A COMPARISON OF PRIORITY RULES FOR NON-PASSING AUTOMATED STACKING CRANES

Héctor J. Carlo, Azaria Del Valle-Serrano, Fernando L. Martínez-Acevedo, Yaritza M. Santiago-Correa
University of Puerto Rico – Mayagüez,
Industrial Engineering Department
Call Box 9000, Mayagüez, PR 00681-9000

Iris F.A. Vis
University of Groningen, Faculty of Economics and Business,
Department of Operations,
P.O. Box 800, 9700 AV Groningen, The Netherlands

Abstract

A recent trend in container ports is to operate dual non-passing Automated Storage Cranes (ASCs) that collaborate to serve storage and retrieval requests from opposite ends of a storage block. Since the ASCs are unable to pass each other, there is an *exchange zone* that serves as a temporary storage location so that one crane can start a request and leave it to the other crane to complete it. In this study, twelve priority rules are introduced and evaluated to determine which rule minimizes the total makespan for serving all requests, given the sequence in which each ASC will serve the requests. Preliminary results from 12 randomly generated experiments indicate that the priority rules favoring the crane furthest away from the origin of the next request (*LonOri*) and the longest individual completion times (*LonTot*) outperformed all other rules in terms of the average percent difference with the best found solution and in terms of the percent of times the priority rule yield the best found solution. Also, combining priority rules *AdvFun* and *LonRem* yields the best makespan in 11 of the 12 (91.67%) problem instances tested. Results of this study transcend container ports as it is applicable to any material handling system composed of non-passing MHE and that has pickup/deposit points at the ends of the system.

1. Introduction

With the globalization of the economies, the freight volume handled at container ports has increased precipitately. Most large container ports worldwide have been forced to explore new technologies to increase their efficiency in order to maintain their level of competitiveness. A recent trend in container ports is to deploy various material handling equipment (MHE) (e.g. gantry cranes) in parallel to the same area to serve a set of storage and retrieval requests. Depending on the system configuration, the interference between the parallel cranes could significantly affect the performance of the system. There are four main multi-crane configurations currently used in container terminals.

The first multi-crane configuration can be seen in the seaside of container ports where multiple identical quay cranes may be used to simultaneously load / unload vessels [1]. In this case, transfer vehicles feed quay cranes perpendicular to the movement of the crane, represented by the arrows in Figure 1. The vessel is typically partitioned into regions (storage bays), which are assigned to the quay cranes. Clearly, the location of the input/output (I/O) point in this configuration does not require that quay cranes collaborate with each other to serve the storage and retrieval requests. Interference between quay cranes is very limited, depending on the partition and the scheduling of the quay cranes (e.g. [2-3]).

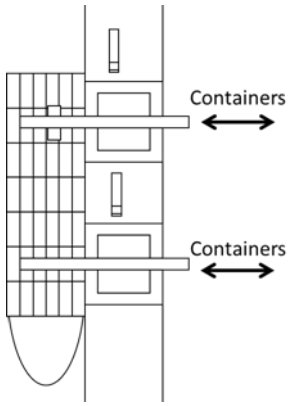

Figure 1: Multi-quay Crane Configuration

The remainder multi-crane configurations can be found in the landside of container ports. One configuration, shown in Figure 2, consists of multiple identical non-automated (typically rubber-tyred) gantry cranes operating in parallel to store and retrieve containers to (from) storage blocks from (to) yard trucks. In this system, the yard truck runs through a truck lane to which the crane travels to pickup (or deposit) containers from (on) the trucks, as shown in Figure 2. Once again, the transfer of containers is done perpendicular to the movement of the (yard) crane. In this configuration the yard cranes operate independently (i.e. do not collaborate) to serve the requests (e.g. [4]). Potential interference between the cranes are easily handled by carefully partitioning the space and scheduling the cranes.

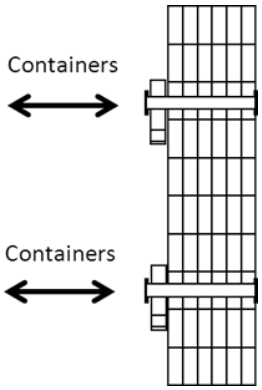

Figure 2: Multi-yard Crane Configuration

The last two multi-crane configurations in container terminals are the (passing or non-passing) Automated Stacking Cranes (ASCs) used in a storage block to store and retrieve containers (see [5-6] for a description of the configurations). ASCs are automated rail-mounted gantry cranes. In this configuration, ASCs typically pickup (deposit) containers from (to) Automated Guided Vehicles (AGVs) located at the end of the block. As shown in Figure 3, the block layout for container terminals using ASCs is different than from non-automated yard cranes as the pickup and deposit points are located at opposite ends of the block instead of in the truck lane. In this configuration the transfer of containers is done parallel to the movement of the ASC. Hence, the ability to pickup or deposit containers from the I/O point is more susceptible to crane interference.

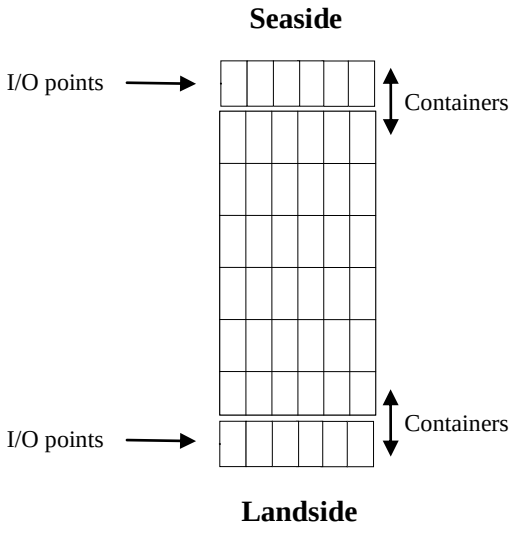

Figure 3: Block Layout with ASCs

The container terminal Altenwerder in Hamburg, Germany uses a system of two passing ASCs. The system is composed of two ASCs, one larger than the other, as shown in Figure 4. This configuration allows each ASC to access I/O points on both sides of the stack. Interference between the ASCs occur when the larger ASC is lifting or putting down a container, in which case the smaller ASC is not able to pass. Interference also occurs when the two ASCs are performing a request in the same bay [7]. However, prioritizing the ASCs under this configuration is not too detrimental to the performance of the system as after the interruption (however it is resolved) the ASCs may pass each other and continue with their schedules.

Figure 4: Passing Stacking Cranes of Kalmar Industries [8]

On the other hand, a configuration of two non-passing ASCs, such as the container terminal Altenwerder in Hamburg (Germany), requires a more careful operational strategy. Notice that in the non-passing configuration, shown in Figure 5, does not allow ASCs to reach both ends of the block. Instead, either the ASCs are operated independently or they collaborate by passing on containers to each other via an *exchange zone*. If the cranes operate independently, the requests must be assigned to the crane serving the corresponding I/O point for each request. In this case, any potential interference between the ASCs is easily handled by carefully scheduling the cranes. However, if the ASCs are allowed to pass a container to each other via an exchange zone (i.e. the ASCs collaborate), the ASC prioritization problem becomes much important and more complex for two reasons: (1) an inefficient priority rule could have a potentially detrimental impact on the system since the ASCs cannot pass each other; and (2) the priority rules need to consider the dynamics of the exchange zone which are non-existent

in the other multi-crane configurations. This paper focuses on evaluating priority rules for the non-passing collaborating ASC configuration.

Figure 5: Two Non-passing ASCs (Gottwald [9])

The ultimate problem that this research stream intends to solve is how to assign a list of requests to two non-passing ASCs, how to spit the requests among the cranes, and how to schedule them for each ASC. A natural strategy to address this problem is to smartly generate schedules for each ASC to serve the requests. These schedules would then be evaluated to incorporate the delays caused the interaction and interference between the ASCs. Hence, fast and reliable prioritization rules that would establish how to resolve interferences between the cranes need to be developed. This study presents a simulation-based comparison of twelve priority rules for a system of two non-passing ASCs that are allowed to pass containers to each other via an exchange zone.

The reminder of the paper is organized as follows. Section 2 discusses the relevant existing literature on the problem. The system and problem description is presented in Section 3. Section 4 describes the proposed priority rules. Section 5 presents the experimental results. Lastly, Section 6 includes the conclusions and further research.

2. Literature Review

There is a dearth of publications regarding identical Material Handling Equipment (MHE) that that collaborate by passing loads from one MHE to the other while sharing the same travel path. Carlo and Vis [5] propose a simple sequencing heuristic for two non-passing ASCs. However, no priority rule is described for resolving interference between the cranes. Dorndorf and Schneider [10] provide a dynamic scheduling algorithm for triple cross-over stacking cranes, which is composed of two non-passing cranes and a large crane able to pass the others. Although the system studied by [10] allows for collaboration between the cranes, no precedence constraints are included in their model. The precedence constraints arise in our problem when requests are split

between the two ASCs. Dorndorf and Schneider [10] dynamically optimize the assignment and sequencing of requests considering collisions via a branch & bound algorithm (B&B). Unfortunately, the B&B is slow, which limits the size of the problem instances. Further, the B&B is not well-suited for evaluating the fitness of crane sequences, which needs to be solved many times in the static version of the problem.

No other study was found on identical MHE that that collaborate by passing loads from one MHE to the other while sharing the same travel path. The objective of this study is to identify fast and reliable prioritization rules for dual non-passing collaborating ASCs.

3. System and Problem Description

Figure 6 depicts the system under study, which is composed of two collaborating non-passing ASCs, *C1* and *C2*, who serve storage and retrieval requests from both the landside and seaside. *C1* is the ASC closer to the seaside, while *C2* is the one closer to the landside. The I/O points for the ASCs are in opposite ends of the block. Since the ASCs are unable to pass each other there is an exchange zone that serves as a temporary storage location so that one crane can start a request and leave it to the other crane to complete it. The ASCs are allowed to operate using a dual-command (dual-cycle) strategy. A minimum safety distance between the ASCs must be observed at all times.

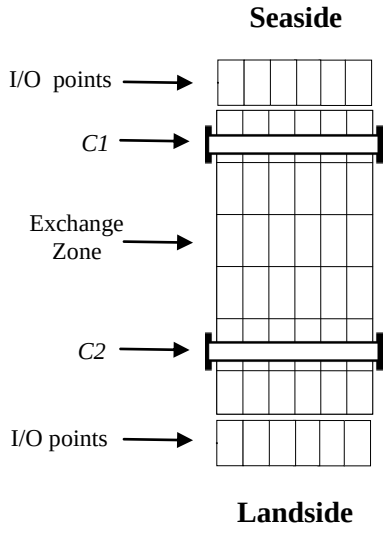

Figure 6: Non-passing Collaborating ASCs System

It is assumed that (storage and retrieval) requests have already been assigned to the cranes, and that the sequence in which the requests will be served for each crane is known. Further, the storage or retrieval location of each requests, as well as the

corresponding I/O point, are assumed given. The objective of this study is to compare simple (i.e. fast) priority rules in terms of the total makespan (i.e. the completion time of both ASCs) to serve all requests in a predetermined list. In practice, this list would correspond to a block of requests (e.g. 1 hour worth of requests).

Priority rules are required when the sequence followed by the cranes creates a situation where the cranes interfere with each other. Hence, it is key to track the position of each ASC in time. Unfortunately, determining how to resolve an interference between the ASCs is not trivial in terms of coding the priority rules. Hence, we use the graphical representation from Vis et al. [11] and Carlo and Vis [12] to identify and resolve interference between the ASCs. We observe that each storage and retrieval request is composed of four types of movements. For example, a storage request starts by performing an empty travel to the pickup location. This is followed by a pickup move, in our case representing the time to lower the shuttle to pickup a container. Then, a full travel is performed to the respective I/O point or exchange zone. Lastly, a drop-off is performed to release the container. The retrieval process follows a similar structure. Table 1 presents the four movements for each type of request.

Table 1. Description of ASC movements to serve storage and retrieval requests

Steps	**Storage Request**	**Retrieval Request**
1	move from current position to pickup point	move from current position to retrieval location
2	pick up container	pick up container from retrieval location
3	move with container to storage location	move with container to drop-off point
4	deposit container at storage location	deposit container at drop-off point

We introduce the indices i and $j \in \{1,2,3,4\}$ to respectively represent the specific step of a request being performed on C1 and C2, as defined in Table 1. Clearly, the type of movement and the state (i.e., full or empty) of the lift related to a step (e.g., i=3) differs per type of request. This information is then incorporated into a (continuous) function to represent the position of a lift as shown below.

f(t) Function that represents the position of C1 (seaside crane) in time *t.*

g(t) Function that represents the position of C2 (landside crane) in time *t.*

fi(t) Continuous i^{th} function *f(t)* that represents the position of C1 in time *t* for a specific request.

gj(t) Continuous j^{th} function *g(t)* that represents the position of C2 in time *t* for a specific request.

In Figure 7 we show by means of an example the various functions for C1 handling a retrieval request. In the example, the seaside I/O is represented by level -5, the I/O for the landside is at level 5, and the exchange point is at level 0. Notice that the ASC starts at level (row on the storage block) -2, moves empty from level 2, pick up item, move to level -20 (the seaside I/O) and release item at this level.

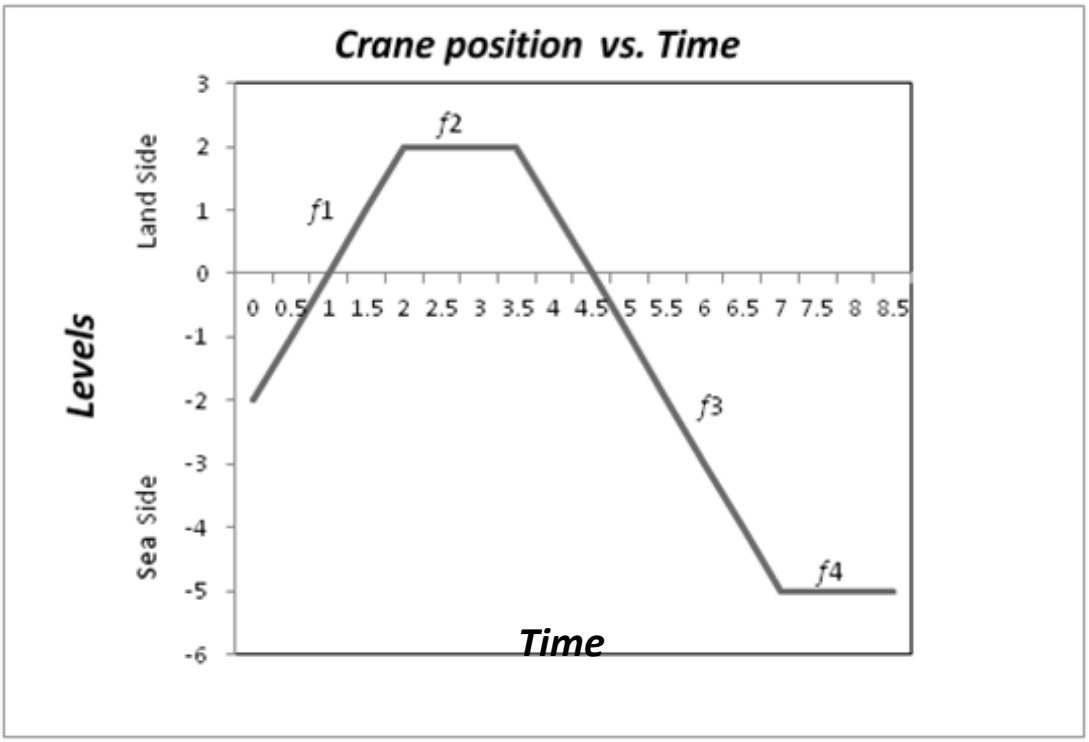

Figure 7: Example of a retrieval request handled by C1.

C2 would have a movement (and Figure) analogous to Figure 7. Figure 8 depicts a scenario where the *f(t)* and *g(t)* are plotted together. From Figure 8 it can be noticed that the constraint that the two cranes cannot pass each other is violated. In this case one needs to prioritize the movement of the ASCs. Clearly, in this example, giving priority to C1 would result in a lower makespan than giving priority to C2.

One of the benefits of the proposed graphical representation is that mathematically it is very simple to determine if and when interference occurs by finding the intersection of the corresponding piece-wise linear functions. For more details on the graphical representation, the reader is referred to Carlo and Vis [12]. In the next Section we propose and describe twelve priority rules that could be used to resolve interference between the ASCs.

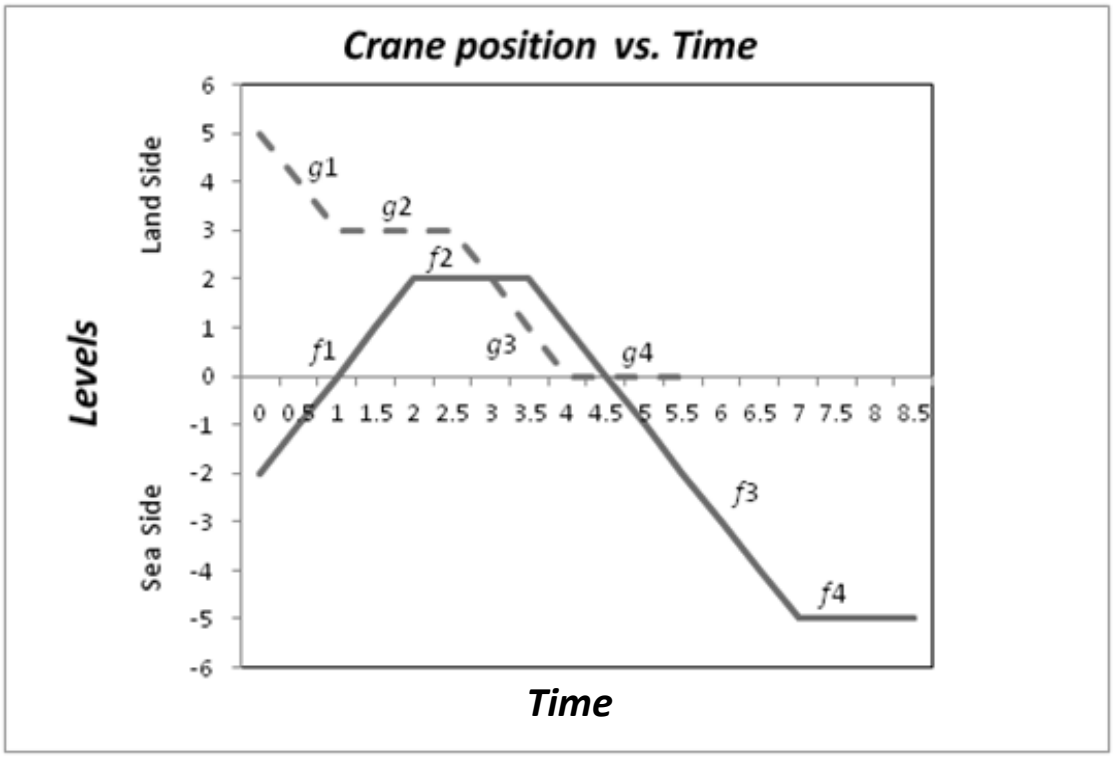

Figure 8: Example of C1 and C2 handling a retrieval request simultaneously.

4. Proposed Priority Rules

Once an interference between the ASCs has been identified a prioritization scheme needs to be executed. Clearly, the optimal priority scheme can be found by exhaustive enumeration, i.e. creating two branches (one with Priority to C1 and the other with Priority to C2). However, obtaining the optimal priority is very time consuming. The following twelve priority rules (P1-P12) are proposed for handling interferences between the ASCs.

P1. (***PriC1***) - Crane (1) always has priority. This priority rule was considered in Carlo and Vis [12] to prioritize two non-passing lifts with a single I/O point. This priority could be advantageous if most of the requests are either associated with the seaside or assigned to C1.

P2. (***PriC2***) Crane 2 always has priority. This priority rule was also considered in [12]. This priority could be advantageous if most of the requests are associated with the landside or assigned to C2.

P3. (***AdvFun***) Favor the ASC whose respective function *f(t)* or *g(t)* has a higher index. This priority rule was designed to favor the crane closest to completing a request (in terms of the number of movements remaining).

P4. (***ShoOri***) Favor the ASC with the shortest travel time to the origin of the next request (upon completion of the current request). This priority rule seeks to minimize the empty travel time to the next request.

P5. (***LonOri***) Favor the ASC with the longest travel time to the origin of the next request (upon completion of the current request). This priority rule was designed recognizing that a long unproductive (i.e. empty) unavoidable move needs to be performed, which is less likely to re-interfere with the non-prioritized ASC.

P6. (***ShoFin***) Favor the ASC with the shortest time to finish the current request. The idea behind this priority rule is to minimize the wait time for the non-prioritized crane.

P7. (***LonFin***) Favor the ASC with the longest time to finish the current request. The idea behind this priority rule is to allow longer requests to be prioritized.

P8. (***AbsDis***) Favor the ASC with the shortest distance to the exchange zone. This priority rule seeks to favor the crane that is furthest away from its I/O point. Notice that in order for the ASCs to interfere one must be past the exchange zone.

P9. (***TotReq***) Favor the ASC to which most requests have been assigned. This priority rule is based on the premise that the more requests a crane has to serve, the most likely it is to be the bottleneck crane.

P10. (***RemReq***) Favor the ASC that has more requests left to serve. The spirit of this priority rule is to identify the bottleneck crane and give it priority.

P11. (***LonTot***) At the beginning of the problem, determine the total time each ASC needs to complete its requests (considering all assigned requests) without considering the other crane (i.e. no interference). Give priority to the crane that requires the most time to finish all the assigned requests. Similar to P9 and P10, this priority rule seeks to give priority to the bottleneck crane.

P12. (***LonRem***) Whenever there is interference between the ASCs, determine the time each ASC would need to complete the remaining requests without considering the

other crane (i.e. no interference). This is equivalent to the Priority Longest rule in Carlo and Vis [12].

5. Experimental Results

In order to compare the twelve proposed priority rules, twelve random instances of a problem were generated using Visual Basic for Applications (VBA). The problem instances considered 25 (storage or retrieval) requests. Half of the requests either originated or ended on the seaside I/O and the other half on the landside I/O. Twenty five percent of the requests were storages to ensure the S/R operates under both single and double command. The probability of a requests being stored (retrieved) on (from) the other side of the exchange zone was set to 50%. We refer to these requests *splittable*, as they can be potentially split among the two ASCs (e.g. in a retrieval request a ASC moves it from the storage location to the exchange zone, while the other ASC moves the container from the exchange zone to the I/O). The probability that a splittable request is actualy split was also set to 50%. Hence, we expect on average that half of the 25 requests potentially create interference. Further, on average 25% of the requests would be split. Notice that although there are only 25 requests to serve, on average, one quarter of these requests will be split creating an average of 31.25 (=25*1.25) requests to be served.

Unfortunately, creating feasible problem instances is not trivial as C1 could be waiting for a split request that C2 has not deposited in the exchange zone, while C2 is also waiting for a split request that C1 has not deposited in the exchange zone. When the randomly generated problems were infeasible, we manually tweaked them by moving the requests creating infeasible instances to different locations in the ASC sequences.

The twelve priority rules were coded in C++. The total makespan for each instance was recorded assuming that the distance between bays was one distance unit (DU) and the ASCs travel at a constant velocity of one DU per time unit (DU/TU). In our experiments, only the travel distance between levels was considered. In practice, the ASCs move according to either in Chebyshev or Rectilinear paths by moving between levels (bays) and rows.

Table A-1 in the Appendix presents the percent difference between the makespan for each priority rule and the best makespan found by any of the priority rules for our twelve experiments. The average, standard deviation, and corresponding rank (based on the average) of the percent differences are included at the bottom of Table A-1. Lastly, the percent of times the priority rule found the best makespan is presented in the row labeled *%Best*. It is observed that rules *LonOri* and *LonTot* have the lowest average percent difference. Both of these priority rules found the best makespan in 7 of the 12 instances (58.33%). We did not find the optimal priority scheme for our problem instances, however, we suspect that the optimal makespan is close to the best found.

When solving the assignment and sequencing of requests for the collaborating non-passing ASC system the recommended priority rules based on our preliminary results are *LonOri* and *LonTot*, which yield solution that are on average 1.6% from the best found. Since the priority rules run extremely fast (even on a personal computer), two different

priority rules could be run and the one that yields the best makespan is used. In the case two priority rules can be used, priority rules *AdvFun* and *LonRem* seems to complement each other the best. When these two priority rules are combined, the best found solution was obtained in 11 of the 12 instances (91.67%).

6. Conclusions and Future Work

This study proposes and evaluates twelve priority rules for resolving interference on a system of two non-passing ASCs that are able to collaborate by passing containers from one ASC to the other via an exchange zone. The sequence of the requests to be served by each ASC is assumed given. The proposed priority rules are key when evaluating candidate solutions for the problem of assigning and sequencing requests of the non-passing collaborating ASCs system. Hence, the proposed priority rules are designed to have very fast runtimes. Twelve randomly generated problem instances with a specific structure to propense interference were created. Results indicate that priority rules *LonOri* and *LonTot* have the lowest average percent difference with respect to the best found makespan (1.6%). However, if two priority rules are run sequentially and the better priority rule is selected, then combining priority rules *AdvFun* and *LonRem* is preferred. The combination of these two priority rules found the best makespan in 11 of the 12 (91.67%) problem instances tested.

Future work should focus on performing additional experiments with multiple factors and levels. The proposed priority rules should then be compared to the optimal priority scheme.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. EEC-0851879. The authors would also like to acknowledge Groningen Seaports for supporting this research.

Appendix

Table A-1. Experimental Results

Inst.	% Diff from Best											
	PriC1	***PriC2***	***AdvFun***	***ShoOri***	***LonOri***	***ShoFin***	***LonFin***	***AbsDis***	***TotReq***	***RemReq***	***LonTot***	***LonRem***
1	9.75%	1.85%	1.85%	10.49%	0.00%	10.49%	0.00%	1.85%	1.85%	0.00%	0.00%	2.84%
2	0.00%	1.75%	0.00%	0.00%	1.75%	0.00%	1.75%	1.75%	1.75%	1.75%	0.00%	0.00%
3	0.00%	13.96%	13.96%	13.96%	0.00%	8.11%	7.17%	8.11%	0.00%	2.08%	0.00%	0.00%
4	32.61%	0.00%	0.00%	9.17%	0.00%	14.41%	5.53%	1.60%	32.61%	3.64%	32.61%	13.97%
5	20.24%	0.00%	0.00%	8.41%	4.60%	8.41%	4.60%	0.00%	0.00%	4.60%	20.24%	4.60%
6	6.81%	0.00%	2.96%	6.81%	0.00%	2.96%	6.81%	0.00%	0.00%	0.00%	0.00%	0.00%
7	18.64%	21.75%	21.75%	21.75%	6.36%	4.88%	6.36%	1.04%	21.75%	4.14%	21.75%	0.00%
8	17.54%	3.12%	0.00%	3.12%	4.48%	3.12%	17.54%	3.12%	3.12%	2.92%	17.54%	17.54%
9	4.16%	0.00%	0.00%	3.80%	0.00%	0.00%	10.85%	0.00%	0.00%	0.00%	0.00%	4.16%
10	0.00%	0.18%	0.18%	0.18%	0.00%	0.18%	0.00%	0.18%	0.18%	0.18%	0.00%	0.00%
11	20.44%	0.00%	10.71%	20.44%	0.00%	20.44%	0.00%	5.16%	0.00%	0.00%	0.00%	0.00%
12	0.00%	2.15%	3.64%	0.00%	2.15%	0.00%	2.15%	2.15%	2.15%	0.00%	2.15%	0.00%
Ave.	10.85%	3.73%	4.59%	8.18%	1.61%	6.08%	5.23%	2.08%	5.28%	1.61%	7.86%	3.59%
Stdev	10.84%	6.89%	7.11%	7.50%	2.30%	6.54%	5.18%	2.43%	10.54%	1.82%	11.74%	5.99%
Rank	12	5	6	11	2	9	7	3	8	1	10	4
%Best	25.00%	41.67%	41.67%	8.33%	58.33%	16.67%	25.00%	25.00%	41.67%	33.33%	58.33%	50.00%

References

[1] Bierwirth, C., Meisel, F., "A Survey of Berth Allocation and Quay Crane Scheduling Problems in Container Terminals," *European Journal of Operational Research*, 202, 615-627 (2010).

[2] Lim, A., Rodrigues, B., and Xu, Z., "A m-parallel Crane Scheduling Problem with a Non-crossing Constraint," *Naval Research Logistics*, 54, 2, 115-127 (2007).

[3] Zhu, Y., and Lim, A., "Crane Scheduling with Non-crossing Constraint," *Journal of the Operational Research Society*, 57, 1464-1471 (2006).

[4] Li, W., Yong, W., Petering, M.E.H., Goh, M., and de Souza, R., "Discrete Time Model and Algorithms for Container Yard Crane Scheduling," *European Journal of Operational Research*, 198, 165-172 (2009).

[5] Carlo, H.J. and Vis, I.F.A., "Routing New Types of Stacking Crane Configurations at Container Terminals," *Progress in Material Handling Research*, K.P. Ellis, R.D. Meller, M.K. Ogle, B.A. Peters, G.D. Taylor, J. Usher (eds.), Material Handling Institute, Charlotte, North Carolina, 55-70 (2008).

[6] Carlo, H.J. and Vis, I.F.A., "New Initiatives in Stacking Cranes Configurations," *Port Technology International*, 44, 32-36 (2009).

[7] Vis, I.F.A. and Carlo, H.J., "Sequencing Two Cooperating Automated Stacking Cranes in a Container Terminal," *Transportation Science*, 44, 2, 169-182 (2010).

[8] Kalmar Industries, http://www.kalmarind.com/show.php?id=1020132 (accessed, June 2008).

[9] Gottwald, http://www.gottwald.com/gottwald/site/gottwald/en/news/gallery/ (accessed, June 2008).

[10] Dorndorf, U. and Schneider, F., "Scheduling Automated Triple Cross-over Stacking Cranes in a Container Yard," *OR Spectrum*, 32, 617-632 (2010).

[11] Vis, I.F.A., Carlo, H.J., Diaz, P., Laboy, M.E., and VanWijgaarden, B., "Control Policies for a Dynamic Storage System with Multiple Lifts and Shuttles," *Progress in Material Handling Research*, K.P. Ellis, K.R. Gue, R.B.M. de Koster, R.D. Meller, B. Montreuil, M.K. Ogle (eds.), Material Handling Institute, Charlotte, North Carolina, 494-509 (2010).

[12] Carlo, H.J. and Vis, I.F.A., "Sequencing Dynamic Storage Systems With Mutliple Lifts and Shuttles," *International Journal of Production Research*, under review (2012).