Optimization of Container Stowage in a Yard Block Using a Genetic Algorithm

Cristina SERBAN[1], Doina CARP[2]

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

[2] Constanta Maritime University,
Mircea cel Batran Str. 104, Constanta, 900663, Romania,
doina.carp@gmail.com

Abstract: One indicator for efficient management in a port is the time spent by a ship in the port quays. The time allowed for loading-unloading into a specialized quay is mentioned in the management contract. Because the cost of the overtime is very high, it is very important to have a special plan to unload the container ship in a short time. Given the number of containers to be unloaded from a vessel and the initial state (in regards of number of slots) of a block, the genetic algorithm that we propose in this paper finds the plan of container stacking in the block, whilst the objective function is to minimize summation of handling time of yard crane in placing the containers in the available storage cells of the stacking area. The performance of the proposed method is evaluated through several sets of tests on control parameters of the algorithm.

Keywords: containers, cranes, stacking, genetic algorithms

1. Introduction

One of the objectives of an efficient management plan in a port is to reduce the berthing time of vessels. The port efficiency is determined by the main container terminal operations, namely the vessel berthing operation, (quay or yard) crane unloading/loading operations and container shifting and storage operations.

In order to develop better operational strategies and investment plans, researches identified several optimization problems along the years and developed different tools based on intelligent techniques in order to achieve optimal solutions (Figure 1).

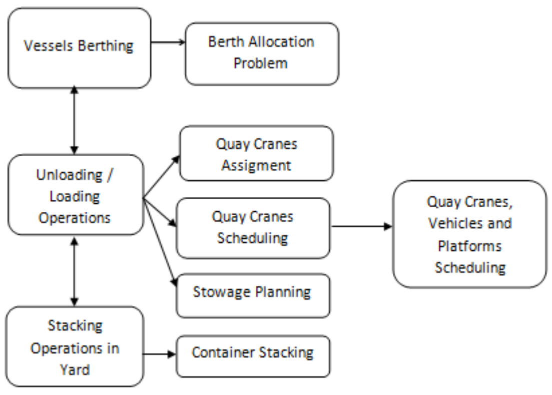

Figure 1. The most common optimization problems in container terminals

The Container Stacking Problem(CSP) consists in relocating the containers to ensure easy access to them so that the yard cranes don't have to do further reshuffles at the expected time of transfer (e.g. [3], [10]). The CSP is classified as a three dimensional bin packing problem in [6], and a genetic algorithm is proposed to solve it. The Berth Allocation Problem (BAP) and the Quay Crane Assignment Problem (QCAP) refer to the allocation of docks and quay cranes to incoming vessels under several constraints and priorities (see [11], where the CSP, BAP and QCAP problems are considered and a computer-based decision support system that integrates the solution of the three problems is provided).

The Quay Crane Scheduling Problem (QCSP) main objectives are first to find the assignment of tasks to quay cranes and then to determine the tasks sequence for each quay crane in order to minimize the handling time while respecting certain constraints (a feasible solution to this problem is found in [8] using a probabilistic technique inspired from ants behaviour). An extension of QCSP is the Integrated Quay Cranes, Vehicles and Platforms Scheduling Problem (IQCVPSP) considered in a split-platform automated storage/retrieval system (see [5], where a genetic algorithm is proposed to solve it).

In this paper we consider a Stowage Planning Problem which occurs during the unloading task of a vessel. In this respect, a genetic algorithm is proposed and optimal solutions for this problem are found in relatively low computational time.

Section 2 gives the main features of the problem considered. In Section 3 we briefly describe the basic principles of genetic algorithms, then we thoroughly depict the one that we proposed. This section also states the results of the several tests that we have been executed on the control parameters of the proposed genetic algorithm. Conclusion remarks and recommendations for further research directions are presented in Section 4.

2. Problem Definition

The unloading task is the set of operations performed by cranes and vehicles to unload a container from the vessel and discharge it into a slot of a bay in the container yard. The quay crane moves from its dwell point to the vessel, picks up the desired container and shifts it ashore to be loaded on a land vehicle (train or truck). The vehicle moves to the yard (Figure 2) which consists of 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). The destination of the container is a slot in a row of a bay in a particular block. The loaded vehicle travels to the load/unload station of the dedicated block and transfers the container to the yard crane. The yard crane moves the container to the predetermined row and places the container in the slot (Figure 3).

Our approach attempts to optimize the container stacking in the block by minimizing the handling time of the yard crane while placing the containers in the slots of the stacking area.

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)

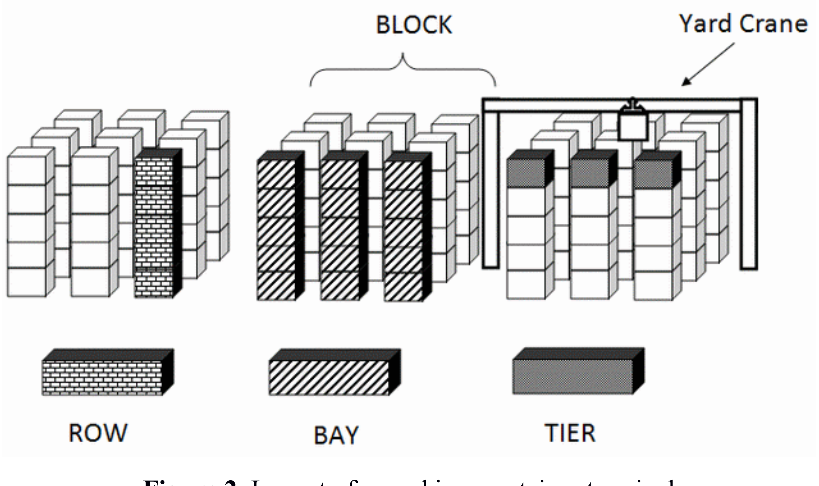

Figure 2. Layout of a yard in a container terminal

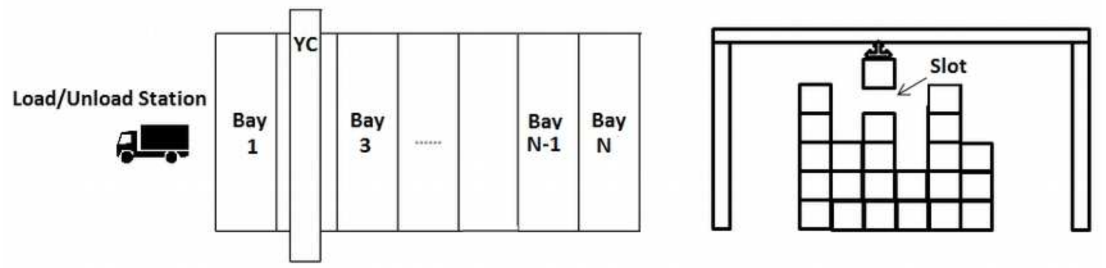

Figure 3. Layout of a block in a container terminal

5. The dwell point policy for the yard crane is "return to start" (to the load/unload station)
6. The initial state of the block, i.e. the number of available cells in each row of the bays in the block, is known before the start of the unloading task
7. The number of containers to be unloaded from a vessel is known; it may be less, equal or greater than the capacity of storage of the block considered
8. The terminal operators ultimately decide which solution is the most appropriate for a particular block in relation to a multi-objective problem: optimization of handling time upon that block or meeting the requirement to storage in that block as many containers as possible.

3. The Proposed Genetic Algorithm

In our model, the yard crane picks up a container from the vehicle and transfers it in a slot at the top of a row (each row has up to 5 tiers) in a block. We assumed that a row could be completed or not, the number of missing containers (up to 5) being retained for each row. We want to find the plan of container stacking in the block, and a genetic algorithm is proposed to find optimal solutions for this problem in a reasonable computational time.

3.1 Basic principles of genetic algorithms

Genetic Algorithms (GAs) are adaptive heuristic search algorithms. Having a highly modular nature, they are used in a wide area of practical problems in science and industry, e.g. optimization, machine learning, economics or population genetic problems (e.g. [1], [2], [7], [9]). GAs are specially designed to find good solutions to problems that were otherwise computationally unsolvable: the solution sets are finite but so large that brute-force evaluation of all possible solutions is not computationally feasible.

GAs are based on the evolutionary ideas of natural selection, applying the principle of 'survival of the fittest' on a population of potential solutions encoded as chromosomes, selecting individuals according to their level of fitness, and mating them together using some recombination operators to produce better approximations to a solution (see [4], [12]).

Solving a problem with GAs means to search a solution in the space of all potential solutions using a population of agents. The search process is based on two mechanisms: exploration (go through different regions in space of solutions and gather information) and exploitation (refine the solution, i.e. the information collected through exploration process). The search is guided through a function called fitness, which measures the closeness to the solution.

An implementation of a genetic algorithm begins with a population of random chromosomes, which are string representations of solutions to a particular problem. 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. The size of population depends on the problem and the type of encoding used and must be chosen with care because it affects greatly the efficiency of a GA (see [12]): if it was too small, the GA would have only a few possibilities to perform crossover, so only a small part of the search space would be explored, and if it was too large, the GA would slow down. The most common size for a population is between 30 and 100 individuals.

The objective function f provides a measure of individuals performance with respect to the problem domain. This performance is transformed by the fitness function F into a measure of allocation of reproductive opportunities for individuals in a GA. The fitness function is problem specific and is derived from the objective function.

For unconstrained optimization problems:

– maximization problems: the fitness function can be considered to be the same as the objective function
– minimization problems: the most fit individuals will have the smallest numerical value of the corresponding objective function; moreover, due to the fact that some operators need non-negative values of the fitness, it is necessary to map the underlying natural objective function to fitness function form. A most commonly adopted fitness mapping is the one from relation (1), which does not alter the location of the minimum, but converts a

minimization problem to an equivalent maximization one.

$$F(x)=\frac{1}{1+f(x)} \quad (1)$$

For constrained optimization problems, one must add penalty functions to the objective function, so problem (2)

$$\min f(x) \quad (2)$$

subject to

$$\begin{cases} g_i(x)=0, & i=1,\dots,m \\ h_j(x)\leq 0, & j=m+1,\dots,q \end{cases}$$

turns into problem (3),

$$\min p(x) \quad (3)$$

$$p(x)=f(x)+\sum_{i=1}^{m} a_i\varphi(g_i(x))+\sum_{j=m+1}^{q} b_j\psi(h_j(x))$$

where φ , ψ are penalty functions

$$\varphi(u)=\begin{cases} 0, u=0 \\ \infty, u\neq 0 \end{cases} \qquad \psi(u)=\begin{cases} 0, u\leq 0 \\ \infty, u>0 \end{cases} \quad (4)$$

and $a_i, b_j>0$ are penalty parameters.

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.

Selection determines the number of times a particular individual is chosen for reproduction, i.e. the number of offspring that an individual will produce. Chromosomes evaluated with higher fitness values will most likely be selected as the parents of a pair of offspring, whereas, those with low values will be discarded from the current population.

Recombination is the process by which chromosomes selected from a source population are recombined to form members of a successor population. After selection has been carried out, the crossover operator is applied to two individuals, randomly paired with a user-definable probability, *pc*, called crossover rate. The recombination operation produces two offspring that inherit traits of both parents and are inserted, one or both of them, into the next population.

Mutation alters one or more gene values in a randomly chosen chromosome with a user-definable probability, *pm*, called mutation rate, and produces a new genetic individual. This operator is used to maintain genetic diversity from one generation of a population to the next, and has two roles, firstly to recover good genetic material that may be lost through selection and crossover, and secondly, to provide the genes that were not present in the initial population.

Evaluation, selection, crossover and mutation (Figure 4) forms one generation in the execution of a GA. After several generations, the best individual (solution) is obtained. In cases where the problem to be solved does not have one individual solution, as is the case in multi-objective optimization and scheduling problems, the GA is useful for providing a number of potential solutions at once, letting the user to choose the best one.

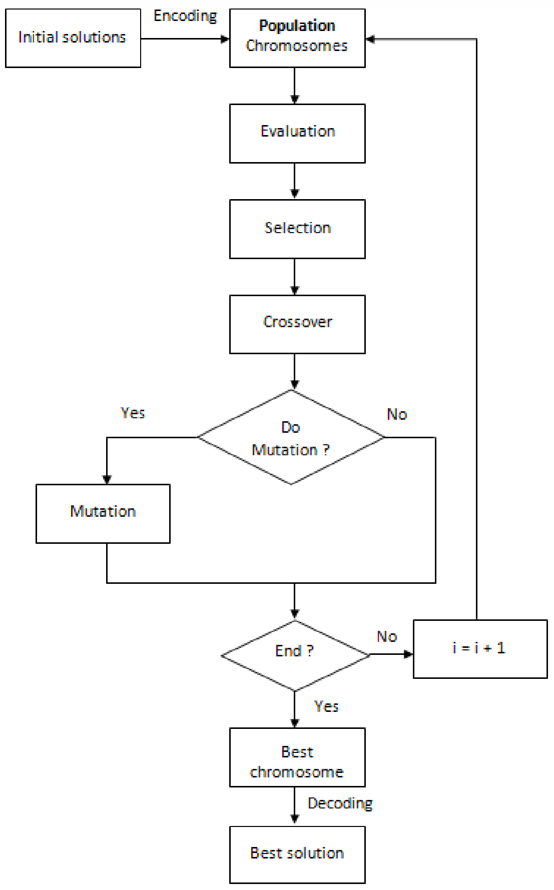

Figure 4. The proposed GA flowchart

3.2 Encoding and initialisation

In our model, the rows in the block are numbered from 1 to *n*. A chromosome is a feasible sequence of the rows in the block and

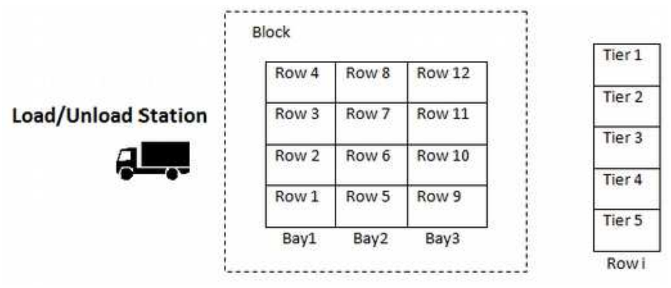

Figure 5. The proposed layout of a block

is encoded using an integer array having size equal to *n*; each element of this array denotes the number of containers missing in that row (maximum number of containers that may be stack in a row is 5). For our problem, we suppose we have a block with 3 bays, each with 4 rows (Figure 5).

The following chromosome: [0 5 3 4 5 1 0 0 2 3 0 1] indicates that the first row is completed, the second is empty and the third has 3 containers missing.

The initial population of size *s* is constructed by randomly generating $n \cdot s$ integers ranging from 0 to 5. We also set the chromosome, denoted by *initChrom*, that will show the initial state of rows in the block (e.g. *initChrom* = [4 2 0 1 0 5 2 1 0 0 2 0]).

3.3 Evaluation

We denote by *M* the number of slots in block, by *n* the number of rows in block and by x_i the number of containers missing in a row, $x_i \in \{0,1,2,3,4,5\}$, $i \in \{1,2 \ldots n\}$. Let T_i be the time needed for yard crane to move to row *i*. We will also consider t_k as the time needed for yard crane to shift from one tier to another, $k \in \{1,2,3,4,5\}$. Let $x_{init_i} \in \{0,1,2,3,4,5\}$ be the number of containers missing in the initial state of the block (values of *initChrom* genes). The goal is to

minimize

$$f(x)=\sum_{i=1}^{n} \sum_{k=x_i+1}^{x_{init_i}} (2T_i+t_k) \tag{5}$$

subject to

$$\sum_{i=1}^{n} (x_{init_i}-x_i) \le M \tag{6}$$

The objective function represents the summation of the time needed by the yard crane to pick the container from the load/unload station, place it in a slot and then return to the station. Constraint (6) imposes that the number of containers stacked in a block shouldn't exceed the capacity of that block.

Being a constrained optimization problem, we must add penalty functions to the objective function (5)

$$\psi(u)=\begin{cases} 0, u \le 0 \\ -u, u>0 \end{cases} \tag{7}$$

(without penalty if restriction is met; the penalty corresponds to the level the restriction is unmet), so the new objective function will be (according to (3) and (7)):

$$p(x)= \begin{cases} f(x), \quad if \sum_{i=1}^{n} (x_{init_i}-x_i) \le M \\ af(x)-b\left(\sum_{i=1}^{n} (x_{init_i}-x_i)-M\right), otherwise \end{cases} \tag{8}$$

where the penalty parameters $a, b>0; a+b=1$ control the weights of the two components of the problem: optimization of the objective function or meeting the restriction.

The fitness function will be given by relation (1).

3.4 Selection, crossover and mutation operators

The chromosomes are evaluated by using the fitness function and the current population is sorted in ascending mode. According to the crossover rate, some of the least fitted individuals will be replaced by a new set of offspring. The parents were selected by one of the most commonly used selection methods, namely the roulette-wheel selection, also known as the fitness proportionate selection. Conceptually, this method can be thought as a game of Roulette, each individual getting a slice of the roulette wheel equal in area to its fitness. The wheel is spun and on each spin, the individual under wheel's arrow is selected to be parent (see Table 1).

Table 1. Pseudo-code of selection operator

```
Sum the fitness F of all individuals
in the population (we call it FSum)
Compute the probability for each
individual i:

            $P(i)=\frac{F(i)}{FSum}$

Compute the cumulative probability
for each individual i:

            $CP(i)=\sum_{k=1}^{i} P(k)$
for i = 1:s do
   Choose a random integer between 0
and 1
  (we call it sLimit)
    if CP(i) >= sLimit then
       select chromosome i

end
```

Next, we propose both a crossover and a mutation operator, dependent of the nature of our problem and designed taking into consideration the restriction (9): the number of containers that may be stack in a row during the unloading task can not be higher than the initial number of containers missing from that row.

$$x_i \leq x_{init_i}, \forall i=1,\ldots,n \tag{9}$$

The crossover operator randomly selects two parents and breeds the offspring by setting in the offspring gene a value less than or equal to that of *initChrom* on the matching position, so that the offspring will not have greater values (i.e. more containers missing) than the initial chromosome. The offspring created replaces one of the parents, randomly selected (see Table 2).

In next step of the GA, we used a version of the uniform mutation operator in which some of the chromosomes belonging to the current population and one gene of each of them are randomly selected; the uniform operator replaces the value of the chosen gene with a uniform random integer selected between the upper and lower bounds for that gene (0 and 5, respectively). In addition, the proposed mutation operator is performed only if

Table 2. Pseudo-code of crossover operator

```
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
 P1 = parents(k); P2 = parents(k+1);
 for i = 1:n do
  if P1(i) = P2(i) then
     offspring(i) = P1(i)
  else
    if P1(i) <= initChrom (i)
and P2(i) <= initChrom(i) then
      offspring(i)= min(P1(i),P2(i))
    else
      if P1(i) <= initChrom(i) then
         offspring(i) = P1(i)
      else
        if P2(i) <= initChrom(i) then
           offspring(i) = P2(i)
        else
          if P1(i) > initChrom i)
and P2(i) > initChrom(i) then
           offspring(i) = initChrom (i)
  end

end
```

replacing the selected gene does not violate the constraint given by relation (9) (see Table 3).

Table 3. Pseudo-code of mutation operator

```
Compute number of mutations:
              $noM = pm \cdot s$

Compute total number of genes in
population:

              $noG = n \cdot s$
while noM > 0 do
    Choose a random integer between 1
and noG (we call it mLimit)
    Choose a chromosome:
              $chrom=\frac{mLimit}{n}$
    Choose a random gene
              $gen=mLimit \ \% \ n$

    The value of mutated gene is
replaced by a random number between 0
and 5 and smaller than the value in the
same position in initChrom
          noM = noM - 1;
end
```

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. The best solution is the one with the smallest numerical value of the objective function and, in addition, meets the constraint (9).

Table 4 illustrates a solution found by the proposed GA. Given the initial state of the block, i.e. *initChrom* = [4 2 0 1 0 5 2 1 0 0 2 0], one may interpret the solution as the plan of container stacking, i.e. "stack 4 containers on row 1, 1 container on row 4, 4 containers on row 6" and so on.

Table 4. Best solution of the proposed GA (an example)

0	2	0	0	0	1	0	0	0	0	0	0

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.

3.5 Results

Performance of the proposed GA was evaluated through several test cases on the control parameters of the GA: the population size and the crossover/mutation rates. We used a small size problem based on the design of a block depicted in Figure 5 and an average operational time for the yard crane, which we considered was suitable for this problem. The most important objective is to minimize the handling time in tasks of yard cranes, so the weights *a* and b were fixed to 0.7 and 0.3, respectively.

Different combinations of population size and crossover/mutation rates have been tested, and each of the test cases was solved using the proposed GA for 30 replications. The mean of the objective function values, the standard deviation (SD) and the CPU time in seconds for the test cases are recorded in Table 5. These results indicate that the combination of 0.9 and 0.1 for crossover and mutation rates have the best performance among all the tested combinations, which can be interpreted as breeding more offspring in each generation is more efficient at obtaining better solutions than exploring new search areas. The results also demonstrates that 25 as the population size shows the best performance among other values. Moreover, the standard deviation values are approximately 10% of the mean values for this test case, which means that the GA can find solutions close to each other in its various runs. The table also shows the average number of containers that may be stacked in a block with maximum capacity of 25, in a period of time given in minutes (e.g. 22 containers may be stacked in 85.63 minutes).

4. Conclusion

In this paper, we approach a Stowage Planning problem and propose a genetic algorithm to optimize the container stacking in a block. The results of the algorithm may be considered as the plan of the container stacking and used to reduce the time spent by a ship in the port quays, by unloading it in a short time. As future work, we intend to refine the objective function based on observations from real case situations and introduce into the objective function more parameters related to the stacking options, like the size, priorities or type of the containers.

Table 5. The proposed GA results

Pop. Size	**Cross./ Mutat. rate**	**Mean (minutes)**	**SD**	**CPUtime(s)**	**Containers stacked (average)**
25	0.9/0.1	85.63	8.12	19.51	22
25	0.9/0.01	79.29	20.24	7.16	20
25	0.7/0.1	73.66	23.36	33.87	20
50	0.9/0.1	85.78	10.20	40.16	22
50	0.9/0.01	71.68	29.73	13.22	18
50	0.7/0.1	67.04	27.88	76.79	18
100	0.9/0.1	78.35	22.90	84.66	20
100	0.9/0.01	80.33	26.84	90.65	17
100	0.7/0.1	47.02	36.51	143.17	13

REFERENCES

1. BARBULESCU, A., E. BAUTU, **Alternative Models in Precipitation Analysis,** An. St. Univ. Ovidius Constanta, Ser. Mat., ISSN 1224-1784**,** 17(3), 2009, pp. 45-68.

2. EL-SEHIEMY, R. A., M. A. EL-HOSSEINI, A. E. HASSANIEN, **Multiobjective Real-Coded Genetic Algorithm for Economic/Environmental Dispatch Problem**, Studies in Informatics and Control, ISSN 1220-1766, vol. 22(2), 2014, pp. 113-122.

3. GHEITH, M. S., A. B. EL-TAWIL, N. A. HARRAZ, **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, 2013, pp. 955-964.

4. HAUPT, R. L., S. E. HAUPT, **Practical Genetic Algorithms**, John Wiley & Sons, Inc. New York, NY, USA, 1998.

5. HOMAYOUNI, S. M., S. H. TANG, O. MOTLAGH, **A Genetic Algorithm for Optimization of Integrated Scheduling of Cranes, Vehicles, and Storage Platforms at Automated Container Terminals**, Journal of Computational and Applied Mathematics vol. 270, 2014, pp. 545-556.

6. KAMMARTI, R., I. AYACHI, M. KSOURI, P. BORNE, **Evolutionary Approach for the Containers Bin-Packing Problem**, Studies in Informatics and Control, ISSN 1220-1766, vol. 18(4), 2009, pp. 315-324.

7. LAGOS, C., B. CRAWFORD, R. SOTO, J. M. RUBIO, E. CABRERA, F. PARADES, **Combining Tabu Search and Genetic Algorithms to Solve the Capacitated Multicommodity Network Flow Problem**, Studies in Informatics and Control, ISSN 1220-1766, vol. 23(3), 2014, pp. 265-276.

8. LAJJAMA, A., M. EL MEROUANI, Y. TABAA, A. MEDOURI, **A New Approach for Sequencing Loading and Unloading Operations in the Seaside Area of a Container Terminal**, International Journal of Supply and Operations Management, vol. 1(3), 2014, pp. 328-346.

9. NOROUZI, A., F. S. BABAMIR, A. H. ZAIM, **An Interactive Genetic Algorithm for Mobile Sensor Networks**, Studies in Informatics and Control, ISSN 1220-1766, vol. 22(2), 2013, pp. 213-218.

10. SALIDO, M. A., O. SAPENA, F. BARBER, **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, 22-25 September 2009, pp. 532-535.

11. SALIDO, M. A., M. RODRIGUEZ-MOLINS, F. BARBER, F., **A Decision Support System for Managing Combinatorial Problems in Container Terminals**, Knowledge-Based Systems, vol. 29, 2012, pp. 63-74.

12. SIVANANDAM, S. N., S. N. DEEPA, **Introduction to Genetic Algorithms**, Springer Berlin Heidelberg, 2008.