A Yard Crane Scheduling Problem with Practical Constraints

Amir Hossein Gharehgozli
Rotterdam School of Management, Erasmus University, Rotterdam, The Netherlands

Yugang Yu
Rotterdam School of Management, Erasmus University, Rotterdam, The Netherlands

René de Koster
Rotterdam School of Management, Erasmus University, Rotterdam, The Netherlands

Gilbert Laporte
Canada Research Chair in Distribution Management, HEC Montréal, Montréal, Canada

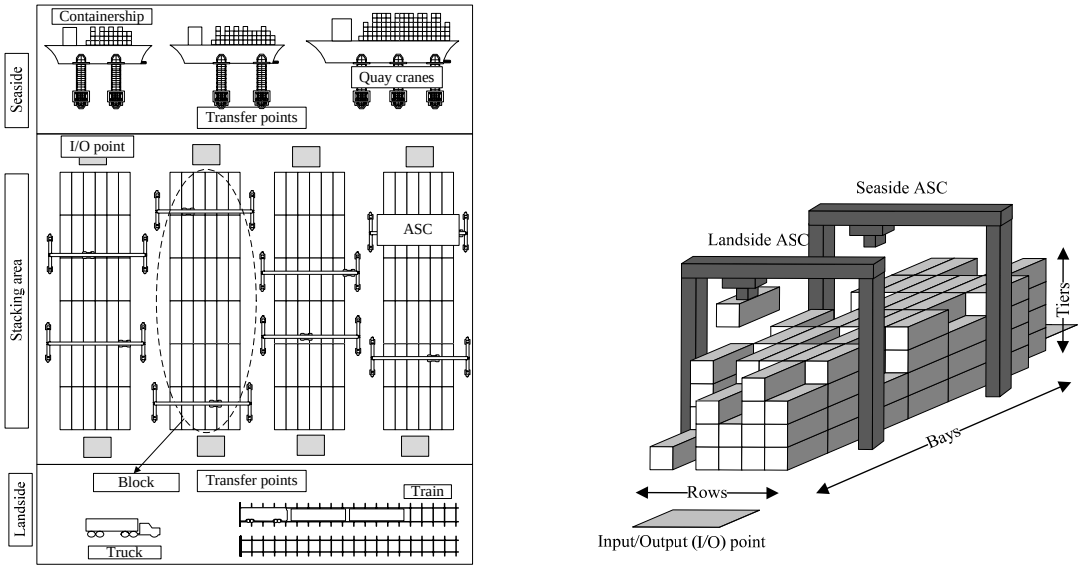
Abstract

The problem considered in this paper is to locate storage containers while a set of container storage and retrieval requests are sequenced. Two automated cranes stack and retrieve containers in a single block of a yard. The cranes cannot pass each other and must be separated by a safety distance. Storage containers are initially located at the seaside and landside input/output (I/O) points of the block. Each must be stacked in a specific location of the block, selected from a set of open locations suitable for stacking the storage container. Retrieval containers are initially located in the block and must be delivered to the I/O points. Due to the importance and acceptable waiting times of different modes of transport, requests have different priorities. The problem is modeled as a multiple asymmetric generalized traveling salesman problem with precedence constraints. The objective is to minimize the makespan. We have developed an adaptive large neighborhood search heuristic to quickly

compute near-optimal solutions. The numerical experiments show that the solution method can obtain near-optimal solutions.

1 Introduction

Containerized transportation has become an essential part of world trade during the past decades [1]. A large terminal handles millions of containers on an annual basis. Figure 1a depicts a typical container terminal layout with several blocks of containers in the stack area. Containers arrive or leave the terminal from the seaside and landside and spend a period of time in these blocks consisting of multiple rows, tiers, and bays, as shown in Figure 1b. An input/output (I/O) point is located at each end of a block and two automated stacking cranes (ASCs) are used to stack and retrieve containers in that block. These can move along the bays and rows of the block simultaneously, but cannot pass each other. Thus, the seaside (landside) ASC carries out all requests that have to be stacked or retrieved to the seaside (landside). Furthermore, for security reasons, the distance between the two ASCs cannot be less than a minimum safety limit.



(a) Top view of a container yard (b) Block of containers

Figure 1: Schematic representation of a container yard layout.

Container terminal managers attempt to efficiently manage the logistic process of the terminals to keep up with the increasing number of containers to be handled. The stacking area is highly critical since most of the containers transiting through a container terminal must be stored for a certain length of time, possibly in different blocks. To improve the stacking operations, many new container terminals use more than one ASC for each block [2, 7]. However, a key factor in efficiently handling the stacking

operations and preventing the block from becoming a bottleneck is not only the number of ASCs, but also the scheduling of their operations.

In this paper, we study the problem of minimizing the makespan of the two ASCs carrying out a set of storage and retrieval requests in a block of containers. In addition to determining the order in which to carry out the requests, we determine a location for each storage request under the following operational constraints. When a container is to be retrieved to the seaside, the seaside ASC picks it up from its location in the block and drops it at the seaside I/O point. When a storage request is to be stacked from the seaside, the seaside ASC picks it up and stores it in a location in the block, selected from a set of available open locations specified for that request. Each open storage location can be occupied by at most one container. Landside operations can be carried out similarly. We consider different priorities in stacking or retrieving containers. We formulate this multi-crane problem as a multiple asymmetric generalized traveling salesman problem with precedence constraints (mAGTSP-PC). The model also contains additional constraints regarding the interactions of the ASCs and the selection of open storage locations which lie in the intersection of multiple sets. We develop an adaptive large neighborhood search heuristic for this problem.

A major body of research on the yard crane scheduling problem is focused on retrieving a set containers. Kim and Kim [1] schedule a yard crane to retrieve containers from several blocks in the stacking area of a terminal. They propose a discrete time network model in which the objective is to minimize the total travel time of the crane to carry out all retrieval requests. Narasimhan and Paleka [3] also study a model in which a single yard crane retrieves containers from a single block. They prove that the problem is NP-hard and develop a branch-and-bound algorithm. For large size instances, they propose a heuristic with a worst-case performance ratio of 1.5. Ng [4] schedules several yard cranes to carry out a set of retrieval requests with different ready times in a yard zone, defined as multiple blocks located behind each other in a row. The yard cranes cannot pass each other and cannot exit the zone. They propose a discrete time mixed integer model and solve it by means of a heuristic based on dynamic programming. The objective function is to minimize the total completion time. In more recent papers, retrieval and storage requests are considered simultaneously. Zhang et al. [8] propose a discrete time mixed integer linear model for a problem in which several yard cranes carry out a given workload in multiple blocks. Based on their definition, a workload can consist of storage and retrieval requests. The objective is to minimize the total unfinished workload at the end of each time period.

Since the use of double cranes limited to a block of containers is a new technology, scheduling models for such configurations can only be found in more recent papers. Li et al. [2] introduce a discrete time model to schedule two ASCs carrying out the storage and retrieval requests in a single block with an I/O point located at one side of the block along the bays. The ASCs cannot pass each other and must be separated by a safety distance. The requests have different due times and the objective is to minimize a weighted combination of earliness and lateness of all requests, compared to their due times. They introduce a rolling horizon algorithm in which a horizon of a specific length is defined,

and all requests falling within this horizon are considered and optimized by CPLEX. The horizon is updated whenever all its requests have been scheduled. Vis and Carlo [7] also consider a similar setting. However, in their problem the ASCs can pass each other but cannot work on the same bay simultaneously. In their problem, requests do not have any due time and can be scheduled in any sequence. They formulate the problem as a continuous time model and minimize the makespan of the ASCs. They solve it by simulated annealing algorithm.

Our scientific contribution is to develop and solve a continuous time model to schedule two interacting ASCs working in a single block of containers. No continuous time model with a similar set of constraints has yet been proposed for this problem. Our model incorporates precedence constraints, ASC interaction constraints, and constraints that assign each container to a storage location selected from a given set. Considering these constraints is valuable not only from a theoretical standpoint but also from a practical point of view. In practice, containers have different priorities, and a set of suitable open locations is available for each container that has to be stacked. The mAGTSP and its variants have been shown to be NP-hard (see, for example, [3]) and the new features introduced in this study make the problem even more difficult to solve.

The remainder of this paper is organized as follows. In Section 2, we describe the technical aspects of the problem and present our mathematical model. Section 3 describes the heuristic. Section 4 presents numerical results, and Section 5 contains the conclusions.

2 Problem description and mathematical model

We seek to determine the sequence to stack containers of n storage requests and retrieve containers of $N-n$ retrieval requests in a single block of containers consisting of X rows, Y bays, Z tiers and two I/O points, one at each end of the block. We denote by S and L the seaside and landside I/O points, respectively. Let $\mathcal{R}$ be the set of all requests, whereas $\mathcal{R}_s$ and $\mathcal{R}_l$ be the sets of requests that must be picked up or dropped off at S and L, respectively.

The objective is to minimize the makespan of the two ASCs carrying out these requests. We denote by ASC_s the seaside ASC assigned to $\mathcal{R}_s$, and by ASC_l the landside ASC assigned to $\mathcal{R}_l$. Each ASC can carry only one container at a time. From a given starting point, each ASC can execute its requests in any sequence in order to minimize its total travel time. We denote by 0_s and $0_s'$, respectively, the starting and ending locations of ASC_s, whereas 0_l and $0_l'$ are the corresponding locations for ASC_l. These locations can be anywhere in the block. The ASCs cannot pass each other and have to be separated by a distance taking τ time units to travel. Every storage or retrieval request $r \in \mathcal{R} = \{1, \dots, N\}$ corresponds to a unique container. For storage request r, an ASC moves a container from the I/O point where it is located to a location i selected from a given set $\mathcal{L}_r$ of available open locations. For retrieval request r, an ASC moves a container from location j to an I/O point. For ease of notation, we write that the location j of retrieval

request r belongs to $\mathcal{L}_r$. Thus, we can define the set $\mathcal{V}$ of all locations. For each storage location in $\mathcal{V}$ lying in the intersection of multiple sets, a copy is created for each set to which it belongs. The copy locations will be used in the model to avoid stacking multiple containers in the same location.

The problem can now be stated mathematically as the following integer linear program. Denote by $\mathcal{V}_k$ the set of locations of requests in $\mathcal{R}_k$, which includes the starting and ending locations of ASC_k. As a result, let x_{ij} be the binary decision variable which equals 1 if and only if location $j \in \mathcal{V}_k$ is visited immediately after location $i \in \mathcal{V}_k, \forall k \in \mathcal{K}$. The objective function minimizes C, the makespan of the two ASCs:

Minmize C. (1)

Constraints (2)-(3) below mean that each set must be entered and exited exactly once. In these constraints, $\mathcal{L}'_r = \bigcup_{\substack{e \in \mathcal{R}_k \\ e \neq r}} \mathcal{L}_e, \forall\, r \in \mathcal{R}_k, \forall k \in \mathcal{K}$:

$$\sum_{i \in \mathcal{L}'_r} \sum_{\substack{i \in \mathcal{L}_r \\ j \neq i}} x_{ij} = 1, \forall r \in \mathcal{R}_k \backslash \{0_k\}, \forall k \in \mathcal{K}, \tag{2}$$

$$\sum_{i \in \mathcal{L}_r} \sum_{\substack{i \in \mathcal{L}'_r \\ j \neq i}} x_{ij} = 1, \forall r \in \mathcal{R}_k \backslash \{0'_k\}, \forall k \in \mathcal{K}. \tag{3}$$

Constraints (4) are the network flow conservation constraints. Constraints (4)-(5) ensure that in an optimal solution, each location can receive at most one container, and each container is stacked somewhere. In constraint (5), $\mathcal{I}$ is the set of storage locations in the intersection of multiple sets, and $\mathcal{Q}_i$ is the set of all copies of storage location $\forall i \in \mathcal{I}$:

$$\sum_{\substack{i \in \mathcal{V}_k \\ i \neq j}} x_{ij} = \sum_{\substack{l \in \mathcal{V}_k \\ l \neq j}} x_{jl} \leq 1, \forall j \in \mathcal{V}_k, \forall k \in \mathcal{K}, \tag{4}$$

$$\sum_{i \in \mathcal{V}} \sum_{\substack{i \in \mathcal{Q}_i \\ j \neq l}} x_{ij} \leq 1, \forall i \in \mathcal{I}. \tag{5}$$

Constraints (6) define C_i, the arrival time of an ASC at location $i, \forall i \in \mathcal{V}$, whereas, t_{ij} is the travel time of the ASC from location i to location j in the block. In these constraints, M is a large constant. Constraints (7) are used to define the makespans of the ASCs based on their arrival times at their finishing points, $0'_s$, and $0'_l$:

$$C_i + T_{ij} - C_j \leq M\big(1 - x_{ij}\big), \forall i, j \in \mathcal{V}_k, \forall k \in \mathcal{K}, \tag{6}$$

$$C \geq C_j, \forall j \in \{0'_s, 0'_l\}. \quad (7)$$

Let $\mathcal{P}_i^-$ be the set of storage or retrieval locations that have to be visited before location $\forall i \in \mathcal{V}$ if they appear in a feasible solution. Constraints (8) ensure that location i is visited after all locations $j, \forall j \in \mathcal{P}_i^-$:

$$C_i \geq C_j, \forall i \in \mathcal{V}, \forall j \in \mathcal{P}_i^-. \quad (8)$$

Let $\mathcal{B}_i$ be the set of all locations behind location i that cannot be visited by the other crane because of the no-passing constraint. Furthermore, $\mathcal{A}_i$ is the set of all locations in front of location i that can only be visited by the other crane at least τ time units after the moment that location i has been visited. Thus, the interaction between the ASCs can be modeled by constraints (9)-(11). Let z_{ij} indicate whether location $\forall i \in \mathcal{V}_k, \forall k \in \mathcal{K}$ has to be visited before location $\forall j \in \mathcal{V}_{k'}, \forall k' \in \mathcal{K}$:

$$z_{ij} + z_{ji} = 1, \forall i, j \in \mathcal{V}, i \neq j. \quad (9)$$
$$C_i + \left(\tau - \left|y_i - y_j\right|\right) \leq M\left(1 - z_{ij}\right) + C_{ij}, \forall i \in \mathcal{V}, \forall j \in \mathcal{A}_i, \quad (10)$$
$$C_i + \left(\tau + \left|y_i - y_j\right|\right) \leq M\left(1 - z_{ij}\right) + C_{ij}, \forall i \in \mathcal{V}, \forall j \in \mathcal{B}_i. \quad (11)$$

Finally, we need the integrality and non-negativity constraints.

3. Adaptive large neighborhood search heuristic

The complexity of the problem means that only relatively small instances can be solved to optimality within a reasonable time. To solve real-size problems, we have developed an adaptive large neighborhood search (ALNS) heuristic. This type of heuristic was initially proposed by Ropke and Pisinger [5], and is based on the LNS search scheme previously developed by Shaw [6]. In the ALNS, the search starts from a feasible solution generated by a simple construction heuristic. The ALNS attempts to improve this solution by sequentially applying several elementary operators to remove a subset of requests from the solution and reinsert them.

3.1. Operators

We have developed several removal and insertion operators, all of which are applied to a feasible solution. Note that in this problem, one of the ASCs needs to visit a unique location for each request. In our operators, we refer to corresponding requests of locations.

Random removal: In this type of removal, q requests are randomly removed from the solution.

Worst removal: Assume that $\Delta(r, X)$ is the travel time of request r to its predecessor and successor requests in the solution. The worst removal operator removes q requests with the largest $\Delta(r, X)$. Note that we do not define $\Delta(r, X)$ as the effect of removing request r

in the objective function. Indeed, because of the interactions between the ASCs, and because of the precedence constraints, fully evaluating the objective function for each partial solution is rather time consuming. Our simplified approach avoids this computation.

Location removal: One of the features of our problem is the presence of sets of open locations to stack storage containers. In the heuristic used to generate a feasible initial solution, the location for each storage container is randomly selected from its set of available open locations. If no operator is applied to assign new locations, the selected locations do not change, which results in lost opportunities to improve the solution. The location removal operator, randomly selects $max\{n, q\}$ storage requests and removes the locations assigned to them.

Greedy reinsertion: Let Δf_r be the travel time of request r to its predecessor and successor request by inserting it in its best position. The greedy operator inserts the request minimizing Δf_r, over all removed requests. After inserting a request, the operator updates the Δf_r values and is reapplied.

Random reinsertion: This operator randomly reinserts the requests into the partial solution.

Location insertion: The location removal operator previously described can only be followed by the location insertion operator. For each storage request with a removed location, this operator randomly selects a new location from its set of available open locations.

3.2. Choosing a removal operator or a insertion operator

We run the ALNS algorithm for a preset number of iterations. At each iteration, the selection of a removal or insertion operator is governed by a roulette-wheel mechanism which selects operator o from the set of removal or insertion operators with probability $w_o / \sum_{o'=1}^{O} w_{o'}$, where w_o is the weight of operator o, and O is the number of operators of the set. In the ALNS, an initial weight is assigned to each operator and the weights are then updated after every δ iterations of the algorithm. The updates are based on the score that each operator has gained during the past δ iterations. At each iteration, the score of each operator is updated based on its performance: the weight of each removal or insertion operator used in the current iteration is incremented by σ_1 if their application result in a new best solution; by σ_2 if result in a better incumbent solution; and by σ_3 if result in a solution which is not better than the best solution found and incumbent solution but is accepted, where $\sigma_1 \geq \sigma_2 \geq \sigma_3$. Let π_o and θ_o be the total score of operator o and the number of times it has been selected after δ iterations, respectively. The weight of operator o is then updated as $w_o \leftarrow w_o(1-\rho) + \rho\pi_o/\theta_o$, where $\rho \in [0,1]$ is the reaction factor which controls how quickly the weight adjustment reacts to changes in the operator performance.

The ALNS metaheuristic is executed within a simulated annealing (SA) framework. In the SA framework, if the solution found at the current iteration is better than the current solution, it is accepted. Otherwise, it is accepted with probability $e^{-\left(z(s')-z(s^*)\right)/T}$

where T is the temperature, $Z(S')$ and $Z(S^*)$ are the objective values of the solution at the current iteration and best solution, respectively. The temperature starts at T_{start} and is updated at each iteration as $T \leftarrow T\phi$, where $0 < \phi < 1$ is a cooling rate. We terminate the algorithm after a number of runs.

4. Computational experiments

In this section, we compare the results of the ALNS with those results of truncated CPLEX, and some other simple heuristics. In our experiments, we consider a block with 40 bays, 10 rows, and four tiers. The number N of requests is set to 10 and 25. We assume an equal number of open locations $|\mathcal{L}_r|$ for storage requests $r = 1, \dots, n$, and we set $|\mathcal{L}_r|$ equal to 1 or 3. As a result we perform basic experiments on five randomly generated instances in each case. Each instance has the same number of retrieval and storage requests. All locations are uniformly distributed over the block, and an equal number of requests have to be picked up or delivered to the landside and the seaside. The landside ASC starts its operations from L and ends either at the storage position of a storage request if this is the last request in the sequence, or at L if a retrieval request is the last request. Similarly, the seaside ASC starts from S and either finishes in a storage position or at S. We assume that the containers are categorized in six priority levels of decreasing importance.

Tables 1 and 2 compare the makespan and computation time of the ALNS and truncated CPLEX over different instances. For each instance, we apply the ALNS 20 times each with a different random initial solution. The CPLEX algorithm is applied to each instance only once in order to find a solution for the mAGTSP-PC and mAGTSP which is a relaxed problem without the precedence constraints and the ASC interaction constraints. The computation time of CPLEX for finding an optimal or even a feasible solution for the mAGTSP-PC is very high and we therefore truncate it after four hours, whereas the mAGTSP can be quickly solved. Column G_Z^{LB} shows that the gap between the lower bound and the optimal mAGTSP-PC solution is large. The gap stems from the fact that the new constraints significantly affect the sequence of requests carried out by the ASCs. This suggests that although the gap between the average ALNS solution value and the lower bound may be large in some cases, this does not necessarily reflect on the quality of the ALNS algorithm. A fair assessment is to compare the average result of the ALNS with the best known objective value.

Table 1: Results of the ALNS and CPLEX for Experiments 1, and 2.

inst	CPLEX results					ALNS results						
	LB	CPU	Z	CPU	G_Z^{LB}	Min	Max	Ave	G_{Ave}^{min}	G_{min}^{Z}	G_{Ave}^{Z}	CPU
Experiment 1 ($N = 10, \|\mathcal{L}_r\| = 1, r = 1, \dots, 5$)												
1	348.83	0.00	387.59	0.10	10.00	387.59	387.59	387.59	0.00	0.00	0.00	0.75
2	288.29	0.01	386.53	0.40	25.42	386.53	386.53	386.53	0.00	0.00	0.00	0.73

3	343.05	0.00	426.17	0.02	19.50	426.17	426.17	426.17	0.00	0.00	0.00	0.80
4	337.50	0.01	398.74	0.12	15.36	398.74	398.74	398.74	0.00	0.00	0.00	0.75
5	234.95	0.01	289.52	0.06	18.85	289.52	289.52	289.52	0.00	0.00	0.00	0.79
Ave	310.52	0.01	377.71	0.14	17.83	377.71	377.71	377.71	0.00	0.00	0.00	0.76
Experiment 2 ($N = 10, \|\mathcal{L}_r\| = 3, r = 1, \dots, 5$)												
1	288.29	0.01	356.32	3.54	19.09	356.32	356.32	356.32	0.00	0.00	0.00	0.81
2	319.38	0.01	419.15	2234.84	23.80	419.15	419.15	419.15	0.00	0.00	0.00	0.93
3	313.97	0.02	346.08	1699.35	9.28	346.08	350.05	346.88	0.23	0.00	0.23	0.89
4	275.06	0.02	291.56	13.80	5.66	291.56	291.56	291.56	0.00	0.00	0.00	0.84
5	336.81	0.01	392.46	6.55	14.18	392.46	419.05	397.53	1.27	0.00	1.27	0.86
Ave	306.70	0.01	361.11	791.62	14.40	361.11	367.23	362.29	0.30	0.00	0.30	0.87

Note. LB is the optimal solution of the AGTSP. Z is the feasible or optimal solution of mAGTSP-PC. Columns CPU show the computation time in seconds. For CPLEX, when CPU < 14400 seconds, the optimum is attained. The computation time of the ALNS is an average over 20 runs. The gaps are calculated as: $G_a^b(\%) = \big((a-b)/a\big) \times 100$.

Table 2: Results of the ALNS and CPLEX for Experiments 3, and 4.

inst	CPLEX results					ALNS results						
	LB	CPU	Z	CPU	G_Z^{LB}	Min	Max	Ave	G_{Ave}^{min}	G_{min}^{Z}	G_{Ave}^{Z}	CPU
Experiment 3 ($N = 25, \|\mathcal{L}_r\| = 1, r = 1, \dots, 13$)												
1	660.26	0.02	728.36	14400	9.35	728.36	756.00	739.19	1.47	0.00	0.11	3.50
2	608.06	0.02	747.62	11730.83	18.67	747.62	789.15	761.44	1.81	0.00	0.14	3.26
3	733.75	0.02	749.54	14400	2.11	749.54	776.57	758.57	1.19	0.00	0.09	3.35
4	675.55	0.01	805.03	14400	16.08	805.03	812.33	806.77	0.22	0.00	0.02	2.97
5	809.92	0.02	827.04	14400	2.07	826.78	854.89	834.26	0.90	-0.03	0.07	3.12
Ave	697.51	0.02	771.52	13866.17	9.66	771.47	797.79	780.05	1.12	-0.01	0.09	3.24
Experiment 4 ($N = 25, \|\mathcal{L}_r\| = 3, r = 1, \dots, 13$)												
1	653.96	0.07	952.04	14400	31.31	952.04	952.04	952.04	0.00	0.00	0.00	4.41
2	715.18	0.06	846.43	14400	15.51	842.57	849.37	845.80	0.38	-0.46	-0.01	4.42
3	624.06	0.06	869.14	14400	28.20	869.14	906.20	885.17	1.81	0.00	0.16	4.09
4	738.29	0.06	776.28	14400	4.89	767.41	824.80	809.12	5.15	-1.16	0.33	4.96
5	659.84	0.06	860.45	14400	23.31	860.45	870.99	863.61	0.37	0.00	0.03	4.66
Ave	678.27	0.06	860.87	14400	20.64	858.32	880.68	871.15	1.54	-0.32	0.10	4.51

Next, we compare the ALNS with some simple constructive heuristics such as nearest neighbor (NN), and random heuristics. In the NN heuristic, each ASC travels to the nearest request until all requests are performed. Due to the precedence constraints, sequencing the requests of each ASC regardless of the other one may result in a poor

solution. As a result, we use a modified NN in which we choose a seaside or landside ASC with 50% probability and travel to the nearest requests of the previously carried out requests of that ASC. Table 3 shows that the ALNS outperforms both.

Table 3: A comparison of the ALNS and two heuristics.

Inst	ALNS	NN	G_{NN}^{ALNS}	Rand	G_{rand}^{ALNS}
Expr. 2 Inst. 1	356.32	473.25	24.71	490.70	27.39
Expr. 4 Inst. 1	952.04	1293.09	26.37	1348.89	29.42

5. Conclusion

We have modeled and solved a problem arising in a container terminal, consisting of scheduling two ASCs to execute a set of storage and retrieval requests in a block. Several constraints were considered: (1) the ASCs cannot pass each other and, for security reasons, the ASCs must be separated by a safety distance; (2) each storage container must be stacked in a location selected from a set of available open locations; and (3) because of waiting times and of the presence of several container transport modes, containers have different storage and retrieval priorities. We have formulated the problem as a multiple AGTSP with precedence constraints and ASC interaction constraints. Due to the complexity of the problem, it can be only solved exactly for small size instances. We have therefore developed an ALNS heuristic capable of solving instances of realistic sizes. Our experiments demonstrate that the ALNS outperforms truncated CPLEX and heuristics.

References

[1] Kim, K. H., and Kim, K. Y., "An Optimal Routing Algorithm for a Transfer Crane in Port Container Terminals," *Transportation Science*, 33, 1, 17-33 (1999).

[2] Li, W., Y. Wu, M. E. H. Petering, M. Goh, and de Souza, R. , "Discrete Time Model and Algorithms for Container Yard Crane Scheduling," *European Journal of Operational Research*, 198, 1, 165-172 (2009).

[3] Narasimhan, A., and Paleka, U. S., "Analysis and Algorithms for the Transtainer Routing Problem in Container Port Operations," *Transportation Science*, 36, 1, 63-78 (2002).

[4] Ng, W. C. Crane Scheduling in Container Yards with Inter-Crane Interference. European Journal of Operational Research, 164, 1, 64-78 (2005).

[5] Ropke, S., and Pisinger, D., "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows," *Transportation Science*, 40, 4, 455-472 (2006).

[6] Shaw, P., "A New Local Search Algorithm Providing High Quality Solutions to Vehicle Routing Problems," *Technical Report*, Glasgow (1997).

[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] Zhang, C., Y. W. Wan, J. Liu, and Linn, R. J., “Dynamic Crane Deployment in Container Storage Yards,” *Transportation Research Part B: Methodological*, 36, 6, 537-555 (2002).