Automated Design of Heuristics for the Container Relocation Problem Using Genetic Programming

Marko Đurasević*

Faculty of Electrical Engineering, University of Zagreb, Zagreb 10000, Croatia

Mateja Đumić

[a]*Department of Mathematics, J. J. Strossmayer University of Osijek, Osijek 31000, Croatia,*

Abstract

The container relocation problem is a challenging combinatorial optimisation problem tasked with finding a sequence of container relocations required to retrieve all containers by a given order. Due to the complexity of this problem, heuristic methods are often applied to obtain acceptable solutions in a small amount of time. These include relocation rules (RRs) that determine the relocation moves that need to be performed to efficiently retrieve the next container based on certain yard properties. Such rules are often designed manually by domain experts, which is a time-consuming and challenging task. This paper investigates the application of genetic programming (GP) to design effective RRs automatically. Experimental results show that RRs evolved by GP outperform several existing manually designed RRs. Additional analyses of the proposed approach demonstrate that the evolved rules generalise well across a wide range of unseen problems and that their performance can be further enhanced. Therefore, the proposed method presents a viable alternative to existing manually designed RRs and opens a new research direction in the area of container relocation problems.

Keywords: Container relocation problem, genetic programming,

*Corresponding author
Email address: marko.durasevic@fer.hr (Marko Đurasević)

hyper-heuristics, relocation rules

1. Introduction

The container relocation problem (CRP) is an important combinatorial optimisation problem that appears in warehouse and yard management. Nowadays, this problem is gaining more importance since most international trade is carried out by the international shipping industry [1].

Usually, containers are placed in a stacking area while waiting for loading. Because of the limited capacity of the stacking area, containers are placed side by side or on top of each other. By storing containers in such a way, blocks are formed. Each block consists of a number of stacks (width), a number of tiers (height), and a number of bays (length).

Loading of containers is done in a predetermined order. If the container that needs to be retrieved next is not on the top of its stack, all containers above it need to be relocated. The relocation of containers is, in most cases, inevitable because of the incomplete or unavailable information. In [2] authors give an estimation that 30-40% of the outbound containers at European terminals do not have correct information about the ship or the destination port. The situation is even worse for inbound containers, and only 10-15% of container shipment is known in advance [2].

CRP deals with the relocation and retrieval of containers at the same time. In literature, a variety of approaches used for solving different variants of CRP can be found. CRP variants differ in restrictions on moves, retrieval priorities of blocks (whether they are distinct or not), whether all blocks need to be retrieved, how many blocks can be moved at a time, and whether the stacks are ordered in a single or multiple bays. A formal problem classification can be found in [3].

The single bay CRP was for the first time introduced by Sculli and Hui in 1988 [4]. Avriel et al. [5] showed that it is an NP-complete problem. In [6] a heuristic based on the expected number of additional relocations (ENAR), which was used in a branch-and-bound algorithm to search for the optimal solution,

was introduced. A beam search algorithm and three relocation rules (RRs): lowest position (TLP), reshuffle index (RI), and reshuffle index with look-ahead (RIL) were proposed in [7]. In [8] a GRASP-based algorithm that found new bounds for many problem instances was proposed. Lee and Lee [9] developed a three-phase heuristic for multibay CRP, which was the first approach in the literature applied for multiple bays. In this problem, besides the number of relocations, it is essential to reduce the crane operation time, which can be significant when moving from one bay to another.

A corridor method combined with dynamic programming was used in [10], while in [11], two mathematical models and the Min-Max rule were introduced. In [12] a tree search procedure for multibay CRP was introduced. This approach defines well placed and bad placed containers, which can help calculate a lower bound and choose moves. Iterative Deepening A* algorithm was used in [13] for solving CRP. In this paper, three lower bounds and four heuristics were used in the nodes of a tree. A new heuristic which, besides the container that needs to move, takes into account the properties of the container that needs to move next is introduced in [14]. In [15] authors propose the heuristic based on groups to solve CRP with multiple bays. In this paper, cranes that can move one or more containers at a time were observed.

Authors in [16] used the A* algorithm for solving CRP exactly and approximately. They also study the average-case asymptotic behaviour of the CRP when the number of stacks increases. CRP with time window was studied in [17] in which an abstraction heuristic is developed to improve the tree-search approach used for solving it. In most recent research, four rules combined with a genetic algorithm were used to determine the best sequence of container retrievals [18], and GRASP for multibay CRP is introduced in [19]. Also, the dynamic stacking problem with uncertainties was studied in [20]. In this paper, the problem was solved using two approaches: hand-crafted rules and model-based. The second one solves the problem by solving a static model of a specific planning horizon and seems to be better of these two approaches.

From the previous overview, it is evident that in most research, heuristic

methods have been applied for solving the CRP, which ranged from simple heuristics to more complex metaheuristics. Metaheuristics have the benefit that they achieve better results than simple heuristic rules at the expense of a longer execution time, especially as the problem size increases. On the other hand, simple RRs obtain good solutions in an almost negligible time since they do not traverse the search space in the quest for good solutions but rather construct a good solution iteratively using a certain strategy. These strategies are manually defined and include domain knowledge defined by experts. However, effective RRs are challenging to design and require good expert knowledge about the problem. This provides motivation for investigating the possibility of automatic design of new RRs for the CRP.

Genetic programming (GP) is an evolutionary computation method that has achieved human-competitive results for many problems [21] and is commonly used for automatic heuristic generation [22]. It was successfully applied in the design of heuristics for different combinatorial optimisation problems including the travelling salesman problem [23], vehicle routing problem [24], capacitated arc routing problem [25], and similar. Automated design of heuristics has probably achieved the largest application in scheduling [26, 27], where GP was used to design heuristics for different machine environments like one-machine problem [28], job shop scheduling [29, 30, 31], unrelated machine environment [32, 33], and resource constrained project scheduling problem [34, 35]. In recent years, various research directions in the automated design of heuristics were investigated like ensemble learning [36, 37, 38, 39, 40], multi-objective optimisation [41, 42], surrogate models [43, 44], multitask GP [45], solution construction strategies [46], feature selection [47], and many others.

In this study, GP is applied for constructing simple RRs, which iteratively determine how the containers should be relocated. For that purpose, a set of terminal nodes for the problem as well as different solution construction procedures are proposed. An extensive experimental analysis demonstrates that automatically generated RRs achieve a better performance than existing rules on an extensive benchmark dataset. Furthermore, a detailed analysis of different

aspects of the proposed approach is performed to understand it better. The obtained results demonstrate that GP can effectively be applied to the considered problem, and thus opens further avenues in the research of CRP.

In summary, the main contributions of this work are:

- development of a GP framework for the automated development of priority functions used in relocation rules,
- modification of the restricted relocation scheme and three modifications of the unrestricted relocation scheme,
- comparison of results for two optimisation criteria obtained by relocation rules evolved with GP and 7 different manually designed relocation rules from the literature,
- detailed analysis of the approach in terms of algorithm convergence, the developed priority functions and their sizes, and the terminal set used in the development of the priority functions.

The remaining part of this paper is organised as follows. In Section 2, the problem description is given. Section 3 provides a short description of the genetic programming algorithm. Section 4 describes the automated design of RRs. The experimental setup is given in Section 5, while experimental results are presented in Section 6. Section 7 provides a further analyses of the results. The discussion about the most important design choices in the automatic design of relocation rules is provided in Section 8. Finally, Section 9 concludes the paper and outlines future research directions.

2. Container relocation problem

2.1. Problem description

In this paper, the single bay CRP will be considered. The bay consists of S stacks with H tiers. The height of each stack s is denoted with $h(s)$, and it has to be smaller than or equal to H. The assumptions are that all containers

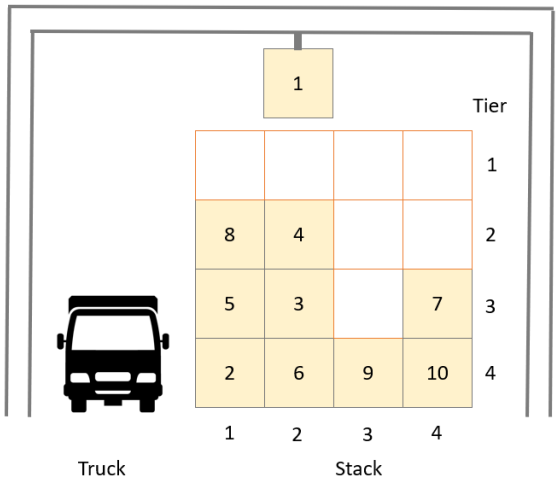


Figure 1: Example of a container bay

are of the same size, the gantry crane can move only one container at a time, all containers must be retrieved, and each container has a unique priority.

As previously mentioned, the CRP consists of two types of operations: relocation and retrieval. Relocation is the operation of moving the container from the top of one stack to another one. The container can be relocated to a stack only if the stack height is smaller than H. Retrieval is the operation of picking a container from the top of the stack and moving it on the truck used for loading. The truck to which the container is retrieved is expected to be located at the beginning of the bay, at position 0. The gantry crane, which is located in the bay, performs the relocation and retrieval operations. An example of a container bay can be seen in Figure 1.

Each container is given an ID that determines the order in which they need to be retrieved. The container with the smallest ID needs to be retrieved first and is called the *target container*. If the target container is not on the top of the stack, relocations needs to be performed. The stack from which the container is moved is called the *origin stack*, while the stack to which it is moved is called the *destination stack*.

In the example given with Figure 1, the container with ID 1 was on the top

of the stack, and it is in the process of retrieval, while the container with ID 2, which needs to be retrieved next, is blocked. Containers with IDs 8 and 5 need to be relocated to retrieve the container with ID 2. If one of these containers is relocated to stack two, it will become full, and further containers cannot be relocated to it.

Feasible solutions of the CRP are all sequences of relocations and retrievals that ensure that the crane can retrieve all containers in a predetermined order. The goal is to find the one that minimises the given objective. Usually, the number of relocations or total crane operation time are used as objectives.

When using the crane operation time as an objective, it is important to consider all parts of relocating and retrieving containers. For each movement, the time needed for the crane trolley to come to the container, pick it up and move to the destination stack needs to be calculated. The speed of moving the crane trolley per container width (moving to the adjacent stack) is 1.2 s, while the total pick-up and place-down time for a container is equal to 30 s [19]. The time invested in placing the crane to the origin stack and moving the container from the origin to the destination stack will be $1.2 \times (\textit{number of passed stacks})$. Consequently, the time for relocating the container can be calculated as: $1.2 \times |\textit{crane location} - \textit{origin stack}| + 1.2 \times |\textit{origin stack} - \textit{destination stack}| +$ *pick-up and place-down time*. The total crane operation time is the sum of times needed for each movement.

In the literature, two types of CRPs are distinguished: the restricted CRP and the unrestricted CRP. In the restricted CRP, only containers that are above the target container can be relocated, while in the unrestricted CRP, it is possible to relocate containers from all stacks. In this study, both variants are considered. Using the classification of [3], the problems considered in this study can be denoted as res|dis|com|ind and unr|dis|com|ind, which means that either restricted or unrestricted moves are considered, with distinct priorities, complete retrieval, and individual container moves.

2.2. Mathematical model

To define the restricted version of CRP more formally, the following variables are introduced [8]:

$$b_{ijnt} = \begin{cases} 1, & \text{if container } n \text{ is at position } (i,j) \text{ at the beginning of period } t \\ 0, & \text{otherwise,} \end{cases}$$

$$x_{ijklnt} = \begin{cases} 1, & \text{if container } n \text{ is relocated from position } (i,j) \text{ to position } (k,l) \text{ in period } t \\ 0, & \text{otherwise,} \end{cases}$$

$$y_{ijnt} = \begin{cases} 1, & \text{if container } n \text{ is retrieved from position } (i,j) \text{ in period } t \\ 0, & \text{otherwise,} \end{cases}$$

$$v_{nt} = \begin{cases} 1, & \text{if } n \geq t, \\ 0, & \text{otherwise.} \end{cases}$$

The variables b_{ijnt} define the feasible configuration, the variables x_{ijklnt} and y_{ijnt} define the feasible movements or relocations and retrievals, and the variables v_{nt} indicate whether a container n is in the stacking area or has already been retrieved from storage in period t. It is important to point out that the period t represents all relocation's moves, from the retrieval of a container with $n = t - 1$ to the retrieval of a container with $n = t$, and consequently may consist of more than one movement.

The formal definition can now be given as follows [11, 8]:

$$\min \sum_{i=1}^{S} \sum_{j=1}^{H} \sum_{k=1}^{S} \sum_{l=1}^{H} \sum_{n=1}^{N} \sum_{t=1}^{T} x_{ijklnt} \tag{1}$$

$$\sum_{i=1}^{S} \sum_{j=1}^{H} b_{ijnt} + v_{nt} = 1, \ n = 1, \ldots N, \, t = 1, \ldots, T \tag{2}$$

$$\sum_{n=1}^{N} b_{ijnt} \leq 1, \ i = 1, \ldots S, \, j = 1, \ldots H, \, t = 1, \ldots T \tag{3}$$

$$\sum_{n=1}^{N} b_{ijnt} \geq \sum_{n=1}^{N} b_{ij+1nt},\ i = 1, \ldots S,\ j = 1, \ldots H-1,\ t = 1, \ldots T \tag{4}$$

$$\begin{aligned} b_{ijnt} = \quad & b_{ijnt-1} + \sum_{k=1}^{S}\sum_{l=1}^{H} x_{klijnt-1} - \sum_{k=1}^{S}\sum_{l=1}^{H} x_{ijklnt-1} - y_{ijnt-1} \\ & i = 1, \ldots S,\ j = 1, \ldots H,\ n = 1, \ldots, N,\ t = 2, \ldots T \end{aligned} \tag{5}$$

$$v_{nt} = \sum_{i=1}^{S}\sum_{j=1}^{H}\sum_{t'=1}^{t-1} y_{ijnt'},\ n = 1, \ldots, N,\ t = 2, \ldots T \tag{6}$$

$$\begin{aligned} M\left(1 - \sum_{n=1}^{N} x_{ijklnt}\right) \geq & \sum_{n=1}^{N}\sum_{j'=j+1}^{H}\sum_{l'=l+1}^{H} x_{ij'kl'nt} \\ & i, k = 1, \ldots, S,\ j, l = 1, \ldots H-1,\ t = 1, \ldots, T \end{aligned} \tag{7}$$

$$\begin{aligned} M\left(1 - \sum_{j=1}^{H} b_{ijtt}\right) \geq & \sum_{j=1}^{H}\sum_{k=1}^{S}\sum_{l=1}^{H}\sum_{n=1}^{N}\sum_{i' \neq i}^{S} x_{i'jklnt} \\ & i = 1, \ldots, S,\ t = 1, \ldots, T \end{aligned} \tag{8}$$

$$x_{ijklnt} = 0,\ i = 1, \ldots, S,\ j = 1, \ldots, H,\ n = 1, \ldots, N\, t = 1, \ldots, T \tag{9}$$

$$\begin{aligned} \sum_{k=1}^{S}\sum_{l=1}^{H}\sum_{n=1}^{N} x_{ij+1klnt} - \sum_{n=1}^{N} b_{ij+1nt} + 1 \geq & \sum_{k=1}^{S}\sum_{l=1}^{H}\sum_{n=1}^{N} x_{ijklnt} + \sum_{n=1}^{N} y_{ijnt} \\ & i = 1, \ldots, S,\ j = 1, \ldots, H-1,\ t = 1, \ldots, T \end{aligned} \tag{10}$$

The objective of the problem is to minimize the total number of crane relocations throughout the entire scheduling period. The constraint 2 ensures that in each time period, each container is in the stack or has been retrieved, while constraint 3 ensures that at most one container can be in each slot (i, j) in any given time period. The condition 4 specifies that there must be no vertical gap between two containers. The condition 5 is called the *flow balancing constraint* because it maintains a feasible solution over time periods. More precisely, a new

feasible solution in time period t can be obtained from the feasible solution in time period $t-1$ by applying a valid set of moves. The relation between the variable v_{nt} and the variables y_{ijnt} is given by the constraint 6. Constraints 7 and 8 are constraints arising from a restricted scheme in which only containers above the target container can be moved (constraint 7) and only containers from the stack in which the target container is located can be moved (constraint 8). Without these constraints the formal model would define the unrestricted variant of the CRP. In these constraints M is the number of containers above the target container. Moving to the same stack is not possible, which is given by constraint 9, while constraint 10 ensures that a container can be moved only if the container above it is moved.

2.3. Notation

S	number of stacks in yard
H	number of tiers, maximum height of stacks
$h(s)$	the height of stack s
N	total number of containers

3. Genetic programming

GP is an evolutionary algorithm that works similarly to a genetic algorithm (GA) [48]. The outline of GP is given in Algorithm 1. At the beginning of the algorithm's execution, an initial population of individuals is randomly generated. To determine how well each individual performs, a numerical value, called *fitness*, is calculated for each individual using the *fitness function*. The fitness function evaluates each individual on a set of problems and assigns a value to the individual based on its performance on the problem set. Based on the fitness, two better individuals are randomly selected from the population and used by the crossover operator. The crossover operator combines parts of the two individuals to create a new, hopefully better child individual. The *mutation* operator is then applied on the child individual with a certain probability to introduce random changes in it. With mutation it is possible to escape from local

optima and introduce genetic material that might not exist in the population. The child individual then replaces a random individual from the population, usually selected among the worse individuals. The entire procedure is repeated until a certain termination criterion is satisfied, like the maximum number of fitness function evaluations, algorithm iterations, or execution time.

Algorithm 1: Standard steady state GP algorithm

```
1 Initialise the population P and evaluate all individuals in it;
2 while termination criterion is not met do
3     Randomly select 2 individuals from the population;
4     Perform the crossover operator on the selected parents and create a
      new individual;
5     Perform the mutation operator on the new individual with a certain
      probability;
6     Select an individual to be replaced with the newly created
      individual;
7 end
```

The previously described evolutionary procedure is the same for both the GA and GP. However, the main difference comes from the representation that these two methods use. Whereas individuals in GAs can be represented as permutations or lists of floating point or integer numbers, the individuals in GP are represented using expression trees. Figure 2 shows an example of such an expression tree that encodes the expression $x - 3 + \frac{x^2}{2}$. The expression tree consists of two types of nodes, *function* and *terminal* nodes. Function nodes can only appear as inner nodes of the tree and they represent various mathematical (summation, multiplication), logical (and, or, not), or other kinds of operators. On the other hand, terminals represent either input variables (x) or constants (2 or 3) that appear only in leaves of the tree. Input variables usually represent some domain specific information about the problem that is available to GP for constructing expressions. The selection of both sets is quite important as

selecting too many nodes results in a huge search space, whereas using small function and terminal sets can lead to myopic expressions. Unlike other solution representations, the tree representation adds an additional layer of flexibility in allowing expressions of different sizes to be constructed, meaning that not all individuals will have the same size. However, the maximum size of the expression is usually controlled by a certain parameter, like the maximum allowed tree depth. As in GAs, the individuals of the initial population are randomly generated. However, to ensure more diversity on the initial population, several population and individual initialisation strategies were proposed, among which the ramped-half-and-half is the most commonly used [48].

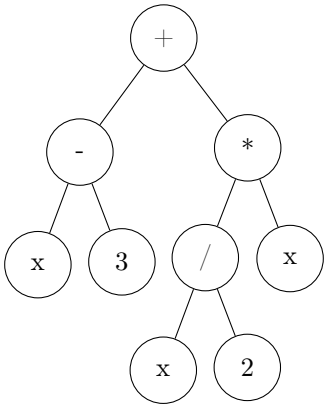


Figure 2: Tree representation of an individual in GP

As a consequence of using a different representation, GP also applies genetic operators (crossover and mutation) that are adapted to such a representation. Figure 3 outlines one possible crossover operator, denoted as *subtree* crossover. This operator randomly selects two nodes in the parent trees (denoted in red) and replaces the subtree rooted at the selected node in the first parent with the subtree rooted at the selected node in the second parent. In that way the genetic material of both individuals is combined to create a better individual. Figure 4 outlines an example of the mutation operator, denoted as the subtree mutation. In this operator a node is selected randomly (denoted in red) and the tree rooted at that node is replaced with a randomly generated subtree. In

that way new genetic material is introduced in the individual, which enhances its diversity. In both operators is required to ensure that after the application of the operators the individual still satisfies constraints like the maximum depth of the individual.

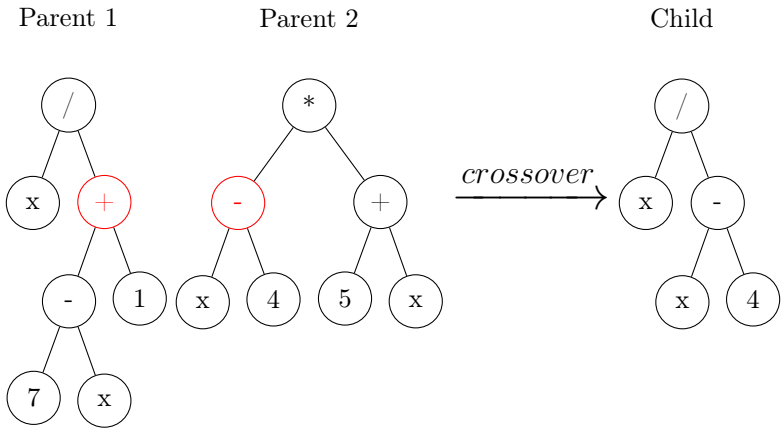


Figure 3: Subtree crossover

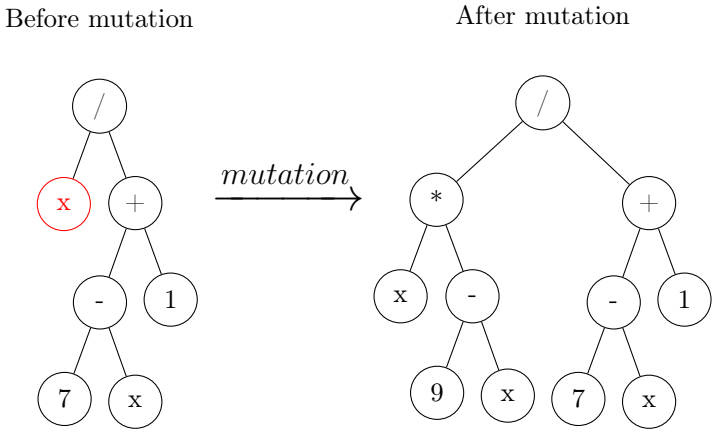


Figure 4: Subtree mutation

Since GP uses a different representation than GAs, it has been used mainly for problems that cannot be solved with GAs, such as symbolic regression [49], classification [50], or feature selection [51]. In particular, GP has become one of the most popular hyperheuristic methods [22], which means that it is used to

develop new heuristics for various combinatorial optimisation problems. This allows GP to evolve a heuristic that can be used to solve multiple problem instances, as opposed to GAs that must be run from the beginning to solve each problem under consideration. Therefore, GP is also used in this study to develop new heuristics for the CRP.

4. Design of relocation rules

RRs can be divided into two components: a relocation scheme (RS) that defines the overall strategy of how containers will be moved and which stacks are eligible for selection, and a priority function (PF) used to rank the available stacks. The RS uses the PF to rank all the stacks and selects the best one to which the current container will be relocated. Since the RS has to consider some constraints and is usually straightforward, it is defined manually. However, the PF which performs the ranking can be quite complex, and therefore it is generated using GP.

The RS can be defined in different ways. However, all schemes function similarly. If the target container is on the top of its stack, it can immediately be removed from the yard. Otherwise, if other containers are on top of the target container, they need to be relocated to other stacks so that the target container is on top. Several relocation schemes are proposed to test how different decisions can affect the performance of RRs. However, all of them can be categorised as restricted or unrestricted versions, depending on which kind of relocation moves are allowed.

Algorithm 2 outlines the restricted RS variant, which will be denoted as RE. In this scheme, the origin stack is the stack where the target container is located, whereas the destination is any other stack that is not full. For each other stack, the priority is calculated using the PF, and the container on top of the origin stack is relocated to the stack with the lowest priority value. After a certain number of relocations, the target container will be on top and can be retrieved. The entire procedure is repeated until the yard is empty. Based

on this scheme, an alternative one denoted as REN will also be used. When calculating the priorities for the stack, this scheme skips the stack with the next target container to prevent a possible relocation of a container to that stack.

Algorithm 2: Restricted relocation scheme

```
1  while container yard not empty do
2      C = GetTargetContainer();
3      S = GetStack(C);
4      while container C is not on top of stack S do
5          foreach stack st, st != S and st not full do
6              π_st = CalculatePriority(st);
7          end
8          Relocate top container from S to the stack with min(π_st);
9      end
10     Retrieve container C;
11 end
```

Algorithm 3 shows the basic unrestricted scheme denoted as UN. The minC function returns the lowest container ID on a stack. The main difference between the UN and RE schemes is that after the destination stack is determined, it is checked whether the stack contains containers with an ID that is smaller than the ID of the container that will be relocated to that stack. If such a container exists, the top container will be relocated again at some point in the future. Therefore, this scheme tries to relocate these containers from the destination stack to stacks where all containers have a larger ID, if they exist. When all these containers have been relocated, or no such stack exists, the procedure proceeds as the RE scheme.

Except for this basic unrestricted scheme, three alternative schemes were also defined. The UNC variant works similarly as the RE scheme; however, instead of only calculating the priority for the destination stack, it calculates the priority for every two pairs of stacks, in which the first one represents the

Algorithm 3: Unrestricted relocation scheme

```
1  while container yard not empty do
2      C = GetNextContainer();
3      S = GetStack(C);
4      while container C is not on top of stack S do
5          foreach stack st, st != S and st not full do
6              π_st = CalculatePriority(st);
7          end
8          D = stack with min(π_st);
9          DC = top container ID of stack D;
10         while stack S1 exists such that minC(S1) > DC do
11             Relocate container DC to S1;
12             DC = top container ID of stack D;
13         end
14         Relocate top container from S to D;
15     end
16     Retrieve container C;
17 end
```

origin stack and the second one the destination stack. The pair with the lowest priority value is selected, and the top container from the origin stack is relocated to the destination stack. Since it is possible that such a strategy could lead to infinite relocations, this procedure is done a certain number of times in each iteration, after which the scheme acts as the RE scheme in the sense that it only relocates containers from the stack where the target container is located. The UNC2 scheme functions similarly, except that it used two expressions, one to select the destination stack and the other one to select the origin stack of the relocation. Finally, the UNP scheme works in the same way as the UN; however, it evolves an additional expression used to determine the destination of the unrestricted moves, i.e., to which stacks the containers from the destination

stack will be relocated.

Although rarely, it is possible that two stacks obtain the same priority value. In this case, the tie can be broken in different ways, but for simplicity in the proposed RSs the first stack for which the best priority value was obtained is selected. Since stacks are always evaluated in the same order, this ensures that the RR is completely deterministic, i.e., given the same system conditions it will always select the same stack and produce the same solution.

As previously outlined, the priority function is evolved by GP. For that purpose, the primitive set needs to be specified. The function set consists of summation, subtraction, multiplication, and protected division (returns 1 if the divisor is close to 0). Although other functions were tested (minimum, maximum, negation, and if), their inclusion did not lead to any improvements in the results, and as such, were not used. The terminal nodes that were used are denoted in Table 1. The proposed set of terminals includes different simple properties (stack height, number of empty positions, ID of the current container) but also more complicated nodes (number of containers with a lower ID than the current container, average of container IDs on stack). By using the previously outlined function and terminal nodes, GP randomly constructs the population of individuals in which solutions (PFs) are represented as expressions (similar to the expression in Figure 2), upon which different genetic operators are applied until the given termination criterion. The evolved PFs are then used as a standard mathematical expression by the RSs to determine the best action at each decision point.

To analyse the time complexity of the proposed approach it is necessary to determine the complexity of both parts, the PF and the RS. Since the PF is a mathematical expression that consists of simple arithmetic operators, it can be calculated quite efficiently. Therefore, we can assume that the calculation of the PF takes constant time, resulting in $\mathcal{O}(1)$ complexity. For each of the N containers that needs to be retrieved, the RS first needs to relocate any containers that are located above it. In the worst case the container that needs to be retrieved is located at the bottom of the stack with all other containers

Table 1: List of the applied terminal nodes

Name	Description
SH	Stack height
EMP	Number of empty places in stack
CUR	ID of the current container that will be relocated
DUR	Time required to transfer the container to the desired stack
RI	Number of containers with a smaller ID on the stack than the ID of selected container
MIN	The smallest container ID of the selected stack
AVG	Average ID of containers in a stack
REM	Remaining containers until the container which needs to be removed is on top
NEXT	Returns 1 if the selected stack contains the next container that will be retrieved and 0 otherwise
DIFF	Difference between the ID of the container with the lowest ID on the stack and the one that needs to be relocated
EMPTY	Returns 1 if a stack is empty and 0 otherwise
WL	Number of containers on the stack for which no container of a larger ID is on top of them
NL	Number of containers on the stack for which a container of a larger ID is on top of them
DSM	Height of the first container in a stack with a smaller ID than that of the considered container

above it. Since the PF needs to be calculated for each container that needs to be relocated, this means that the PF would be calculated for all containers except the one that needs to be retrieved. For the first container that needs to be retrieved this would mean that $N-1$ containers need to be relocated, for the second $N-2$, and so on. We see that this sequence equals to the sum of first $N-1$ numbers and as such we know that the total number of relocations will be

approximately equal to N^2. Since each container can be relocated to any stack other than the source stack, it means that the PF needs to be calculated $S-1$ times for each container that is relocated. This results in the total complexity of $\mathcal{O}(N^2S)$. Since usually $N >> S$, the complexity of the RS can be reduced to $\mathcal{O}(N^2)$. Therefore, we see that the time complexity, in the worst case, depends quadratically on the number of containers. This complexity is tied with the execution of the RS for one problem and is the complexity of interest when applying the RS for new problems. However, the complexity of generating a new PF is significantly larger, since during the evolution many PFs need to be evaluated and they are evaluated on a larger set of problems. As such, the complexity of the learning algorithm (GP) is several orders of magnitude higher. However, since GP is executed offline, prior to the application of the RR, its complexity does not influence the execution time of the RR, and the RRs can be applied in real time. Regarding space complexity, all RRs have a complexity of $\mathcal{O}(1)$, since at each point in time they only need to store the best priority value obtained until now and the stack to which it is associated. Thus, the space complexity does not depend on the size of the problem that is being solved.

5. Experimental setup

The Caserta [10] and Zhu [13] datasets are used to test the proposed method. In the Caserta dataset, the number of stacks and containers per stack can be between 3 and 10. All stacks have the same initial height in all problem instances. The maximum allowed height of a stack is $h+2$, where h represents the stack height at the start. The Zhu dataset contains problems with 6 to 10 stacks and between 15 and 69 containers in the bay. The difference in comparison to the Caserta dataset is that the maximum stack height is specified for each instance in this dataset. Usually, the problem instances in the Zhu dataset have a smaller maximum height and consequently restrict the possible relocation moves at the start. The Caserta dataset contains 840 instances, whereas the Zhu dataset contains 125000 instances.

In the experiments, three independent datasets were used, train, validation, and test. The train set was used to generate the expressions for the RRs, the validation for parameter tuning, and the test set to determine the final quality of the evolved rules. For the test set, the original Caserta and Zhu sets are used with all instances. The train and validation sets are generated randomly using the procedures described in [10, 13]. For the Caserta dataset, those sets contained the same number of instances as the original (840 instances). On the other hand, for the Zhu instances, these sets contained 1000 problems as matching the size of the original set would significantly increase the training time.

All GP parameters were optimised in preliminary experiments. After these experiments, the population size was set to 1000 individuals, the mutation probability to 0.3 for the restricted and 0.1 for the unrestricted schemes, the tree depth to 5, and the termination criterion to 50000 function evaluations. GP used a tree structure to represent the PFs it evolves, similar as described in Section 3. The initial population is constructed using the standard ramped-half-and-half procedure [48]. Regarding the genetic operators, for crossover the subtree, uniform, context preserving, size fair, and one point operators were used, whereas for mutation the subtree, hoist, node complement, node replacement, permutation, and shrink mutation were used [48]. Since several genetic operators are defined, in each iteration the algorithm randomly selects, with an equal probability, an operator with which it will perform the crossover and mutation of individuals. When GP had to evolve two PFs, the GP individual consisted of 2 expressions on which genetic operators were executed independently from each other.

Due to the stochastic nature of GP, each experiment was run 30 times to obtain statistically significant results. Each time the method is run, the best individual (expression) on the training set obtained during the execution of the algorithm is stored. Therefore, after training, a single expression is obtained for each run of the algorithm, resulting in 30 expressions. Each expression is then evaluated on all instances in the test set and the minimum, median and maxi-

mum values of these 30 executions are reported in the results in the tables. Two optimisation criteria are considered, namely the number of relocations and the crane execution time. The Mann-Whitney test is used to compare whether there is a statistically significant difference between the results. The non-parametric test was used because normality of all data could not be confirmed with the Shapiro-Wilk test. A significance level of $\alpha = 0.05$ was used for all tests.

The proposed method is compared with seven manually designed RRs from the literature. These include rules that only perform restricted moves, namely TLP [9], RI [9], Min-Max [11], PR3 [13], PR4 [13], as well as rules that use unrestricted moves, such as PU1 and PU2 [13]. To enable comparison, these RRs were implemented in our framework according to the above literature.

The problem instances and the source code of the methods can be obtained from `http://www.zemris.fer.hr/~idurasevic/CRP/CRP.7z`.

6. Results

The results obtained by the proposed method for the Caserta dataset are shown in Table 2. The first part of the table outlines the results obtained by the manually designed RRs. The second part of the table shows the results obtained by automatically designed RRs with different relocation schemes when optimising the two objectives. The values for both criteria are denoted for the obtained rules regardless of which criterion was optimised, to gain a better insight into how rules evolved for one criterion perform on the other. The best results obtained in the table are denoted in bold.

The obtained results demonstrate that the automatically designed RRs in most cases outperform all the manually designed RRs. This is evident because, for all RSs except UNP, the worst rules outperform all manually designed RRs. This shows that the proposed method is stable and can easily evolve new rules that outperform existing manually designed rules. When considering only the restricted rules, the best-generated rule performs better by 7.7% for the relocation criterion and 4.3% for the crane operation time criterion than the best

manually designed rule PR4. For the unrestricted variant, the best-generated rule achieved an improvement of 5.1% for the number of relocations and 3.8% for the crane operation time compared to the best manually designed rule (PU2). One thing that can be observed from the results is that when optimising the number of container relocations, the values for the crane operation time are similar to the values obtained when optimising this objective. However, the reverse is not always true, which might mean that optimising the number of container relocations also implicitly optimises the crane operation time.

To more easily compare the results, they are also illustrated using violin plots in Figures 5 and 6. These plots show that the UN scheme obtains the overall best results. When considering only the restricted schemes, it is clear that there is no statistically significant difference between the RE and REN variants. On the other hand, the differences are much more evident for the unrestricted variants. In this case, the UN scheme obtains significantly better results than any other RS.

Table 3 shows the results obtained on the Zhu dataset. The results again demonstrate that most of the automatically generated RRs outperform the existing manually designed rules. In this case, the best-evolved rules for the restricted variant outperform the best existing rule (PR4) by 7% for the number of relocations and 3.3% for the crane operation time. For the unrestricted variant, the best manually designed rule achieves a better performance than the best manually designed rule (PU2) by 4% for the number of relocations and 3% for the crane operation time. The difference in the obtained results is statistically significant.

Figures 7 and 8 show the violin plots for the tested RSs. Again the UN scheme achieves the best performance. However, in this case, its superiority over the other methods is much more clear. For the other schemes, the conclusions are mostly similar to the Caserta data set. Again, there is no statistically significant difference between the two restricted schemes for both criteria. For the unrestricted schemes, the UN scheme achieves significantly better results than any other scheme.

Table 2: Results obtained on the Caserta dataset

	Container relocations	Crane operation time
TLP	35982	2430770
RI	29524	2162170
MM	28996	2173290
PR3	25859	2064600
PR4	25787	2063070
PU1	25049	2034230
PU2	24962	2031130

Optimising crane relocations

	Container relocations			Crane operation time		
	Min	Med	Max	Min	Med	Max
RE	23816	24122	24508	1978200	1993387	2004644
REN	23849	24154	24449	1978772	1991568	2001708
UN	**23679**	**23967**	**24122**	1975008	1985987	1996640
UNC	23777	24236	24579	1979806	1995411	1996640
UNT	24041	24312	24709	1986174	1997805	2016432
UNP	24341	24876	25405	1997278	2019489	2038289

Optimising crane operation time

	Container relocations			Crane operation time		
	Min	Med	Max	Min	Med	Max
RE	23717	24298	24609	1974102	1992614	2001919
REN	23965	24276	24608	1982433	1991391	2005142
UN	23757	24053	24463	**1975159**	**1985758**	**1995911**
UNC	23850	24291	24684	1977746	1991795	2005178
UNT	23924	24292	24674	1981213	1993391	2009551
UNP	24389	24969	26741	1998595	2020428	2060607

Table 3: Results obtained on the Zhu dataset

	Container relocations	Crane operation time
TLP	551023	41038200
RI	469502	37508600
MM	473358	38155300
PR3	436717	37015300
PR4	435886	36994500
PU1	423058	36482700
PU2	422555	36476100

Optimising container relocations

	Container relocations			Crane operation time		
	Min	Med	Max	Min	Med	Max
RE	413469	414885	417092	35920974	35989754	36228750
REN	410993	414126	416897	35880669	35933913	36142543
UN	**405571**	**407103**	**409266**	35625504	35681901	35926484
UNC	413179	415247	417865	35897273	35982310	36267563
UNT	412713	415548	419194	35905778	36003943	36191417
UNP	414597	418379	424787	35948660	36103360	36341047

Optimising crane operation time

	Container relocations			Crane operation time		
	Min	Med	Max	Min	Med	Max
RE	414007	416447	422213	35815092	35907664	36067638
REN	410645	415655	418710	35791829	35890043	36002463
UN	406609	408286	412797	**3537753**	**35629511**	**35736631**
UNC	414030	416100	428039	35821219	35915527	36106539
UNT	414003	416458	639548	35797176	35956742	36399481
UNP	415371	418549	424566	35900349	36030791	35151525

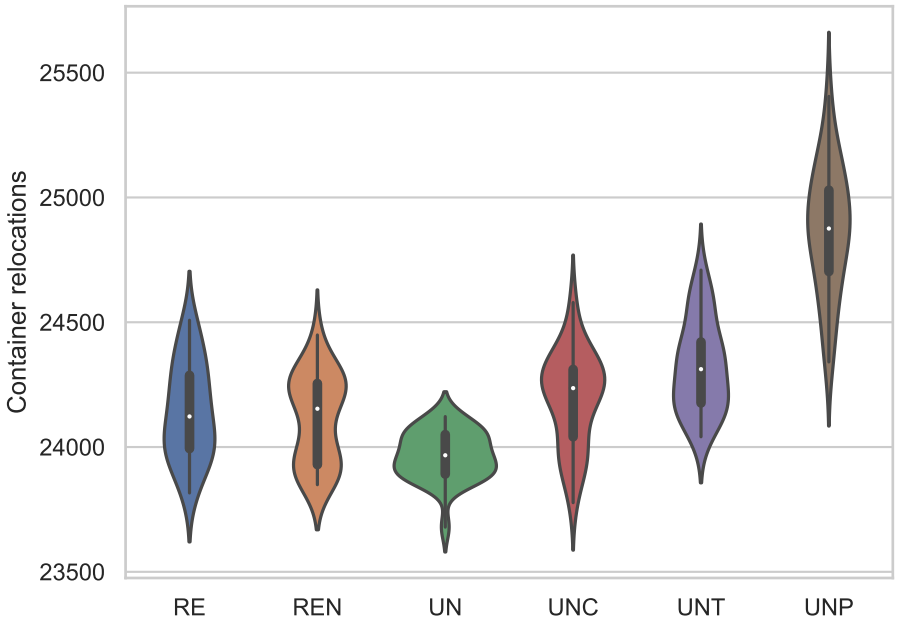


Figure 5: Optimising the number of relocations on the Caserta dataset

The previous results demonstrate that the automatically designed RRs almost always achieved a better performance than their manually designed counterparts, almost regardless of the applied RS. The results for the different RSs show that certain schemes lead to better results. For the restricted schemes, it was demonstrated that not considering the stack which contains the next target container as in the REN scheme did not significantly improve the results. This suggests that the evolved PFs can determine on which stack the next target container is located and not relocate containers there if not necessary. Thus, including this strategy in the RS is redundant. For the unrestricted scheme it is clear that evolving two rules which perform different decisions is difficult. It is most appropriate to select the destination container, and use a simple manually designed strategy that determines how to perform the unrestricted moves.

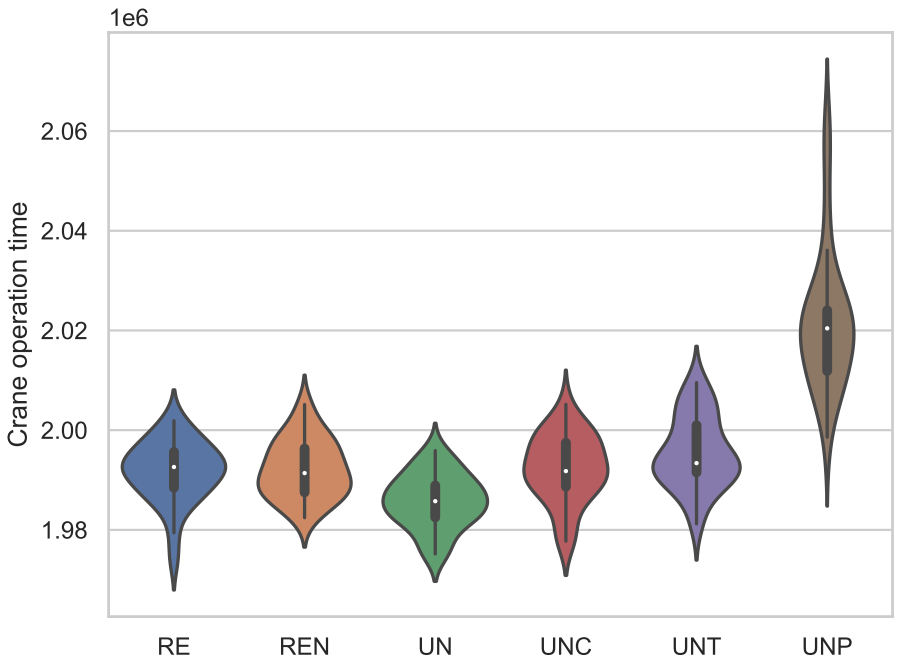


Figure 6: Optimising the crane operation time on the Caserta dataset

7. Further analysis

7.1. Dependency between the optimised criteria

The previous section demonstrated that by optimising one criterion, quite good solutions were obtained for the other criterion, especially when the number of relocations is optimised. This suggests that the two objectives are not conflicting and that it is enough to optimise only a single objective. To investigate the dependency between the objectives, NSGA-II [52] was applied to optimise both criteria simultaneously and obtain Pareto fronts of solutions. The Pareto front obtained for the unrestricted RS is shown in Figure 9 (denoted with red points). Based on 30 executions, only 5 non-dominated solutions were obtained. The figure shows that the range of the criteria values for these objectives is relatively small, especially when considering the time to perform the relocations. Although the improvement in one criterion leads to the worsening of the other,

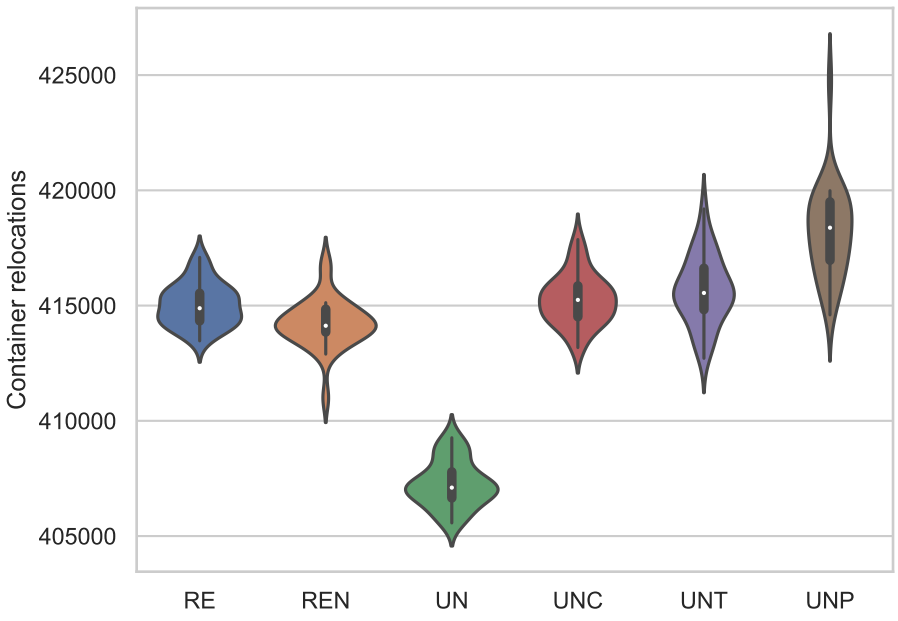


Figure 7: Optimising the number of relocations on the Zhu data set

these scales are quite small, especially for the crane operation time.

The figure also shows the two best solutions obtained in single-objective optimisation (denoted with blue points). One of them was obtained when optimising the number of container relocations (denoted with 1 in the figure) and the other when optimising the crane operation time (denoted with 2 in the figure). It is interesting to observe that the solution obtained by optimising the number of containers dominates all the solutions obtained by NSGA-II. This shows that by optimising one criterion, the other is also optimised to a significant extent and that it does not seem necessary to optimise each criterion individually or to use multi-objective approaches for optimisation.

To support this conclusion, we additionally apply Spearman's Rho correlation test, which examines the correlation between the two objectives on the results obtained in Table 2. When optimising the number of relocations, a correlation coefficient of 0.83 was obtained for the RE scheme, while a correlation value of 0.85 was obtained for the UN scheme. On the other hand, when op-

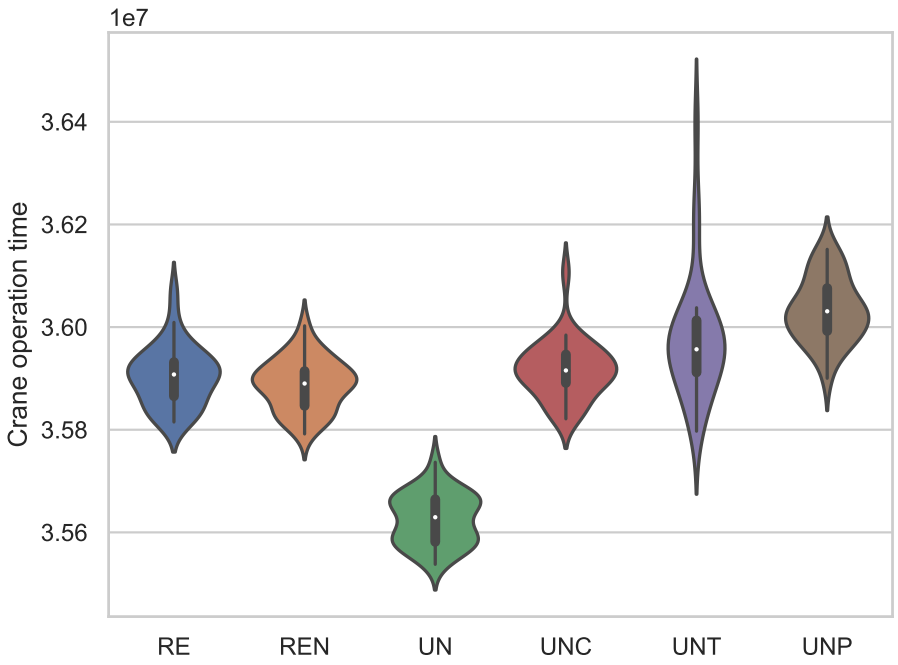


Figure 8: Optimising the crane operation time on the Zhu data set

timising the criterion of crane operation time criterion, a correlation value of 0.75 was obtained for the RE scheme and a value of 0.83 for the UN scheme. Since quite high correlation values between the two criteria were obtained, we can conclude that optimising one of the criteria indirectly optimises the other. Thus, a single rule that optimises both criteria can be obtained by optimising one of the two considered objectives.

7.2. Algorithm convergence

Figures 10 and 11 show the convergence of the algorithm on all three datasets for the restricted and unrestricted relocation schemes, respectively. In the figures, the container relocations are calculated as the sum of relocations made on all instances of the observed set and are given on the y-axis, while the number of evaluations is on the x-axis. In order to determine the number of evaluations to be performed, it is important to monitor the fitness values for the training and

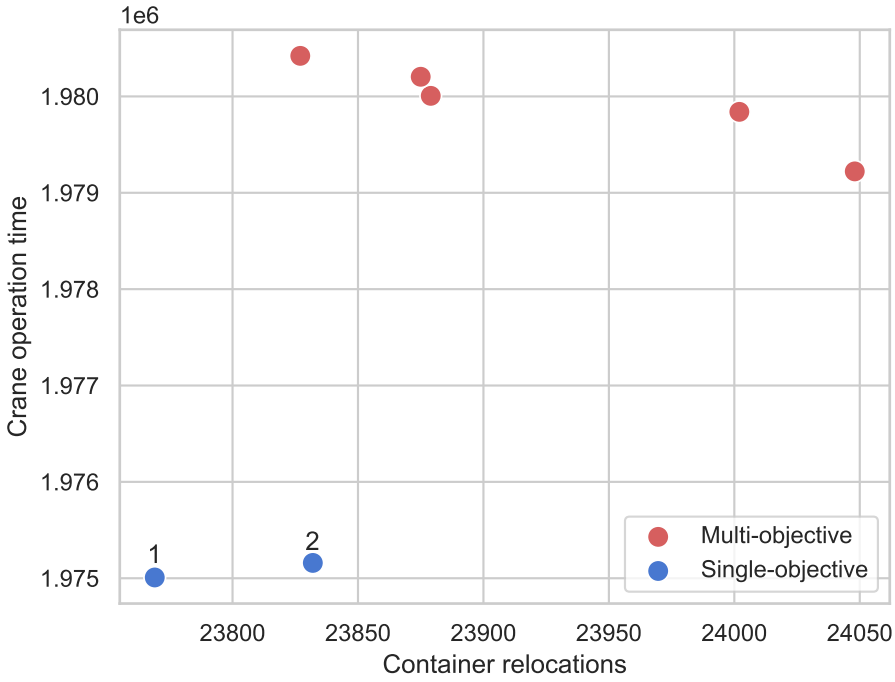


Figure 9: Pareto front of the solutions obtained when optimising both criteria with NSGA-II and the best solutions obtained when individually optimising the number of container relocations (1) and crane operation time (2)

validation sets simultaneously. From the figure it can be seen that the fitness of the training set is constantly improving during the training process, which is to be expected. Similar observations can be made for the fitness of the validation and test sets. However, the rate of improvement for these two sets is much lower, and in some cases the solutions may even deteriorate slightly. Such behaviour suggests that the expressions have succeeded in extracting general knowledge that makes them perform well on unseen problems, and that further training causes these expressions to adapt to the training set instead of improving their generalisation capabilities.

Since fitness is still improving after 50000 evaluations on all three sets, we want to determine whether these improvements are relevant. To find out, we use the validation set to test the generalisation ability of the developed expres-

sions on unseen problem instances. We perform a statistical test between the results obtained at the end of the algorithm (after 50000 evaluations) and between the solutions obtained during the execution of the algorithm after each step of 1000 evaluations. That is, we compare whether there is a significant difference between the results at the end of the algorithm and the results after 1000 evaluations, 2000 evaluations, 3000 evaluations, etc. in the validation set. These tests show that between 30000 and 36000 evaluations there is no significant difference between the solutions obtained with this number of evaluations and the final solutions. From this we can conclude that although the solutions have still improved, these improvements are not really significant. Furthermore, since from the figures we can see that the convergence is quite similar for the validation and test sets, we can conclude that if there is no significant improvement for the validation set, this should also be true for the test set as well. Based on all the observations so far, we can say that the algorithm has converged after the given time and that the number of evaluations could even have been reduced by a certain amount (to about 36000 evaluations) without any significant deterioration in the obtained results.

7.3. Run time analysis

As with other hyper-heuristic approaches, the runtime of the proposed method can be divided into two parts: the generation and execution of RRs. Since the generation of RRs is performed by GP, this part is more computationally intensive. For the default parameter values, a single run of GP took on average 50 and 40 minutes for the restricted and unrestricted relocation schemes, respectively. However, this part is performed offline at any time to obtain RRs. On the other hand, the execution time of the evolved RRs is severely smaller. When applied to solve the entire Caserta dataset consisting of 840 instances, the restricted and unrestricted rules require around 0.03 seconds on average. To solve all instances in the Zhu dataset 0.4 seconds were required on average. Therefore, the execution time of the RR is almost negligible, especially for a single decision.

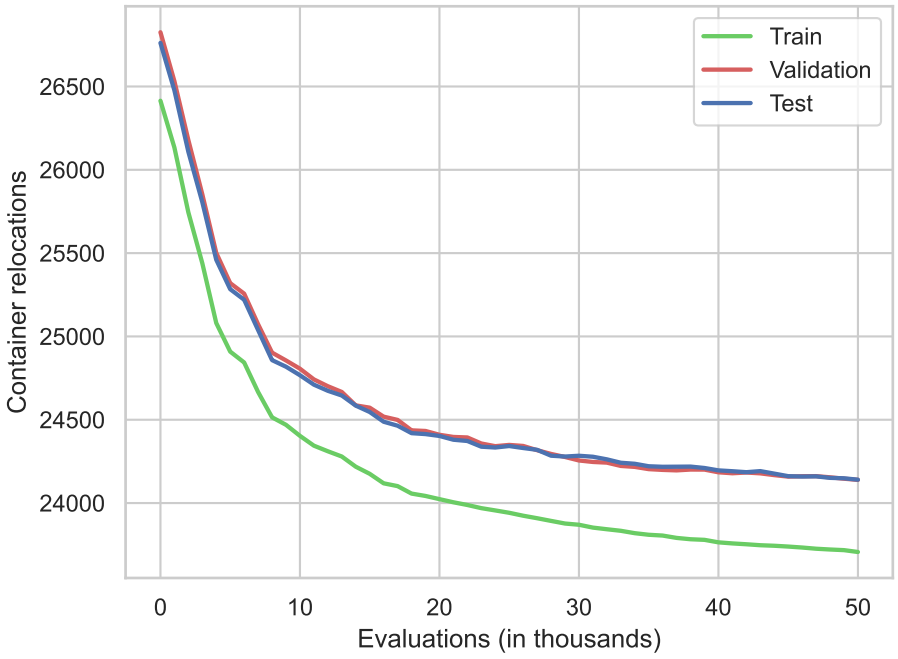


Figure 10: Convergence on the problems sets for the restricted variant

7.4. Relocation rule analysis

To gain a better insight into the workings of the generated RRs, in this section the behaviour of a selected RR for the restricted relocation scheme will be explained. The tree representation of the rule is shown in Figure 12, which translates to the following expression

$$\frac{\frac{\mathrm{RI}*\mathrm{MIN}}{\mathrm{AVG}*\mathrm{AVG}} - \frac{\mathrm{DIF}}{\mathrm{RI}*\mathrm{EMP}*\mathrm{EMP}}}{\mathrm{MIN}}.$$

The first part of the expression $\frac{\mathrm{RI}*\mathrm{MIN}}{\mathrm{AVG}*\mathrm{AVG}}$ favours stacks with a small reshuffle index (containers that have a smaller ID than the container that will be relocated), but which also contain containers with larger IDs. The second expression $-\frac{\mathrm{DIF}}{\mathrm{RI}*\mathrm{EMP}*\mathrm{EMP}}$ favours stacks with a large difference between the container with the minimum ID on the stack and the one that is being relocated, but which also contain as few free spaces as possible. Thus, the second part prefers stacks that are almost full but have a small reshuffle index, and thus the number of

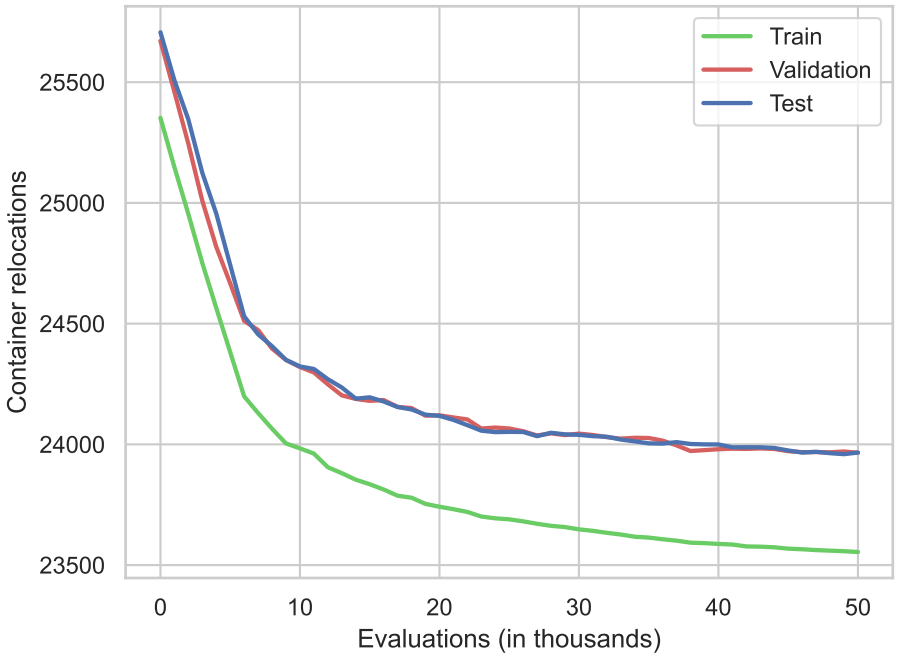


Figure 11: Convergence on the problems sets for the unrestricted variant

unnecessary relocations is minimised. In the end, the rules try to balance between the stacks that have a small reshuffle index for the current container and are as full as possible, and have a high average of IDs, and with that, it tries to minimise the number of relocations.

7.5. Analysis of the priority function sizes

The size of the evolved priority functions is an essential factor when designing RRs. The evolved PFs should be small in size so that they are easy to interpret and understand. Unfortunately, GP is prone to bloating, and the evolved expressions will usually contain many redundant parts. Although different methods can be used to remedy this problem, the easiest way is to restrict the depth of the evolved trees. Therefore, the influence of different tree depths on the quality and size of the evolved PFs are explored. Tree depths between 3 and 10 were used, and the obtained results on the Caserta dataset are outlined

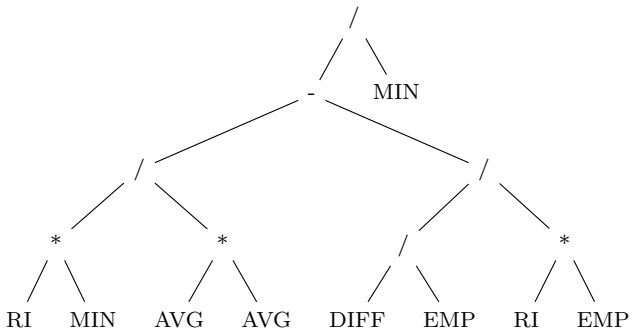


Figure 12: Relocation rule example

in Table 4. The SOB column represents the number of nodes in the expression of the best solution, whereas the MES column denotes the median size of the evolved expressions. The smallest tree depth of 3 and the largest depths from 8 until 10 usually lead to worse results. These depths offer either a too small or too large search space. The best results are usually achieved for depths between 4 and 6. Regarding the tree sizes, it is clear that a certain tree depth is required to obtain good solutions. For example, the median tree size for the depth of 3 is equal to the maximum number of nodes in the tree. Based on the figure, it can be seen that the best RRs are obtained when they contain more than 25 nodes in their PF. Therefore it is preferable to use larger tree depths to get better results. Consequently, a tree depth of at least 4 should be used, or even larger depths if it is required to obtain better results.

To gain a better understanding on the dependency of the tree sizes and their fitness, all solutions obtained for the tested tree depths were plotted against their size and fitness in Figure 13. It should be noted that the tree sizes were cut off at 200 nodes to make the figure more readable. This figure shows that until a specific tree size, it is not possible to obtain good expressions. Although it is difficult to determine a fixed size at which this happens, the figure shows that after tree sizes of 25 nodes, the number of good solutions increases. The increase of tree size does not guarantee that better solutions are obtained, but it does increase the probability of it.

Table 4: Influence of the tree depth on rule sizes

		Container relocations			Size	
	Depth	Min	Med	Max	SOB	MES
RE	3	24230	24512	24743	13	15
	4	23898	24297	24615	27	25
	5	23816	24122	24508	41	49
	6	23808	24113	24531	81	66
	7	23907	24148	24610	61	90
	8	23906	24238	25576	191	177
	9	23816	24285	25060	409	180
	10	23956	24286	25219	93	241
UN	3	24002	24230	24444	15	15
	4	23862	24032	24406	31	27
	5	23679	23967	24122	61	46
	6	23818	23989	24407	103	62
	7	23731	24009	24303	111	99
	8	23809	24082	24447	65	87
	9	23852	24144	24648	37	156
	10	23835	24151	24630	39	247

7.6. Generality of rules

In the previous section, the RRs were applied on the data set of the same type as the one on which they were evolved. Naturally, the best results should be obtained if a data set used to generate the rules is similar to the one on which the rules will be applied. However, the obtained rules should be general enough and perform well on other types of data sets as well. To test this, the rules that were evolved on the Caserta dataset were tested on the Zhu data set and vice versa. Table 5 outlines the results obtained when such a switch is performed. The first 4 rows show the results obtained when the Caserta dataset is used

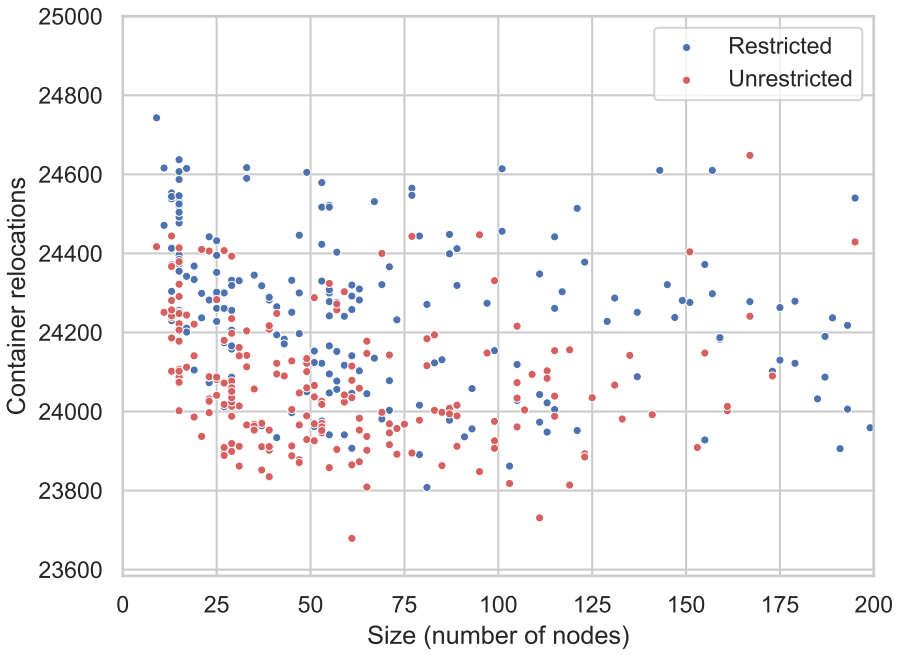


Figure 13: Dependency between the fitness of the rules and expression size

for testing, whereas the last 4 rows denote the results obtained when the Zhu dataset is used as the test set. The postfixes "-C" and "-Z" denote that the rules were trained on the Caserta and Zhu data set types, respectively.

When using the Caserta data set type as the test set, an interesting thing can be observed. For the unrestricted variant, there is no significant difference between the rules generated using either dataset. However, for the restricted variant, the rules generated using the Zhu dataset perform significantly worse than those generated on the Caserta data set. On the other hand, when the Zhu data set is used to test the rules generated using the Zhu and Caserta data sets, it performs similarly for both relocation schemes. This seems to suggest that using the Caserta data set for training might result in more general rules. A possible reason could be that the yards in the Zhu instances are almost full, limiting the relocation possibilities. This might make finding good rules more difficult as there are only a few possible relocations in the first steps, and thus

Table 5: Results when the RRs are applied on the other data set type

		Container relocations			Crane operation time		
		Min	Med	Max	Min	Med	Max
Caserta	RE-C	23816	24122	24508	1978200	1993387	2004644
	RE-Z	24281	24871	25600	2000882	2019038	2052329
	UN-C	23679	23967	24122	1975008	1985987	1996640
	UN-Z	23754	24057	24673	1979928	1991150	2018590
Zhu	Re-Z	413469	414885	417092	35920974	35989754	36228750
	RE-C	412559	416128	422553	35868280	36011830	36449254
	UN-Z	405571	407103	409266	35625504	35681901	35926484
	UN-C	406239	407828	411046	35658306	35706072	35967755

the rules have to focus more on the latter part when there are more available places.

By investigating the PFs evolved by GP using the two problem sets, it was noticed that the PFs are of similar sizes, regardless on which problem set they were trained. Furthermore, the frequency of terminal and function nodes in the evolved PFs was also quite similar, with only a few percent of difference between the rules evolved using the Caserta and Zhu data set. Based on these observations we can conclude that the expressions evolved across different data sets do not differ significantly in their structure. Since the rules perform well even when applied on the problem type that was not used for training, it is safe to assume that when trained on both data sets GP is capable of generating rules that work in a similar way and use similar strategies for relocating containers.

7.7. Performance on large problems

To stress test the proposed approach, the RRs evolved in the previous section when optimising the number of relocations for the Caserta dataset are applied on a large scale Caserta data set that was proposed in [53]. This dataset is composed of 400 instances that contain from 10 to 100 stacks and 100 containers

Table 6: Results for large scale Caserta dataset

	Container relocations			Crane operation time		
TLP	29717794			3.1E+9		
RI	15104213			1.6E+9		
MM	18902574			2.0E+9		
PR3	11119625			1.3E+9		
PR4	10966282			1.3E+9		
PU1	10951307			1.3E+9		
PU2	10797383			1.2E+9		
	Min	Med	Max	Min	Med	Max
RE	5212033	7304279	10657770	6.9E+8	8.9E+8	1.2E+9
UN	5720948	7370982	9444975	7.4E+8	9.2E+8	1.1E+9

per stack, meaning that the largest instances contain 10000 containers. The results obtained on this data set are denoted in Table 6. The table shows that the automatically designed RRs significantly outperform all the manually designed rules. The median value of the automatically designed rules is better by around 30% when compared with the best manually designed rule PU2. This shows that the gap between the automatically and manually designed rules increases as the instances become larger. Since the automatically designed rules were evolved on the small scale Caserta data set, this result also demonstrates that the obtained rules generalise well on unseen instances.

7.8. Improving results by evolving two PFs

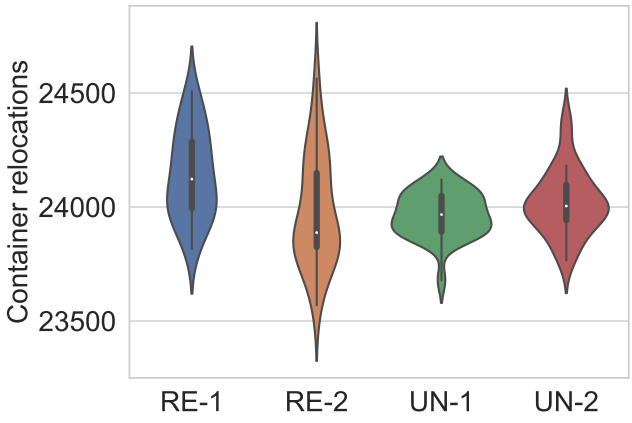
During preliminary analyses, it was observed that frequently a single rule had difficulties performing well across a broader range of instances. This naturally raises the question of whether evolving several expressions and using them simultaneously can improve the results. Therefore, the RE and UN schemes were extended to decide where to relocate the current container based on two PFs that are evolved simultaneously. Both PFs are evaluated individually for

Table 7: Results for RRs with two expressions

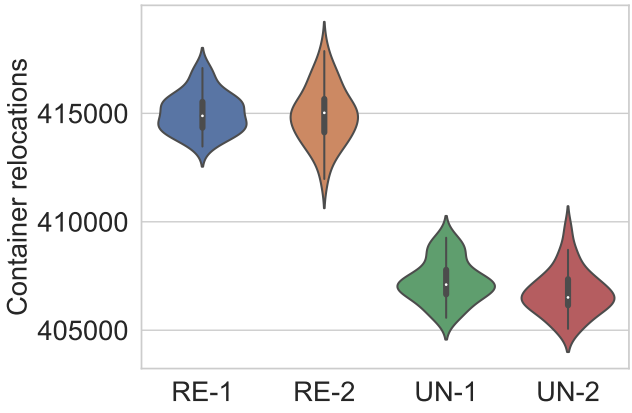
		Container relocations			Crane operation time		
		Min	Med	Max	Min	Med	Max
Caserta	RE-1	23816	24122	24508	1978200	1993387	2004644
	RE-2	23569	23888	24564	1972846	1983899	2015048
	UN-1	23679	23967	24122	1975008	1985987	1996640
	UN-2	23765	24005	24377	1978301	1991808	2001211
Zhu	RE-1	413469	414885	417092	35920974	35989754	36228750
	RE-2	411973	415025	417873	35888624	35986230	36231655
	UN-1	405571	407103	409266	35625504	35681901	35926484
	UN-2	405062	406512	409652	35623466	35688671	35894462

each candidate stack, and the current container will be relocated to the stack for which the lowest priority was obtained by any of the two PFs.

The results for this approach (denoted with the suffix "-2") are compared to the standard single expression RRs (denoted with the suffix "-1") and denoted in Table 7 and Figure 14. The results demonstrate that using two PFs has an effect on the obtained results, however, more in the case when using the RE scheme. This is especially evident for the Caserta data set, on which RRs that use 2 PFs significantly outperform the version that uses only a single PF. Not only that, in this case, the RRs using the RE scheme obtained a better median than the rules using the UN scheme. Furthermore, RE-2 achieved the best overall results across all the performed experiments. This clearly shows that the RE scheme is not inferior to the UN scheme. The reason it performed worse than the UN scheme could have been because a single PF was not enough to incorporate all the relevant information to perform such good decisions. Using two rules gives GP the ability that each one specialises for different situations. These results provide motivation for further research in which ensemble learning methods could be applied to generate sets of RRs.



(a) Caserta dataset



(b) Zhu dataset

Figure 14: Violin plots of RRs using one and two PFs

7.9. Terminal set analysis

In the proposed method, 14 terminal nodes, which encompass the characteristics of the problem, were proposed. Although using all those in the construction of PFs leads to good results, an open question is whether all those nodes are required. To gain a better insight into which nodes are the most important, a short analysis is performed. First, based on the initial experiments for the RE and UN, the occurrences of all nodes were calculated and are denoted in Figure 15. The figure shows the total number of times that a particular

node occurred in the trees (marked as total) and the unique number of times it occurred in an expression (denoted as unique), meaning that the number of expressions that contain a particular node is counted. For the latter metric, the maximum value is equal to the number of runs, meaning 30. Therefore, a vertical blue line was plotted at that value to better denote which nodes appear in most individuals. The figure denotes that two terminals are clearly more important than others. The two most important terminals are the RI and DIFF terminals. They appear not only in almost all generated rules but also appear several times in them. This is expected because they give direct information about whether it makes sense to relocate the current container to a stack. Out of the other nodes, the CUR, DUR, and SH also appear quite often, but not with such a large frequency. The least informative nodes are DSM, NEXT, NL, and WL.

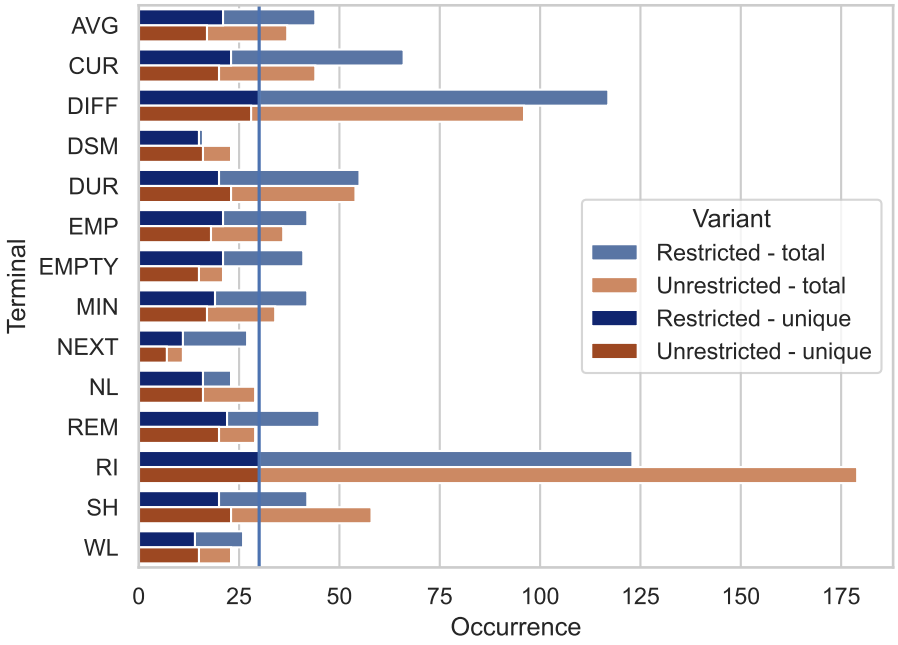


Figure 15: Terminal node occurrence frequency in the evolved PFs

Based on the initial observation that certain terminal nodes are less impor-

Table 8: Results of RRs with a reduced terminal set

		Container relocations			Crane operation time		
		Min	Med	Max	Min	Med	Max
Caserta	RE	23816	24122	24508	1978200	1993387	2004644
	RE-R	23758	23983	24252	1974572	1983939	1994616
	UN	23679	23967	24122	1975008	1985987	1996640
	UN-R	23786	23901	24126	1978344	1981985	1990452

tant, a series of experiments were performed to decrease the number of terminals in the set. At the start, the set contained all terminals, and then the node, which, when removed from the set, leads to the best results on the validation set, is excluded from the terminal set. The procedure is then repeated until all the nodes are removed. Although not all node combinations are tested (which would be hardly possible due to the sheer number of combinations), such a procedure still leads to a reduced set of terminal nodes. For the RE scheme, the terminal set was reduced to six nodes: SH, RI, EMP, DIFF, AVG, and CUR. On the other hand, in the UN scheme, the best results are obtained using only 4 nodes, SH, EMP, DIFF, and RI. The results obtained by the reduced terminal sets are denoted in Table 8, and are denoted with the suffix "-R". For both schemes, it was possible to improve the performance in comparison with the original terminal set. Although the differences to the original set are not large, the results obtained with the reduced terminal set were significantly better.

8. Findings on the use of GP for the CRP

Based on the performed experiments and analyses of the proposed approach, it is possible to outline important steps in the design and application of the proposed hyper-heuristic method to the considered problem. The first and most important part in the design of the method was to determine the general structure of the RRs consisting of the RS and PF. As the experiments have demonstrated, the design of the RS and PF are equally important for the performance

of the RR. For the RS it was demonstrated that the unrestricted RS outperformed the restricted variant, which is expected since it includes an additional flexibility by allowing the relocation of containers from the destination stack. This shows that including additional manually designed heuristic rules in the RS can help improve its performance. Furthermore, it was also interesting to observe that when GP was used to also evolve an expression for that heuristic part in addition to the main PF, the RR performed significantly worse. This demonstrates that the best RSs are actually those which provide a synergy of human designed heuristics for simple and intuitive decisions and more complex machine designed heuristics for more complicated and less intuitive decisions.

For the PF, the choice of elements that will be used to construct it has demonstrated to be crucial. When a large set of terminal nodes was used, the results, although good, were worse than when using a reduced set of terminals. Even though with more terminals GP can evolve PFs that have a better overview of the problem, the search space becomes too large and GP exhibits difficulties in obtaining good PFs. Therefore, it is crucial to perform an analysis on the importance of each terminal node and reduce the terminal set to improve the search capabilities of GP and reduce the complexity of the evolved expressions. Furthermore, the parameter controlling the maximum size of expressions also affects its performance directly, as restricting the size too much or allowing too large expressions to be generated again significantly decreases the performance of GP. These parameters have a more significant influence on the quality of the results than other GP parameters like the population size or mutation probability, and thus deserve more attention in the parameter tuning phase.

GP was demonstrated to be a computationally expensive procedure, and it requires some time to evolve efficient RRs. However, this procedure is performed only once, prior to solving a real problem, and as such does not influence the execution of the RR. The execution time of the RR, on the other hand, has demonstrated to be almost negligible, as the rules were able to solve several hundreds of instances in less than a second. Therefore, a single decision of the

RR is even faster and can be executed without a problem in real time. This allows that the RR is combined with some more sophisticated procedure to improve its performance, but still be able to perform the decision in a negligible amount of time.

Finally, in order for GP to be able to design RRs for a certain problem, it is required to have a certain set of instances that can be used for training. As such, when applying the GP to a problem with different characteristics, in order to create an appropriate RR it would be required to prepare a set that would be used by GP to evolve a new and specialised RR. However, the analysis of the results demonstrated that the evolved rules are quite general and perform well even when used on problem instances with different characteristics. This observation is important as it shows that GP is able to learn rules that perform well in general and thus it is not required to evolve a large number of rules for each problem variant, but rather existing rules can be efficiently reused on new and slightly different problem types.

9. Conclusion

This study proposed the application of GP to automatically design RRs for the CRP. In the proposed method, GP is used to evolve an expression that assigns priorities to stacks based on which it is determined on which stack the container will be placed. An RS is then used to relocate and retrieve containers from the yard using the expression generated by GP. The goal of this approach was to generate RRs which perform better than existing manually designed rules from the literature.

Experimental results demonstrate that automatically designed RRs outperform several existing rules from the literature for the two considered criteria. Even more, it was demonstrated that the worst automatically designed rules perform better than their manually designed counterparts for the best relocation schemes. The additional analysis further showed that the generated RRs are general enough to perform well on problems of different properties than those

they were evolved on, that the training time can be further reduced without a significant impact on the obtained results, and that their performance can be improved by using a subset of terminal nodes or evolving two PFs for container ranking. The obtained results and observations demonstrate that the proposed method represents a viable alternative to manually designed RRs, which can be utilised by themselves to construct good solutions in a relatively small amount of time. Additionally, because of their low execution time, they also have a potential of being used in synergy with other methods similarly as manually designed RRs have been used.

In future work, it is planned to extend this work in several directions. Firstly, the method will be adapted and tested on other variants of the CRP, like the multibay and duplicate priority variants. Another direction is to improve the obtained results further using ensemble learning methods or the rollout heuristics. Finally, the last research direction would be to adapt the proposed method and apply it to the dynamic variant of the CRP problem, in which the sequence of container retrievals is not entirely known beforehand, and the order of retrievals becomes known during the execution of the system.

References

[1] R. Jovanovic, S. Tanaka, T. Nishi, S. Voß, A GRASP approach for solving the Blocks Relocation Problem with Stowage Plan, Flexible Services and Manufacturing Journal 31 (3) (2019) 702–729. doi:10.1007/s10696-018-9320-3.

[2] D. Steenken, S. Voß, R. Stahlbock, Container terminal operation and operations research - A classification and literature review, OR Spectrum 26 (1) (2004) 3–49. doi:10.1007/s00291-003-0157-z.

[3] C. Lu, B. Zeng, S. Liu, A Study on the Block Relocation Problem: Lower Bound Derivations and Strong Formulations, IEEE Transactions on Automation Science and Engineering (2020) 1–25arXiv:1904.03347, doi:10.1109/tase.2020.2979868.

[4] D. Sculli, C. Hui, Three dimensional stacking of containers, Omega 16 (6) (1988) 585–594. doi:https://doi.org/10.1016/0305-0483(88)90032-1.

[5] M. Avriel, M. Penn, N. Shpirer, Container ship stowage problem: complexity and connection to the coloring of circle graphs, Discrete Applied Mathematics 103 (1-3) (2000) 271–279. doi:10.1016/s0166-218x(99)00245-0.

[6] K. H. Kim, G. P. Hong, A heuristic rule for relocating blocks, Computers and Operations Research 33 (4) (2006) 940–954. doi:10.1016/j.cor.2004.08.005.

[7] K.-C. Wu, C.-J. Ting, A beam search algorithm for minimizing reshuffle operations at container yards, Proceedings of the International Conference on Logistics and Maritime Systems (2010) 703–710.

[8] C. Díaz, M. C. Riff, New bounds for large container relocation instances using grasp, Proceedings - 2016 IEEE 28th International Conference on Tools with Artificial Intelligence, ICTAI 2016 (2017) 343–349doi:10.1109/ICTAI.2016.56.

[9] Y. Lee, Y. J. Lee, A heuristic for retrieving containers from a yard, Computers and Operations Research 37 (6) (2010) 1139–1147. doi:10.1016/j.cor.2009.10.005.

[10] M. Caserta, S. Voß, M. Sniedovich, Applying the corridor method to a blocks relocation problem, OR Spectrum 33 (4) (2011) 915–929. doi:10.1007/s00291-009-0176-5.

[11] M. Caserta, S. Schwarze, S. Voß, A mathematical formulation and complexity considerations for the blocks relocation problem, European Journal of Operational Research 219 (1) (2012) 96–104. doi:10.1016/j.ejor.2011.12.039.

[12] F. Forster, A. Bortfeldt, A tree search procedure for the container relocation problem, Computers and Operations Research 39 (2) (2012) 299–309. doi:10.1016/j.cor.2011.04.004.

[13] W. Zhu, H. Qin, A. Lim, H. Zhang, Iterative deepening A* algorithms for the container relocation problem, IEEE Transactions on Automation Science and Engineering 9 (4) (2012) 710–722. doi:10.1109/TASE.2012.2198642.

[14] R. Jovanovic, S. Voß, A chain heuristic for the blocks relocation problem, Computers and Industrial Engineering 75 (1) (2014) 79–86. doi:10.1016/j.cie.2014.06.010.

[15] D.-Y. Lin, Y.-J. Lee, Y. Lee, The container retrieval problem with respect to relocation, Transportation Research Part C: Emerging Technologies 52 (2015) 132–143. doi:https://doi.org/10.1016/j.trc.2015.01.024.

[16] S. Borjian, V. Galle, V. H. Manshadi, C. Barnhart, P. Jaillet, Container Relocation Problem: Approximation, Asymptotic, and Incomplete InformationarXiv:1505.04229.

[17] D. Ku, T. S. Arthanari, Container relocation problem with time windows for container departure, European Journal of Operational Research 252 (3) (2016) 1031–1039. doi:https://doi.org/10.1016/j.ejor.2016.01.055.

[18] L. Maglić, M. Gulić, L. Maglić, Optimization of Container Relocation Operations in Port Container Terminals, Transport 35 (1) (2019) 37–47. doi:10.3846/transport.2019.11628.

[19] C. D. Cifuentes, M. C. Riff, G-crem: A grasp approach to solve the container relocation problem for multibays, Applied Soft Computing 97 (2020) 106721. doi:https://doi.org/10.1016/j.asoc.2020.106721.
URL https://www.sciencedirect.com/science/article/pii/S1568494620306591

[20] S. Raggl, A. Beham, S. Wagner, M. Affenzeller, Solution approaches for the dynamic stacking problem, GECCO 2020 Companion - Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (2020) 1652–1660doi:10.1145/3377929.3398111.

[21] J. R. Koza, Human-competitive results produced by genetic programming, Genetic Programming and Evolvable Machines 11 (3) (2010) 251–284. doi:10.1007/s10710-010-9112-3.

[22] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724. doi:10.1057/jors.2013.71.

[23] G. Duflo, E. Kieffer, M. R. Brust, G. Danoy, P. Bouvry, A gp hyper-heuristic approach for generating tsp heuristics, in: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2019, pp. 521–529. doi:10.1109/IPDPSW.2019.00094.

[24] J. Jacobsen-Grocott, Y. Mei, G. Chen, M. Zhang, Evolving heuristics for dynamic vehicle routing with time windows using genetic programming, in: 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 1948–1955. doi:10.1109/CEC.2017.7969539.

[25] Y. Liu, Y. Mei, M. Zhang, Z. Zhang, A Predictive-Reactive Approach with Genetic Programming and Cooperative Coevolution for the Uncertain Capacitated Arc Routing Problem, Evolutionary Computation 28 (2) (2020) 289–316. arXiv:https://direct.mit.edu/evco/article-pdf/28/2/289/1858868/evco_a_00256.pdf, doi:10.1162/evco_a_00256. URL https://doi.org/10.1162/evco_a_00256

[26] J. Branke, S. Nguyen, C. W. Pickardt, M. Zhang, Automated design of production scheduling heuristics: A review, IEEE Transactions on Evolutionary Computation 20 (1) (2016) 110–124. doi:10.1109/TEVC.2015.2429314.

[27] S. Nguyen, Y. Mei, M. Zhang, Genetic programming for production scheduling: a survey with a unified framework, Complex & Intelligent Systems 3 (1) (2017) 41–66. doi:10.1007/s40747-017-0036-x.
URL https://doi.org/10.1007/s40747-017-0036-x

[28] F. J. Gil-Gala, C. Mencía, M. R. Sierra, R. Varela, Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time, Applied Soft Computing 85 (2019) 105782. doi:https://doi.org/10.1016/j.asoc.2019.105782.
URL https://www.sciencedirect.com/science/article/pii/S1568494619305630

[29] D. Jakobović, K. Marasović, Evolving priority scheduling heuristics with genetic programming, Applied Soft Computing 12 (9) (2012) 2781 – 2789. doi:https://doi.org/10.1016/j.asoc.2012.03.065.

[30] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem, IEEE Transactions on Evolutionary Computation 17 (5) (2013) 621–639. doi:10.1109/TEVC.2012.2227326.

[31] S. Nguyen, Y. Mei, B. Xue, M. Zhang, A Hybrid Genetic Programming Algorithm for Automated Design of Dispatching Rules, Evolutionary Computation 27 (3) (2019) 467–496. arXiv:https://direct.mit.edu/evco/article-pdf/27/3/467/1858661/evco_a_00230.pdf, doi:10.1162/evco_a_00230.
URL https://doi.org/10.1162/evco_a_00230

[32] M. Đurasević, D. Jakobović, K. Knežević, Adaptive scheduling on unrelated machines with genetic programming, Applied Soft Computing 48 (2016) 419–430. doi:https://doi.org/10.1016/j.asoc.2016.07.025.
URL https://www.sciencedirect.com/science/article/pii/S1568494616303519

[33] K. Jaklinović, M. Đurasević, D. Jakobović, Designing dispatching rules with genetic programming for the unrelated machines environment with constraints, Expert Systems with Applications 172 (2021) 114548. doi:https://doi.org/10.1016/j.eswa.2020.114548.
URL https://www.sciencedirect.com/science/article/pii/S0957417420311921

[34] S. Chand, Q. Huynh, H. Singh, T. Ray, M. Wagner, On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems, Information Sciences 432 (2018) 146 – 163. doi:https://doi.org/10.1016/j.ins.2017.12.013.
URL http://www.sciencedirect.com/science/article/pii/S0020025517311350

[35] M. Đumić, D. Šišejković, R. Čorić, D. Jakobović, Evolving priority rules for resource constrained project scheduling problem with genetic programming, Future Generation Computer Systems 86 (2018) 211 – 221. doi:https://doi.org/10.1016/j.future.2018.04.029.
URL http://www.sciencedirect.com/science/article/pii/S0167739X1732441X

[36] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Zhang, An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling, Applied Soft Computing Journal 63 (2018) 72–86. doi:10.1016/j.asoc.2017.11.020.
URL http://dx.doi.org/10.1016/j.asoc.2017.11.020

[37] M. Đurasević, D. Jakobović, Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment, Genetic Programming and Evolvable Machines 19. doi:10.1007/s10710-017-9302-3.

[38] M. Đurasević, D. Jakobović, Creating dispatching rules by simple ensemble

combination, Journal of Heuristics 25 (6) (2019) 959–1013. doi:10.1007/s10732-019-09416-x.

[39] M. Đumić, D. Jakobović, Ensembles of priority rules for resource constrained project scheduling problem, Applied Soft Computing 110 (2021) 107606. doi:https://doi.org/10.1016/j.asoc.2021.107606.
URL https://www.sciencedirect.com/science/article/pii/S1568494621005275

[40] F. J. Gil-Gala, M. R. Sierra, C. Mencía, R. Varela, Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling, Natural Computing 4 (2016). doi:10.1007/s11047-020-09793-4.

[41] M. Đurasević, D. Jakobović, Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment, Genetic Programming and Evolvable Machines 19 (1-2) (2017) 9–51. doi:10.1007/s10710-017-9310-3.
URL https://doi.org/10.1007/s10710-017-9310-3

[42] F. Zhang, Y. Mei, M. Zhang, Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics, in: 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 1366–1373. doi:10.1109/CEC.2019.8790112.

[43] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, K. C. Tan, Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling, IEEE Transactions on Evolutionary Computation 25 (4) (2021) 651–665. doi:10.1109/TEVC.2021.3065707.

[44] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Collaborative multifidelity-based surrogate models for genetic programming in dynamic flexible job shop scheduling, IEEE Transactions on Cybernetics (2021) 1–15doi:10.1109/TCYB.2021.3050141.

[45] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, M. Zhang, Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling, IEEE Transactions on Cybernetics (2021) 1–14doi:10.1109/TCYB.2021.3065340.

[46] M. Đurasević, D. Jakobović, Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment, Applied Soft Computing 96 (2020) 106637. doi:https://doi.org/10.1016/j.asoc.2020.106637.
URL https://www.sciencedirect.com/science/article/pii/S1568494620305755

[47] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling, IEEE Transactions on Cybernetics 51 (4) (2021) 1797–1811. doi:10.1109/TCYB.2020.3024849.

[48] R. Poli, W. B. Langdon, N. F. McPhee, A field guide to genetic programming, Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008, (With contributions by J. R. Koza).
URL http://www.gp-field-guide.org.uk

[49] M. Đurasević, D. Jakobović, M. S. R. Martins, S. Picek, M. Wagner, Fitness landscape analysis of dimensionally-aware genetic programming featuring feynman equations, in: Parallel Problem Solving from Nature – PPSN XVI, Springer International Publishing, 2020, pp. 111–124. doi:10.1007/978-3-030-58115-2_8.
URL https://doi.org/10.1007/978-3-030-58115-2_8

[50] K.-H. Liu, C.-G. Xu, A genetic programming-based approach to the classification of multiclass microarray datasets, Bioinformatics 25 (3) (2008) 331–337. arXiv:https://academic.oup.com/bioinformatics/article-pdf/25/3/331/16891260/btn644.pdf, doi:

10.1093/bioinformatics/btn644.
URL https://doi.org/10.1093/bioinformatics/btn644

[51] F. Viegas, L. Rocha, M. Gonçalves, F. Mourão, G. Sá, T. Salles, G. Andrade, I. Sandin, A genetic programming approach for feature selection in highly dimensional skewed data, Neurocomputing 273 (2018) 554–569. doi:https://doi.org/10.1016/j.neucom.2017.08.050.
URL https://www.sciencedirect.com/science/article/pii/S0925231217314716

[52] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197. doi:10.1109/4235.996017.

[53] F. Tricoire, J. Scagnetti, A. Beham, New insights on the block relocation problem, Computers and Operations Research 89 (2018) 127–139. doi:10.1016/j.cor.2017.08.010.