RAIRO-Oper. Res. 55 (2021) 115–133
https://doi.org/10.1051/ro/2020135

RAIRO Operations Research
www.rairo-ro.org

SOLVING METHODS FOR THE QUAY CRANE SCHEDULING PROBLEM AT PORT OF TRIPOLI-LEBANON

Ali Skaf[1,*], Sid Lamrous[1], Zakaria Hammoudan[2] and Marie-Ange Manier[1]

Abstract. The quay crane scheduling problem (QCSP) is a global problem and all ports around the world seek to solve it, to get an acceptable time of unloading containers from the vessels or loading containers to the vessels and therefore reducing the docking time in the terminal. This paper proposes three solutions for the QCSP in port of Tripoli-Lebanon, two exact methods which are the mixed integer linear programming and the dynamic programming algorithm, to obtain the optimal solution and one heuristic method which is the genetic algorithm, to obtain near optimal solution within an acceptable CPU time. The main objective of these methods is to minimize the unloading or the loading time of the containers and therefore reduce the waiting time of the vessels in the terminals. We tested and validated our methods for small and large random instances. Finally, we compared the results obtained with these methods for some real instances in the port of Tripoli-Lebanon.

Mathematics Subject Classification. 90B06, 90B35, 90C11, 90C39.

Received November 4, 2019. Accepted November 22, 2020.

1. Introduction and related works

This study addresses the QCSP at port of Tripoli-Lebanon. Lebanon has two ports, the first one is located in the north (port of Tripoli) and the other one is located in Beirut (port of Beirut) which is the largest port. The port of Tripoli-Lebanon has 3 zones: water zone, land zone and dump zone with areas 2 200 000 m^2, 320 000 m^2 and 480 000 m^2 respectively. It receives about 400 ships every year.

In August 2016, the container transportation and shipping company CMA CGM (Compagnie Maritime d'Affrètement (CMA) and Compagnie Générale Maritime (CGM)) released a route from south Asia to north Europe passing by the port of Tripoli-Lebanon, as a transshipment port. This will transfer the port from the secondary to the main center. Therefore, about 800 containers will pass through the port of Tripoli-Lebanon in a week, and finally the vessels will move to their intended ports.

The quay crane is a machine used to unload containers from the vessels or to load containers on the vessels, and the container is an enclosure for holding products such as cars, food, wood... (Fig. 1). Each vessel is divided into many rows, and many bays and contains a specific number of containers (Fig. 2). (Figure reference: `https://www.morethanshipping.com`).

Keywords. Optimization, quay crane, scheduling problem, MILP, dynamic programming, genetic algorithm, case study.

[1] Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, UTBM, Belfort 90010, France.

[2] Jinan University (JU), Tripoli 1300, Lebanon.

[*]Coresonding author: `ali.skaf@utbm.fr`

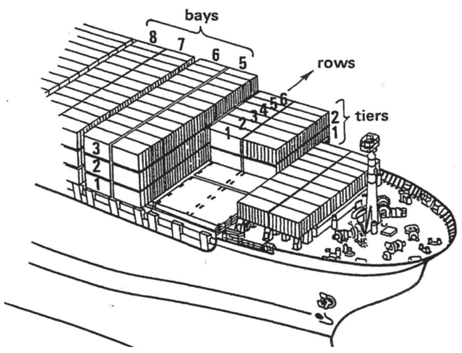


FIGURE 1. Quay crane.

FIGURE 2. Vessel-quay cranes-bays.

The scheduling problem is a popular problem in many maritime ports. In this paper we propose a solution for the scheduling problem in the port of Tripoli-Lebanon. Many researchers have proposed the QCSP and have proposed methods for solving problems such as exact and heuristic methods. Some of them are mentioned in the list below:

Daganzo [8] is one of the first searcher who studied the QCSP for multiple container vessels. His suggestion was to divide the container vessels into bays, where only one quay crane can work on a bay at a time. These quay cranes are free to move quickly from one bay to another, and container vessels cannot depart until all their bays are unloaded. Minimizing the cost of delay is the main purpose, with an exact and an approximate solution method. Furthermore, a branch and bound method for the static QCSP was developed by Peterkofsky *et al.* [21], only if the quay cranes could cross over each other, while Kim *et al.* [14] discussed the QCSP with non-interference constraints, considering the case of a single container vessel. The main purpose was to reduce the total completion time of all quay cranes.

Some of the researchers proposed a mixed-integer linear programming model:

Azza *et al.* [4] found the handling sequence of tasks at ship. Their objective is to minimize the time spent by the vessel at berth and minimize the total cost. Moreover, they proposed an ant colony algorithm to solve the problem. Meanwhile, Alnaqbi *et al.* [2] aimed to minimize the latest starting time for the vessel and its handling time. Furthermore, they proposed a hybrid algorithm to solve the problem. At the same time, Al Awar *et al.* [3] proposed a solution to minimize the entire processing time for all vessels. While, Zhen *et al.* [32] studied the quay crane and yard truck scheduling problem to minimize the total handling time of all tasks, a particle swarm optimization algorithm-based solution was proposed with the MILP. Finally, Xiazhong *et al.* [30] found the optimal storage location of container groups while minimizing both the maximum completion time and the traveling time of trucks. Moreover, they proposed a tabu search to solve the problem.

Others proposed a mixed-integer programming model:

Moghaddam *et al.* [27] presented a novel, MIP model for the quay crane scheduling and assignment problem, and Al-Dhaheri *et al.* [1] determined the sequence of unloading operations of a vessel that a set of quay cranes will perform so that the completion time of the operations is minimized.

And also the genetic algorithm was proposed by many researchers:

Liang *et al.* [15] studied the berthing position problem and determined the number of quay cranes assigned to each ship. The objective was to reduce the handling time and the waiting time for each vessel. While, Chung *et al.* [7] proposed a modified genetic algorithm to deal with the problem and to test the optimization reliability of the proposed algorithm. A set of well-known benchmarking problems was solved, Furthermore, Diabat *et al.* [9] developed a formulation for the quay crane scheduling problem and quay crane assignment which accounts for crane position. Moreover, Salhi *et al.* [22] proposed a model that combines three distinct problems, namely berth allocation, quay crane assignment, and quay crane scheduling that arises in container ports.

And several resolution algorithms were proposed:

Lim *et al.* [16] assumed that a container vessel is a job. When this job is assigned to quay crane, there is a profit value. They found an assignment matching which maximizes the total profit. A dynamic programming algorithm and a Tabu search are used to provide a solution to the problem. Moreover, Yi *et al.* [31] proposed a general model for the QCSP and introduced a heuristic model to solve it. Their objective is to minimize the total handling time of all tasks. Furthermore, Liu *et al.* [17] proposed models and algorithms for the general double-cycling problem with internal-reshuffles, where reshuffle containers can move directly from one stack to another. A branch-and-price framework is used to solve the problem. And Msakni *et al.* [19] studied the problem of determining the assignment of quay cranes to containers vessels and the scheduling operation by each quay crane. They developed an exact method based on a branch-and-price algorithm.

Finally, some researchers have suggested studies and classification schemes:

Steeken *et al.* [26] studied the QCSP to increase the competition's capacity of port container terminals. Then, Wang *et al.* [28] determined the sequence of unloading operation, with the objective to minimize the weighted operation time of jobs and travel time. After that, Wang *et al.* [29] made a study to improve knowledge for quay crane and berth scheduling. Boysen *et al.* [6] presented a classification scheme for the QCSP without considering the non-interference constraints between the quay cranes. A scheme was applied to classify the existing literature, to determine the status-quo of complexity results, and to identify future research. While, De Oliveira *et al.* [20] focused on the integration of quay side operations and the QCSP taking in consideration the minimization of the time spent and the fuel consumption. And, Liu *et al.* [18] studied a scheduling problem with two uniform quay cranes. Their objective was to minimize the turn-around time of a vessel. Finally, Haoyuan *et al.* [11] proposed a simulation model of container terminal, to solve its quay crane scheduling problem, with the objective to optimize the shortest total delay time for all vessels.

In our previous studies, Skaf *et al.* [23] studied the QCSP with non-interference constraints in the port of Tripoli-Lebanon. They proposed a mixed-integer linear programming (MILP) model and a dynamic programming algorithm to solve the problem. After that, Skaf *et al.* [24] studied the QCSP with non-interference constraints in the port of Tripoli-Lebanon, they proposed a genetic algorithm to solve the problem, and finally they compared the obtained results with the mentioned port results.

The above review suggests that methods such as genetic algorithm are adapted to solve the Quay Crane Scheduling Problem (QCSP).

Meanwhile, Skaf *et al.* [25] aimed to study both the quay crane and yard truck scheduling problem for a single quay crane and multiple yard trucks, the main objective is to minimize the total completion time for all containers. They proposed a MILP and a dynamic programming to solve the problem.

Table 1 shows a classification of works according to the resolution methods (exact or heuristic) and according to the variant of the studied problem. We note that most researchers have used the MILP as an exact method and the genetic algorithm (GA) as a heuristic method to solve the QCSP.

Table 1. Classification of works.

References	Variant	Resolution methods	
		Exact	Heuristic
Lim *et al.* [16]	QCSP	DPA	Tabu search
Liang *et al.* [15]	QCSP		GA
Diabat *et al.* [9]	QCSP		GA
Al-Dhaheri *et al.* [1]	QCSP	MIP	
Moghaddam *et al.* [27]	QCSP	MIP	
Chung *et al.* [7]	QCSP		GA
Azza *et al.* [4]	QCSP	MILP	Ant-colony
Alnaqbi *et al.* [2]	QCSP	MILP	Hybrid
Al Awar *et al.* [3]	QCSP	MILP	
Salhi *et al.* [22]	QCSP		GA
Zhen *et al.* [32]	QCYTSP	MILP	PSO
Xiazhong *et al.* [30]	QCYTSP	MILP	Tabu search
This study	QCSP	MILP, DPA	GA

Notes. QCSP: quay crane scheduling problem. QCYTSP: quay crane and yard truck scheduling problem.

In this study, we propose three solving methods for the scheduling problem of unloading tasks performed by several cranes assigned to a single vessel in ports like that of Tripoli-Lebanon. Those methods are a MILP, a dynamic programming algorithm (DPA) and a genetic algorithm. After that, we compare all obtained results for those methods with real results from the mentioned port. The proposed MILP and the DPA allow us to find all possible assignments for the quay cranes to the bays, taking into consideration the non-interference constraints between quay cranes. Their main objective is to minimize the total completion time.

The proposed genetic algorithm aims to minimize the total completion time and to find the near optimal solutions. This study allows to find the optimal completion time for a container's vessel. We observe that the dynamic programming algorithm provided better results than the mixed-integer linear programming in terms of CPU time and completion time for large number of bays. Nevertheless, the drawback of the DPA is that for larger instances, for example, for 22 bays, it may take 4 h to give a result, and therefore in such cases, we propose a genetic algorithm to obtain optimal or near optimal results with a quick execution time for the quay cranes scheduling problem without cranes crossing.

Later in this paper, in Section 2 we define the studied problem. In Section 3 we propose three solving methods. In Section 4 we compare the results of the three solving methods and the real results in the port of Tripoli-Lebanon. Finally, in Section 5 we give a conclusion and a direction for future research.

2. Problem definition and mathematical formulation

In this section, we provide the problem definition and the mathematical formulation such as the assumptions, the needed parameters and the decision variables used in the mathematical model.

We study a QCSP with non-interference constraints for a single vessel unloaded by several quay cranes, as can be encountered in the port of Tripoli-Lebanon, using three solving methods to improve the results of such port.

2.1. Assumptions

The QCSP we study has the following characteristics:

- A single vessel is considered, which transports containers. Its surface is divided into many bays and each bay contains a specific number of containers put in a row or over each other. (Cranes number $<$ bays number).
- The quay cranes move along a single-track, this means that they work without any crossing between them.

– Each quay crane is able to unload containers from only one bay at a time, when it completes its work in the current bay, it can pass to the next one.
– Each quay crane can unload one container at a time.
– Each bay is handled by at most one quay crane at a time.
– We ignore the travel time between two bays because it is short compared to the unloading times.
– No idle quay crane.

2.2. Mathematical formulation

In this sub-section, we present the mathematical formulation of the problem with the different parameters, variables and the mathematical model.

2.2.1. Data

– Q: set of quay cranes, with $|Q|$: number of quay cranes.
– i: index of quay crane ($i = \{1 \dots |Q|\}$).
– q_i: the ith quay crane ($i = \{1 \dots |Q|\}$).
– B: set of bays, with $|B|$: number of bays.
– j: index of bay ($j = \{1 \dots |B|\}$) on the vessel.
– C_j: number of containers in bay j.
– Tc: time to unload a container by a quay crane and put it in the storage. It is supposed to be the same for all containers in all bays.
– M: great integer number.

2.2.2. Decision variables

$\forall i \in Q, \forall j, j' \in B.$

– $x_{(j,i)} = \begin{cases} 1 & \text{if bay } j \text{ is handled by quay crane } i \\ 0 & \text{otherwise.} \end{cases}$
– $z_{(j,j')} = \begin{cases} 1 & \text{if the unloading of bay } j \text{ finishes before starting to unload bay } j' \\ 0 & \text{otherwise.} \end{cases}$
– t_j : completion time of bay j.
– $C_{\max}$: makespan.

2.2.3. Mathematical model

Here we show some equations to facilitate the understanding of the problem.

$$t_j \leq t_{j'} \qquad \forall j, j' \in B \tag{2.1}$$
$$t_j \leq t_{j'} - (Tc \cdot C_{j'}) \qquad \forall j, j' \in B \tag{2.2}$$
$$t_j > t_{j'} - (Tc \cdot C_{j'}) \qquad \forall j, j' \in B \tag{2.3}$$
$$q_i + 1 \leq q_{i+1} \qquad \forall q_i \in Q. \tag{2.4}$$

Equations (2.1) means, if there are two successive bays to be worked inside, then the completion time of the second one is greater than the first one. Equation (2.2) is used when the work at bay j finishes before than starting the work at bay j', while equation (2.3) is used when the work at bay j finishes after the one at bay j' starts. Equation (2.4) is used if quay crane q_i works in bay j and quay crane q_{i+1} works in bay j' to prevent the crossing between quay cranes.

3. Solving methods

In this section, we will propose three solving methods: MILP solved by CPLEX and dynamic programming algorithm coded with Java to obtain exact solutions, and a genetic algorithm which is a meta-heuristic method coded in Java to obtain near optimal solutions.

3.1. Mixed integer linear programming

The mixed integer linear formulation was found by Leonid Kantorovich [13]. It is a technique for the optimization of an objective function, subject to linear inequality and equality constraints.
The MILP we propose is as follows:

Objective

$$\text{Minimize} \quad C_{\max} \tag{3.1}$$

Subject to

$$\sum_{i=1}^{|Q|} x_{(j,i)} = 1 \qquad \forall j \in B \tag{3.2}$$

$$t_j \geq (Tc \cdot C_j) \qquad \forall j \in B \tag{3.3}$$

$$t_j - t_{j'} + (Tc \cdot C_{j'}) + z_{(j,j')} \cdot M > 0 \qquad \forall j, j' \in B \tag{3.4}$$

$$t_j - t_{j'} + (Tc \cdot C_{j'}) - (1 - z_{(j,j')}) \cdot M \leq 0 \qquad \forall j, j' \in B \tag{3.5}$$

$$\sum_{i=1}^{|Q|} i \cdot x_{(j,i)} + 1 \leq \sum_{i'=1}^{|Q|} i' \cdot x_{(j',i')} + (z_{(j,j')} + z_{(j',j)}) \cdot M \qquad \forall j, j' \in B, j < j' \tag{3.6}$$

$$C_{\max} = \max_j \; t_j \tag{3.7}$$

$$x_{(j,i)} = \{0,1\} \qquad \forall j \in B, \forall i \in Q \tag{3.8}$$

$$z_{(j,j')} = \{0,1\} \qquad \forall j, j' \in B, j < j'. \tag{3.9}$$

Equation (3.1) is the objective function that it is desired to minimize the makespan (completion time of all bays). Constraint (3.2) ensures that every bay must be handled only by one quay crane. Constraint (3.3) ensures that the completion time is bigger than the working time in each bay (Working time = number of containers $\times$ time needed to unload a container and store it). Constraint (3.4) indicates that if $z_{(j,j')} = 1$, then bay j finishes before bay j' starts (Eq. (2.2)) and constraint (3.5) indicates if $z_{(j,j')} = 0$, bay j finishes after bay j' starts (Eq. (2.3)). Constraint (3.6) avoids the interference between the quay cranes: Suppose that bays j and j' are performed simultaneously and $j < j'$, if the first quay crane works in bay j and the second quay crane works bay j', then $q_1 + 1 \leq q_2$ (Eq. (2.4)). Constraint (3.7) defines the $C_{\max}$ value, finally constraints (3.8) and (3.9) define the property of the decision variables.

3.2. Dynamic programming algorithm

3.2.1. Definition

Dynamic programming algorithm was found by Richard Bellman [5]. It is used for problems optimization to explore the optimal solution. The dynamic programming algorithm (DPA) consists in dividing the problem into many small sub-problems and after solving each sub-problem only once, and ultimately, the problem has been solved. Finally, we can save the results and select the obtained optimal solution. The dynamic programming problems can be classified into two categories:

– **Optimization problems**

Selecting the feasible solution with the value of the function is maximized or minimized.

– **Combinatorial problems**

Figuring out the number of ways to do something, or the probability of some events happening.

3.2.2. Hierarchy tree diagram

A hierarchical chart represents a system of hierarchy progressing from the top to the bottom and can also be referred to as a structure chart.

The first node of the tree is called the root. If this root node is connected by another node, the root is then a parent node and the connected node is a child. All Tree nodes are connected by links called edges. It's an important part of trees, because it manages the relationship between nodes.

We start by generating all parameters and we finish by obtaining the optimal solution.

The DPA works depending on the hierarchy tree diagram system. As shown below, a detailed tree example of the DPA is provided:

We start by developing all the needed conditions in the problem. After we get all possible choices whatever the CPU time will be high, with a single solution for each choice. Finally, we choose the optimal solution which is the smallest one.

3.2.3. Algorithm description

Algorithm 1. Dynamic programming algorithm.

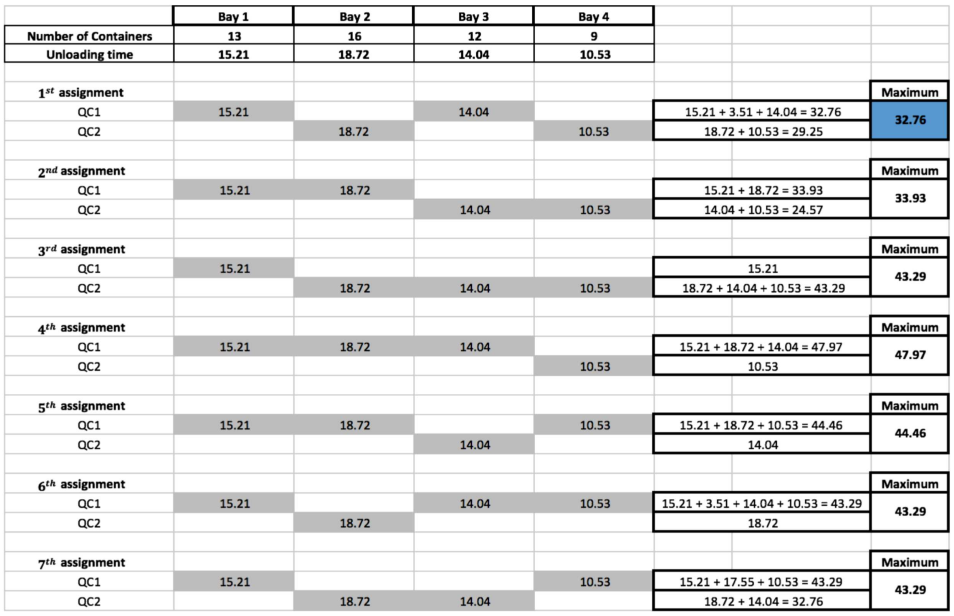
```
 1: Generate_variables();
 2: Generate_assignments();
 3: for each assignment do
 4:    if crossing = false then
 5:       calculate_completion_time();
 6:    else
 7:       go_to_next_assignment;
 8:    end if
 9: end for
10: min←value_first_assignment;
11: for each assignment do
12:    if min > value_next_assignment then
13:       min←value_next_assignment;
14:       End;
15:    else
16:       go_to_next_assignment;
17:    end if
18: end for
```

Lines explanation

- Line 1: generating all needed variables (number of quay cranes, number of bays, number of containers in a specific bay, and time to unload a container by the quay crane).
- Line 2: generating all possible choices for the assignment of all bays to quay cranes.
- Lines 3–9: for each assignment, we test if there is any crossing between the quay cranes. If yes, we test another assignment; if there is no crossing, we calculate the completion time.
- Line 10: create a new variable called min and give it the calculated value of the first assignment.
- Lines 11–18: for each assignment, we test if the completion time is less than the variable min, if yes set min to the new completion time, or else go to the next assignment.

3.2.4. Process example

We suppose that we have 2 quay cranes with 4 bays containing 13, 16, 12 and 9 containers respectively to be unloaded. The container's unloading times of these 4 bays are respectively 15.21, 18.72, 14.04 and 10.53 time units (Fig. 3 describes the full process for this example).



	Bay 1	Bay 2	Bay 3	Bay 4		
Number of Containers	13	16	12	9		
Unloading time	15.21	18.72	14.04	10.53		
1st assignment						Maximum
QC1	15.21		14.04		15.21 + 3.51 + 14.04 = 32.76	32.76
QC2		18.72		10.53	18.72 + 10.53 = 29.25	
2nd assignment						Maximum
QC1	15.21	18.72			15.21 + 18.72 = 33.93	33.93
QC2			14.04	10.53	14.04 + 10.53 = 24.57	
3rd assignment						Maximum
QC1	15.21				15.21	43.29
QC2		18.72	14.04	10.53	18.72 + 14.04 + 10.53 = 43.29	
4th assignment						Maximum
QC1	15.21	18.72	14.04		15.21 + 18.72 + 14.04 = 47.97	47.97
QC2				10.53	10.53	
5th assignment						Maximum
QC1	15.21	18.72		10.53	15.21 + 18.72 + 10.53 = 44.46	44.46
QC2			14.04		14.04	
6th assignment						Maximum
QC1	15.21		14.04	10.53	15.21 + 3.51 + 14.04 + 10.53 = 43.29	43.29
QC2		18.72			18.72	
7th assignment						Maximum
QC1	15.21			10.53	15.21 + 17.55 + 10.53 = 43.29	43.29
QC2		18.72	14.04		18.72 + 14.04 = 32.76	

FIGURE 3. Process example.

Our DPA generates all variables and then all crane-to-bay assignments. In this case, for 4 bays and 2 cranes, there is initially 2^4, or 16 possible assignments which are: 1–1–1–1, 1–1–1–2, 1–1–2–1, 1–1–2–2, 1–2–1–1, 1–2–1–2, 1–2–2–1, 1–2–2–2, 2–1–1–1 , 2–1–1–2, 2–1–2–1, 2–1–2–2, 2–2–1–1, 2–2–1–2, 2–2–2–1, 2–2–2–2. In these x–y–z–t type lists, the value in ith position represents the number of the crane assigned to unload bay i.

In a second step, the 2 assignments (1–1–1–1 and 2–2–2–2) are eliminated because all the cranes reserved for the vessel are all supposed to unload containers. We are therefore forbidden here to use only one crane.

In the next step, a crossing test is carried out, which leads to eliminating the assignments which start with 2. In fact, if the second crane works in the first bay there will be a crossing between the cranes. Note that the reverse is not true: the quay crane 1 can unload the containers from bay 4, however it cannot start with this bay, which will be limiting in terms of the scheduling phase.

- The first assignment is 1–2–1–2, means that the first quay crane (QC1) unloads the containers from bays 1 and 3 for a theoretical completion time of 29.25 (15.21 + 14.04), while the second quay crane (QC2) unloads bays 2 and 4 with an effective working time of 29.25 (18.72 + 10.53). The *makespan* should theoretically be 29.25. However, once bay 1 has been unloaded, QC1 cannot start to unload bay 3 because QC2 does not finish its work in bay 2 until 18.72. QC1 must therefore wait 3.51 units of time, and then the two cranes can move simultaneously towards bays 3 and 4 respectively. The actual end of work date of QC1 is therefore not 29.25 because it is necessary to add a time of waiting for 3.51 to avoid a collision, which is 32.76. The *makespan* is therefore equal to 32.76.
- The second assignment is 1–1–2–2, means that QC1 unloads containers 1 and 2 with a completion time of 33.93, while QC2 unloads containers in bays 3 and 4 with a time completion of 24.57. The total time to perform the unloading is therefore equal to 33.93.
- The third assignment is 1–2–2–2, means that QC1 unloads the 13 containers from bay 1 with a completion date of 15.21, while QC2 unloads the containers from bay 2, 3 and 4 and completes work on date 43.29. The *makespan* is therefore equal to 43.29.
- The fourth assignment is 1–1–1–2: QC1 unloads bay 1, 2, and 3 until date 47.97, while QC2 takes care of bay 4 with a completion date of 10.53. The total time for unloading the vessel is therefore equal to 47.97.

- The fifth assignment is 1–1–2–1: QC1 unloads the containers from bays 1, 2 and 4 and ends at 44.46, QC2 finishes unloading bay 3 at 14.04. The time to unload the 50 containers is therefore equal to 44.46.
- The sixth assignment is 1–2–1–1: QC1 unloads the containers from bays 1, 3 and 4 and theoretically ends at 39.78 (15.21 + 14.04 + 10.53), but actually at 43.29. Indeed, QC2 takes care of bay 2 until the date 18.72, and once bay 1 is unloaded at 15.21, QC1 must wait for the end of the work of QC2 to be able to move to bays 3 and 4. QC1 must therefore undergo a 3.51 wait $(18.72 - 15.21)$. The time to unload the 50 containers is therefore equal to 43.29 (39.78 + 3.51).
- The seventh assignment is 1–2–2–1: the QC1 crane (respectively QC2) unloads the containers from bays 1 and 4 (respectively 2 and 3) with an end date of 43.29 (respectively 32.76). The *makespan* is therefore equal to 43.29. As for scenario 6, this date includes a waiting time for QC1, here of 17.55 time units between the unloading of bays 1 and 4, in order to allow QC2 time to finish its work in bays 2 and 3 and avoid thus a collision.

3.2.5. Summary

The dynamic programming algorithm (DPA) allows to find all possible completion times which grow when the number of quay cranes and bays increases, taking into consideration the non-crossing conditions for the quay cranes. Finally, it allows to choose the optimal completion time.

3.3. Genetic algorithm

After testing large instances using the MILP and dynamic programming algorithm, we observed that those two methods took a long CPU time to solve the problem and obtain the optimal solution (as we will see later in the next section). We propose a meta-heuristic method which is the genetic algorithm (GA) to obtain near optimal solution with fast CPU time. The genetic algorithm (GA) was proposed by John Holland [12] and developed and applied by David Goldberg [10].

In this paper, GA depends on Roulette wheel selection, order crossover and swap mutation.

The genetic algorithm steps are presented as follows: generation of an initial population of solutions, after that, test of the non-interference constraints for the quay cranes to avoid the crossing between them, if there is no crossing, calculation of the fitness value, if not setting of the fitness to zero. If the current generation is the final one, then the program is terminated, and the results are provided. If not, we execute the GA steps which are roulette wheel selection, order crossover and swap mutation.

We choose the genetic algorithm due to its simplicity, as it is easy to be implemented. Figure 4 describes the flowchart of this genetic algorithm.

3.3.1. Representation of a chromosome

The solution in the genetic algorithm is called a chromosome and each element inside is called gene. In our study, the chromosome is composed of a series of bays. Table 2 shows a chromosome and each gene contains the bay number. For example, the third element (third gene) in the chromosome represents the bay number 4.

3.3.2. Initial positions of quay cranes

To fix the initial position of a quay crane we use the following equation: $1 + (i - 1) \times\ R$, where i is the index of the quay crane and R a random number between 1 and L, with $L =$ bays number/quay cranes number. If L is a rational number, we take its natural part number. Be aware that the position of the first quay crane is always at the first bay.

Example: Let us suppose that we have 12 bays and 3 quay cranes, then R should be a random number between 1 and 12/3 (hence between 1 and 4). If $R = 2$, then the main position of quay crane 1 is at bay 1, quay crane 2 is at bay 3 and quay crane 3 is at bay 5.

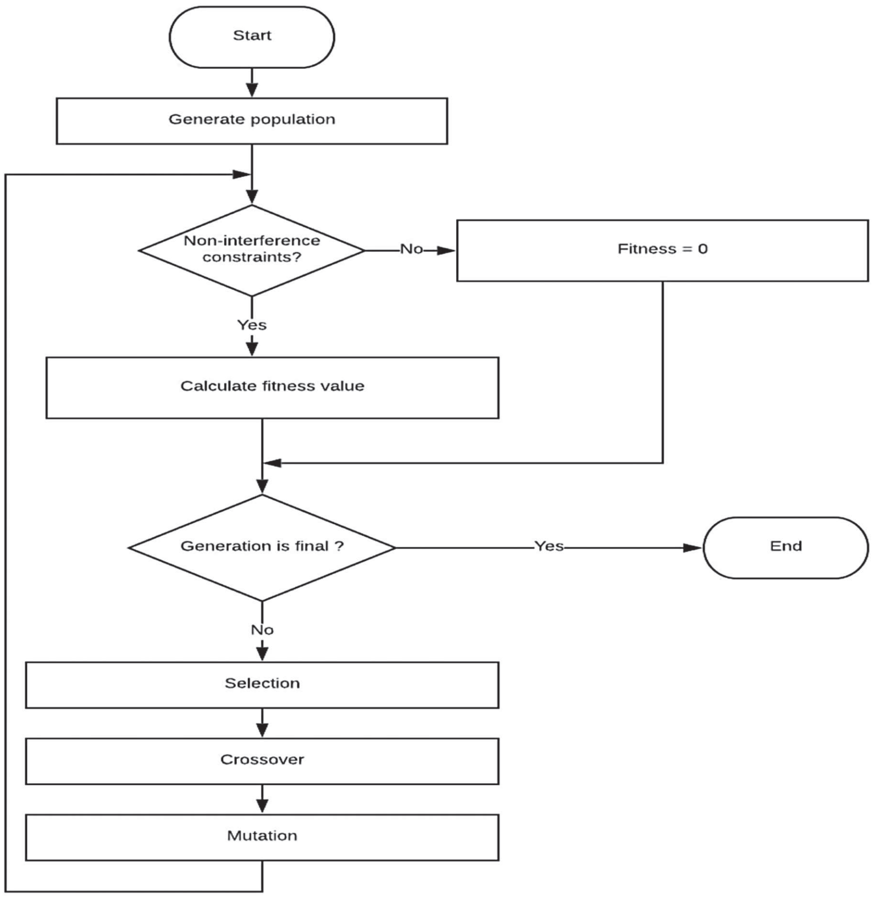


FIGURE 4. Genetic algorithm flowchart.

TABLE 2. Chromosome and genes.

<u>7</u>	9	4	10	3	8	6	1	2	5

3.3.3. Quay cranes scheduling

Algorithm 2 explains how the scheduling problem works. The initial positions of the quay cranes, and therefore the bays will be assigned to the quay cranes. A sequential procedure is used to consider the order of the bays in the chromosome.

Algorithm 2. Quay cranes scheduling.

```
if $q_{nb} = 1$ then
    $q_1 \leftarrow bay_j$;
    $bay_j \leftarrow -1$;
    j ← j+1;
    $comp_q \leftarrow comp_q + w_{bay_j}$;
end
else
    if $comp_{q_i}$ Not in $\{comp_q\}/comp_{q_i}$ then
        $comp_{q_i} = \mathrm{Min}\{comp_q\}$;
        $bay_j \leftarrow -1$;
        j = Next_bay();
        $comp_{q_i} \leftarrow comp_{q_i} + w_{bay_j}$;
    end
    else
        if $\Delta(bay) > 0$ then
            $q_i \leftarrow bay_j$ with shorter distance;
            $bay_j \leftarrow -1$;
            j = Next_bay();
            $comp_{q_i} \leftarrow comp_{q_i} + w_{bay_j}$;
        end
    end
    else
        $q_i \leftarrow bay_j$ with smaller index number;
        $bay_j \leftarrow -1$;
        j = Min_bay();
        $comp_{q_i} \leftarrow comp_{q_i} + w_{bay_j}$;
    end
end
```

$comp_q$: quay crane's completion time // bay_j: bay number j // q_{nb}: number of quay cranes // q_i: the ith quay crane // w_{bay_j}: quay crane's time to finish working in bay j // $\Delta = |bay_{k2} - bay_{k1}|$: calculate the distance between bays// Next_bay(): get the next bay, Min_bay(): get the smaller bay number

The four following steps of this algorithm are featured. For each step we clarify it with the numerical example used in Section 3.3.1 with 3 quay cranes and 10 bays. In this example, the main makespan (completion time) of all the quay cranes is initialized to 0, because no unloading operation has been performed until now (iteration init in Tab. 3).

In the chromosome, we consider the current bay (current gene) which corresponds to an unassigned bay, and we assign it to a crane. For example, let us take the first gene of the chromosome (bay 7), and we suppose the processing time for all bays is 10.

Non-crossing (step 1): the first step consists to determine which quay cranes can unload the first unassigned bay in the chromosome without crossing with the other quay cranes. If there is only one available quay crane, this bay is assigned to this quay crane, else we continue to the next step. For our example, quay crane 2 and quay crane 3 can handle bay 7 without interference with quay crane 1.

Selection based on time criteria (step 2): among these two cranes, we choose the one which is available sooner. In case of equality, that means if the two cranes have the same completion time, then we go to step 3 to discriminate them according to a distance criterion. In our example, the completion times of quay crane 2 and quay crane 3 are equal.

Table 3. Detailed example.

Iteration		QC1	QC2	QC3
init	Bay position	1	3	5
	Partial scheduling	∅	∅	∅
	Completion time	0	0	0
1	Bay position	1	3	7
	Partial scheduling	∅	∅	7
	Completion time	0	0	10
2	Bay position	1	3	9
	Partial scheduling	∅	∅	7, 9
	Completion time	0	0	20
3	Bay position	1	4	9
	Partial scheduling	∅	4	7, 9
	Completion time	0	10	20
4	Bay position	1	4	10
	Partial scheduling	∅	4	7, 9, 10
	Completion time	0	10	30
5	Bay position	3	4	10
	Partial scheduling	3	4	7, 9, 10
	Completion time	10	10	30
6	Bay position	3	8	10
	Partial scheduling	3	4, 8	7, 9, 10
	Completion time	10	20	30
7	Bay position	6	8	10
	Partial scheduling	3, 6	4, 8	7, 9, 10
	Completion time	20	20	30
8	Bay position	1	8	10
	Partial scheduling	3, 6, 1	4, 8	7, 9, 10
	Completion time	30	20	30
9	Bay position	1	2	10
	Partial scheduling	3, 6, 1	4, 8, 2	7, 9, 10
	Completion time	30	30	30
10	Bay position	2	5	10
	Completed scheduling	3, 6, 1	4, 8, 2, 5	7, 9, 10
	Completion time	30	40	30

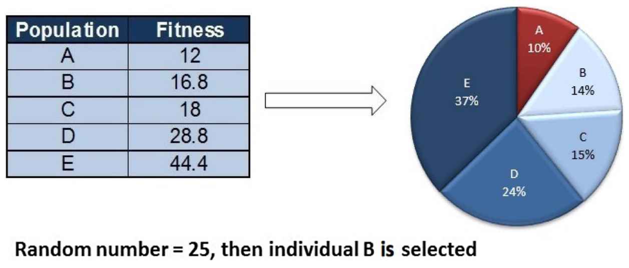
Selection based on distance criteria (step 3): if no crane has been chosen in step 2 with the time criterion, we assign the current bay to the nearest crane. In case of equality (if the two cranes are at equal distance from the bay), we choose the crane at the left (with the smallest index). The distance between bay 7 and bay 3 (which quay crane 2 works inside) is 3 bays and the distance between bay 7 and bay 5 (which quay crane 3 works inside), is 1 bay. Then crane 3 is nearer to bay 7 than crane 2.

Assignment (step 4): assign bay 7 to quay crane 3. Update the position of quay crane 3 and the associated completion time (see iteration 1 in Tab. 3). Then update the current gene (following one) in the chromosome (gene 2, bay 9 in our example). Repeat the four steps until all bays are assigned to one quay crane.

Algorithm 2 describes the process of the quay cranes scheduling while Table 3 shows a detailed numerical example for the scheduling.

3.3.4. Fitness

In the previous section, we have seen a procedure to prevent the interference between the quay cranes. Nevertheless, we should check if the quay cranes satisfy the constraints or not, so we used the two constraints



Population	Fitness
A	12
B	16.8
C	18
D	28.8
E	44.4

Figure 5. Roulette wheel selection.

(3.4) and (3.5) and then checked the interference constraint (3.6) to ensure that there is no collision between quay cranes. The fitness value will be calculated from the equation (3.6). If the constraints are not satisfied, the fitness value will be set to zero.

$$\text{Fitness} = 1/C_{\max}. \tag{3.10}$$

3.3.5. Roulette wheel selection

The roulette wheel selection is a process to choose individuals from a population, according to the "fitness" function or the values of their cost, to form a new population. Individuals will be evolved by some successive iterations of selection, called generations.

All individuals are selected proportionally to their "fitness" function, so an individual with a higher fitness function will be more likely to be selected than another with a lower fitness value. This function can be considered as a measure of profit or quality that one wishes to maximize. A simple operator of selection is the technique of the weighted roulette where each individual of a population occupies a surface of the roulette proportional to the value of its function "fitness". For reproduction, candidates are selected with a probability proportional to their "fitness". For each selection of an individual, a simple rotation of the wheel gives the selected candidate. Roulette Wheel Selection works as follows:

Calculate the sum of all fitness, then generate a number between 0 and this sum; add the fitness values to a partial sum X starting from the top of population, and finally the chosen chromosome is the first chromosome for which X overpasses the random number. The procedure is illustrated in Figure 5.

3.3.6. Order crossover

The order crossover procedure aims to select randomly a substring from a random parent. After that we create an offspring by copying the selected substring in their corresponding positions. Then, we delete from the second parent all existing in the substring and place the genes in the empty positions of the offspring from left to right. Finally, we should take into consideration the order of the sequence of the offspring creation. The procedure is illustrated in Figure 6.

3.3.7. Swap mutation

The swap mutation procedure aims to test all the individuals bit by bit.

From the chromosome, we randomly select two positions (two genes) and then we swap the values of these two positions.

The procedure is illustrated in Figure 7.

4. Experimental results

In this section, we evaluate the performance of our methods (MILP, dynamic programming, and genetic algorithm). For this goal, we test them on randomly instances, as well as on a real case related to the port of Tripoli-Lebanon.

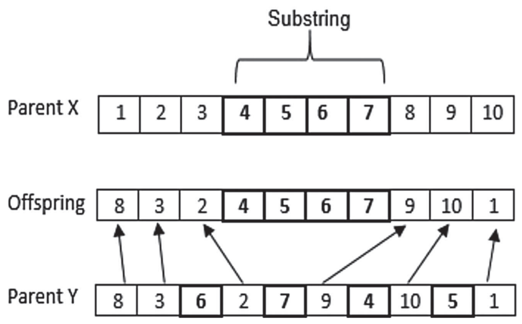


FIGURE 6. Order crossover.

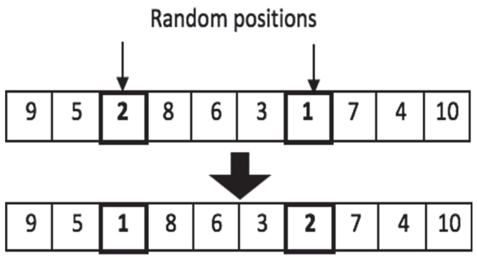


FIGURE 7. Swap mutation.

Our programs were executed in a MacBook Pro 2,7 GHz Intel Core i5 with 8 GB RAM 1867 MHz DDR3 running in OSX 10.11.6.

We fix the size of the population, the probability of the mutation, the probability of the crossover and the max number of generations when executing the genetic algorithm which are respectively 300, 0.2, 0.25 and 1000.

Small and large sizes instances were generated randomly with 2 or 3 quay cranes, 4–15 bays, and containers number in each bay were also generated in a random way between 5 and 30 containers.

4.1. MILP *versus* DPA

The MILP was solved by CPLEX 12.7.1, a software for solving integer programming, while the DPA is coded with JAVA J2EE. JAVA is a programming language developed by Sun Microsystems. It was designed to develop programs and creating web applications. It is an object-oriented programming language and its syntax is similar to C++.

As shown in Table 4, CPLEX results and DPA results are logically the same (same makespan) for small and large instances, but DPA is better in terms of CPU time.

For instance 21 with 14 bays and 2 quay cranes, CPLEX did not provide any result after 2 h, so we interrupted the execution, while DPA solved the problem to optimality. But in instance 22 for 14 bays and 3 quay cranes CPLEX gives us a result like DPA. For instances 23 and 24, we waited more than 3 h using CPLEX without obtaining any result. We can therefore state that DPA is more efficient in terms of time than MILP. We will adopt DPA as the exact method to be compared with the meta-heuristic method.

4.2. DPA *versus* GA

The genetic algorithm (GA) and the DPA are both coded with JAVA J2EE.

Table 5 shows us, that both GA and DPA gave the same optimal solution with an acceptable and realistic CPU time but only for small instances.

TABLE 4. Small and large instances.

No.	($\|Q\| \times \|B\|$)	Makespan		CPU Time	
		CPLEX (min)	DPA (min)	CPLEX (s)	DPA (s)
1	2×4	32.76	32.76	<1	<1
2	3×4	24.57	24.57	<1	<1
3	2×5	39.78	39.78	<1	<1
4	3×5	29.25	29.25	<1	<1
5	2×6	51.48	51.48	<1	<1
6	3×6	37.44	37.44	<1	<1
7	2×7	58.5	58.5	<1	<1
8	3×7	39.78	39.78	<1	<1
9	2×8	73.71	73.71	<1	<1
10	3×8	51.48	51.48	<1	<1
11	2×9	90.09	90.09	<1	<1
12	3×9	60.84	60.84	<1	<1
13	2×10	99.45	99.45	4.77	1.33
14	3×10	70.2	70.2	1.16	0.35
15	2×11	104.13	104.13	24.43	3.89
16	3×11	73.71	73.71	5.57	1.13
17	2×12	107.64	107.64	115.27	13.34
18	3×12	73.71	73.71	7.39	3.28
19	2×13	114.66	114.66	1723.85	32.59
20	3×13	76.05	76.05	20.38	4.57
21	2×14	N.A.	119.34	N.A.	82.44
22	3×14	81.9	81.9	197.83	34.86
23	2×15	N.A.	127.53	N.A.	131.77
24	3×15	N.A.	91.26	N.A.	104.82

Notes. $|Q| \times |B|$: number of quay cranes $\times$ number of bays. N.A. = No result after interrupting execution.

TABLE 5. Small instances.

No.	($\|Q\| \times \|B\|$)	Makespan			CPU Time	
		DP (min)	GA (min)	GAP (%)	DP (s)	GA (s)
1	2×4	32.76	32.76	0	<1	<1
2	3×4	24.57	24.57	0	<1	<1
3	2×5	39.78	39.78	0	<1	<1
5	2×6	51.48	51.48	0	<1	<1

As shown in Table 6, from instance 1 to instance 20, DPA gives us the optimal solutions and GA gives us near optimal solutions, but the difference is the higher CPU time for DPA. The GAP between DPA results and GA results is between 0% and 3.33%. From instance 21, DPA was unable to provide any solution after more than 4 h of CPU processing time, meanwhile GA provides solutions in a very short time. We therefore state that the proposed GA is successful and practical for our problem. (GAP = ((DPA − GA)/DPA) * 100).

4.3. Stability analysis of the genetic algorithm

To evaluate if our genetic algorithm is stable, we executed the same instance 5 times and we got the same result as shown in Table 7 for small instances.

Table 6. Large instances.

No.	$(\|Q\| \times \|B\|)$	Makespan			CPU Time	
		DPA (min)	GA (min)	GAP (%)	DPA (s)	GA (s)
1	2×12	107.64	109.98	2.13	13.34	<1
2	3×12	73.71	74.88	1.56	3.28	<1
3	2×13	114.66	115.83	1.01	32.59	<1
4	3×13	76.05	76.05	0	4.57	<1
5	2×14	119.34	122.85	2.86	112.44	<1
6	3×14	81.9	83.07	1.41	54.86	<1
7	2×15	127.53	129.87	1.8	231.77	<1
8	3×15	91.26	92.43	1.27	104.82	<1
9	2×16	132.21	132.21	0	303.04	2.31
10	3×16	95.94	98.28	2.38	112.94	1.59
11	2×17	139.23	142.74	2.46	509.96	4.36
12	3×17	101.79	105.3	3.33	203.34	3.12
13	2×18	146.25	147.42	0.79	1144.12	9.67
14	3×18	105.3	106.47	1.1	708.32	7.39
15	2×19	153.27	155.61	1.5	1392.84	13.34
16	3×19	113.49	113.49	0	912.48	10.62
17	2×20	161.46	164.97	2.13	1601.13	32.99
18	3×20	118.17	119.34	0.98	1044.19	28.64
19	2×21	170.82	173.16	1.35	3563.7	58.28
20	3×21	125.19	127.53	1.83	2390.13	34.72
21	2×22	N.A.	180.18	–	N.A.	77.14
22	3×22	N.A.	128.7	–	N.A.	60.12
23	2×23	N.A.	186.03	–	N.A.	104.35

Table 7. Stability analysis for small instances.

	$(\|Q\| \times \|B\|)$	$(\|Q\| \times \|B\|)$	$(\|Q\| \times \|B\|)$	$(\|Q\| \times \|B\|)$
Execution	2×4	3×4	2×5	3×5
1	32.76	24.57	39.78	29.25
2	32.76	24.57	39.78	29.25
3	32.76	24.57	39.78	29.25
4	32.76	24.57	39.78	29.25
5	32.76	24.57	39.78	29.25

For the large instances, we also executed the same instance 5 times (Tab. 8). At least two executions have the same near optimal result, with a percentage of 40% while the three other executions give results more or less 3% away from the near optimal result.

As an example, for 2 quay cranes and 12 bays, the first, third and fourth executions give the near optimal solution obtained by the genetic algorithm (109.98), while the second and the fifth executions give results more or less 3% away from the near optimal solution.

Percentage calculation for second execution

- $112.32 - 109.98 = 2.34$
  $(2.34/109.98) * 100 = 2.12\%$ from the near optimal result.

Table 8. Stability analysis for large instances.

	$(\|Q\| \times \|B\|)$	$(\|Q\| \times \|B\|)$
Execution	2×12	3×12
1	109.98*	74.88*
2	113.49	76.39
3	109.98*	77.22
4	109.98*	77.22
5	112.32	74.88*

Notes. *The best near optimal solution.

Table 9. DPA results *versus* Port results *versus* GA results.

Ex.	Cranes	Bays	Containers	Port makespan	DPA	GA	Saved time % Port-DPA
1	2	6	25	40	19.5	21.84	51.25
2	2	4	62	95	44.78	45.95	52.86
3	2	3	90	103	70.2	72.54	31.84
4	2	4	84	96	49.14	52.01	48.81

– $113.49 - 109.98 = 3.51$
 $(3.51/109.98) * 100 = 3.19\%$ from the near optimal result.
– $77.22 - 74.88 = 2.34$
 $(2.34/74.88) * 100 = 3.12\%$ from the near optimal result.
– $76.39 - 74.88 = 1.51$
 $(1.51/74.88) * 100 = 2.01\%$ from the near optimal result.

We therefore state that the proposed genetic algorithm is stable and good to be applied.

4.4. Comparison between both DPA, GA and port results

In Table 9, the real results in the port of Tripoli-Lebanon are compared with the dynamic programming algorithm (DPA) results and with the genetic algorithm (GA) results.

In the second example in Table 9, there are two quay cranes and four bays containing in total 62 containers. The time to unload a container by a quay crane is about 1.17 min. DPA makespan is 44.78 min, whereas in the port, the unloading time of 62 containers is about 95 min. Finally, the DPA enables us to save time by 52.86%, while GA result is 45.95 min and it enables to save time by 51.63%. (we add the percentage of the saved time only for DPA because it is the best method). The better one should have the higher percentage of saved time but although the percentage of the saved time by GA is smaller than DPA percentage, the GA results are good and can be applied. Note that this model accommodates the true conditions of the port of Tripoli-Lebanon.

5. Conclusion

This paper studies the QCSP in its multi-cranes variant, for which we proposed a mixed-integer linear programming (MILP) model, a dynamic programming algorithm (DPA) and a genetic algorithm (GA). Then, we make results comparison with the real instances from the port of Tripoli-Lebanon.

This paper shows the difference between two exact methods and a meta-heuristic method. We saw that for the large instances, the QCSP should be solved by a meta-heuristic method which is genetic algorithm, to have good solutions quickly.

To improve the quality of solutions in terms of execution time and convergence towards the optimal solution, other meta-heuristic methods should be tested in future work.

References

[1] N. Al-Dhaheri and A. Diabat, The quay crane scheduling problem. *J. Manuf. Syst.* **36** (2015) 87–94.

[2] B. Alnaqbi, H. Alrubaiai and S.A. Alawi, Combination of a dynamic-hybrid berth allocation problem with a quay crane scheduling problem. In: *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)* (2016).

[3] K.A. Awar, M. Alawani and S.A. Jaberi, A multi-vessel quay crane scheduling problem. In: *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)* (2016).

[4] L. Azza, M.E. Merouani and A. Medouri, Ant colony system for solving quay crane scheduling problem in container terminal. In: *2014 International Conference on Logistics Operations Management* (2014).

[5] R. Bellman, Dynamic Programming. Princeton University Press (1957).

[6] A.N. Boysen, B.D. Briskorn and C.F. Meisel, A generalized classification scheme for crane scheduling with interference. *Eur. J. Oper. Res.* **258** (2017) 343–357.

[7] S. Chung and K. Choy, A modified genetic algorithm for quay crane scheduling operations. *Expert Syst. App.* **39** (2012) 4213–4221.

[8] C. Daganzo, The quay crane scheduling problem. *Transp. Res.* **23** (1989) 159–175.

[9] A. Diabat and E. Theodorou, An integrated quay crane assignment and scheduling problem. *Comput. Ind. Eng.* **73** (2014) 115–123.

[10] D. Goldberg, Genetic algorithms in search. Addison-Wesley Professional, Reading, MA (1989).

[11] L. Haoyuan and S. Qi, Simulation-based optimization on quay crane scheduling of container terminals. In: *2017 29th Chinese Control And Decision Conference (CCDC)* (2017).

[12] J. Holland, Genetic algorithms, computer programs that evolve in ways that resemble natural selection can solve complex problems even their creators do not fully understand. http://www2.econ.iastate.edu/tesfatsi/holland.GAIntro.htm (1960).

[13] L. Kantorovich, Mathematical methods of organizing and planning production. *Manage. Sci.* **4** (1939) 366–422.

[14] K. Kim and Y. Park, A crane scheduling method for port container terminals. *Eur. J. Oper. Res.* **156** (2004) 752–768.

[15] C. Liang, Y. Huang and Y. Yang, A quay crane dynamic scheduling problem by hybrid evolutionary algorithm for berth allocation planning. *Comput. Ind. Eng.* **56** (2008) 1021–1028.

[16] A. Lim, B. Rodrigues, F. Xiao and Y. Zhu, Quay crane scheduling with spatial constraints. *Nav. Res. Logistics* **51** (2004) 386–406.

[17] M. Liu, S. Wang and C. Chu, A branch-and-price framework for the general double-cycling problem with internal-reshuffles. In: *2015 IEEE 12th International Conference on Networking, Sensing and Control* (2015).

[18] M. Liu, F. Zheng, Y. Xu and C. Chu, Approximation algorithm for uniform quay crane scheduling at container ports. *Discrete Math. Algorithms App.* **8** (2016) 1650018.

[19] M.K. Msakni, A. Diabat, G. Rabadi and M. Kotachi, An integrated quay crane assignment and scheduling problem using branch-and-price. In: *International Conference on Computational Science and Computational Intelligence* (2016).

[20] J.P.R.D. Oliveira, J.D. Barbosa and M. Lamprou, Multi-objective optimization of the quay crane assignment and scheduling problem: time and movement optimization. In: *2016 7th International Conference on Information, Intelligence, Systems and Applications (IISA)* (2016).

[21] R. Peterkofsky and C. Daganzo, A branch and bound solution method for the quay crane scheduling problem. *Transp. Res.* **24** (1990) 159–172.

[22] A. Salhi, G. Alsoufi and X. Yang, An evolutionary approach to a combined mixed integer programming model of seaside operations as arise in container ports. *Adv. Theor. Appl. Comb. Optim.* (2017).

[23] A. Skaf, S. Lamrous, Z. Hammoudan and M.-A. Manier, Exact method for single vessel and multiple quay cranes to solve scheduling problem at port of tripoli-lebanon. In: *2018 International Conference on Industrial Engineering and Engineering Management* (2018).

[24] A. Skaf, S. Lamrous, Z. Hammoudan and M.-A. Manier, Genetic algorithm to optimize unloading of large containers vessel in port of tripoli-lebanon. In: *2019 International Conference on Control, Decision and Information Technologies* (2019).

[25] A. Skaf, S. Lamrous, Z. Hammoudan and M.-A. Manier, Single quay crane and multiple yard trucks scheduling problem with integration of reach-stacker cranes at port of tripoli-lebanon. In: *2019 IEEE International Conference on Systems, Man, and Cybernetics* (2019).

[26] D. Steeken and R. Stahlbock, Container terminal operation and operations research – classification and literature review. *OR Spectr.* **26** (2004) 3–49.

[27] R. Tavakkoli-Moghaddam, A. Makui, S. Salahi, M. Bazzazi and F. Taheri, An efficient algorithm for solving a new mathematical model for a quay crane scheduling problem in container ports. *Comput. Ind. Eng.* **56** (2009) 241–248.

[28] S. Wang and W. Hu, Multi quay crane scheduling problem based on aco in container terminals. In: *2009 International Conference on Management and Service Science* (2009).

[29] S. Wang and W. Hu, An investigation into berth and quay crane scheduling for container terminals based on knowledge. In: *International Conference on Future Information Technology and Management Engineering* (2010).

[30] C. Xiazhong, Z. Ye and H. Hongtao, Optimization research of joint quay crane scheduling and block selection in container terminals. In: *2017 International Conference on Service Systems and Service Management* (2017).

[31] D. Yi, L. GuoLong and L. ChengJi, Model and heuristic algorithm for quay crane scheduling at container terminal. In: *9th International Conference on Fuzzy Systems and Knowledge Discovery* (2012).

[32] L. Zhen, S. Yu, S. Wang and Z. Sun, Scheduling quay cranes and yard trucks for unloading operations in container ports. *Ann. Oper. Res.* **273** (2016) 455.