A Genetic Algorithm for Solving a Container Storage Problem Using a Residence Time Strategy

Cristina SERBAN, Doina CARP

Ovidius University of Constanta, Mamaia Bld. 124, Constanta, 900527, Romania.
cgherghina@gmail.com

Abstract: At each port of destination, some containers are unloaded from a vessel and stored in the terminal to be delivered to their customers. One of the strategies used to arrange the containers in a terminal is *residence time strategy*: based on their delivery deadlines, each incoming container being assigned to a priority class. The aim of this study is to determine a valid arrangement of incoming containers in a block (part) of the terminal, in the shortest amount of time, with higher priority containers located above lower priority ones. In this way, some of the main objectives of a container terminal may be achieved: avoiding further reshuffles (number of relocations) and reducing the vessel berthing time. We developed a genetic algorithm and its performance is evaluated against a random stacking strategy used as benchmark for the experiments, and through several sets of tests on control parameters. All the tests showed that, if a reliable estimation of the delivery time can be assigned to every incoming container, the proposed method may be a useful tool for container terminal operators.

Keywords: Genetic algorithms, optimization, container terminals, container stacking, residence time strategy.

1. Introduction

Container terminals are important nodal points in the global network of the maritime transport and efficiency of container handling activities is an important requirement in order to survive in this highly competitive environment. The performance of a container terminal, which often must handle thousands of containers in a short amount of time, is determined by its ability to provide services that meet the customer needs and depends on factors such as its physical capacity, handling equipment at the quay and storage areas, inland accessibility, etc.

In order to reduce the operational costs and improve the service quality in a container terminal, researches identified along the years several operational issues (Figure 1) that arise in the planning and scheduling of container terminals, such as berth allocation, crane assignment (see e.g. [5]), stowage planning, storage planning and stacking strategies (see e.g. [3,6]), and developed different tools to solve them. Some tools are based on mathematical models, such as linear programming, nonlinear programming or mixed integer programming; because of the computational complexity of solving these problems, other methods such as decision support systems or genetic algorithms were used to obtain an approximate solution (see [8,11]).

The operations of loading and unloading containers are among the most important tasks

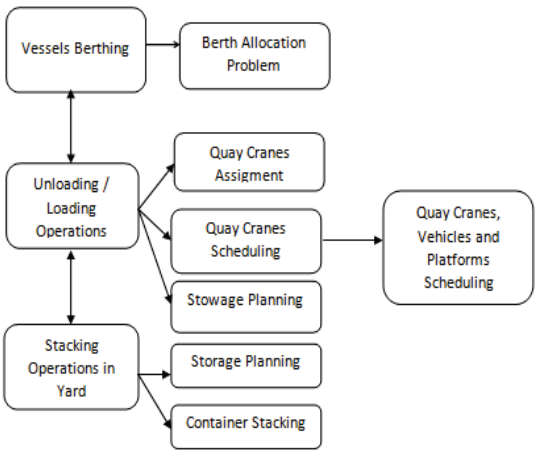

Figure 1 The most common optimization problems in container terminals

in a container terminal, the vessel berthing time being a measure of the quality of services of a terminal to ship liners. In order to maximize the efficacy of these tasks, each incoming container should be placed in a slot of the yard block in the shortest time possible and each outgoing container should be easily reachable at the time of its departure.

Due to continuous growth of the maritime container transport, the storage space has become a deficient resource; the problem of locating each container to the most suitable storage area in accordance with some objectives is known as the container storage problem and it falls into the category of NP

hard problems. In order to solve it, several types of storage and stacking strategies have been developed along the years (a review of the literature focused on them is to be found in [1] and [9]). Their main objectives are efficient usage of storage space, efficient loading and unloading operations and minimizing of reshuffles. In [13] the strategies are classified into storage planning and scattered stacking. Storage planning sets before the vessel's arrival the storage space of each block to stack different types of containers; in scattered stacking, the containers of the same type are stochastically spread over some blocks assigned to a berthing place, into a position determined in real time. In [1] there are mentioned three types of scattered stacking: random stacking, which randomly distributes containers over the blocks; category stacking, which implies defining some categories (based on a stowage plan) and stacking containers of the same category on top of each other; and residence time stacking, which means stacking a container on top of others if its departure time is earlier than that of all containers stored beneath it. As stated in [1], the residence time strategy is a good one to use: containers can be stacked more optimally if a reliable estimation of the departure time can be assigned to every container.

The problem employed in this paper depicts a form of residence time stacking (RTS). We extend the storage problem we presented in [12] by taking into consideration the priority ranks of the unloaded containers established based upon their departure time, thereby ensuring no reshuffles occur in the future, and we propose a genetic algorithm to solve it. Furthermore, we used the random stacking (RS) algorithm, implemented in [12], as benchmark for our tests.

Currently, the problem of minimizing the reshuffles (the container stacking problem) is treated after the completion of a vessel unloading task. The proposed method originality consists both in the way it attempts to solve, simultaneously, two of the main problems of a container terminal, minimization of reshuffles and of vessel berthing time, using a residence time strategy at the time of unloading a vessel, as well as in the original formulation of a GA over a container storage problem. Thus, if a reliable estimation of the departure time can be assigned to every incoming container, it may be a useful tool for terminal operators.

Section 2 gives the main features of the problem considered. In Section 3 the proposed method for solving the problem is presented. This section also provides the results of the proposed algorithm. Conclusion remarks and directions for future research are presented in Section 4.

2. Problem definition

The unloading task consists of a set of operations performed by (quay and yard) cranes and vehicles to unload a container from the vessel and discharge it into a slot of a bay in the container yard. The yard comprises several blocks, each one having several bays; each bay contains a set of rows, and each row consists of a set of tiers, usually 4 or 5 (Figure 2). The quay crane moves from its dwell point to the vessel, lifts a container and transfers it ashore to be loaded on a vehicle (train or truck). The loaded vehicle goes to the load/unload station of the appropriate block and shifts the container to the yard crane. The yard crane moves the container to a predetermined row of a bay in a particular block and places the container in the slot (Figure 3).

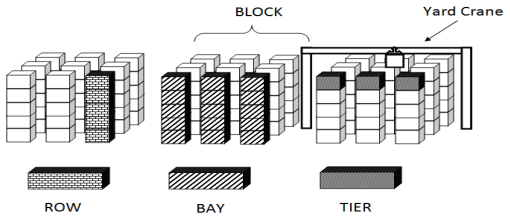

Figure 2 Layout of a yard in a container terminal

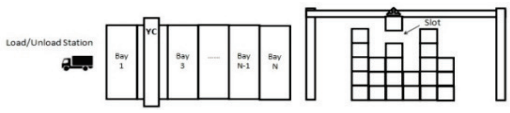

Figure 3 Layout of a block in a container terminal

The loading operation is also carefully planned in a container terminal. In order to have an efficient stowage plan, the outgoing containers should be put into storage in a good configuration, i.e. if need be, an outgoing container should be picked up immediately, instead of having several containers being stacked on top of it and needing to be moved elsewhere. Although the order in which containers will be picked up is usually not known in advance, a rough estimate of residence time can usually be made (see e.g. [1]) which could be used to identify which

containers should (not) be put on top of each other.

In this paper we consider the case when the containers unloaded from vessels (incoming containers) are given ranks according to their priorities which depend on their time of departure: the lower the rank number, the higher the priority associated, i.e. the sooner the container will be scheduled for departure. In order to facilitate the work of the yard crane and improve the containers handling time when they have to be picked up from the block and loaded on vessels, the containers should be stacked above each other with higher priority (lower rank number) containers located at the top of the row (Figure 4). This type of layout ensures easy access to outgoing containers at the expected time of transfer without further reshuffles.

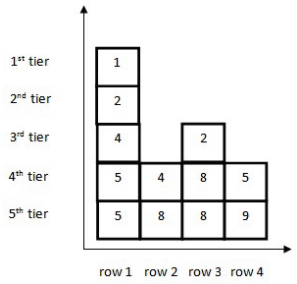

Figure 4 A desired layout of a block with containers stacked according to their priority

Our approach is focused on the operational decisions that have to be made by terminal operators and attempts to optimize the container stacking in the block as well as the incoming vessel berthing time by minimizing the handling time of the yard crane while placing the containers in the slots of the stacking area in the right priority order, i.e. no higher priority container is placed beneath a lower priority container.

2.1. Assumptions

While considering some block in the container yard, the following assumptions were considered:

1. Transportation times of loaded and empty yard crane are the same
2. Transferring time between yard crane and vehicles is assumed small enough to be ignored
3. The load/unload station of the block is predetermined and fixed
4. The bays are numbered as in Figure 3 (bay no.1 is the one near the load/unload station)
5. The dwell point policy for the yard crane is "return to start" (to the load/unload station)
6. The initial layout of the block (i.e. the priority ranks and the number of available cells in each row of the bays in the block) is known before the start of the unloading task
7. In the initial layout of the block the containers are stacked in the right priority order
8. The priority ranks of (most of) containers to be unloaded from a vessel are known; we assume that there is an emergency block for containers without priority ranks and leave them out of consideration; we also know the number of containers of each priority ranks
9. A container may be placed in a slot of a row if its priority rank does not violate the right priority order (higher priority container is placed upon lower priority container).

2.2 Input parameters

We will use the following notations:

- PR the number of priority ranks considered

- n_p the number of incoming containers of priority rank p, $C = \{n_p\}_{1 \le p \le PR}$

- R the number of rows in the block

- s_r the number of slots (i.e. empty cells) in row r in initial layout of the block, $S = \{s_r\}_{1 \le r \le R}$

- N the number of incoming containers, $N = \sum_{\mathrm{p}=1}^{\mathrm{PR}} n_p$, $N \le \sum_{r=1}^{\mathrm{R}} s_r$

- p_{i} the priority rank of incoming container i, $p_i \in \{1,2,\dots,\mathrm{PR}\}$, $i \in \{1,2,\dots,N\}$

- P the list of possible valid allocations, $P = \{(\mathrm{r},\mathrm{p}) | 1 \le r \le R, 1 \le \mathrm{p} \le PR\}$

An allocation of a slot of a row to a container is valid if the container priority rank does not violate the right priority order for that row. Given the number of incoming containers and their priority ranks, the list P is generated. For example, for the initial layout from Figure 4, assuming there are five tiers, the list of number of slots will be $S = \{0,3,2,3\}$, so the set of all valid allocations will be *{(2,1), (2,2), (2,3),*

(2,4) (3,1), (3,2), (4,1), (4,2), (4,3), (4,4), (4,5)}; if $C = \{0, 2, 3\}$ (the list of incoming containers comprises of two containers with priority rank equal to 2 and three containers with priority rank equal to 3), then the list of possible valid allocations will be *P = {(2,2), (2,3),(3,2),(4,2),(4,3)}*.

- $T_1(r)$ the time needed for yard crane to move to row r, $\mathrm{r} \in \{1,2,\dots,\mathrm{R}\}$

- K the number of tiers in a row

- k a tier in a row, $\mathrm{k} \in \{1,2,\dots,\mathrm{K}\}$; $k = 1$ is the top of the row, $k = K$ is the bottom of the row

- $\mathrm{T}_2(k)$ the time needed for yard crane to shift to tier $k, \mathrm{k} \in \{1,2,\dots,\mathrm{K}\}$

2.3. Mathematical model

1) Decision Variables

The decision variables will be $c_i = (r_i, p_i)$, $i \in \{1,2,\dots,N\}$, c_i representing the incoming container i of priority rank p_i placed in row r_i, $r_\mathrm{i} \in \{1,2,\dots R\}$, $p_\mathrm{i} \in \{1,2,\dots,\mathrm{PR}\}$.

2) Objective function

The objective function is the handling time of the yard crane. It is composed of the time needed for crane to move to a row from the dwell point, to shift to a tier and to return to the dwell point. The objective function is determined by summing the handling time of the yard crane while placing the containers in the slots (row/tier). The priority rank of each container is also taken into consideration, favoring the allocation of lower priority ranks on top of the rows.

$$f(c_1,\dots,c_N) = \sum_{\mathrm{i}=1}^{\mathrm{N}}\left(2\mathrm{T}_1(\mathrm{r_i}) + \frac{1}{\mathrm{p_i}}T_2\left(\mathrm{s_{r_i}} - \sum_{\mathrm{j}=1}^{\mathrm{i}-1}[\mathrm{r_j} = \mathrm{r_i}]\right)\right) \quad (4)$$

where the notation [.] implies the Iverson bracket, defined by

$$[M] = \begin{cases} 0, if\ M\ is\ false \\ 1, if\ M\ is\ true \end{cases}$$

where *M* is a mathematical statement.

3) Constraints

(*C1*) the number of containers that may be placed in a row can not be higher than the number of slots of that row

$$\sum_{i=1}^{N}[r_i = r_j] \le s_{r_j}, \forall\ 1 \le j \le N, 1 \le r \le R \quad (1)$$

(*C2*) the number of containers stacked with a particular priority rank must be equal to the number of unloaded containers of that priority rank

$$\sum_{i=1}^{N}[p_i = p_j] = n_{p_j}, \forall\ 1 \le j \le N, 1 \le p \le PR \quad (2)$$

(*C3*) the incoming containers must be stacked in the right priority order

$$if\ \ r_i = r_j\ and\ k_i \le k_j\ then\ p_i \le p_j$$
$$\forall\ 1 \le i, j \le N, 1 \le p \le PR \quad (3)$$

3. The proposed genetic algorithm

Genetic algorithms (GAs) are well-known meta-heuristic methods used in a wide area of practical problems in science and industry, e.g. optimization, machine learning, economics or population genetic problems (e.g. [4], [10]). They are inspired by the natural evolution of the living organisms, with a strong exploration ability to search the feasible space.

3.1. Encoding and initialisation

A GA starts by chromosome encoding and initialisation. A chromosome is composed of genes whose values can be either bit-strings, real numbers, symbols or characters, the interpretation of these strings being entirely problem dependent.

We considered genes to be composed of two cells (Figure 5(a)), the row (left cell) and the priority rank (right cell) that is valid for that row (according to the right priority order).

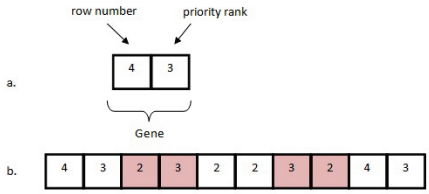

Figure 5 Chromosome representation

A chromosome is a set of genes randomly selected from the list *P*. The chromosome length is fixed and is equal to the number of incoming containers, *N*. In order to establish the slot (row/tier) where each of the containers will be placed, the chromosome will be decoded from left to right. For each placement, the row r is given by the left cell of a gene *g*, and the tier *k* is obtained by considering the number of occurrences of gene *g* in chromosome from left to right and the number of slots for row r (e.g. for chromosome in Figure 5: r = *4, k = 3* for the first gene, r = *2, k*

$= 3$ for the second gene, $r = 2,\ k = 2$ for the third gene, etc.).

In order to generate feasible chromosomes, the constraints *(C1), (C2)* and *(C3)* must be applied for each chromosome (Figure 6).

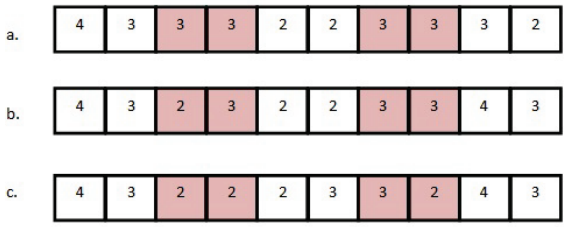

Figure 6 a.According to *C1*, this chromosome is not feasible as it proposes to place three containers on the third row, whilst this row has only two slots; b. According to *C2*, this chromosome is not feasible as it locates four containers of priority rank equal to three, whilst only three containers of that priority rank are to be unloaded; c. According to *C3*, this chromosome is not feasible as it places on the second row a container with a priority rank equal to two beneath a container with a priority rank equal to three

After encoding the chromosomes, we set the algorithm parameters (population size, maximum number of generations, crossover and mutation probabilities) and generate the initial population.

3.2. Evaluation

The fitness function is problem specific and measures the quality of the solutions. In our model, the fitness function is the one described by (4) and is determined by decoding the chromosomes from left to right, a gene being c_i, $i \in \{1,2,\dots,N\}$. Since one of the objectives of this problem is to minimize the handling time of the yard crane, the smaller the value of fitness obtained, the better the solution (chromosome).

3.3. Selection, crossover and mutation operators

Genetic operators used in GAs maintain genetic diversity and are analogous to those which occur in the natural world: selection (or reproduction), crossover (or recombination) and mutation. These operators are implemented to produce new offspring, which are in charge of exploration and exploitation of the feasible solution space (see for details [4]).

According to the crossover probability, some of the least fitted individuals will be replaced by a new set of offspring. The parents were selected by the tournament selection method, which works by running several tournaments, i.e. selecting a number of individuals from the population at random and then selecting only the best of those individuals (with the best fitness).

Next, we used a version of the three parents crossover, dependent of the nature of our problem and designed by taking into consideration the constraints (*C1*) and (*C2*). The crossover operator randomly selects three parents and breeds three children setting in the offspring gene a value taken from one of the three parents from the matching position (see Table 1). The offspring created replaces the parents.

Table 1 Pseudo-code of crossover operator

```
% s: population size
% n: chromosome length
% pc: crossover probability


for i = 1:s do
 Choose a random integer between 0 and 1
(we call it cLimit)
 if cLimit < pc then
    select chromosome i as parent
end


for k = 1:numberOfParents-2
 P1 = parents(k);
 P2 = parents(k+1);
 P3 = parents(k+2);
  for i = 1:n do
% creating Child1
  if P1(i) = P2(i) AND P1(i) follows C1
and C2 then
     offspring1(i) = P1(i)
  else
    if P2(i) follows C1 and C2 then
       offspring1(i)= P2(i)
    else
      if P1(i) follows C1 and C2 then
         offspring1(i) = P1(i)
      else
        if P3(i) follows C1 and C2 then
           offspring1(i) = P3(i)
        else
```

```
            for j = 1:n do
              if P3(j) follows C1 and C2
                 offspring1(i) = P3(j)
            end
% create Child2 in an analogous manner:
combine P2 with P3 and if one of the
genes does not follow constraints C1 and
C2, get a gene from P1
% create Child3 in an analogous manner:
combine P3 with P1 and if one of the
genes does not follow constraints C1 and
C2, get a gene from P2
  end
end
```

In next step of the GA, we used the swap mutation operator to enhance the chromosomes by changing their sequence of genes. This operator works by generating two different random numbers within the length of the chromosome that will specify the places of the genes that will be swapped.

Owing to the random nature of the GA, violations of constraint (*C3*) may occur in any chromosome during crossover or mutation. Hence, a repair subroutine is applied to impose the right priority order and make the chromosome a feasible one.

The algorithm was implemented in Matlab R2011a and the terminating condition was to either a predefined number of generations reached or 97% of the population had same fitness value. After several generations, the best individual (solution) is obtained, i.e. the one with the smallest numerical value of the fitness function.

Figure 7 illustrates a possible solution found by the proposed GA, which may be interpreted as the plan of container stacking. The order of execution of tasks by the yard crane is arbitrary, and, given the way the objective function is defined, does not affect the solution of the problem.

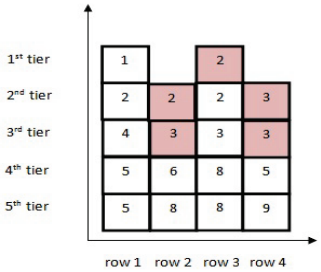

Figure 7 Best solution of the proposed GA (an example)

3.4. Results

Table 2 shows the characteristics of the problem tested in this paper. The basic configuration of the stacking area (see Figure 8) is the same with the one from the earlier work of the same authors. Likewise, we used the same data as in [12] for the operational time for the yard crane. We used minutes as the base time unit; hence, if a yard crane movement takes 14 seconds in real life, it takes 0.23 time units in the problem.

In a real-world situation, it is uncommon for a terminal operator to know exactly when a container will arrive or depart. However, at any time, a container has a residence time left in the block. In order to implement our residence time stacking strategy, we used a residence time divided into 5 classes.

Table 2 Characteristics of the selected problem

Number of rows(R)	12
Maximum number of tiers(K)	5
Number of slots	25
Number of incoming containers(N)	19
Number of priority ranks(PR)	5

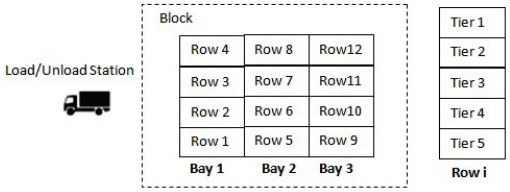

Figure 8 The proposed layout of a block

We evaluated the performance of the proposed GA through several test cases on the control parameters of the GA: the crossover/mutation probabilities. Currently, there are no well-established optimum values for these parameters and they are problem specific [7]. In order to get statistically robust results, each of the test cases was solved using the proposed GA for 10 runs (see Table 3). All the cases have the same population size (50) and the same maximum number of generations (100).

Table 3 GA parameters

Case number	Crossover probability	Mutation probability
1	0.90	0.01 : 0.1, step 0.01
2	0.75	
3	0.60	
4	0.45	
5	0.30	
6	0.15	

The results of the test cases are presented, in terms of the mean of the fitness values and the standard deviation (SD), in Figures 9-12. The standard deviation values are less than 10% of the mean values for these test cases, which means that the GA can find solutions close to each other in its various runs. Based on the minimum fitness value, the best combination of crossover and mutation probabilities is a crossover value of 0.9 and a mutation value of 0.01.

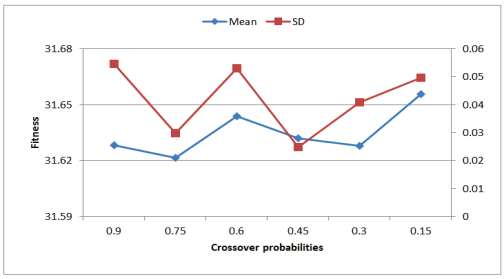

Figure 9 Average fitness using different crossover probabilities

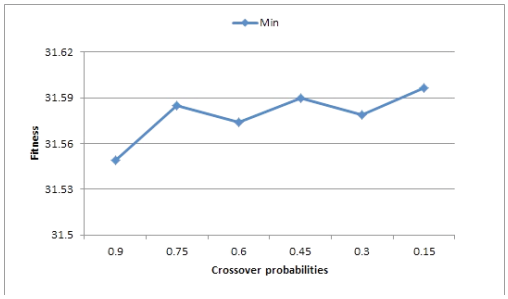

Figure 10 Minimum fitness using different crossover probabilities

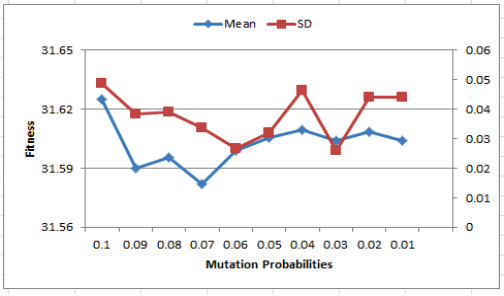

Figure 11 Average fitness using different mutation probabilities

For further evaluation of the performance of the proposed GA, as well as of the RTS strategy, we used the GA described in [12], which implements a random stacking (RS) strategy. Using a RS strategy, a container is placed at a randomly chosen allowed location, with every possible location having an equal probability of being chosen.

The RS strategy is the most basic stacking strategy and is often used as a benchmark algorithm. The results of the comparison can be found in Figure 13.

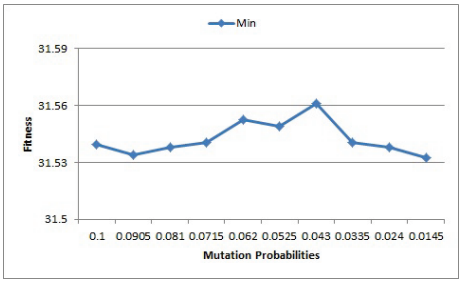

Figure 12 Minimum fitness using different mutation probabilities

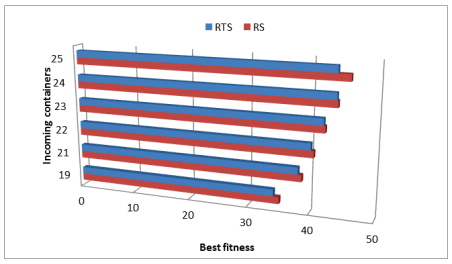

Figure 13 RTS and RS strategy results

Having the same configuration of the stacking area (Table 4), the RTS algorithm performs better, in terms of handling time, than the RS algorithm, confirming the results from [1] and [2]. Therefore, we can conclude that the random strategy is good when there are no additional information regarding the incoming containers, but getting a good estimate for the residence time of containers may lead to better results in a real life scenario in terms of berthing time, as well as to a stacking plan that will minimize further reshuffles.

Table 4 RTS and RS strategy results

Strategy		RTS	RS
GA parameters	Crossover probability	0.75	0.9
	Mutation probability	0.06	0.1
Problem characteristics	Rows	12	12
	Tiers	5	5
	Slots	25	25
	Priority ranks	5	-

4. Conclusion

This paper addresses a container storage problem, which involves making efficient decisions fast in order to meet the requirements of the terminal operators as well as those of the ship owners, and solve it by means of a residence time stacking strategy. We proposed a genetic algorithm to optimize the process of stacking the containers in the right priority order, i.e. no lower priority container is placed upon a higher priority container. The type of layout given by the solution of the proposed method minimizes the reshuffles and reduces the time spent by a ship in the port quays, by allowing it to be loaded it in a short time. Likewise, the berthing time of the incoming vessel is also reduced by minimizing the handling time of the yard crane while stacking the incoming containers.

In a real-world situation, a terminal operator rarely knows exactly when a container is going to arrive or depart; however, at any time, a container has a residence time left in the block. Hence, based on the residence time stacking strategy adopted, this optimization method may lead to reasonable solutions in real life settings.

As future work, we intend to refine the problem characteristics based on observations from real case situations and introduce more parameters related to the stacking options, such as the container weight (lighter containers are stored on top of heavier ones) or nature (regular, tank, empty or refrigerated). Furthermore, the length and width of blocks are important variables in designing the layout of yard and they also influence the container handling process.

REFERENCES

1. Borgman, B., Asperen, E., Dekker, R. (2010) Online rules for container stacking, *OR Spectrum,* 32, pp. 687–716.
2. Dekker, R., Voogd, P., Asperen, E. (2006) Advanced methods for container stacking, *OR Spectrum*, 28, pp. 563–568.
3. Gheith, M.S., EL-Tawil, A.B., Harraz, N.A. (2013) A Proposed heuristic for Solving the Container Pre-marshalling Problem, *In the 19th International Conference on Industrial Engineering and Engineering Management, edited by Ershi Qi, Jiang Shen and Runliang Dou, Springer Berlin Heidelberg,* pp. 955-964.
4. Haupt, R.L,. Haupt, S.E. (1998) Practical Genetic Algorithms, *John Wiley & Sons, Inc. New York,* NY, USA.
5. Homayouni, S.M., Tang, S.H., Motlagh, O. (2014) A genetic algorithm for optimization of integrated scheduling of cranes, vehicles, and storage platforms at automated container terminals*, Journal of Computational and Applied Mathematics*, 270, pp. 545-556.
6. Kammarti, R., Ayachi, I., Ksouri, M., BORNE, P. (2009) Evolutionary Approach for the Containers Bin-Packing Problem, *Studies in Informatics and Control,* 18 (4), pp. 315-324
7. Kumar, A., Prakash, A., Tiwari, M.K., Shankar, R. & Baveja, A. (2005) Solving machine-loading problem of a flexible manufacturing system with constraint-based genetic algorithm, *European Journal of Operational Research, 175*, pp. 1043-1069
8. Lajjama, A., EL Merouani, M., Tabaa, Y., Medouri, A. (2014) A new approach for sequencing loading and unloading operations in the seaside area of a container terminal, *International Journal of Supply and Operations Management* 1(3), pp. 328-346.
9. Luo, J., Wu, Y., Halldorsson, A., Song, X. (2011) Storage and stacking logistics problems in container terminals, *OR Insight,* 24, pp. 256–275.
10. Norouzi, A., Babamir, F.S., Zaim, A.H. (2013) An Interactive Genetic Algorithm for Mobile Sensor Networks, *Studies in Informatics and Control,* 22(2), pp. 213-218.
11. Salido, M.A., Sapena, O., Barber, F. (2009) An Artificial Intelligence Planning tool for the Container Stacking Problem, *In Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation, edited by IEEE Press Piscataway,* NJ, USA, pp. 532-535.
12. Serban, C., Carp, D. (2016) Optimization of container stowage in a yard block using a genetic algorithm, *Studies in Informatics and Control,* 25(1), pp. 123-130.
13. Steenken, D., VOß, S., Stahlbock, R. (2004) Container terminal operation and operations research - a classification and literature review, *OR Spectrum,* 26*,* pp. 3-49.