RAIRO-Oper. Res. 49 (2015) 473–494
DOI: 10.1051/ro/2014039

RAIRO Operations Research
www.rairo-ro.org

THE DYNAMIC BERTH ALLOCATION PROBLEM: A LINEARIZED FORMULATION

Ahmed Simrin[1] and Ali Diabat[1]

Abstract. International shipping is a multi-billion dollar business which has undergone significant growth during the last decade. Considerable benefits could be gained by improving and optimizing container terminal operations. One specific challenge facing container terminals is the berth allocation problem, referred to as (BAP). In this paper, a new formulation is proposed for the dynamic berth allocation problem (DBAP). Initially the problem was formulated as a non-linear mixed integer program, followed by incorporating techniques to present an equivalent mixed integer program (MIP). A genetic algorithm (GA) heuristic was developed and applied to different instances of the problems, and through computational experiments the best, average, and worst case performances were analyzed to determine the efficiency of the algorithms.

Keywords. Container terminal, linear program, non-linear program, dynamic berth allocation.

Mathematics Subject Classification. 90Bxx.

1. Introduction

An important challenge to optimizing container terminal transshipment operations is the berth allocation problem (BAP), which is the process of assigning quay space to vessels in a container terminal for loading or unloading of containers [4]. Optimizing the assignment of quay space to vessels can increase the productivity of terminals without additional costs caused by other approaches such as constructing new berths.

Received January 2, 2014. Accepted July 3, 2014.

[1] Department of Engineering Systems & Management, Masdar Institute of Science & Technology, Abu Dhabi, United Arab Emirates. adiabat@masdar.ac.ae

Most of the BAP formulations in the literature aim to minimize the total time a container terminal spends in servicing a certain number of vessels. Data like arrival times and handling times of vessels are used in a model to produce an optimized schedule. Other data like draft, length, and width of vessels may also be used in the model.

The BAP can be classified into two types based on the arrival of vessels to the terminal: static arrival and dynamic arrival. Static arrival of vessels means that all vessels arrive before scheduling starts, and this guarantees that any schedule will be feasible. Conversely, dynamic arrival means that vessels may arrive before or after the schedule is constructed, and this puts more constraints on the schedule, as not any schedule will be feasible. However, dynamic arrival is more realistic than static.

Several styles of BAP models exist based on the layout of the berth [2]:

i. **Discrete layout:** the quay is divided into a number of berths, and only one vessel can be moored at each berth at a time.
ii. **Continuous layout:** a vessel can be moored at any location along the quay with the condition of not exceeding the boundaries of the quay.
iii. **Hybrid layout:** the quay in this layout is partitioned into a set of berths. However, unlike in the normal discrete layout, small vessels in this layout can share one berth simultaneously, and large vessels can take more than one berth too.

Models of the BAP also differ in the vessels' handling time parameter; some models consider handling time as a fixed parameter known for each vessel in advance, while other approaches deal with handling time as a berth-dependent parameter. There are different, yet fixed handling times for vessels at different berths. Handling time could also depend on the number of cranes assigned to vessels, work schedules of cranes, or a combination of all of the previous factors.

In this paper, a new realistic model for the DBAP that imitates real life is proposed. The DBAP is believed to be more realistic than the SBAP, because it, as mentioned previously, considers the dynamic arrival of vessels, instead of assuming that all vessels are already at the terminal waiting for service, which is the case in the SBAP. In the model, a discrete quay layout which imitates the ones usually used in newly built terminals such as Mina Zayed in Abu Dhabi, United Arab Emirates (UAE) and handling time is considered as constant berth-dependent parameters.

Words vessel and ship are used interchangeably in this paper.

2. Literature review

Some research has been done in the past fifteen years on the berth allocation problem. Some addressed the static ones, while others addressed the dynamic. In this section, we conduct a literature review of the research done in the field of berth allocation, and we explain what considerations the model took into account, besides the techniques that were used to solve it.

Imai *et al.* [11] proposed a static berth allocation model with respect to minimum waiting and handling time of the vessels in addition to the deviation between the arrival order of vessels and the service order. It was also assumed that the handling time of a vessel depends on the berth; in other words, a vessel has different handling times on different berths. The problem is then reduced to a classical assignment problem.

Another formulation of discrete SBAP for the public berth system was presented by Imai *et al.* [12], in which they did planning only with respect to waiting and handling time of vessels. They presented a Lagrangian relaxation-based heuristic to solve the problem. Lee *et al.* [16] presented a model for a discrete SBAP considering the minimization of waiting and handling time of vessels only. They assumed that the handling time of a vessel depends on its berthing position. Moreover, they assumed that the handling time depends on the Quay Crane (QC) operation schedule.

Imai *et al.* [14] used genetic algorithms (GA) to solve their model of SBAP that considers the minimization of the weighted number of vessel rejections as an objective function. A vessel is rejected if it cannot be serviced without exceeding the due date, represented by the maximum acceptable waiting time.

In addition to the static model that Imai *et al.* [12] proposed, they presented another model for the DBAP, and they developed a Lagrangian-based heuristic to solve the problem. They showed via a large number of experiments that their algorithm is adaptable to real world applications.

Hansen and Oguz [8] also provided compact reformulations of the dynamic problem as well as the static one. Similar to Imai's models, their objective in the two models was to minimize the total waiting and handling time of vessels. The same objective was also considered by Monaco and Sammarra [18]; they proposed another formulation of the discrete DBAP minimizing waiting and handling time, which was believed to be a stronger and a more compact formulation than other formulations in literature. They developed a Lagrangian relaxation to solve the problem. Lagrangian relaxation has also been implemented in other problems pertaining to quayside operations, such as the works of [1, 4, 23].

For multi-user container terminals (MUT's), Imai *et al.* [13] proposed a modified formulation of the DBAP taking into consideration different service priorities of the vessels. They developed a heuristic based on genetic algorithms (GA) for the solution of their non-linear formulation.

Theofanis *et al.* [19] have also formulated a linear mixed integer program for the dynamic and discrete berth allocation problem with the objective being minimizing the total weighted service time of vessels. They developed an optimization based genetic algorithm to solve the problem. Hansen and Oguz [9] also considered the earliness or tardiness of completion besides the minimization of waiting and handling time of vessels in their model as an objective function. Berthing of a vessel apart from the desired berth was also taken into consideration in their model. Moreover, they proposed a variable neighborhood search (VNS), and compared it with the results of multi-start (MS), genetic algorithms (GA) and mimetic algorithms (MA). The VNS was found to outperform the other algorithms on large-scale instances.

A two-objective BAP was considered by Imai *et al.* [15] - vessel service quality expressed by the vessels' delays, and berth utilization expressed by the total service time of all vessels in a certain period of time. Sub-gradient Lagrangian relaxation and genetic algorithm based heuristics were used to solve the problem, and the genetic algorithm outperformed the Sub-gradient in most of the cases.

Cordeau [3] also proposed two formulations for the discrete DBAP with the minimization of the weighted service time of the vessels considered as the objective function, and he developed a tabu search heuristic to solve them. Mauri [17] proposed an improved hybrid column generation approach for their berth allocation problem model, which was modeled as a vehicle routing problem. The approach combines the population training algorithm with linear programming, denoted PTA/LP.

In the model that Imai *et al.* [16] developed, they considered the dynamic case in which some vessels are assigned to external terminals when their waiting time exceeds the limit. Their objective was to minimize the service time of the vessels in external terminals. A genetic algorithm was used to solve the model, and it showed good performance in reducing the usage of external terminals.

Han *et al.* [7] proposed a new hybrid optimization strategy, which they called GASA. They introduced a non-linear model that aims to reduce the turnaround time as well as operation costs of vessels. They applied both GASA and GA on the model and they found that GASA accelerated the evolution process, and avoided getting stuck in a local optimal solution early.

In order to simulate the real decision-making processes, Zhou *et al.* [24] proposed a new dynamic and stochastic BAP model, where quay crane handling times are stochastic. They aimed at decreasing the vessel waiting time. Therefore, the model ignored the "first come, first served" rule. A heuristic based on GA was developed to solve the problem and they got satisfactory and efficient results.

Imai *et al.* [10] considered berth allocation and crane assignment at multi-user container terminals simultaneously in their model. They solved their models by a genetic algorithm based-heuristic and results showed that it is applicable to solve the problem. Liang *et al.* [20] presented a simultaneous and integrated formulation of berth allocation with quay crane assignment problems, and their objective was to minimize the sum of handling time, waiting time and also the tardiness of each vessel against the given due date. They used a genetic algorithm heuristic to solve the problems.

Finally, Giallombardo *et al.* [21] presented two formulations for the integrated problem of berth allocation and quay crane assignment. The first one was a mixed integer quadratic problem, while the second reduces to a mixed integer linear problem. The objective function was to maximize the total value of chosen quay crane profiles from one hand, and to minimize the housekeeping costs generated by transshipment between vessels.

In this paper, we propose a new model for the dynamic berth allocation problem. We assume the handling time of vessels to be dependent on the berth to which they are assigned, because different berths may vary in the number of cranes attached to them or the specifications of the cranes. Berths also demonstrate different productivity levels, especially when berths are located in different corners. In our model, we minimize the total waiting and handling time of vessels, which results in increasing productivity of the terminal and reducing operational costs.

3. Problem formulation

Our model has different assumptions that were set before starting with the process of formulation. We consider dynamic arrival of vessels which, as mentioned previously, allows vessels to arrive before or after the berthing plan is determined. The model, therefore, takes into consideration the constraint that a vessel cannot be serviced before its arrival. Moreover, all vessels in our problem must be completely serviced at one berth only without interruption from other vessels.

It was also assumed that a vessel has different, yet deterministic, handling times on different berths. Berths in the problem are assumed to be discrete, which results in the assumption that one berth can service only one vessel at a time.

The objective of our model is to minimize the total time of the whole service process for all vessels, which is the total duration of time vessels spend at a terminal between their arrival to the terminal and their departure. The duration of time to minimize has been divided into two types in our model: handling time and waiting time of vessels. Handling time of a vessel is the time a vessel spends at the terminal between the point servicing a vessel starts until it finishes, where waiting time for a vessel is the time a vessel spends at the container terminal waiting before service starts.

Having all this information, the model will accordingly make decisions regarding berth-to-vessel assignments as well as the order in which each vessel is serviced at the berth assigned to it. These factors must be determined so as to minimize the sum of handling and waiting time of all vessels while satisfying the constraints and the conditions in the problem.

3.1. Model constraints

In order to maintain the solution's feasibility, the model must satisfy the following constraints:

1. Each vessel must be serviced exactly once and with no interruption from other vessels.
2. A vessel must be serviced at one berth only.
3. A berth cannot handle more than one vessel at a time.
4. A vessel cannot be berthed before its arrival.

3.2. Model development

To formulate the problem, the following notation is used:

Sets:

B	=	set of berths, indexed by i
V	=	set of vessels, indexed by j
O	=	set of service orders, indexed by k
P_k	=	subset of O such that $P_k = \{p \mid p < k \in O\}$
W_i	=	subset of vessels with $A_j \geqslant S_i$

Parameters:

S_i	=	time when berth i becomes idle for allocation planning
A_j	=	arrival time of vessel j at the terminal
C_{ij}	=	handling time spent by vessel j on berth i
M	=	big number used for linearization

Decision variables:

$$x_{ijk} = \begin{cases} 1 \text{ if vessel } j \text{ is serviced at} \\ \quad \text{berth } i \text{ in the } k\text{th order} \\ 0 \text{ otherwise} \end{cases}$$

$t_i =$ integer equal to the number of vessels to service at berth i

$h_{ik} =$ number of vessels waiting to be serviced at berth i when the kth vessel is being serviced at that berth

$w_{ijk} =$ summation of the handling time of vessel j at berth i in the order k and the total waiting time resulted for all vessels waiting to be serviced at that berth

$y_{ijk} =$ time of berth i between the departure of the kth vessel and arrival of the kth vessel when vessel j is serviced as the kth vessel.

Our objective function is as follows:

$$\text{Min}\left(\sum_{i\in B}\sum_{j\in V}\sum_{k\in O}\{w_{ijk} + (S_i - A_j)\,x_{ijk}\} + \sum_{i\in B}\sum_{j\in Wi}\sum_{k\in O}\left(h_{ik}\cdot y_{ijk}\right)\right). \quad (3.1)$$

The first part of the objective function considers the handling time of vessels as well as the waiting time that results for vessels from servicing others. The variable w_{ijk} is equal to the summation of the handling time of vessel j serviced at berth i in the order k, and the waiting time resulted from that service on all vessels that will be moored at berth i after vessel j. The variable w_{ijk}, however, doesn't take into consideration the fact that some vessels may not be already in the terminal waiting when service of vessel j starts, and hence the waiting time that is added to those vessels is not valid. We will see in the constraints how w_{ijk} is forced to be 0 if x_{ijk} is equal to 0.

The $(S_i - A_j)\, x_{ijk}$ term solves the problem of the invalid waiting time; if a vessel arrives late at the terminal then the term $((S_i - A_j)\, x_{ijk})$ will be negative, and it will cancel any extra waiting time that is added by w_{ijk}.

The second part is the time during which the terminal remains idle, and that too adds waiting time to any vessel that is to be serviced after that idle time $\big(h_{ik} \cdot y_{ijk}\big)$, where y_{ijk} is the idle time of the terminal before vessel j is serviced at berth i in the order k and h_{ik} is the number of vessels that may be affected by that delay (idle time). Similar to w_{ijk}, some vessels may not be at the terminal at that time, and hence it's not affected by the time delay although it's counted in the term $\big(h_{ik} \cdot y_{ijk}\big)$. That is also treated by $(S_i - A_j)\, x_{ijk}$ which will be negative if vessel j arrives after planning starts. A detailed graphical example that shows how the objective function sums all handling, waiting, and idle times together is discussed later on.

Our model is subject to the following constraints:

$$\sum_{i\in B}\sum_{k\in O} x_{ijk} = 1 \qquad \forall\; j \in V, \tag{3.2}$$

$$\sum_{j\in V} x_{ijk} \leqslant 1 \qquad \forall\, i \in B, k \in O \tag{3.3}$$

$$t_i = \sum_{j\in V}\sum_{k\in O} x_{ijk} \qquad \forall\, i \in B \tag{3.4}$$

$$h_{ik} \geqslant t_i - k + 1 \qquad \forall\, i \in B,\;\; k \in O \tag{3.5}$$

$$h_{ik} \geqslant 0 \qquad \forall i \in B,\;\; \forall k \in O \tag{3.6}$$

$$w_{ijk} \geqslant \quad h_{ik} \cdot C_{ij} + (x_{ijk} - 1)M \qquad \forall i,\, \forall j,\, \forall k \tag{3.7}$$

$$w_{ijk} \leqslant \quad h_{ik} \cdot C_{ij} + (1 - x_{ijk}) \quad M \qquad \forall i,\, \forall j,\, \forall k \tag{3.8}$$

$$\sum_{j\in V}\sum_{m\in Pk} (C_{il} x_{ilm} + y_{ilm}) + y_{ijk} - (A_j - S_i) x_{ijk} \geqslant 0 \qquad \forall i \in B, \forall j \in Wi,\, \forall k \in O \tag{3.9}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i \in B, \forall j \in V,\, \forall k \in O \tag{3.10}$$

$$t_i,\, w_{ijk} \geqslant \quad 0 \qquad \forall i \in B, \forall j \in V,\, \forall k \in O \tag{3.11}$$

Where

$i\,(=1,\ldots,I)\in B$	Set of berths
$j\,(=1,\ldots,T)\in V$	Set of vessels
$k\,(=1,\ldots,T)\in O$	Set of service orders
S_i	Time when berth i becomes idle for berth allocation planning
A_j	Arrival time of vessel j at the terminal
C_{ij}	Handling time spent by vessel j at berth i
x_{ijk}	1 if vessel j is serviced as the kth vessel at berth i, and 0 otherwise
t_i	Total number of vessels to service at berth i
h_{ik}	The number of vessels that are waiting to be serviced at berth i, at order k
w_{ijk}	Total waiting time caused by servicing vessel j at berth i at the order k, added to it is the handling time of vessel j itself.
M	A big number used for linearization, and has to be at least equal to biggest possible value of w_{ijk}
y_{ijk}	Idle time of berth i between the departure of the $k-1$th vessel and the arrival of the kth vessel when vessel j is serviced as the kth vessel.
P_k	Subset of O such that $P_k=\{p\mid p<k\in O\}$
W_i	Subset of vessels with $A_j\geqslant S_i$.

Constraint (3.2) ensures that every vessel in the problem will receive service exactly once and at one berth only, while constraint (3.3) guarantees that no two vessels are serviced at the same berth simultaneously.

In our formulation, we use a variable that represents the number of vessels berth i is assigned to, defined as t_i, and constraint (3.4) is responsible for that.

Constraints (3.5) and (3.6) are used to find the value of $h_{ik}$for every berth i in every order k, which represents the number of vessels that are waiting to be serviced at berth i, when schedule reaches the order k. Although it's a '$\geqslant$' sign, the solution will always consider the value (t_i-k+1) and not any greater value because it's a minimization problem and because h_{ik} has a positive sign in the objective function. A '$\geqslant$' sign was used, not '$=$' because after all vessels are serviced *i.e.* large values of k (t_i-k+1) becomes negative, and that should not happen, so we use '$\geqslant$' with another constraint that forces variable h_{ik} not to go below 0 (constraint (3.5)).

The variable w_{ijk} represents the handling time of vessel j on berth i in the order k added to it the waiting time of all vessels waiting to be serviced after it. The value of w_{ijk} must be equal to the handling time of that vessel multiplied by the number of vessels that are not serviced at that berth yet, which is $(h_{ik}\,.\,C_{ij})$. That should be the case only if x_{ijk} is equal to 1 and it must be 0 otherwise.

Constraints (3.7), (3.8) and (3.11) guarantee that if x_{ijk} =1, constraints (3.7) and (3.8) both force w_{ijk} to be exactly equal to $(h_{ik}\,.\,C_{ij})$, while if $x_{ijk} = 0$, w_{ijk} takes the lowest value possible since our problem is a minimization problem, and in this case constraints (3.7) and (3.11) both set a lower bound for the value, but constraint (3.11) dominates because its lower bound is higher than that of constraint (3.7); therefore w_{ijk} takes the value 0.

Finally in constraint (3.9), we make sure that a vessel is serviced after its arrival, not before. The constraint can be written as $\sum\limits_{i\in V}\sum\limits_{m\in Pk}(C_{il}x_{ilm}+y_{ilm})+y_{ijk} \geqslant (A_j - S_i)\,x_{ijk}$. The left-hand side of the constraint represents the time, with respect to S_i, at which vessel j is serviced at berth i in the order k, while the right-hand side represents the arrival time of vessel j with respect to the planning time S_i. Therefore by having the time of service greater than or equal to the arrival time, we make sure that a vessel is not serviced before its arrival.

4. Genetic algorithms

Different approaches are followed to solve optimization problems; some "easy" problems can be solved using exact methods, while in "hard" problems some algorithms may be used to give optimal or nearly optimal solutions to the problem. One of these approaches is the genetic algorithm (GA), which mimics the natural evolution principles; they process a group of solutions, called population, into a fitter population. The process is repeated until termination criteria are met [5, 6].

4.1. Introduction to genetic algorithms

The basic concept of GAs is designed to mimic the processes of evolution in natural systems, specifically Charles Darwin's principles of survival of the fittest individuals.

A genetic algorithm first generates a population of solutions, which is usually feasible and generated by the programmer. Then evaluation of all solutions is done by calculating the so-called fitness value to every solution in the generation, which reflects the level of goodness of the solution. A set of "couples" is then selected from the current generation to mate and produce a subsequent "fitter" generation. Fitter solutions from the older generation will have better chances to mate and to produce offspring. Production of offspring is done through a process called crossover, in which a child will have one portion of its characteristics from one parent and the other portion from the second parent [22].

A random mutation, usually with low probability, may be applied to some parts of the solutions in the new generation to help prevent getting stuck in a local optimal point, and it also guarantees that the probability of any solution is never zero. The new generation's fitness value is calculated again and the same processes are repeated until one or all of the termination criteria are met. The solution with the best objective value throughout all the generations is eventually selected as

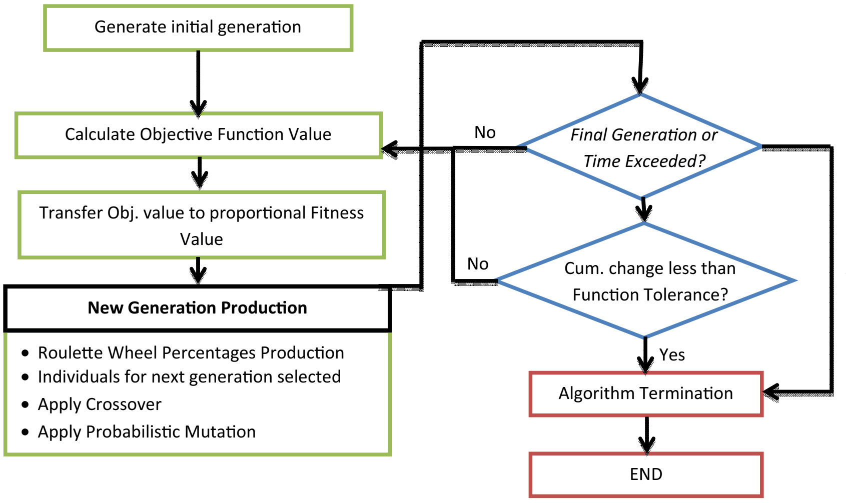

Generate initial generation
Calculate Objective Function Value
Transfer Obj. value to proportional Fitness Value
New Generation Production
Roulette Wheel Percentages Production
Individuals for next generation selected
Apply Crossover
Apply Probabilistic Mutation
Final Generation or Time Exceeded?
No
Cum. change less than Function Tolerance?
No
Yes
Yes
Algorithm Termination
END

Figure 1. Genetic Algorithms process.

the final output of the algorithm. In this study, a genetic algorithm heuristic for the DBAP was built from scratch on MATLAB, and the heuristic was tested on several instances of the problems.

The flowchart in Figure 1 explains the process a genetic algorithm follows to solve a model.

4.2. The genetic algorithm used in the dynamic berth allocation problem

4.2.1. Chromosome representation

A chromosome (sometimes called genome) that includes the decision variables of the problem is used to represent a solution in terms of an individual. Chromosome representations are normally fixed-length bit strings. Every position in the chromosome, so-called gene, represents a part of the solution. The value of a gene represents the decision made for one part of the solution.

In our problem, for each solution it is important to decide the values of the decision variables x_{ijk}. If that is known, we can calculate the total handling and waiting time of all vessels of that solution. In order to calculate that by coding, we need to take into consideration the time berths become ready for service, the arrival times of vessels, as well as the handling times of vessels on different berths.

Among several possible chromosome representations, we used the shortest representation we could determine, which consisted of two rows of integer values. Each chromosome is implemented as two rows, and both of them are of length equal to the number of vessels in the problem. Each value in the first row represents the berth assigned to the corresponding vessel, while the second row is a number that is used to decide the order in which the corresponding vessel will be serviced according to that solution. Therefore, normally the range of integer values for the first row will be {1, 2,. . . , number of berths}, while in the second row, it takes a value in the range {1, 2,. . . , number of vessels}. In our implementation, the values in the second row do not represent the actual order of service, but they are used for comparison to decide the order of service at berths.

For example if we have a problem of five berths and ten vessels, the two rows in the chromosome will have a length of 10 genes. Below is an example of a chromosome to depict the problem:

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
4	2	4	5	1	3	3	1	5	2
2	1	3	8	1	5	10	2	7	6

Looking at the first row, we know that the chromosome (solution) above suggests that vessel 1 is serviced at berth 4, vessel 2 is serviced at berth 2, vessel 3 at berth 4, and so on. The second row decides the order of service, we see that vessels 1 and 3, which are both serviced at berth 4, have the values 2 and 3 in the second row respectively, and since 2 is less than 3 we conclude that vessel 1 will be serviced before vessel 3. The same applies for vessels 2 and 10, they both are serviced at berth 2, but vessel 2 is serviced before vessel 10, because it has the value 1 in the second row while vessel 10 has the value 6. In the case when two vessels are serviced at the same berth and having the same value in the 2$^{\text{nd}}$ row, a random selection of which vessel will be serviced before the other is done.

4.2.2. Population initiation and constraints satisfaction

Our chromosome representation of the problem guarantees that any combination of values of the variables is always feasible, because any vessel can physically go to any berth and be serviced in any order between first and last. Besides that, by feasible we mean that all of the constraints are satisfied. The four constraints that we want to satisfy are listed in Section 3 under model constraints.

The first constraint forces each vessel to be serviced exactly once with no interruption from other vessels., and that is satisfied by coding; for a vessel to be serviced we make sure by coding that the berth has finished servicing all the previous vessels assigned to it, and that the vessel itself will be arriving at the port at or before that time. If any of those two conditions are not satisfied for a period of time, we add waiting time to any vessels that are already in port waiting for service at that berth during that time until both conditions are satisfied, we then proceed with the next vessel on schedule.

The second constraint allows a vessel to be serviced on one berth only. That is satisfied by assigning one berth to each of the vessels in the problem, as mentioned previously.

The third constraint, which prevents a berth from servicing more than a vessel at a time, is also satisfied by sorting the vessels assigned to each of the berths in the problem on a shortest first basis. Doing this, we make sure that the vessels are serviced one after another on each berth without any conflict.

Finally, the fourth constraint states that a vessel cannot be berthed before its arrival. That is satisfied by adding idle time in case the berth is free and the next vessel to be serviced based on the chromosome has not arrived yet.

4.2.3. Fitness and relative fitness evaluation

There exist three important terms that are used in the selection step: objective value, fitness value, and relative fitness value. The objective value of a chromosome is simply the value of the objective function for that solution. A fitness value of a chromosome is a transformed value of the objective function, while a relative fitness value is the value assigned to a chromosome that shows how good it is compared to other chromosomes. In our implementation a relative fitness value lies in the range 0-1 and it is used in the roulette selection. After we find the objective value of the proposed solutions, we find what's called the fitness value of that chromosome, which, as mentioned above, is a transformed value of the objective function value.

In this work, we implement three different fitness functions and test them. Fitness functions differ in the weight they give to solutions. The three different fitness functions that we studied are as follows:

1. $\mathit{Fitness}\ (x) = (\text{worstObj} - \text{ObjVal}\,(\text{x})\,)^2 + m$
2. $\mathit{Fitness}\ (x) = \frac{\text{bestObj}}{\text{ObjVal(x)}} - \frac{\text{bestObj}}{\text{worstObj}} + m$
3. $\mathit{Fitness}\ (x) = \left(\frac{\text{bestObj}}{\text{ObjVal(x)}} - \frac{\text{bestObj}}{\text{worstObj}}\right)^2 + m$

where ObjVal(x) is the objective value of the individual x. The value bestObj represents the best objective value among the objective values of all chromosomes in the current generation. Similarly, worstObj represents the worst objective value among all chromosomes, and finally m is a very small number used to prevent having all fitness values equal to 0, when they all have the same optimal value.

In the second and third fitness functions, we subtract $\frac{\text{bestObj}}{\text{worstObj}}$ from the first term, so that we ensure the chromosome with the worst objective value will have a very low chance (equal to the negligible value m) of being selected for the next generation. In what follows (and for simplicity), we will use the following representative names: *subtraction*^{2}, *division*, and *division*^{2} for the three fitness functions, respectively, described above. In previous work, it was found that a subtraction fitness function raised to the power of 2 gives better results than a subtraction fitness function without a power of 2. Therefore, it was not included in our experiments in this work.

In Table 1, we show an example of 5 chromosomes with 5 different objective values. We apply the three fitness functions on the chromosomes and then find the corresponding relative fitness values for better understanding of the selection process.

From the table above, we can see that fitness values can take any real positive value depending on the problem. However, relative fitness values only can be

Table 1. Relative Fitness Values of different Fitness Functions.

	Subtraction^2 function		Division function		Division^2 function	
Obj. value	Fitness value	Relative Fit. value	Fitness value	Relative Fit. value	Fitness value	Relative Fit. value
10	0	0/30 = 0%	0	0/0.8 = 0%	0.00	0.0/0.26 = 0.0%
9	1	1/30 = 3%	0.07	0.07/0.8 = 8%	0.01	0.005/0.26 = 2%
8	4	4/30 = 13%	0.15	0.15/0.8 = 17%	0.02	0.02/0.26 = 8.5%
7	9	9/30 = 30%	0.26	0.26/0.8 = 29%	0.07	0.07/0.26 = 26%
6	16	16/30 = 54%	0.4	0.4/0.8 = 46%	0.16	0.16/0.26 = 61.5%
–	30	100%	0.88	100%	0.26	100%

represented in a percentage form and the summation of all relative values in any population normally equals 100%.

We conclude from the table above that raising a fitness function to the power of 2, as in the first and third functions, makes it more biased to the fitter chromosomes, and more aggressive with the weak ones. For example, chromosome 5, which has the best objective value, has a relative fitness value of only 46%. Using the second function, it has a chance of 46% to be selected for the next generation, while it has relative fitness values equal to 54% and 61.5% using the first and third fitness functions respectively. On the other hand chromosome 2, which is the second weakest chromosome in the population, has a relative fitness value of 8% using the second fitness function, and the values 3% and 2% using the first and third fitness functions respectively. Another point that can be concluded from the table above is that the division^2 fitness function gives the fittest chromosome a higher relative fitness value than subtraction^2 does.

We test the three functions on different instances of the problem, and the results are shown thereafter.

4.2.4. Roulette selection

After relative fitness values are calculated, next we pursue a roulette wheel selection. As the name says, a number of chromosomes are selected randomly as needed for the next generation, one after another; each chromosome has a chance of being selected equal to its relative fitness value. Therefore, the chromosomes with higher fitness values have a better chance to appear more than once in the pairs for crossover. We perform this random selection a number of times equal to the required number of chromosomes for crossover.

4.2.5. Crossover (recombination)

When individuals are selected using roulette wheel selection, couples are selected randomly for recombination, commonly referred to as crossover. Crossover can be done in different ways; in our experiments, we tested three different functions: single-point, double-point, and uniform crossover. Below, we explain how each of these functions works. It is important to remember that in our chromosome representation, a chromosome consists of two rows, and in crossover whatever happens to the upper gene happens to the lower.

- **Single-point Crossover:**

In single crossover, a point, that splits the two chromosomes in each couple into two segments, is selected randomly; the two lines of each chromosome are split on the same point. Recombination is then applied by having the first segment from the first chromosome combined with the second segment of the second chromosome, and similarly the second segment from the first chromosome is combined with the first segment from the second chromosome. That results in two new children born from the two parents.

The figure below shows how single crossover works with the chromosomes in our implementation. Single-point crossover is applied for two parents on a point between the fourth and fifth cells of each chromosome. Two children were produced as shown below.

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
Parent #1	4	2	4	5	1	3	3	1	5	2
	2	1	3	8	1	5	10	2	7	6
Parent #2	4	2	5	1	3	5	4	2	2	1
	3	4	2	7	8	5	9	10	2	6

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
Child #1	4	2	4	5	3	5	4	2	2	1
	2	1	3	8	8	5	9	10	2	6
Child #2	4	2	5	1	1	3	3	1	5	2
	3	4	2	7	1	5	10	2	7	6

In our solution, for each couple in each generation the crossover point is selected randomly, and it takes a value between 1 and length of the chromosome-1.

- **Two-point Crossover:**

As the name says, in two-point crossover, two points on the chromosome are chosen to apply the crossover. Those two points split both chromosomes in each couple into three segments; the two lines in each chromosome are split at the same point. In our implementation of the problem, recombination is done by exchanging the middle segments in the two chromosomes, and that produces two new children from the two parents.

An example of double-point crossover is shown below; crossover is applied on two parents at two points, and two children with characteristics from both parents are produced.

In our implementation, for each couple in each generation, the two crossover points are selected randomly, where the first point (P1) takes a value between 1

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
Parent #1	4	2	4	5	1	3	3	1	5	2
	2	1	3	8	1	5	10	2	7	6
Parent #2	4	2	5	1	3	5	4	2	2	1
	3	4	2	7	8	5	9	10	2	6

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
Child #1	4	2	4	5	3	5	4	2	5	2
	2	1	3	8	8	5	9	10	7	6
Child #2	4	2	5	1	1	3	3	1	2	1
	3	4	2	7	1	5	10	2	2	6

and length of the chromosome-2, and the second point (P2) takes a value between the value of (P1)+1 and length of the chromosome-1.

- **Uniform Crossover:**

Unlike one-point and two-point crossover, uniform crossover deals with individual genes instead of chromosome segments. For every couple selected for crossover, a binary vector of length equal to the length of chromosomes is generated and then genes in location i along the parent chromosomes are switched if the value of cell i along the binary vector is 1, and no switch happens otherwise.

Below is an example of a uniform crossover:

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
Parent #1	4	2	4	5	1	3	3	1	5	2
	2	1	3	8	1	5	10	2	7	6
Parent #2	4	2	5	1	3	5	4	2	2	1
	3	4	2	7	8	5	9	10	2	6
		1	1	0	1	0	0	1	1	0

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
Child #1	4	2	5	5	3	3	3	2	2	2
	2	4	2	8	8	5	10	10	2	6
Child #2	4	2	4	1	1	5	4	1	5	1
	3	1	3	7	1	5	9	2	7	6

4.2.6. Mutation

Before we reinsert the newly produced population as the next generation, we apply mutation to it. Mutation, in binary representation, means flipping a bit from 0 to 1 or the opposite, and it is done with a low probability. In our algorithm, we have an integer representation of chromosomes; we implemented mutation by doing a shuffle for a gene value to produce a new value in the allowed range of the specific problem.

In our experiments, we tested more than a mutation rate in order to find the optimal mutation probability and then adopted it for a final algorithm implementation. In each generation, if a chromosome is selected for mutation, only one bit, along with the bit beneath it, is flipped, and that bit is selected randomly.

Mutation helps guarantee that the probability of searching for a specific string will never be zero, and it can also help recover good chromosomes that may be lost during selection or crossover.

4.2.7. Reinsertion

After doing selection, crossover, and mutation, a new population is now ready to be inserted to the next iteration (generation). The average fitness value of individuals in each generation is expected to go higher as we produce more generations. That is expected because of the fact that least-fit individuals are likely to disappear in newer generations while most-fit ones are likely to survive and to reproduce more individuals of similar kind in the next generations.

The same steps are applied all over again to the newly inserted generation until the algorithm terminates.

4.2.8. Algorithm termination

We consider three criteria for algorithm termination. The first one is setting a maximum number of generations; once that number is reached the algorithm terminates. The second criterion is function tolerance, which forces the algorithm to terminate if the cumulative change in fitness function values found over a specific number of generations is less than a specific number. Finally, we specify a maximum duration of time for the algorithm to run, after which the algorithm is terminated. If one of the previous three criteria is satisfied, our GA terminates, and the individual that had the best objective value in all the previous generations is taken as the final solution of the problem.

5. Experimental results

We tested our genetic algorithm on 6 different instances of the DBAP problem considering different numbers of vessels and berths every time (Tab. 2).

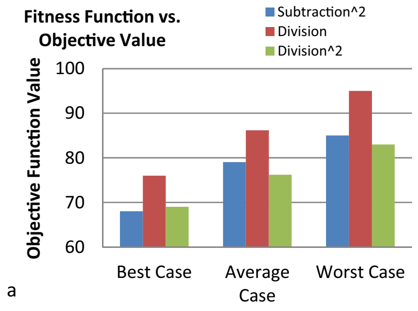
We used the CPLEX solver on a powerful Dell XPS 8300 with 16GB of installed RAM and a 3.4 GHz i7 CPU to solve the instances, but we found that it was able to solve only the two smallest problems, while it failed to find any good solution for the last four instances. Table 2 shows the solvability of the instances using CPLEX and genetic algorithms.

In our experiments we compared the results on running the algorithm on different values for the following parameters: relative fitness function, crossover function, mutation rate, population size, and number of iterations.

In our analysis we considered the two largest instances in the table above: 20 Berths $\times$ 60 Vessels and 30 Berths $\times$ 80 Vessels, as the effect of changing parameters can be seen more clearly in large problems than in small ones. The default

TABLE 2. Solvability of the DBAP using the CPLEX solver and our developed GA.

	Instance	Number of Integer Decision Variables	Solvability using CPLEX solver	Solvability using GA on MATLAB
1	3 Berths $\times$ 5 Vessels	243	Yes	Yes
2	5 Berths $\times$ 10 Vessels	1,555	Yes	Yes
3	8 Berths $\times$ 15 Vessels	5,528	No	Yes
4	10 Berths $\times$ 30 Vessels	27,310	No	Yes
5	20 Berths $\times$ 60 Vessels	217,220	No	Yes
6	30 Berths $\times$ 80 Vessels	578,430	No	Yes



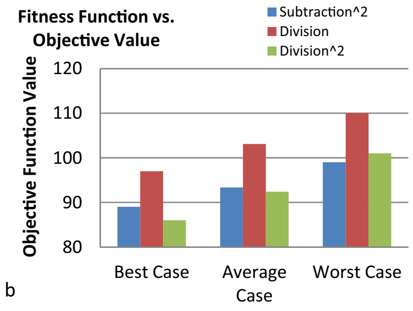


FIGURE 2. Best, Average, and Worst cases for the fitness functions on 2 instances of the DBAP (a) 20 Berths $\times$ 60 Vessels; (b) 30 Berths $\times$ 80 Vessels.

values that we used for genetic algorithm parameters are: 3000 iterations, population size of 50, mutation rate of 50%, uniform crossover and division^2 fitness function.

5.1. EFFECT OF FITNESS FUNCTIONS ON THE ACCURACY

As mentioned previously, we proposed three different fitness functions. The three fitness functions are:

a. $Fitness\ (x) = (\text{worstObj} - \text{ObjVal}(x))^2 + m$

b. $Fitness\ (x) = \frac{\text{bestObj}}{\text{ObjVal}(x)} - \frac{\text{bestObj}}{\text{worstObj}} + m$

c. $Fitness\ (x) = \left(\frac{\text{bestObj}}{\text{ObjVal}(x)} - \frac{\text{bestObj}}{\text{worstObj}}\right)^2 + m.$

We applied each of the three fitness functions 50 times on each of the two DBAP instances. We then took the best, mean, and worst objective values of each function. Figure 2 above shows the results.

As we can see in the graphs, the first and third fitness functions, which are raised to the power of 2, gave better results: *i.e.*, smaller objective values than the second. The reason is believed to be that in such problems, the range of values of

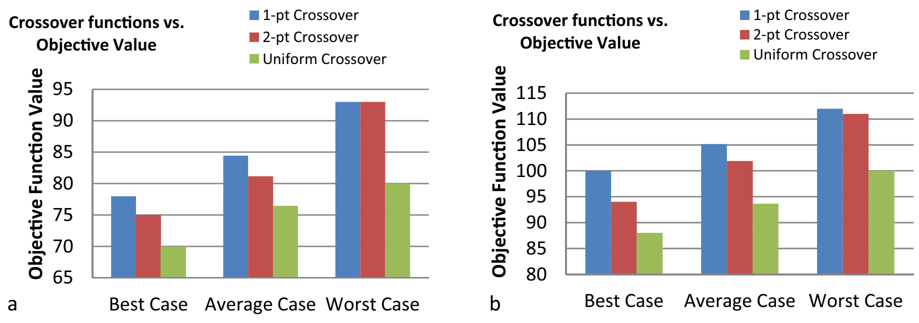


FIGURE 3. Best, Average, and Worst cases for the crossover functions on two instances of the DBAP (a) 20 Berths $\times$ 60 Vessels; (b) 30 Berths $\times$ 80 Vessels.

the objective function is small, and hence subtraction or division to the first power does not give the fit individuals significant weights (high relative fitness values) compared to others, but raising it to a power higher than 1 does.

We can also see that the performance of Division^2 and Subtraction^2 is close to each other; the difference between the two is not significant in best, average, and worst cases.

5.2. Effect of crossover function on the accuracy

As we did in the fitness functions tests, we applied the three proposed forms of crossover functions 50 times on both instances, and the best, average, and worst results are plotted in Figure 3 below.

It is clear that uniform crossover outperforms both single-point and double-point crossover in all cases (best, average, and worst). We also see that 2-point crossover gives better results than 1-point in best and average cases. We can therefore verify that the smaller we go in partitioning chromosomes for crossover, the children produced are more fit.

5.3. Effect of mutation rate on the accuracy

Three different mutation rates were tested on the two instances. The mutation rates were 25%, 50%, and 75%, respectively. The following graphs (Fig. 4) resulted.

From the graphs below, we conclude that 50% mutation, which means that one out of every two chromosomes in average will have one of its genes shuffled, gives the best results. A mutation rate of 25% seems to be too low to give best results while 75% seems to be too high. That can be seen in the valley-shaped graph for the three mutation rates.

5.4. Effect of population size on the accuracy

We tested the effect of population size on the accuracy of the algorithm. We considered the population sizes: 30, 50, and 70, and recorded the mean value of the objective function of 50 runs of the algorithm for three instances of the

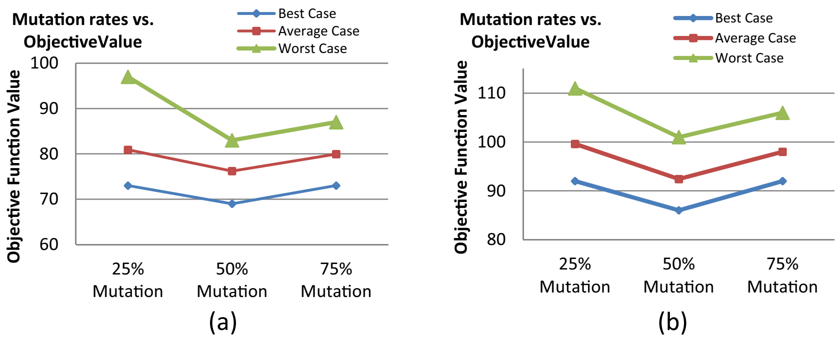


FIGURE 4. Best, Average and Worst cases for ratios of mutation on three instances of the DBAP (a) 20 Berths × 60 Vessels; (b) 30 Berths × 80 Vessels.

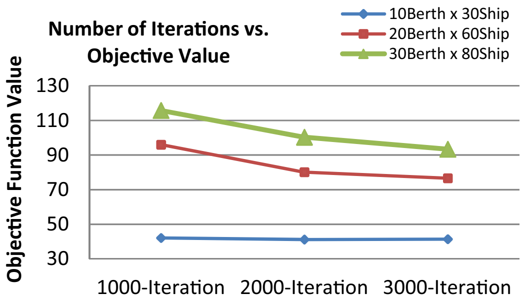


FIGURE 5. Effect of population size on the accuracy of the algorithm in 3 instances.

DBAP: 10 Berths × 30 Vessels, 20 Berths × 60 Vessels, and 30 Berths × 80 Vessels. The results are plotted in Figure 5.

Usually increasing the population size is better for the accuracy of the problem as we will consider more solutions of the problem during the execution. However, there are points for any problem after which the best solution doesn't get any better even if the population size is increased. For example in a 10 Berths × 30 Vessels problem – the smallest among the problems tested here – a population size of 30 gives almost the same objective function value as do the population sizes of 50 and 70. In problems 20 Berths × 60 Vessels and 30 Berths × 80 Vessels, population sizes 50 and 70 proved to have better results than smaller population sizes in the problems respectively.

Moreover, we see that the amount of improvement in the objective function decreases as we increase the population size. For example, in the 30 Berths × 80 Vessels problem, the objective function improved by about 15 units of handling and waiting time when the population size was increased from 30 to 50, while the improvement was only 5 units of time when the size of population increased from 50 to 70.

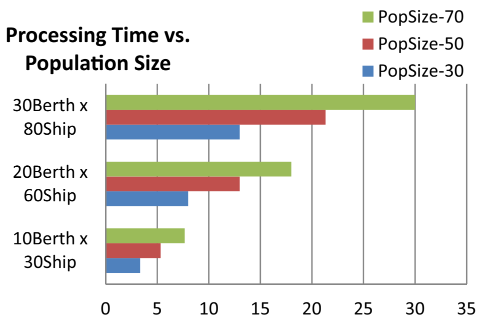


FIGURE 6. Effect of number of iterations on the accuracy of the algorithm in three instances.

5.5. Effect of number of iterations on the accuracy

By default, 3000 iterations were used in our experiments, but we did some experiments to test the effect of the number of iterations on the accuracy of the algorithm. We tried 50 runs of 1000, 2000 and 3000 iterations on each of our largest problems (10 Berths × 30 Vessels, 20 Berths × 60 Vessels, and 30 Berths × 80 Vessels). The mean value of the 50 runs was recorded for each number of iterations on each problem and the results were plotted in Figure 6 below.

Increasing the number of iterations may sometimes result in better solutions of the problem as it is the case in problems 20 Berths × 60 Vessels and 30 Berths × 80 Vessels. The effect of number of iterations on the goodness of solutions depends mainly on the number of variables in the problem. For example, no improvement in the objective function occurred in the 10 Berths × 30 Vessels problem when the number of iterations went above 1000 iterations. In other words, one thousand iterations were enough to find a very good solution, and no better solutions are usually found after that number.

However, in the two larger problems, results improved when the number of iterations increased from 1000 to 2000 and then to 3000. It is also possible that completing more iterations for those two problems can lead to better solutions. It is also important to notice that results can never get worse as we increase the number of iterations; if it doesn't improve, it will not get worse.

5.6. Population size vs. processing time

We also studied the effect of increasing population size on the time of execution of the algorithm. We see in Figure 7 below that increasing the population size, as expected, increases the execution time of the algorithm. We can also see that the execution time increases linearly with increasing the population size, which can be considered a good advantage for our algorithm. Therefore, although increasing population size results in a longer execution time, it is sometimes worth doing so, as that may lead to better solutions.

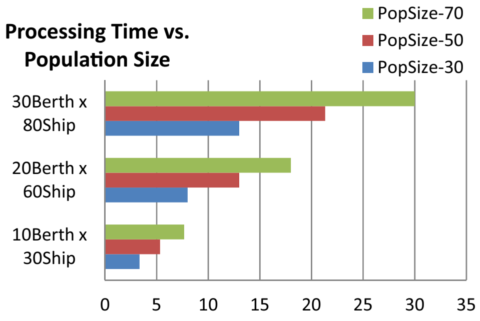


FIGURE 7. Effect of population size on processing time of the algorithm in three instances.

6. Conclusion

In this paper, we developed a new model for the dynamic berth allocation problem. Our objective was to minimize the total handling and waiting time of vessels at the container terminal. The problem, as the name says, considers dynamic arrival of vessels to the port and also assumes different handling times of vessels on different berths. The DBAP is a nondeterministic polynomial time (NP) problem; therefore we developed a genetic algorithm-based heuristic to solve it.

In our experiments, we tested different values for the different parameters of the algorithm; we tested three different fitness functions, and found out that raising a fitness function to the power of 2 in this specific problem can improve the results. We also tested three types of crossover functions: single point, two-point, and uniform crossover. Uniform crossover was noticeably more powerful than the other two. In addition, we tried different mutation rates and determined that mutation rate must not be too low or too high, as they both hurt the quality of the solution.

Our genetic algorithms showed very good results; for the instances that CPLEX was able to solve, our genetic algorithms could reach 0% in most of the cases as a best case, while in the average case the gap was around 1%. Moreover, the developed algorithms are very fast: for example it takes less than a minute to reach the optimal solution for a 576 000-variable problem.

Finally, we tried different population sizes and different numbers for iterations and concluded that depending on the size of the problem, a certain population size and a certain number of algorithm iterations are sufficient to reach the best answer. Figures and tables were provided listing all the results that we obtained, along with the discussion in the previous section.

References

[1] N. Al-Dhaheri and A. Diabat, The quay crane scheduling problem. *J. Manuf. Syst.*, in press, 2015.

[2] C. Bierwirth and F. Meisel, A survey of berth allocation and quay crane scheduling problems in container terminals. *Eur. J. Oper. Res.* **202** (2010) 615–627.

[3] J.-F. Cordeau, G. Laporte, P. Legato and L. Moccia, Models and tabu search heuristics for the berth-allocation problem. *Transp. Sci.* **39** (2005) 526–538.

[4] A. Diabat, Y.-M. Fu, A Lagrangian relaxation approach for solving the integrated quay crane assignment and scheduling problem. *Appl. Math. Modell.* **32** (2015) 1194–1201.

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

[6] Y.-M. Fu, A. Diabat, I.-T. Tsai, A multi-vessel quay crane assignment and scheduling problem: Formulation and heuristic solution approach. *Exp. Syst. Appl.* **41** (2014) 6959–6965.

[7] M. Han, P. Li and J. Sun. The algorithm for berth scheduling problem by the hybrid optimization strategy GASA, in *Proc. of the ninth international conference on Control*, automation, robotics and vision (ICARCV'06). IEEE Comput. Soc., Washington DC (2006) 1–4.

[8] P. Hansen and C. Og_uz, A note on formulations of static and dynamic berth allocation problems. Les Cahiers du GERAD **30** (2003) 1–17.

[9] P. Hansen and C. Og_uz, N. Mladenovic, Variable neighborhood search for minimum cost berth allocation. *Eur. J. Operat. Res.* **191** (2008) 636–649.

[10] A. Imai, H.C. Chen, E. Nishimura and S. Papadimitriou, The simultaneous berth and quay crane allocation problem. *Transp. Res. Part E* **44** (2008) 900–920.

[11] A. Imai, K. Nagaiwa and C.W. Tat, Efficient planning of berth allocation for container terminals in Asia. *J. Adv. Transp.* **31** (1997) 75–94.

[12] A. Imai, E. Nishimura and S. Papadimitriou, The dynamic berth allocation problem for a container port. *Transp. Res. Part B* **35** (2001) 401–417.

[13] A. Imai, E. Nishimura and S. Papadimitriou, Berth allocation with service priority. *Transp. Res. Part B* **37** (2003) 437–457.

[14] A. Imai, E. Nishimura and S. Papadimitriou, Berthing ships at a multi-user container terminal with a limited quay capacity. *Transp. Res. Part E* **44** (2008) 136–151.

[15] A. Imai, J.-T. Zhang, E. Nishimura and S. Papadimitriou, The berth allocation problem with service time and delay time objectives. *Marit. Econ. Logist.* **9** (2007b) 269–290.

[16] D.-H. Lee, L. Song and H. Wang, Bilevel programming model and solutions of berth allocation and quay crane scheduling, in *Proc. of 85th annual meeting of Transportation research board* (CD-ROM), Washington DC (2006).

[17] G.R. Mauri, A.C.M. Oliveira and L.A.N. Lorena, A hybrid column generation approach for the berth allocation problem, in Eighth European conference on Evolutionary computation in combinatorial optimisation EvoCOP, edited by J. van Hemert, C. Cotta. *Lect. Note Comput. Sci.*, **4972**, Springer, Berlin (2008) 110–122.

[18] M.F. Monaco and M. Sammarra, The berth allocation problem: a strong formulation solved by a Lagrangean approach. *Transp. Sci.* **41** (2007) 265–280.

[19] S. Theofanis, M. Boile and M. Golias, An optimization based genetic algorithm heuristic for the berth allocation problem, in Proc. of IEEE congress on Evolutionary computation (CEC 2007). *IEEE Computer Society*, Washington DC (2007) 4439–4445.

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

[21] G. Giallombardo, L. Moccia, M. Salani and I. Vacca, The tactical berth allocation problem with quay crane assignment and transshipment-related quadratic yard costs, in *Proc. of the European Transport Conference* (ETC) (2008) 1–27.

[22] A. Popov, Genetic algorithms for optimization, User Manual (2005).

[23] E. Theodorou, A. Diabat, A joint quay crane assignment and scheduling problem: formulation, solution algorithm and computational results. *Optim. Lett.*, in press, 2014.

[24] P. Zhou, H. Kang and L. Lin, A dynamic berth allocation model based on stochastic consideration, in *Proc. of the Sixth World Congress on Intelligent Control and Automation WCICA*, IEEE Comput. Soc. Vol. 2, Washington DC (2006) 7297–7301.