Solving the container pre-marshalling problem using variable length genetic algorithms

Mohamed Gheith\textsuperscript{a*}, Amr B. Eltawil\textsuperscript{a} and Nermine A. Harraz\textsuperscript{b}

\textsuperscript{a}Department of Industrial Engineering and Systems Management, Egypt–Japan University of Science and Technology (E-JUST), Alexandria, Egypt; \textsuperscript{b}Production Engineering Department, Alexandria University, Alexandria, Egypt

(Received 1 April 2014; accepted 11 March 2015)

In container terminals, the yard area consists of a set of blocks, which consists of a set of bays. Each bay consists of a set of stacks, which consists of a set of tiers. In the container pre-marshalling problem, an initial layout of a bay is converted to a final desired layout. The final layout follows the given loading schedule of this bay. This has a direct impact on the most important container terminal performance measure: the vessel loading time. The deviation between the current layout and the desired layout is expressed by the value of the mis-overlays. The objective of the pre-marshalling problem is to eliminate the mis-overlays with the minimum number of container movements. In this article, a variable chromosome length genetic algorithm was applied to solve the problem. The results of the new solution approach were compared against benchmark instances and the results were remarkably better.

Keywords: container pre-marshalling; variable chromosome length; genetic algorithm; container terminals; heuristics

1. Introduction

A container terminal comprises three areas: the quay side, the container yard and the land side. Containers arrive at the yard from either the quay side or the land side via trucks, and then they are stacked in the yard by yard cranes. After that, and according to the schedule of departure times of the containers, the containers are moved via trucks to either the land side if they are to be delivered to a customer or the quay side if they are to be loaded on a vessel. The schedule of departures determines the loading priorities of the containers on the vessels. The containers are classified into categories according to their priorities: the lower the category number, the higher the priority of this category.

One of the most important performance indicators in a container terminal is the berthing time of vessels (Stahlbock and Voss 2008). It is desired to minimize the berthing time of a vessel. If the containers are not stacked in the yard in a configuration that facilitates the work of both the yard crane and the quay crane, the berthing time will be negatively affected (Gheith, Eltawil, and Harraz 2012a). This is why the containers must be reordered to improve their retrieval rate when it is time to pick them up from the bay and load them on the vessels. This reordering is

\*Corresponding author. Email: mohamed.gheith@ejust.edu.eg

© 2015 Taylor & Francis
As shown in Figure 1, a container yard consists of blocks; each block consists of a set of bays, each bay consists of a set of stacks, and each stack consists of a set of tiers. When containers arrive at the yard, they are piled up above each other waiting for further operations. To collect a container that is not located on the top of the bay, additional movements have to be made to change the location of the containers that are above the desired one, to make it accessible. These additional movements are called ‘reshuffles’ and they consume a lot of time, leading to reduced work efficiency of the yard crane and quay crane, especially during vessel loading operations.

In the container pre-marshalling problem, an initial layout of a bay is converted into a desired final layout in which the containers are stacked above each other with higher priority containers located at the top of the stack, while minimizing the total number of movements required to achieve the final layout (Gheith, Eltawil, and Harraz 2012a). This will minimize or eliminate future additional reshuffles by the yard crane.

As shown in Figure 2, the pre-marshalling problem is restricted to a single bay, a specific number of tiers (maximum height) and a specific number of stacks (maximum width). These two dimensions are determined by the characteristics of the yard crane that is used in operations. The most used pieces of equipment in stacking and retrieving the stacked containers are rail-mounted gantry cranes, which can only access the containers from the top.

The concept of mis-overlay was introduced by Lee and Hsu (2007). A mis-overlay exists when two or more containers are stacked above each other in the wrong priority order. The degree of mis-overlay is measured by the mis-overlay index, which is the maximum depth of mis-overlays in a stack, and it is calculated for each stack as shown in Figure 3. In the first two stacks, all containers are stacked in the right order (e.g. no higher priority container is placed beneath a lower priority container); hence, the mis-overlay indices of the first two stacks are zero. In the
third stack, a container with higher priority (priority of one) is placed beneath a container with lower priority (the top container of the stack with priority of two); hence, the mis-overlay index of this stack is one. In the fourth stack, the bottom container has a priority higher than the priority of the container above it, so that, starting from this container to the top container of the stack, all the containers are considered to be placed in the wrong order, and the mis-overlay index is calculated by counting these containers; therefore, the mis-overlay index for stack number four is three. The objective of the pre-marshalling problem is to achieve zero mis-overlay indices for all the stacks while minimizing the number of movements required to reorder the containers.

Following this introduction, a review of related work is presented in Section 2. In Section 3, the proposed methodology for solving the container pre-marshalling problem is presented. In
Section 4, analysis of the results of the proposed approach is provided. Finally, conclusions and directions for future research are discussed in Section 5.

2. Related work

The literature discussing the container pre-marshalling problem is very limited (Caserta, Voss, and Schwarze 2011). Gheith, Eltawil, and Harraz (2012b) provide a review on the container pre-marshalling problem. Stahlbock and Voss (2008) provide an extensive review of container terminal operations and describe the container pre-marshalling problem and its constraints. Bortfeldt and Forster (2012) provide a tree search heuristic for solving the container pre-marshalling problem. The search was carried out by means of a recursive procedure.

Lee and Hsu (2007) proposed an integer programming model to solve the container pre-marshalling problem for small size instances. For large size problems, the authors developed a simple heuristic that is capable of providing a solution while minimizing the computation time. Lee and Lee (2010) proposed a three-phase heuristic based on heuristic rules to solve the container stacking problem. First, it generates an initial feasible movement sequence according to a simple rule. The second phase quickly reduces the total number of required movements in the sequence by repeatedly solving a binary integer program. In the third phase, the heuristic uses a mixed integer program to reduce the total working time that the crane needs to complete the entire sequence.

Caserta and Voss (2009) developed a heuristic-based corridor method to solve the container pre-marshalling problem. The basic idea of the corridor method is to use an exact method over a restricted portion of the solution space of a given problem to minimize the solution space of the problem. The authors used a modified dynamic programming algorithm as an exact method with the corridor method.

In this work, a genetic algorithm (GA) approach based on the variable chromosome length genetic algorithm (VCLGA) is used to solve the container pre-marshalling problem. No one has previously applied the proposed method to the container pre-marshalling problem.

3. The variable chromosome length genetic algorithm solution approach

GAs are a type of random search algorithm that try to mimic the mechanism of natural genetics. An exact algorithm (e.g. branch and bound, branch and cut) may take a long time to solve complex problems, and therefore the use of random search algorithms (e.g. GAs) has increased significantly.

Holland (1975) was the first to popularize the idea of GAs based on Darwin’s principle of survival of the fittest. The GA procedures start by chromosome representation and generation. The chromosome represents the solution to the problem. In this step, a representation for the chromosome is specified as well as the number of generations that will be generated at each iteration. After that, the crossover operators and the mutation operators are applied in order to produce the new generation aiming to enhance the generations over time (Beligiannis et al. 2009). The crossover and mutation operators are devised to be computationally simple and effective (Back et al. 2000).

The classical GA has a fixed chromosome length. This type of chromosomes is suitable for some problems such as the vehicle routing problem, within which it is required to cover a pre-specified number of stops. This number of stops does not change, and the target of solving such a problem is to obtain the best sequence to visit the stops, to minimize either the total time or the
cost. In such a case, there is no need to change the length of the chromosome as it represents the sequence that will be followed by the vehicle.

In the case considered in this article, the situation is different in that the goal of the problem is to achieve zero mis-overlays for all the stacks with the minimum number of container movements. So, the number of movements may change according to the final layout and it is not pre-specified. Therefore, another type of GA is needed; that is, the variable chromosome length genetic algorithm (VCLGA). This type includes the same procedure of the classical GA except that the chromosomes have a variable number of genes.

The classical VCLGA needed to be modified with functional subroutines to cope with the nature of the pre-marshalling problem. The modified VCLGA will be called MVCLGA. The MVCLGA approach consists of the main procedures of the VCLGA, which are the same as the classical GA, and the functional subroutines. The procedures of the classical VCLGA are: (1) chromosome representation and generation, (2) crossover, (3) mutation, (4) and chromosome selection using fitness function. In the chromosome representation and generation step, a representation for the chromosome as well as the number of generations that will be generated at each iteration are specified. The new generations are generated by the crossover operators, while mutation operators enhance the new generations.

The six main functional subroutines of the MVCLGA are: ‘Initialization’, ‘Parent generation’, ‘Cyclic movements elimination’, ‘Minimum chromosome length preservation’, ‘Feasible solution implementation’ and ‘Selection’. Beside these subroutines, additional ones exist: different mutation types subroutines and generation subroutines. In the ‘Initialization’ subroutine all the parameters of the algorithm are specified (e.g. number of generations, maximum number of generations, crossover and mutation probabilities, and the initial layout of the bay). The total mis-overlay index of the bay is also determined.

In the ‘Cyclic movements elimination’ subroutine, any redundant movements are removed from the chromosome, while in the ‘Minimum chromosome length preservation’ subroutine, a minimum length of the chromosome is kept if the length of the chromosome is less than a pre-specified length.

In the ‘Feasible solution implementation’ subroutine, the chromosome is decoded to apply the solution presented by the chromosome. In the ‘Selection’ subroutine, the best solutions are selected, aiming to enhance the next generations.

The pseudo-codes for the main subroutines are provided in the Appendix (see Supplemental data). In the next section, the procedures of the VCLGA and the main functional subroutines will be explained.

### 3.1. Chromosome representation

A chromosome represents a solution for the problem. It consists of a number of genes, and each gene represents the current status of the problem solution. In the MVCLGA, the length of the chromosome is not fixed, so it has a variable number of genes. To cope with the nature of the pre-marshalling problem, the chromosomes are represented as shown in Figure 4. In Figure 4(a) an example of a chromosome is provided: the length of the chromosome is 12 cells and each pair of cells represents a gene. Each gene represents a movement from a stack (left cell) to another stack (right cell) decoded from left to right. Chromosomes can have different lengths; therefore, another example for a shorter chromosome shown in Figure 4(b) has a length of four genes.

### 3.2. Crossover

In crossover, a pre-specified probability called the ‘crossover probability’ is defined, and in each iteration a random probability is generated called the ‘crossover generated probability’. If the
The purpose of the mutation operators is to prevent the algorithm from getting stuck in a local minimum (Braysy and Gendreau 2005). This is typically done by modifying some genes in the chromosome using mutation operators (e.g. remove gene, add gene). Different types of mutation operator exist. For the VCLGA algorithm, four types of mutation operator are used: growth mutation, shrink mutation, swap mutation and replace mutation. The first two types of mutation operator are used to change the length of the chromosomes, while the last two types are used to enhance the chromosomes while keeping their lengths. In each type of mutation operator a pre-specified probability is defined by the name of each type (e.g. growth probability, shrink probability) and a random probability is generated in each iteration defined by the name of each type (e.g. growth mutation generated probability, shrink mutation generated probability). If the ‘mutation type generated probability’ is larger than the pre-specified one, then mutation occurs, and so on.

3.3.1. Growth mutation

In growth mutation the chromosome is enlarged, as shown in Figure 6. This is done by generating a random number within the length of the chromosome to specify the place where a new gene will be added. The new gene is selected from a list that contains all possible movements between

![Figure 4. Chromosome representation in the variable chromosome length genetic algorithm (VCLGA).](image-url)
the stacks. This list is called the ‘list of all possible movements’. For example, if there are four
stacks, then the list of all possible movements will be [(1,2), (1,3), (1,4), (2,1), (2,3), (2,4), (3,1),
(3,2), (3,4), (4,1), (4,2), (4,3)].

3.3.2. **Shrink mutation**

The purpose of shrink mutation is to reduce the length of the chromosome (Figure 7). Using the
same principle as growth mutation, a random number is generated to specify the place where a
gene will be deleted.
3.3.3. Swap mutation

In swap mutation two different random numbers are generated within the length of the chromosome to specify the places of the genes that will be swapped. As shown in Figure 8, the two genes that will be swapped are gene number one and gene number four. This results in a new chromosome that has the same length as the original chromosome but with a different sequence of movements.

3.3.4. Replace mutation

In replace mutation a randomly selected gene in the chromosome will be replaced by a new gene that is selected from the list of all possible movements. This is done by generating a random number within the length of the chromosome to specify the place of the gene that will be replaced, and then the gene is replaced by the selected gene from the list of all possible movements. As in swap mutation, the new chromosome has the same length as the original one but a different movement sequence (Figure 9).
3.4. Fitness function

The fitness function is used to determine the quality of the produced solution. In this concern, the fitness function used in the presented problem is the calculation of the total number of mis-overlays. The smaller the number of mis-overlays the better the solution. The fitness function can be determined from the final layout of the bay, as each chromosome represents a solution that reflects the layout of the bay. In the pre-marshalling problem, it is desired to achieve the minimum number of mis-overlays with the minimum number of container movements. Hence, the second step in the fitness function procedure is to sort the chromosomes that have the same mis-overlay index according to their lengths, which represent the number of container movements. The result is a list of chromosomes sorted according to the mis-overlay index and chromosome length.

3.5. Modified variable chromosome length genetic algorithm

After the explanation of the main procedures of the VCLGA, an explanation of the main functional subroutines used to produce the MVCLGA will be presented.

3.5.1. ‘Initialization’ subroutine

Given the initial layout of a bay, this subroutine is responsible for generating the list of all possible movements as well as determining the total number of mis-overlays. The parameters of the algorithm are specified in this subroutine (e.g. maximum number of generations, population size, crossover probability, growth mutation probability).

3.5.2. ‘Parent generation’ subroutine

The role of this subroutine is to produce a new generation at each iteration of the algorithm. The number of chromosomes in each generation is equal to the population size. In this subroutine, two different random numbers are generated within the length of the ‘list of all possible movements’, representing the length of the first and second chromosomes (parents), respectively. After that, and according to the length of each chromosome, a number of genes are selected randomly to form each chromosome.
3.5.3. ‘Cyclic movements elimination’ subroutine

Owing to the random nature of the algorithm, redundant container movements may occur in any chromosome. Thus, this subroutine is added to the algorithm to eliminate redundant container movements. As shown in Figure 10, the original chromosome has two movements (e.g. genes number four and five) that could be compensated by only one movement (e.g. gene number four) that appear in the new chromosome.

3.5.4. ‘Minimum chromosome length preservation’ subroutine

Theoretically, the number of movements required to achieve a zero mis-overlay index for all stacks must be equal to or greater than the total number of mis-overlays. Owing to the random nature of the GA, the length of the chromosome may be reduced to a length that is less than the total number of mis-overlays, and as each chromosome represents a solution to the problem, a minimum length for each chromosome should be retained, and this length is equal to the total number of mis-overlays. If a chromosome with a length less than the total number of mis-overlay indices is generated, it is eliminated and a new chromosome is generated.

3.5.5. ‘Feasible solution implementation‘ subroutine

The purpose of this subroutine is to decode the produced chromosome into a solution, and to determine the total number of mis-overlays in the final layout of the bay. It also ensures the feasibility of the movements in the chromosome. Infeasible movements may exist in two cases, as shown in Figure 11. The movements either start from an empty stack or end at a stack that is fully occupied.
3.5.6. ‘Selection’ subroutine

This subroutine is used to calculate the fitness function of each solution, and then order these solutions in ascending order based on their lengths. A tournament selection is applied in this case to select the best solutions.

4. Analysis and results

4.1. Comparison against the Lee and Hsu benchmark instance

The GA parameter selection and the significance of these parameters were first studied by De Jong (1975). Nowadays, there are no well-established optimum values for these parameters and they differ from one problem to another (Kumar et al. 2005). In this concern, a benchmark instance was selected. This instance was introduced by Lee and Hsu (2007) and is the only instance that has been solved to optimality. The optimum solution for this instance is nine movements. The selected instance was first solved using different values of GA parameters to select the best values for the container pre-marshalling problem.

The initial layout of the selected instance is shown in Figure 12. Table 1 shows the characteristics of the instance.

Once again, the total number of mis-overlay containers represents the total number of containers that are stacked in incorrect order, so that the number of movements required to achieve a zero mis-overlay index must be equal to or greater than the total number of mis-overlays.

The instance was tested using six different scenarios, each of which has unique parameter values. Table 2 shows the different parameters of the six scenarios. All scenarios have the same population size and the same maximum number of generations. The crossover and mutation probabilities are shown in the last two columns of the table. Furthermore, the mutation probabilities for all the operators are assumed to be equal (i.e. growth mutation probability, shrink mutation
Table 1. Characteristics of the selected instance.

| Characteristic          | Value |
|-------------------------|-------|
| Number of containers    | 14    |
| Number of categories    | 3     |
| Maximum number of tiers | 4     |
| Number of stacks        | 6     |
| Occupancy rate          | 58%   |
| Total number of mis-overlays | 6  |
| Percentage of total number of mis-overlays | 43% |

Table 2. Genetic algorithm parameters values setting experiment.

| Scenario number | Population size | Maximum number of generations | Crossover probability | Mutation probability |
|-----------------|-----------------|-------------------------------|-----------------------|----------------------|
| 1               | 50              | 300                           | 0.15                  | 0.05 to 0.5, step 0.05 |
| 2               | 50              | 300                           | 0.3                   |                      |
| 3               | 50              | 300                           | 0.45                  |                      |
| 4               | 50              | 300                           | 0.6                   |                      |
| 5               | 50              | 300                           | 0.75                  |                      |
| 6               | 50              | 300                           | 0.9                   |                      |

probability, swap mutation probability, etc.). The procedures of the VCLGA and all subroutines are coded using MATLAB. Starting from 0.05 to 0.5, with a step size of 0.05, 10 runs were made for each scenario. Table 3 shows the details of these runs. The results of each run in each scenario are presented in terms of the number of movements and the calculation time.

At small values of crossover and mutation probabilities, the dispersion in the resulting chromosome length is high (Figure 13a), while at large values, the dispersion is low (Figure 13b). This is consistent with the nature of the genetic process, because at small values of crossover and mutation probabilities, there is a higher chance for the crossover and mutation processes to take place, and hence the dispersion increases. Also, the convergence to the optimum chromosome length (i.e. 18 cells or nine genes) in the first case appears in late generations, while in the second case, the convergence starts during early generations.

Figures 14–17 show the average number of movements as well as the average time, for crossover and mutation probabilities, respectively. From the results and based on minimum
Table 3. Summary of the results of the genetic algorithm parameters setting experiment.

| Scenario | Mutation probabilities | 0.05 | 0.1 | 0.15 | 0.2 | 0.25 | 0.3 | 0.35 | 0.4 | 0.45 | 0.5 |
|----------|------------------------|------|-----|------|-----|------|-----|------|-----|------|-----|
| 1        | Number of movements    | 9    | 9   | 9    | 9   | 9    | 9   | 9    | 9   | 10   |     |
|          | Time (s)               | 91   | 243 | 291  | 161 | 32   | 38  | 80   | 11  | 27   | 174 |
| 2        | Number of movements    | 9    | 9   | 9    | 10  | 9    | 9   | 9    | 10  | 9    |     |
|          | Time (s)               | 186  | 85  | 171  | 31  | 113  | 48  | 95   | 44  | 22   | 85  |
| 3        | Number of movements    | 9    | 9   | 10   | 9   | 9    | 9   | 10   | 10  | 9    |     |
|          | Time (s)               | 66   | 35  | 99   | 22  | 35   | 64  | 31   | 19  | 17   | 23  |
| 4        | Number of movements    | 10   | 10  | 10   | 9   | 10   | 9   | 12   | 10  | 9    |     |
|          | Time (s)               | 71   | 13  | 149  | 55  | 90   | 7   | 54   | 21  | 38   | 7   |
| 5        | Number of movements    | 9    | 10  | 10   | 10  | 9    | 9   | 9    | 9   | 10   |     |
|          | Time (s)               | 154  | 51  | 90   | 61  | 43   | 23  | 13   | 80  | 26   | 17  |
| 6        | Number of movements    | 10   | 9   | 10   | 9   | 10   | 9   | 10   | 9   | 9    |     |
|          | Time (s)               | 96   | 59  | 53   | 56  | 57   | 29  | 21   | 49  | 21   | 14  |

Note: MVCLGA = modified variable chromosome length genetic algorithm.

Figure 13. Genetic algorithm convergence at different values of crossover and mutation probabilities.

Figure 18 shows the final layout of the bay using a crossover probability of 0.15 and mutation probabilities for all the operators of 0.3. The total number of movements required to achieve the final layout with zero mis-overlays is nine movements, and the movements, expressed as (from stack, to stack), are: [(1,4), (3,1), (2,3), (6,3), (2,1), (5,3), (5,2), (4,2), (4,2)]. Table 4 shows the different methods used to solve the instance in the literature and its results.
Figure 14. Average number of movements using different crossover probabilities.

Figure 15. Average time using different crossover probabilities.

Figure 16. Average number of movements using different mutation probabilities.

4.2. Comparison against the Lee and Chao benchmark instance

The second instance used is the instance proposed by Lee and Chao (2009). As shown in Figure 19, the bay has 10 stacks with five tiers, and 35 containers grouped in 10 categories numbered from 0 to 9. The occupancy rate of the instance is 70%. In this instance, crossover and mutation probabilities of 0.15 and 0.3, respectively, will be used. Table 5 shows a comparison between different methods used to solve the Lee and Chao instance. Figure 20 shows the
final layout of the bay. The resultant total number of movements of the proposed MVCLGA to achieve such a layout with zero mis-overlays is 16 movements. The movements in this particular solution are: (2,5), (8,4), (7,2), (3,8), (6,5), (7,8), (10,7), (3,8), (9,7), (3,8), (9,2), (3,8), (9,3), (9,3), (10,3), (6,3)). This means that the solution obtained by the proposed MVCLGA is better than the best reported solution for the instance by three movements (Table 5).
Table 5. Comparison between different methods used to solve the Lee and Chao instance.

| Authors                     | Method                                | Class          | Best number of movements | Best calculation time (s) |
|-----------------------------|---------------------------------------|----------------|--------------------------|--------------------------|
| Lee and Chao (2009)         | Neighbourhood search heuristic        | Heuristic      | 31                       | 5                        |
| Caserta and Voss (2009)     | Corridor-based method                 | Heuristic      | 19                       | 20                       |
| Bortfeldt and Forster (2012)| Tree search heuristic                 | Heuristic      | 17                       | 1                        |
| Gheith, Eltawil, and Harraz | Variable length genetic algorithm     | Heuristic      | 16                       | 85                       |

Figure 20. Final layout of the Lee and Chao instance obtained by genetic algorithm heuristic.

4.3. Comparison against Caserta and Voss randomly generated instances

Forty randomly generated instances were developed by Caserta and Voss (2009). The parameters used in solving these instances are shown in Table 6. Table 7 shows a comparison between different methods used to solve Caserta and Voss instances. Although the upper bounds of the
Table 6. Parameters used in solving the Caserta and Voss instances.

| Parameter                        | Value  |
|----------------------------------|--------|
| Population size                  | 50     |
| Maximum number of generations    | 10,000 |
| Crossover probability            | 0.15   |
| Growth mutation probability      | 0.3    |
| Shrink mutation probability      | 0.3    |
| Swap mutation probability        | 0.3    |
| Replace mutation probability     | 0.3    |

Table 7. Comparison between different methods for the Caserta and Voss instances.

| Group number               | CV1   | CV2   | CV3   | CV4   |
|---------------------------|-------|-------|-------|-------|
| Corridor-based method     |       |       |       |       |
| Number of movements       | 21.30 | 28.27 | 46.91 | 50.14 |
| Time (s)                  | 20    | 20    | 20    | 20    |
| Tree search heuristic     |       |       |       |       |
| Number of movements       | 10.5  | 19.1  | 30.4  | 44.4  |
| Time (s)                  | < 1   | < 1   | 9.1   | 20    |
| Rule-based heuristic       |       |       |       |       |
| Number of movements       | 9.2   | 17.4  | 30.5  | 49.5  |
| Time (s)                  | 4.15 × 10^{-4} | 1.47 × 10^{-3} | 3.61 × 10^{-3} | 5.97 × 10^{-3} |
| MVCLGA                     |       |       |       |       |
| Number of movements       | 11.5  | 14.3  | 24.5  | 40.8  |
| Time (s)                  | 395   | 2736  | 630   | 584.16 |
| Upper bound improvement    | 0%    | 17.1% | 19.1% | 8.1%  |

Note: MVCLGA = modified variable chromosome length genetic algorithm.

Table 8. Comparison between different methods for the Lee and Chao instances.

| Group number               | LC1   | LC2   | LC3   | LC4   |
|---------------------------|-------|-------|-------|-------|
| Lee and Chao (2009)       |       |       |       |       |
| Number of movements       | 61    | 75.8  |       |       |
| Time (s)                  | 675   | 2106  |       |       |
| Bortfeldt and Forster (2012) |       |       |       |       |
| Number of movements       | 22.6  | 38.4  | 23.7  | 42.7  |
| Time (s)                  | 1     | 2.1   | 1.4   | 4.2   |
| Gheith, Eltawil, and Harraz |       |       |       |       |
| Number of movements       | 23.4  | 45    | 23.7  | 45    |
| Time (s)                  | 872   | 904   | 889   | 920   |

instances are improved, this reflects on the computation time compared with other methods, as it takes a long time compared with these methods.

4.4. Comparison against Lee and Chao randomly generated instances

Forty randomly generated instances were also developed by Lee and Chao. The parameters used in solving these instances are the same as in case of the Caserta and Voss instances. Table 8 shows a comparison between different methods used to solve Lee and Chao instances. Compared with Lee and Chao, the solutions obtained by the proposed method are better. Compared with Bortfeldt and Forster, the obtained solutions are worse. In one instance, the proposed method obtained the same average number of movements as the solution obtained by Bortfeldt and Forster, while in other instances it obtained, on average, worse results. This may attributed to the following
factors: (1) the randomness of the proposed method, (2) the size of the instance, (3) crossover and mutation techniques, or (4) the crossover and mutation probabilities used not being appropriate for this size of instances. All of these factors may be investigated in future work.

Bortfeldt and Forster (2012) present 32 randomly generated instances. The minimum size of the bay among these instances is $16 \times 5$. According to well-known commercial companies specializing in yard cranes, the maximum dimensions of a yard crane are $8 \times 6$. Hence, the dimensions of such instances are not applicable in real life. Therefore, such instances were not solved. The dimensions of the Lee and Chao instances are enough to check the performance of the proposed method.

5. Conclusions and future work

The container pre-marshalling problem is a very important decision problem for efficient container terminal operation. Container pre-marshalling is used to rearrange the layout of a container stack in order to reduce the time needed to reload this stack on its targeted vessel. The container pre-marshalling problem is an NP-hard problem with a single recorded closed form solution able to solve the problem to optimality only for small size instances. Larger instances are solved using heuristic methods. In this article, a new approach based on a variable chromosome length genetic algorithm (MVCLGA) was used to solve the container pre-marshalling problem. The proposed algorithm was validated by comparison to the only problem solved to optimality, and its performance was tested by comparison against several benchmark instances reported in the literature.

The new algorithm is much faster than the exact method and has very high convergence rates. This is a very important characteristic because the container pre-marshalling problem is an operational problem and the decision must be taken within an hour to match the requirements of the terminal operators.

In the future, other techniques of crossover operators (e.g., two-point crossover, uniform crossover) and mutation operators should be investigated as these may result in better solutions. The VCLGA can be used in similar problems, such as the towers of Hanoi problem, vehicle routing problems, scheduling problems and other problems that have similar characteristics.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research project is sponsored by an Egyptian Ministry of Higher Education grant and the Japanese International Cooperation Agency (JICA) in the scope of the Egypt Japan University of Science and Technology (E-JUST).

Supplemental data

Supplemental data for this article can be accessed at http://dx.doi.org/10.1080/0305215X.2015.1031661

Note

1. These instances are downloadable from ftp://ftp.fernuni-hagen.de/pub/fachp/wiwi/winf/forschng/CPMP/
References

Back, T., D. Fogel, and Z. Michalewicz. 2000. *Evolutionary Computation 1: Basic Algorithms and Operators*. Bristol: Institute of Physics.

Beligiannis, G. N., C. Moschopoulos, and S. D. Likothanassis. 2009. “A genetic algorithm approach to school timetabling.” *Journal of the Operational Research Society* 60: 23–42.

Bortfeldt, A., and F. Forster. 2012. “A Tree Search Procedure for the Container Pre-Marshalling Problem.” *European Journal of Operational Research* 217: 531–540.

Braysy, O., and M. Gendreau. 2005. “Vehicle Routing Problem with Time Windows, Part II: Metaheuristics.” *Transportation Science* 39: 119–139.

Caserta, M., and S. Voss. 2009. “A Corridor Method-Based Algorithm for the Pre-Marshalling Problem.” *Applications of Evolutionary Computing* 5484: 788–797.

Caserta, M., S. Voss, and M. Sniedovich. 2011. “Applying the Corridor Method to aBlocs Relocation Problem.” *OR Spectrum* 33: 915–929.

De Jong, K. A. 1975. “Analysis of the Behaviour of the Class of Genetic Adaptive System.” Ph.D. Thesis, University of Michigan, Ann Arbor, MI.

Gheith, M. S., A. B. Eltawil, and N. A. Harraz. 2012a. “A Proposed Heuristic for Solving the Container Pre-Marshalling Problem.” Proceedings of IEEE 19th International Conference of Industrial Engineering and Engineering Management, Springer Berlin Heidelberg, 955–964.

Gheith, M. S., A. B. Eltawil, and N. A. Harraz. 2012b. “Container Pre-Marshalling Problem: A Review and Solution Framework.” Proceedings of the IEEE International Conference of Industrial Engineering and Engineering Management, Hong Kong, 283–287.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

Hung, S., and T. Lin. 2012. “Heuristic Algorithm for the Container Pre-Marshalling Problem.” *Computers & Industrial Engineering* 62: 13–20.

Kumar, A. Prakash, M. K. Tiwari, R. Shankar, and A. Baveja. 2005. “Solving Machine-Loading Problem of AFlexible Manufacturing System with Constraint-Based Genetic Algorithm.” *European Journal of Operational Research* 175: 1043–1069.

Lee, Y., and S. Chao. 2009. “A Neighbourhood Search Heuristic for the Pre-Marshalling Export Containers.” *European Journal of Operational Research* 196: 468–475.

Lee, Y., and N. Hsu. 2007. “An Optimization Model for the Container Pre-Marshalling Problem.” *Computer & Operations Research* 34: 3255–3313.

Lee, Y., and Y. Lee. 2010. “A Heuristic for Retrieving Containers from aYard.” *Computer & Operations Research* 37: 1139–1147.

Stahlbock, R., and S. Voss. 2008. “Operations Research at Container Terminals: A Literature Update.” *OR Spectrum* 30: 1–52.