step 2.

Integer Linear Program for Step 2 - One ASC (for a given move m)

Indices

c	container above the container exported in move m, $c \in \{1, 2, ..., C_m\}$
x	bay (1 is nearest WS, X is nearest LS), $x \in \{1, 2, ..., X\}$
y	row, $y \in \{1, 2, ..., Y\}$

Parameters

$aTiers_{x,y,m}$	number of empty tiers at x, y in the beginning of move m
$resMaxT_m$	time available to reshuffle during move m [min]
$resT_{c,x,y}$	time to move container c to position x, y [min]
$valRes_{c,x,y}$	value of reshuffling container c to position x, y

Decision Variables

$RES_{c,x,y,m}$	1 if container c is reshuffled to position x, y

Formulation (for a given m)

$$\max \sum_{c,x,y} valRes_{c,x,y} RES_{c,x,y,m} \tag{29}$$

s.t.

$$\sum_{x,y} RES_{c,x,y,m} = 1 \ \forall c \tag{30}$$

$$\sum_{c,x,y} resT_{c,x,y} RES_{c,x,y,m} \leq resMaxT_m \tag{31}$$

$$\sum_{c} RES_{c,x,y,m} \leq aTiers_{x,y,m} \ \forall x, y \tag{32}$$

$$RES_{c,x,y,m} \in \{0,1\} \ \forall c, x, y$$

The objective function expresses the total value of reshuffled containers. Constraint set (30) ensures that containers above a leaving container are moved. Constraint set (31) assigns time limits. Constraint set (32) assigns height limits.

In step 3, we schedule house-keeping jobs if time is available. These jobs attempt to improve container placement in the block by moving some containers to new positions so future requests can be satisfied quicker. For example, a container designated for the LS, but currently stored near the WS could be moved closer to LS to facilitate future export of the container. We calculate the value of potential positions as we calculate the position values in Step 1. We use a simple heuristic to evaluate the best house-keeping jobs that can be achieved within the available time. Hence, no ILP is solved in Step 3. Specifically, the heuristic evaluates all possible locations within the available time and their position values for all containers on top of stacks. The heuristic selects a container and a new position with the best value and subtracts the time required for the selected move from the available time. This repeats until there is no available time to accomplish an improving house-keeping job.

4. **Mathematical formulation for two ASCs.** We now consider two identical ASCs working in the same block (Figure 4). We divide the block into two parts: the LS work area and the WS work area. Each work area has a primary ASC. ASCs can move outside their work area. Because the two ASCs cannot crossover, interaction between the ASCs complicates schedules. To circumvent this difficulty and facilitate a generalization of our optimization model for one ASC, we restrict the movements of ASCs slightly.

Specifically, we impose the following restrictions: (i) every leaving container designated for the LS (WS) buffer zone and every entering container sitting in the LS (WS) buffer zone must be transported by the LS (WS) ASC only, (ii) every entering container sitting in the LS (WS) buffer zone must be placed in the LS (WS) work area only, and (iii) a container above a leaving container in the LS (WS) work area is reshuffled by the LS (WS) ASC to another position in the LS (WS) work area. Restriction (i) may have the following effect when a container located in the WS work area is designated for the LS buffer zone: the LS ASC moves into the WS work area to pickup the container and move it to the LS buffer zone. During part of this move, the WS ASC may be idle at the WS buffer zone to avoid obstructing the other ASC.

We note that the proposed restriction tends to waste less time than a work-division rule frequently used in practice. This rule imposes a buffer zone in the middle of the block and restricts the ASCs to work in different ends of the block. When a container needs to be transported from one end of the block to the other, one ASC moves it to the mid-block buffer zone, where it is unloaded, and later picked up by the other ASC and moved to its destination. Hence, when applying this rule some containers are loaded and unloaded twice resulting in increased handling time.

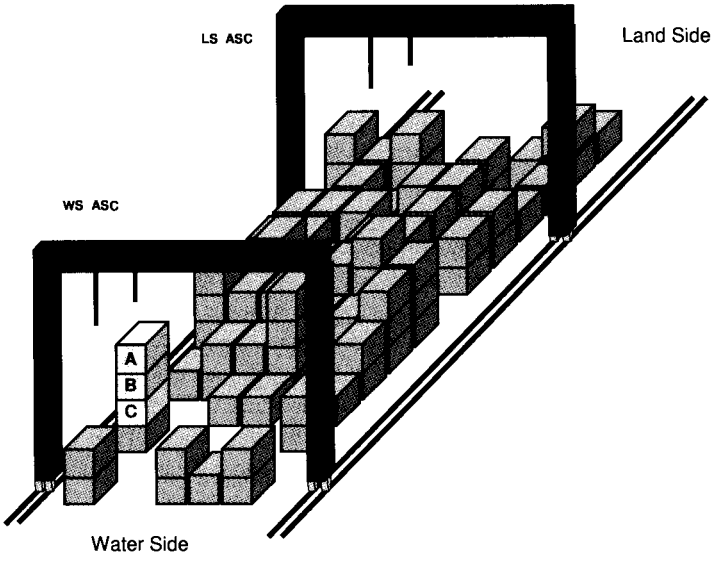

FIGURE 4. Two identical ASCs working in the same block

In view of the restriction imposed above, we now generalize the three-step for one ASC (Section 3). In Step 1, we schedule containers entering and leaving the block. As in Section 3, we divide the total path of an ASC during the time horizon into moves. In contrast to the one-ASC case with four possible routes per move, we now have only one possible move for each ASC. A move for the WS ASC starts and ends at the WS buffer zone. Similarly, a move for the LS ASC starts and ends at the LS buffer zone. Consequently, ASCs may finish their respective moves at different points in time.

As in the one-ASC case, we derive an ILP for Step 1. This ILP is similar to the one in Section 3, but it also incorporates the three restrictions imposed above. Furthermore, to avoid obstructing the other ASC, an ASC about to enter the opposite work area waits until the other ASC finishes its current move. An ASC may also wait for the other ASC to reshuffle containers above a leaving container it is about to move. And finally, an ASC, with its work area occupied by the other ASC, must wait until the other ASC returns to its work area before it can start its next move. A detail description of the ILP is found [23].

As in Section 3, Step 2 of our procedure consists of finding the best positions for containers located above leaving containers during the time reserved for this

activity in Step 1. However, we now impose the additional restriction (as above) that reshuffled containers are placed in the same work area as their current position. We note that an ASC may reshuffle containers while waiting for the other ASC to complete its current work area. The Step-2 ILP is similar to the one in Section 3 and a complete description is given in [23].

As in Section 3, Step 3 of our procedure consists of scheduling house-keeping jobs. Contrary to the one-ASC case where house-keeping jobs took place only at the end of the time horizon, we may now have several additional time windows available. Every time window when an ASC is idle, i.e., no task is scheduled in Step 1 and 2, we examine the possibility of house-keeping jobs. We employ the same heuristic as in Section 3 with the additional restriction that we limit an ASC's house-keeping job to the ASC's work area and it cannot interfere with the other ASC.

5. **Computational experience.** In this section, we provide some computational experience suggesting that the proposed procedures are suitable for online decision support with a 15-minute moving time-horizon. We examine a numerical example constructed using real-world data from the Port of Rotterdam. We note that the Port of Rotterdam uses ASCs with AGVs (automated guided vehicles), not straddle carriers, but such data appears reasonable for AGVs and straddle carriers as well. Our motivation to consider ASCs working with straddle carriers was in response to a port operator considering such equipment for future port operations. In our computational study we investigate how the use of one or two ASC, number of bays, and block area fullness could influence future port operations.

The data from Port of Rotterdam describe a block with 60 bays. Each bay has six rows and a maximum height of four containers, a total of 1,440 available container positions. Initially, the block contains 318 containers (22% block area fullness). Based on this block of 60 bays, we artificially create two blocks, one with 20 bays and 22% fullness (106 containers), and one with 20 bays and 66% fullness (318 containers).

In this example, we schedule one and two identical ASCs during a typical "half-day" work period involving 57 and 53 imported and exported containers, respectively. We adopt a 15-minute moving time-horizon, where each 15-minute period involves a variable number of imported and exported containers usually in the range of 3 to 12 containers. The activity on the WS and LS is approximately the same in each 15-minute period, which highlights the differences between the one- and two-ASC cases. We assume that all information about importing and exporting containers for the next 20 minutes is known at any point in time. This allows for five minutes data transfer and computing time for the three-step procedure described in Sections 3 and 4. The next 15-minute time-horizon starts when the previous time-horizon ends or when all entering and leaving containers of the previous time-horizon have been imported and exported, whichever event occurs first. Hence, there is no idle time towards the end of a time horizon for the one-ASC case. The two-ASC case may have one idle crane towards the end of a time-horizon, but not both ASCs.

For the given data, the Step-1 ILPs consist of about 2,600 continuous variables, 2,400 binary variables and 5,500 constraints. The Step-2 ILPs consist of about 6,000 continuous variables, 6,000 binary variables and 170 constraints. Note that an ILP of Step 1 is solved only once, while a Step-2 ILP is solved once for each leaving container blocked by some other container(s). We implement the ILPs and

associated data processing algorithms in GAMS [3] and solve the ILPs using CPLEX [6] on a standard desktop computer.

Table 1 reports computing times for ILPs in different types of blocks. Column four of Table 1 gives the total computing time for the three-step procedure, averaged over all the time horizons. The total computing time includes generation, run, and output times for the ILPs of Step 1 and 2 as well as the Step-3 heuristic. Column five and six of Table 1 show the minimum and maximum total computing times, respectively, over all the time horizons. Column seven and eight display the maximum run time (excluding problem generation and output times) over the time horizons for Step-1 and 2 ILPs, respectively. Note that the last column may represent run times required to solve multiple Step-2 ILPs. It is observed that the ILPs are always solved in just a few seconds of run time due in part to tight linear programming relaxations. Most of the computing time consists of problem generation. The problem generation time could be reduced significantly by using a different computational platform than GAMS. Even with GAMS, the total computing times are rarely more than 5 minutes. Hence, our three-step procedure can easily be incorporated into an online decision support system.

TABLE 1. Computing times (sec.) of ILPs.

ASCs	Bays	Fullness	Aver. total time	Min total time	Max total time	Max run time Step 1	Max run time Step 2
1	20	22 %	11	4	20	2	2
1	60	22 %	52	38	75	2	4
1	20	66 %	14	4	29	2	2
2	20	22 %	29	5	57	2	2
2	60	22 %	156	40	499	2	4
2	20	66 %	42	7	111	2	2

We compare the performance of one and two ASCs in two different blocks with bay size 20 and 60 and present the results in Table 2. The fullness of the blocks is 22%. Columns three, four, and five of Table 2 give the total number of containers imported, exported, and reshuffled over all time horizons, respectively. Column six specifies the total number of times an ASC moves into the opposite work area (irrelevant for the one-ASC case). "Travel and idle time" (column seven) is the time needed for crane movements added to the time spent waiting. "Total time" (column eight) adds the time to load and unload containers to the travel and idle time total.

The results of Table 2 show that the number of bays significantly affects the performance of one and two ASCs. As the total number of bays is tripled, the travel and idle time increases by factors of 2.6 and 2.5 for the one and two ASCs, respectively. We note that the 60-bay case leads to more entries into the opposite work area compared to the 20-bay case. However, it does not result in a disproportional increase in travel and idle time compared to the one-ASC case. In fact, travel and idle time increases with approximately the same factor in both cases when the number of bays is tripled. In the 20-bay block, Table 2 gives that one ASC needs 57% more time than two ASCs to finish the same job. In the 60-bay block, that

percentage is 41%. This suggests as the number of bays increases the advantage of having two ASCs does not similarly increase.

TABLE 2. ASCs in blocks with different number of bays (22% fullness).

ASCs	Bays	Number Imported	Number Exported	Number Reshuffled	Times in opposite workarea	Travel & idle time (min)	Total time (min)
1	20	57	53	38	-	95	243
1	60	57	53	41	-	244	395
2	20	57	53	38	11	82	155
2	60	57	53	41	21	205	280

Table 3 shows the performance of one and two ASCs in two different blocks with bay size 20 and block fullness 22% and 66%. The results in Table 3 indicate that when the fullness increases from 22% to 66%, the total time to complete the job for one ASC increases by 10%. In the case of two ASCs, the total time increases only 2%. This effect is due to the increase in number of reshuffles for both one and two ASCs when fullness is increased. The increase in reshuffles is less in the two-ASC case because each ASC, on average, makes 3.5 reshuffles instead of 9 reshuffles as is the case for one ASC. When the block is 22% full, one ASC needs 55% more time to finish its schedule than two ASCs. When the block is 66% full, one ASC needs 70% more time. These results suggest the advantage of using two ASCs grows as the block fullness increases.

TABLE 3. Performance of ASCs in blocks with different fullness (20 bays).

ASCs	Full ness	Number Imported	Number Exported	Number Reshuffled	Times in opposite workarea	Travel & idle time (min)	Total time (min)
1	22	57	53	38	-	95	243
1	66	57	53	47	-	111	268
2	22	57	53	38	11	82	155
2	66	57	53	45	11	81	158

Figures 5, 6 and 7 display the bay position of ASCs every 7.5 second during the time it takes to complete all the moves in the various cases. Bay number zero refers to the WS position, where the ASC either remains idle or loads/unloads containers at the WS buffer zone. Bay number 21 (or 61) refers to the LS position, where the ASC either remains idle or loads/unloads containers at the LS buffer zone. Each of the figures displays both the WS (lower part of figure) and LS (upper part of figure) ASCs. We observe from Figures 5-7 that one of the two ASCs is occasionally idle (marked with a horizontal line in Figures 5-7). Idleness can be caused by an ASC finishing its jobs earlier than the other ASC and by an ASC being blocked by the other ASC entering its work area.

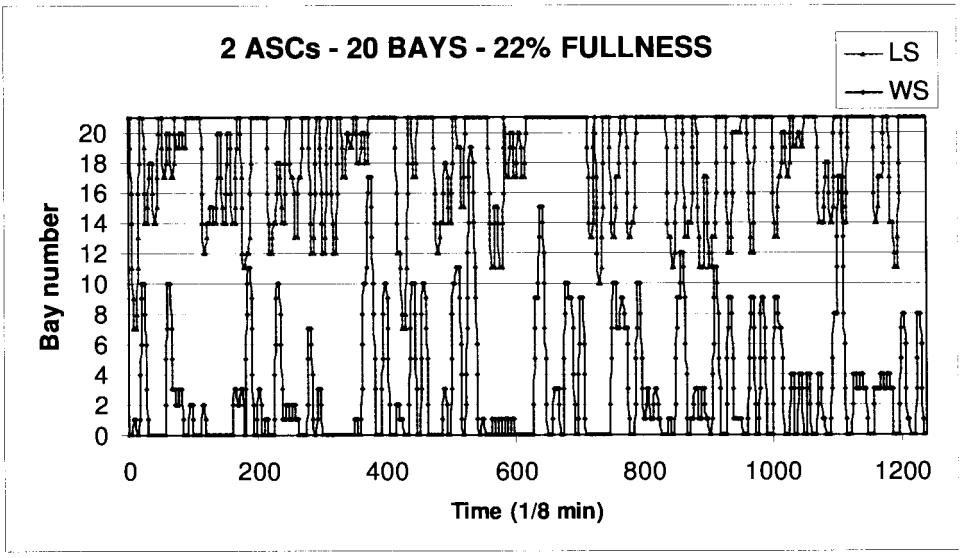

FIGURE 5. Positions of two ASCs in a block with 20 bays and 22% fullness

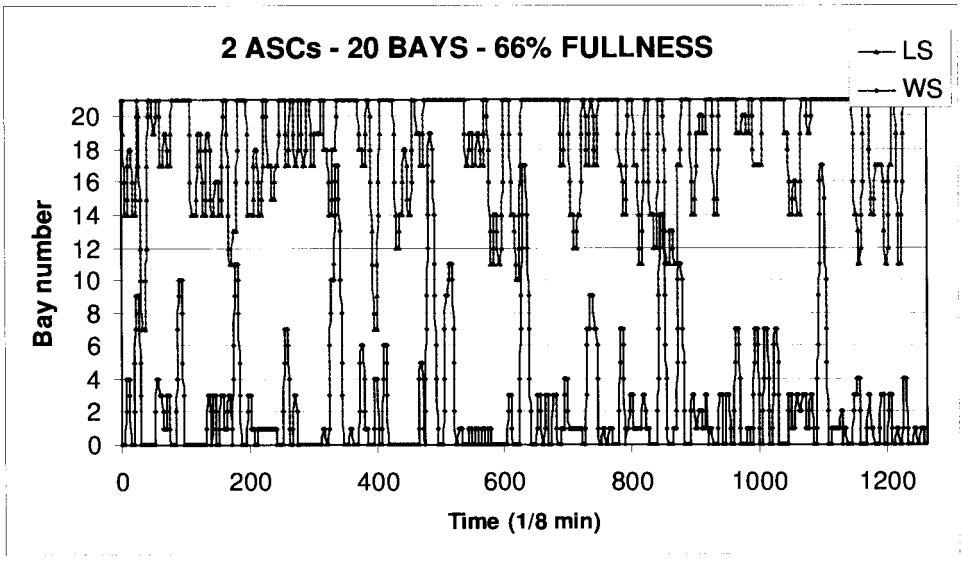

FIGURE 6. Position of two ASCs in a block with 20 bays and 66% fullness

6. **Conclusions.** This paper develops integer linear programs for scheduling one and two equally-sized Automated Stacking Cranes (ASCs) working in a single block with straddle carriers. The integer linear programs solve, on average, in 150 seconds or less and, hence, are suitable for implementation in online decision support

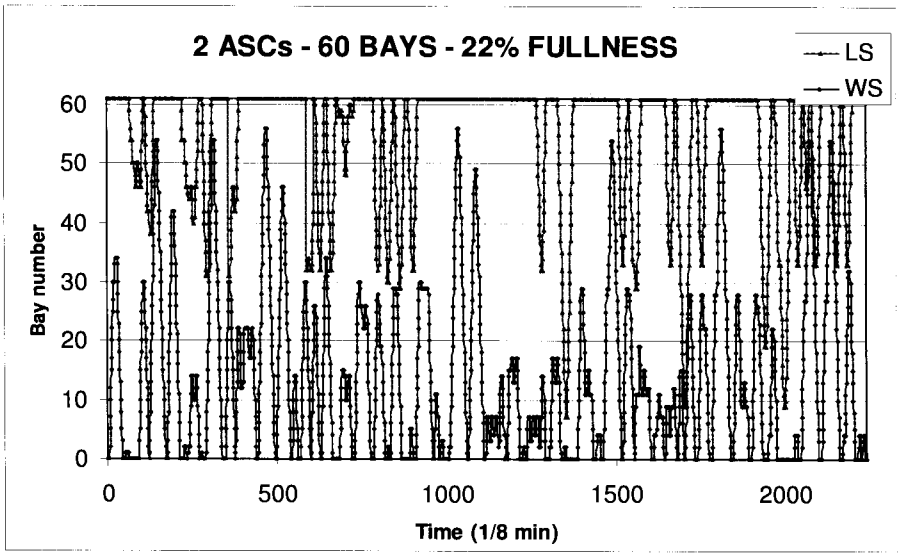

FIGURE 7. Position of two ASCs in a block with 60 bays and 22% fullness

systems. We compare the performance of one and two ASCs working in blocks with different characteristics over several hours. Using real world data, we find that length and fullness of a block significantly affect the performance of one ASC. For two ASCs working together in the same block, only length influences performance. We also find that the performance of two ASCs is dependent on the number of times an ASC enters the opposite work area. Generally, two ASCs outperform one ASC in all our test cases.

REFERENCES

[1] M. Ayik, "Personal Communication," Navis Llc, Oakland, California, May, 2005.
[2] G. G. Brown and R. F. Dell, *Formulating linear and integer linear programs: A rogues' gallery*, INFORMS Transactions on Education, **7** (2007).
[3] GAMS, `http://www.gams.com` accessed, accessed December 1, 2005.
[4] S. Hartmann, *A general framework for scheduling equipment and manpower at container terminals*, OR Spectrum, **26** (2004), 51–74.
[5] E. Henesey, "Enhancing Container Terminal Performance: A Multi Agent Systems Approach," Ph.D thesis, Blekinge Institute of Technology, Sweden, 2004.
[6] ILOG, "CPLEX 9.0, ILOG Inc.," Mountain View, California, 2003.
[7] K. H. Kim and J. W. Bae, *Re-marshaling export containers in port container terminals*, Computers & Industrial Engineering, **35** (1998), 655–658.
[8] K. Y. Kim and K. H. Kim, *A routing algorithm for a single transfer crane to load export containers onto containership*, Computers & Industrial Engineering, **33** (1997), 673–676.
[9] K. H. Kim and K. Y. Kim, *An optimal routing algorithm for a transfer crane in port container terminals*, Transportation Science, **33** (1999), 17–33.
[10] K. Y. Kim and K. H. Kim, *Heuristic algorithms for routing yard-side equipment for minimizing loading times in container terminals*, Naval Research Logistics, **50** (2003), 498–514.
[11] K. H. Kim and K. T. Park, *A note on a dynamic space-allocation method for outbound containers*, European Journal of Operational Research, **148** (2003), 92–101.

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

[13] A. Lim, B. Rodriques, F. Xiao and Y. Zhu, *Crane scheduling using Tabu search*, in "Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence," (2002).

[14] W. Lin, "On Dynamic Crane Deployment in Container Terminals," M.S. thesis, Hong Kong University of Science and Technology, Hong Kong, 2000.

[15] R. Linn, J. Liu, Y. Wan, C. Zhang and K. Murty, *Rubber tired gantry crane deployment for container yard operation*, Computers & Industrial Engineering, **45** (2003), 429–442.

[16] K. G. Murty, J. Liu, Y. Wan and R. Linn, *A DSS (Decision Support System) for operations in a container terminal*, Working paper, University of Michigan, https://www-personal.engin.umich.edu/ murty/terminal-10.pdf, accessed January 15, 2003.

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

[18] P. Preston and E. Kozan, *An approach to determine storage locations of containers at seaport terminals*, Computers & Operations Research, **28** (2001), 983–995.

[19] D. Stenken, S. Vob and R. Stahlbock, *Container terminal operation and operations research - a classification and literature review*, OR Spectrum, **26** (2004), 3–49.

[20] M. Taleb-Ibrahimi, B. De Castilho and C. F. Daganzo, *Storage space vs. handling work in container terminals*, Transportation Research B, **27** (1993), 13–32.

[21] G. Wan, *An intelligent decision support system for crane scheduling in a container terminal*, in "Proceedings of 14th annual conference of the Production and Operations Management Society, POM," (2003).

[22] C. Zhang, Y. Wan, J. Liu and R. Linn, *Dynamic crane deployment in container storage yards*, Transportation Research B, **36** (2002), 537–555.

[23] I. Zyngiridis, "Optimizing Container Movement Using one and two Automated Stacking Cranes," M.S thesis, Naval Postgraduate School, Monterey, California, 2005.

Received April 2007; 1st revision February 2008; 2nd revision February 2009.

E-mail address: dell@nps.edu
E-mail address: joroyset@nps.edu