Solving Container Pre-Marshalling Problem using Monte-Carlo Tree Search and Deep Neural Network

by

Jianwei Liu

B.S., University of California San Diego, 2019

Submitted to the Graduate Faculty of the

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Industrial Engineering

University of Pittsburgh

2023

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Jianwei Liu

It was defended on

April 12, 2023

and approved by

Jayant Rajgopal, Ph.D, Professor, Department of Industrial Engineering, University of Pittsburgh

Taewoo Lee, Ph.D, Assistant Professor, Department of Industrial Engineering, University of Pittsburgh

Thesis Advisor: Bo Zeng, Ph.D, Associate Professor, Department of Industrial Engineering, University of Pittsburgh

Solving Container Pre-Marshalling Problem using Monte-Carlo Tree Search and Deep Neural Network

Jianwei Liu, M.S.

University of Pittsburgh, 2023

The container pre-marshaling problem (CPMP) is a significant challenge in container terminal operations, aiming to optimize the relocation of containers to improve efficiency. Despite extensive research on exact and heuristic methods for the container relocation problem (CRP) and CPMP, reinforcement learning (RL) remains underexplored in the literature. This thesis proposes a Monte-Carlo Tree Search (MCTS) method combined with the Lowest Priority First Heuristic (LPFH) to solve the CPMP efficiently.

The MCTS method incorporates the LPFH heuristic to achieve consistent simulation results and minimize the number of movements needed for container relocation. Our approach achieves near state-of-the-art results in several instances with acceptable inference speed. Additionally, this thesis introduces a machine learning-based method to estimate the number of relocations required for a given CPMP configuration. We train a deep learning model with both convolutional neural network (CNN) and multi-layer perceptron (MLP) architectures on self-generated data, identifying important features to achieve over 90% classification accuracy in small instances.

The proposed approach has the potential to provide more efficient and effective solutions to the CPMP than traditional optimization methods. Overall, this thesis contributes to the CPMP literature by introducing novel methods for solving the problem and providing valuable insights into the potential of machine learning and RL for solving complex optimization problems.

Table of Contents

1.0 Introduction..1

1.1 Container Pre-Marshalling Problem...2

1.2 Research Focus..4

2.0 Related Works..6

2.1 Container Pre-Marshalling Problem...6

2.2 Monte-Carlo Tree Search..7

2.3 Machine Learning in CRP and CPMP...8

3.0 CPMP-MCTS..10

3.1 Methods..10

3.1.1 MCTS in CPMP Setting...10

3.1.2 The Four Phases..10

3.2 Computational Results...12

4.0 Estimate Optimal Solutions with DNN...13

4.1 Methods..13

4.1.1 Feature Extraction...14

4.1.2 Network Structure..16

4.2 Computational Experiments..18

4.2.1 Data Generation..18

4.2.2 Performance Metrics...19

4.2.3 Performance..23

4.2.3.1 Regression Model..23

4.2.3.2 Classification Model 24

5.0 Conclusion and Future Research 28

Bibliography 29

List of Tables

Table 1 Comparison to other heuristic methods on CV dataset 12

Table 2 Influencing Factors of Container Relocated Operation 15

Table 3 Performance Evaluation of the Regression Model on the Different Test Sets.........24

Table 4 Performance Evaluation of the Classification Model on the Different Test Sets 26

Table 5 Performance Evaluation of Different Models and Features on ZSS Sets 27

List of Figures

Figure 1 A illustration of containers with RMGC 3

Figure 2 Example of a initial configuration and its solved state...4

Figure 3 One iteration of MCTS process 8

Figure 4 Overview of estimation DNN 13

Figure 5 Network structure 17

Figure 6 Loss and Accuracy Curve for Regression Model 24

Figure 7 Loss and Accuracy Curve for Classification Model 25

1.0 Introduction

The global shipping industry has experienced significant growth in recent decades, with container throughput reaching 798 million twenty-foot equivalent units (TEU) in 2020, a 47.5% increase from 2010 (UNCTAD 2022). However, delays are inevitable as the number of containers continues to rise, with stacking-related issues being a major contributor. Intra-terminal operations, such as incorrect stacking, and inter-terminal transportation both contribute to delays and uncertainty, leading to increased safety stock and maintenance costs for companies. Furthermore, unnecessary container movements can increase fuel consumption by up to 40%, as noted by Świeboda and Zając (2016). Therefore, reducing delays and improving logistics efficiency is critical for both economic and environmental reasons.

This thesis focuses on delays during the storing and retrieving process, with the container relocation problem (CRP) and the container pre-marshaling problem (CPMP) being two major issues in stacking. While there are numerous studies addressing these issues, this thesis proposes a novel solution to CPMP and a machine learning-based approach for estimating the number of required relocations.

The CPMP is defined as a problem of sorting containers in advance to optimize their efficient transfer during busy periods while minimizing the number of relocations needed for container relocation. This thesis introduces a new Monte-Carlo Tree Search (MCTS) method that incorporates the Lowest Priority First Heuristic (LPFH) to achieve consistent simulation results and minimize container movements. Furthermore, we propose a machine learning model that combines a convolutional neural network (CNN) and a deep neural network (DNN) to estimate the number of required relocations for a given CPMP configuration. Experimental results demonstrate over 90% classification accuracy in small instances, indicating the potential of our approach to improve logistics efficiency while reducing costs and environmental impact.

Overall, this thesis contributes to the CPMP literature by proposing novel methods for solving the problem and providing insights into the potential of machine learning for solving complex optimization problems. The proposed approaches have the potential to significantly improve logistics efficiency, both for economic and environmental reasons.

1.1 Container Pre-Marshalling Problem

In summary, the container pre-marshaling problem (CPMP) addresses the issue of sorting containers in a yard before transferring them to ships or trucks to minimize the number of movements required for container relocation. The terminal usually use a rail-mounted gantry crane (RMGC) to move containers to other locations. The delays and uncertainty in container operations can lead to economic and environmental costs, making it crucial to improve logistics efficiency. The CPMP considers re-ordering containers in the yard based on their departure times to ensure there are no misoverlaid containers in the current configuration. To understand the problem, common terminology such as stack, tier, bay, configuration, and priority are defined.

Stack: Containers in the maritime terminals that are stacked one over another form a stack. The crane operates across different stacks and can only move the top container of the stack.

Tier: The number of slots in a stack is called tiers. Usually there is a height restriction for a stack, which can be defined as the maximum number of tiers.

Bay: As shown in Fig. 1, a bay is one row of container stacks that has multiple tiers.

Configuration: An arrangement of items in the storage area.

Priority: A sequence of numbers that reflect the departure times of containers. The lower the number, the higher the priority, meaning the container will be retrieved before the next number.

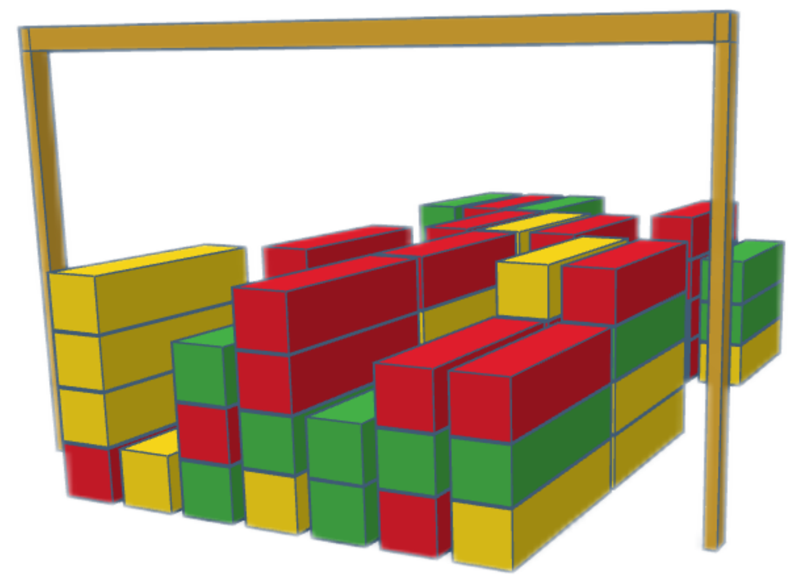

Figure 1 A illustration of containers with RMGC

In the pre-marshalling problem, the focus is on sorting the containers in a single bay in a maritime terminal. The bay consists of a specific number of stacks, denoted as S, and a maximum number of tiers, denoted as T. We define priority (s, t) as the container located in stack s at tier t. The objective of the pre-marshalling problem is to eliminate all misoverlays in the bay, which means that each stack should be sorted in descending order from bottom to top, based on the priority of each container. A configuration is considered sorted if priority (s, t) is greater than or equal to priority (s, t+1) for all 1 ≤ s ≤ S and 1 ≤ t < T. Some assumptions are made in advance which was presented in Hottung and Tierney (2016):

1. The problem is restricted to a single bay.
2. We can only move one container at a time from the top of one stack to the top of another stack. We have to make sure we don't exceed the maximum height of the configuration.
3. The priorities of containers are known in advance of solving the CPMP.

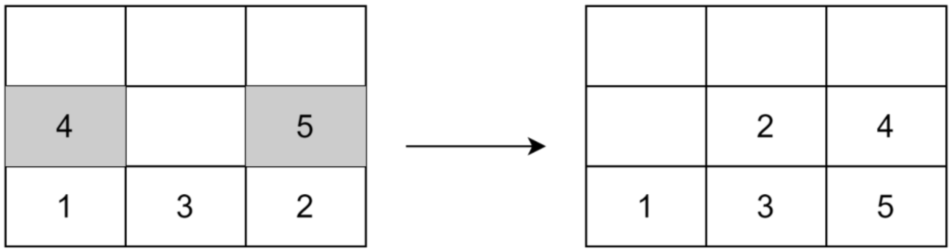

Figure 2 Example of a initial configuration and its solved state

1.2 Research Focus

This thesis focuses on applying MCTS and DNN to solve CPMP. The CPMP has been proved to be NP-hard in Caserta, Schwarze, and Voß (2011). It is computationally very challenging, especially on large scale instances. Our goal is to design methods that can solve practical-scale problems efficiently. MCTS gives us the tool to utilize powerful simulation methods to assist tree search process. DNN helps us estimate the number of relocations directly in large scale problems.

In Section 2, we will talk about the works related to CPMP, MCTS as well as machine learning applications in CRP and CPMP. In Section 3, we will introduce CPMP-MCTS, our MCTS method to solve CPMP. In Section 4, we will demonstrate the DNN to estimate optimal solution. Finally, in Section 5, we will conclude and talk about the future works.

2.0 Related Works

In this section, we first give an overview of existing methods for CPMP. Then we introduce the basic concepts of MCTS as well as the works that inspire us. Lastly, we discuss the use of machine learning in CRP and CPMP.

2.1 Container Pre-Marshalling Problem

The Container Pre-Marshalling Problem introduced by Lee and Chao in 2009 has received significant attention from researchers, leading to the development of various exact and heuristic methods. Exact approaches include the constraint programming model in Rendl and Prandtstetter (2013), which models all the slots with variables to avoid network formulation. van Brink and van der Zwaan (2014) introduced the branch-and-price algorithm to improve performance by using sub-problems. Tierney et al. (2016) demonstrated an A*/IDA* technique, using lower bound to help calculating the number of misoverlaid containers. Tanaka and Tierney (2018) gave an iterative deepening branch-and-bound algorithm to archive better performance in larger instances.

On the other hand, heuristic approaches focus on generating solutions quickly and are applicable to real-world problems, even when dealing with a large number of stacks and tiers. Caserta and Voß (2009) proposed the corridor method, which creates a "corridor" within the bay to limit the number of possible moves, and uses a local search procedure combined with predefined rules. In Exp´osito-Izquierdo et al. (2012), the lowest priority first heuristic (LPFH) was introduced to consider containers with a low priority as targets, and move them first. LPFH outperformed the corridor method. Wang et al. (2015) proposed a target-guided approach within a beam search, fixing the items at chosen locations to avoid further movements. Hottung and Tierney (2016) used a biased random-key genetic algorithm (BRKGA) with a decoder to construct a solution. Notably BRKGA requires less than a minute for solution generation. Jovanovic et al. (2017) extended LPFH with a multistart strategy and a complex set of problem-specific rules.

Overall, researchers have proposed various exact and heuristic methods for solving the CPMP, with heuristic approaches being more applicable to real-world problems. LPFH, in particular, has been extensively studied and extended with additional strategies, while the BRKGA and beam search approaches have shown significant improvements in solution generation.

2.2 Monte-Carlo Tree Search

MCTS was introduced by Coulom in 2006 by combining Monte-Carlo evaluation and tree search. The algorithm will return the solution in the allocated iterations. Each iteration consists of 4 steps:

1. **Selection**: the algorithm searches the portion of the tree that has already been saved in the memory. Given a node, it will select the best child of this node.
2. **Expansion**: expansion adds one new child node to the tree.
3. **Simulation**: simulation performs a random simulation of the problem, and give a score based on the policy used. This is the “Monte Carlo” part of the algorithm.

4. **Backpropagation:** backpropagation propagates the scores, back to all nodes along the path from the last visited node in the tree to the root.

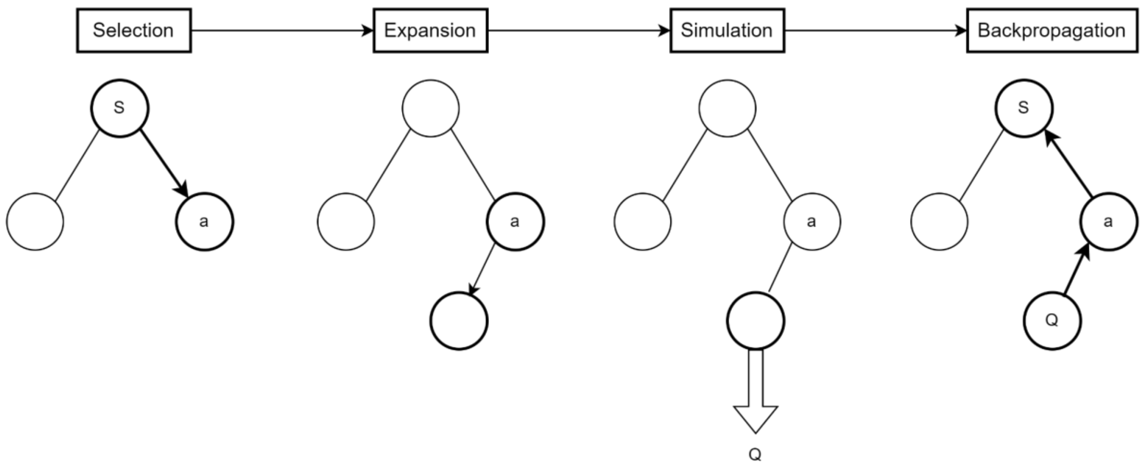

Figure 3 One iteration of MCTS process

2.3 Machine Learning in CRP and CPMP

In recent years, there has been a growing interest in applying deep learning and reinforcement learning techniques to solve combinatorial optimization problems. One such problem is container scheduling, where researchers have explored the use of machine learning algorithms to improve solution quality and efficiency. Hottung et al. (2020) proposed the deep learning heuristic tree search (DLTS), which uses a deep neural network to determine the search space's lower bound and prune it, resulting in high-quality heuristic solutions to the pre-marshaling problem.

Jiang et al. (2021) and Wei et al. (2021) used reinforcement learning methods to train instances with varying dimensions, proposing various heuristic rules to minimize the number of container relocations required. The experimental results indicated that their approach outperformed conventional optimization methods. Overall, these studies demonstrate the potential of deep learning and reinforcement learning techniques in improving the efficiency and quality of container scheduling and related combinatorial optimization problems.

3.0 CPMP-MCTS

3.1 Methods

3.1.1 MCTS in CPMP Setting

Monte Carlo Tree Search is widely used in solving many board games. For CPMP, changes must be made to adapt into MCTS setting. The major differences between CPMP and board games like Go are as follows: First, there is only one player in CPMP, which is the crane that move the containers around. Then it is very difficult to define a perfect reward function that fits every scenario in CPMP. Finally, random policy used in most of MCTS can easily lead to an infinite loop of actions.

Each node of the tree represents a configuration of containers. An action performed results in the tree to move to next level. Thus the children of a node represent all the possible configurations after performing actions. The reward we use is simply the number of relocations required to find a feasible solution. The node stores the least number of relocations and the corresponding action.

3.1.2 The Four Phases

Below we demonstrate the four phases in CPMP-MCTS.

Selection Strategy: The selection strategy is responsible for choosing one of the children of a given node. It balances the tradeoff between exploitation and exploration, where exploitation prioritizes the moves that have led to the best results so far, and exploration focuses on less promising moves that still need to be explored due to the uncertainty of their evaluation. In the Monte Carlo Tree Search (MCTS) algorithm, a child must be selected at each node starting from the root until a leaf node is reached.

Expansion Strategy: When a leaf node is reached, the expansion strategy determines which nodes are stored in memory. Coulom et al. proposed expanding one child per simulation, where the expanded node corresponds to the first position encountered that was not present in the tree. This strategy was also utilized in our implementation.

Simulation Strategy: Simulation helps us decide which node we should choose. To simulate the remainder of the sequences of actions starting from a leaf node, we use random moves at first until a solution is found. However, the random method can’t find a feasible solution in most cases. To get a meaningful simulation, we choose the lowest priority first heuristic (LPFH) as our simulation method.

Back-Propagation Strategy: During the back-propagation phase, the simulation result at the leaf node is propagated backward to the root. Various back-propagation strategies have been proposed in the literature, but we achieved the best results by using the best score so far.

3.2 Computational Results

Table 1 Comparison to other heuristic methods on CV dataset

			Avg. Moves						Avg. Time (s)				
Group	S	T	Opt.	LPFH	BRKGA	DLTS-G1	DLTS-G123	CPMP-MCTS	LPFH	BRKGA	DLTS-G1	DLTS-G123	CPMP-MCTS
CV 3-5	5	5	10.15	11.98	10.33	10.35	10.40	10.875	0.01	1.19	1.06	1.03	3.11
CV 4-5	5	6	17.85	22.13	18.75	17.90	18.05	20.775	0.01	5.38	12.11	10.47	8.12
CV 5-5	5	7	24.95	31.78	27.88	25.10	25.10	30.77	0.01	25.23	46.32	36.73	31.01
CV 3-7	7	5	12.80	15.40	12.93	12.90	13.30	14.10	0.01	1.17	42.40	0.30	5.35
CV 4-7	7	6	21.82	27.88	22.73	22.07	22.30	26.3	0.01	4.41	59.84	4.04	33.21
CV 5-7	7	7	31.48	41.43	33.83	31.98	32.08	41.25	0.01	20.77	59.91	42.26	47.22

Comparing to state-of-the-art heuristic algorithms BRKGA and DLTS, CPMP-MCTS stays behind by 0.5 moves to 9 moves. In terms of average runtime, CPMP-MCTS is generally faster than DLTS-G1 and slower than BRKGA. Compared to LPFH, CPMP-MCTS uses less moves in all the testing groups.

4.0 Estimate Optimal Solutions with DNN

4.1 Methods

Previous research by Ye et al. (2022) presented a preliminary study on using machine learning to predict the number of container relocations required for CPMP. We extend this work by building a custom machine learning model that maps extracted features to the number of required relocations. Our approach also involves feature selection to provide more informative data for the model to learn from. Additionally, we extend the problem's dimension from a 3x3 grid to a 7x10 grid to improve its adaptability to different scales. Overall, our proposed unified model is designed to improve the efficiency and effectiveness of CPMP in seaport container yards.

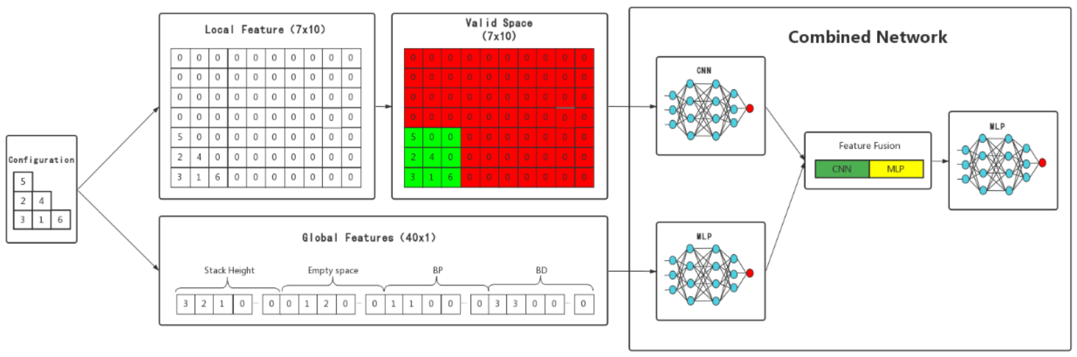

Figure 4 Overview of estimation DNN

4.1.1 Feature Extraction

The features are divided into two parts: local features and global features. The local features are configuration and valid space for each container. The global features contain the height, the number of empty slots, the blocking count and blocking degree of each stack. The total number of features is 180. The detailed explanation is as follow:

1. Features 1 to 70: the container priority and location information. The features start from the bottom of the first stack, and end with the highest tier 7. The rest of the stacks follow the same order until we finish stack 10. In this way we can preserve the location information of each slot. The priority of each slot will be extracted. If there is no container in the current slot, we put 0 instead.
2. Feature 71 to 140: the valid space of configuration. Since we use 0 to represent the empty slot, we need another feature to represent the valid workspace. The containers can only be moved inside the valid space. We use 1 to represent valid and 0 for invalid.
3. Feature 141 to 150: the current height of each stack. This group provides more spatial information for each stack.
4. Feature 151 to 160: the number of empty slots of each stack. Combine with the previous group, the model has a better understanding of the configuration.
5. Feature 161 to 170: the blocking count of each stack. Follow the description in Wei et al. (2021), we can calculate the blocking count of each stack.
6. Feature 171 to 180: the blocking degree of each stack. Similar to blocking count, blocking degree puts the priority into consideration.

The following pseudo code demonstrates how to calculate the blocking count and the blocking degree. We follow the illustration in Wei et al. (2021):

```
define blockingDegree, blockingCount = 0, 0
// elements in list are priorities
Define stack = initialStack
while (stack.elementCount > 1)
    define m = stack.MaxPriority
    // define upper stack includes m
    define upperStack = stack[m.index, end]
    if (upperStack.elementCount > 1)
        for each (x in upperStack exclude m)
            blockingCount += 1
            blockingDegree += x - m
            stack = stack[0, m.index]
return blockingCount, blockingDegree
```

Table 2 Influencing Factors of Container Relocated Operation

Serial number	Features	Notations or calculations
1-70	The container retrieval priority corresponding to each slot in the initial state	$P_{ij} = slot(s_i, t_j)$, $(i\epsilon\{1,2,\dots,10\}, j\epsilon\{1,2,\dots,7\})$
71-140	The valid space in the initial state	$V_{ij} = Valid(s_i, t_j)$, $(i\epsilon\{1,2,\dots,10\}, j\epsilon\{1,2,\dots,7\})$
141-150	Height of each stack	$H_i, i\epsilon\{1,2,\dots,10\}$
151-160	Number of empty slots per stack	$E_i = H_{max} - H_i$, $(i \in \{1,2,\dots,10\})$
161-170	Number of BP containers per stack	$BP_i = countBP(S_i)$, $(i \in \{1,2,\dots,10\})$
171-180	Number of BD containers per stack	$BD_i = countBD(S_i)$, $(i\epsilon\{1,2,\dots,10\})$

4.1.2 Network Structure

In summary, both CNNs and MLPs are types of artificial neural networks used in deep learning. CNNs are commonly applied to analyze visual imagery due to their shared-weight architecture of convolution kernels, which provide translation-equivariant responses known as feature maps. However, CNNs are not always invariant to translation due to the downsampling operation applied to the input. In contrast, MLPs are function approximators that consist of multiple layers of perceptrons, where each neuron accepts weighted inputs and applies an activation function to produce an output that is sent to the next layer. In this work, CNNs are used to process local features with positional information, while MLPs are used to process global features. Combining the strengths of CNN and MLP, we can fully utilize both local features as well as global features.

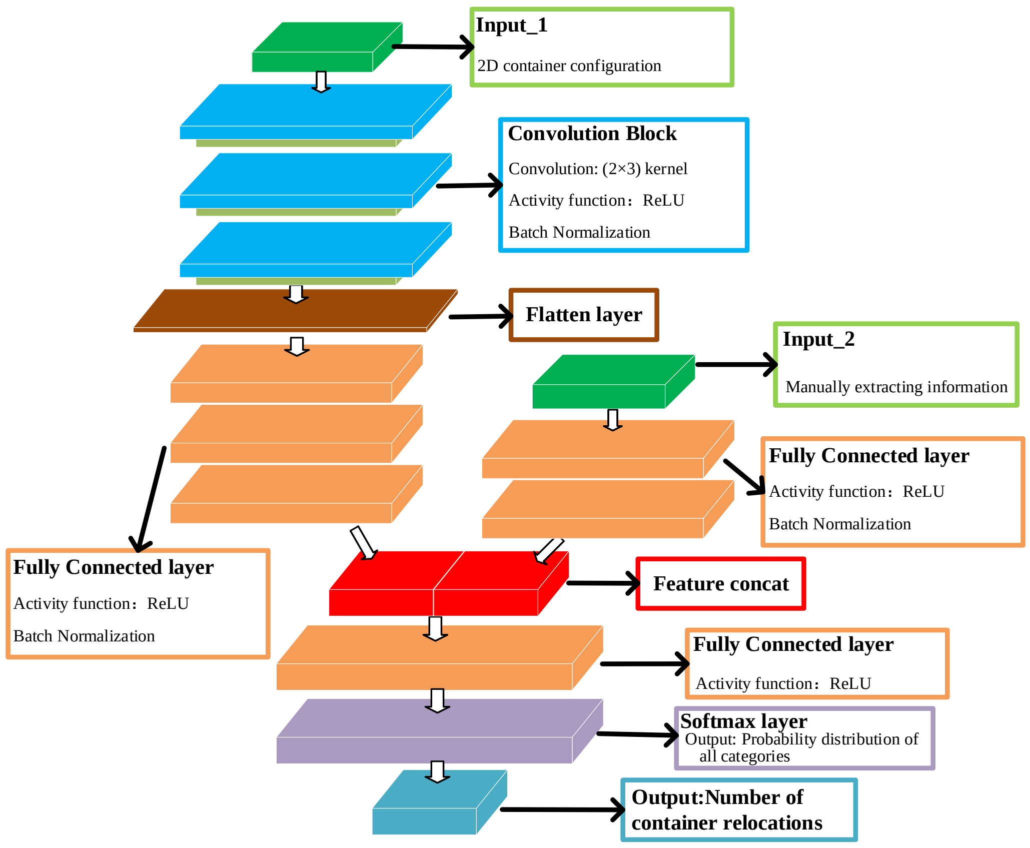

Figure 5 Network structure

We trained two separate models for the local and global features using CNN and MLP architectures, respectively. For the CNN model, we used a three-layer architecture with 32, 64, and 128 filters and a kernel size of 2x3. We applied a ReLU activation function after each convolutional layer. We used a fully connected layer with 32 units after the convolutional layers to map the extracted features to the target number of relocations. For the MLP model, we used a two-layer architecture with 64 and 32 units, respectively. We applied a ReLU activation function after each layer to produce the predicted number of relocations.

4.2 Computational Experiments

4.2.1 Data Generation

To train an effective neural network, a significant amount of data is required. We utilized three initial datasets, including CV, CV-like (generated by our custom data generator), and ZSS in Ye et al. (2022). To ensure realism, we only considered data with container configurations that adhered to certain constraints, such as a maximum of 10 stacks and 7 layers. Since the CV dataset did not have height restrictions, we imposed a height limit of H+2. Consequently, we only used data in the dataset with a maximum height of 5, which was also applied to the CV-like dataset we created. For the ZSS dataset, $H \in \{2,3\}$, $S \in \{3\}$, $N \in \{5,6,7\}$, indicating that the maximum height of each configuration in the dataset is the height limit of the corresponding configuration.

Tanaka's Branch-and-bound algorithm was utilized to obtain the optimal solution as the label for our neural network prediction. This algorithm is an iterative deepening branch-and-bound search algorithm. During the process of obtaining the optimal solution for the initial configuration, we were also able to determine how to proceed at each step. For instance, after each relocation operation, we could obtain a new configuration that had an optimal solution one less than that of the original configuration. On the other hand, after each removal operation, the new configuration that was obtained had the same optimal solution as the original configuration. To ensure that the highest priority of the container remained at 1 in each new configuration that was obtained, we subtracted 1 from the priority of all containers in the new configuration after the removal operation. This data augmentation operation helped us to expand our dataset significantly for the container tipping prediction problem.

4.2.2 Performance Metrics

We designed both regression and classification models for predicting container tipping, and in order to better evaluate the performance of our models, we used four different metrics for the regression model: R-Square, MSE (Mean Squared Error), RMSE (Root Mean Squared Error), and MAE (Mean Absolute Error); and five different metrics for the classification model: CCE (categorical cross-entropy), accuracy, F1-score, recall, and precision.

Before introducing R-Square, we first introduce the regression sum of squares (SSR) and the residual sum of squares (SSE). SSR refers to the sum of squared differences between the predicted values and the mean value of the samples after fitting the data with the regression model, representing the degree of influence of the independent variable on the dependent variable. Its formula is as follows:

$$\boldsymbol{SSR} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 \tag{4. 1}$$

Where$\hat{y}_i$ is the predicted value of the sample$i$,$\bar{y}$ is the mean value of all samples. The larger the value of SSR, the greater the influence of the independent variable on the dependent variable, and the better the fitting ability of the model.

SSE refers to the sum of squared differences between the predicted values and the true values in the regression model, representing the unexplained part of the model. Its formula is as follows:

$$\boldsymbol{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4. 2}$$

Wherey_i is the true value of the sample$i$. The smaller the value of SSE, the better the model's ability to fit the data.

SST refers to the sum of squared differences between all observed values and the mean value of the sample. It can be regarded as a measure of the overall variability of the data population. Its formula is as follows:

$$\boldsymbol{SST = \sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4. 3}$$

The relationship between the above three is:

$$\boldsymbol{SST = SSR + SSE} \tag{4. 4}$$

R^2,also known as the coefficient of determination, is a metric used to evaluate the performance of a regression model. It represents the degree to which the model fits the data, and its formula is as follows:

$$\boldsymbol{R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4. 5}$$

$$\boldsymbol{R^2 = 1 - \frac{SSE}{SST}} \tag{4. 6}$$

MSE, RMSE, and MAE are common metrics used to measure the performance of regression models. MSE represents the average of the squared differences between the predicted values and the true values. RMSE is the square root of the average of the squared differences between predicted and true values. Similar to MSE, a smaller RMSE indicates a better predictive performance of the model. Unlike MSE, RMSE is more interpretable as it has the same unit as the original data, whereas MSE is in square units. MAE represents the average of the absolute differences between predicted and true values. Unlike RMSE and MSE, MAE is not affected by outliers. If the data contains outliers, MAE can better reflect the predictive performance of the model. The formulas for these metrics are shown below:

$$\boldsymbol{MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{4. 7}$$

$$\boldsymbol{RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}} \tag{4. 8}$$

$$\boldsymbol{MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}} \tag{4. 9}$$

CCE is a commonly used loss function for multi-class classification problems, usually used to evaluate the difference between the model's predicted results and the actual results. In multi-class classification problems, if each category is treated as a binary classification problem, each sample has multiple binary classification problems. For a sample, its true label can be represented as a one-hot encoding vector$y$, where the element$i$ is 1, indicating that the sample belongs to the ith category, and the rest of the elements are 0. The model's predicted result can be represented as a probability distribution vector$\widehat{\ }$, where the ith element represents the probability that the model predicts the sample belongs to the$i$ th category. Its calculation formula is as follows:

$$\boldsymbol{CCE = -\sum_{i=1}^{C} y_i\, log(\hat{y}_i)} \tag{4. 10}$$

where$C$ is the number of categories, $y_i$represents the element of the true label vector of the sample$i$, and$\hat{y}_i$ represents the element of the model's predicted results vector. The smaller the CCE value, the better the model's prediction performance. Because the calculation formula of cross-entropy includes the logarithmic function of the predicted value, when the predicted value deviates further from the true value, its contribution to the loss function becomes greater, thereby reflecting the prediction effect of the model more sensitively.

Accuracy represents the proportion of correctly classified samples to the total number of samples. Recall represents the proportion of samples correctly classified as positive examples to the actual number of positive samples. Precision represents the proportion of samples correctly classified as positive examples to the number of samples predicted as positive examples. F1-score is the harmonic mean of Precision and Recall. The calculation formulas are as follows:

$$\boldsymbol{Accuracy = \frac{TP + TN}{TP + FP + TN + FN}} \tag{4. 11}$$

$$\boldsymbol{Precision = \frac{TP}{FP + TP}} \tag{4. 12}$$

$$\boldsymbol{Recall = \frac{TP}{FN + TP}} \tag{4. 13}$$

$$\boldsymbol{F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}} \tag{4. 14}$$

TP is the True Positive, TN is the True Negative, FP is the False Positive, FN is the False Negative. We use the Weighted-average method for all of the four metrics mentioned above, which means that different weights are assigned to different classes (the weights are determined based on the true distribution proportion of each class), and each class is multiplied by its weight before being summed up.

4.2.3 Performance

In this section, we separately validated the performance of regression model and classification model on CPMP problems.

4.2.3.1 Regression Model

For training, we used our own generated CV dataset to obtain a total of 338,538 samples after data augmentation. Then, we took 270,609 examples out of them in an 8:2 ratio for model training and used the remaining data for validation. The original CV data that was not augmented were used as test data to verify the generalization of the model. Figure 6 shows the change in Loss and R-Square during model training and validation. Table 1 shows the performance of our model tested on different datasets. The R-Square tested on our generated dataset exceeds 0.99, and the MAE is only 0.2240, which means that the average error between the predicted and actual relocation counts is within 0.23. The R-Square tested on the CV dataset also reaches 0.97. Based on the above results, we believe that our model can effectively capture the effective features of different configurations and predict reliable relocation counts for different configurations. Meanwhile, its generalization ability is also good.

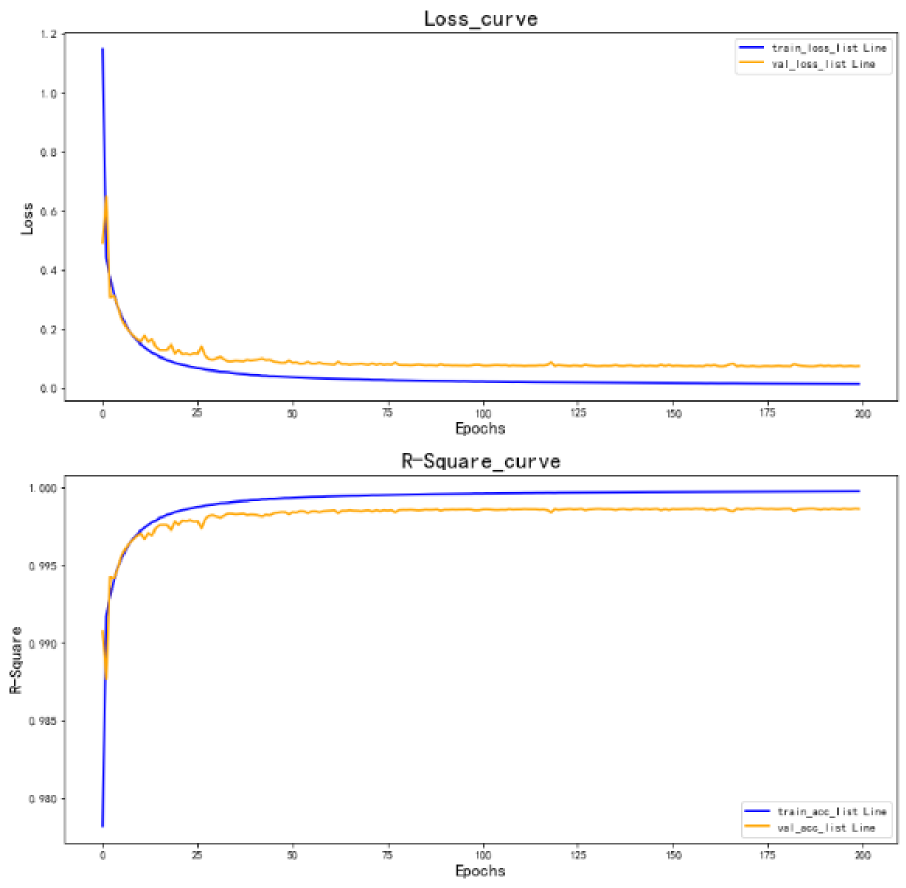

Figure 6 Loss and Accuracy Curve for Regression Model

Table 3 Performance Evaluation of the Regression Model on the Different Test Sets

problem category	Datasets	R-Square	MSE	RMSE	MAE
CPMP	CV-like	0.9933	0.1239	0.2719	0.2240
	CV	0.9744	0.7641	1.0382	1.0779

4.2.3.2 Classification Model

In this module, we conducted the following experiments: First, we trained and tested our model using the dataset we generated, and tested the model on the CV dataset. Then we trained and tested our model using the ZSS dataset to compare our model and features with those of the model and features in Ye et al. (2022). We also trained and tested the ZSS model using our feature dataset.

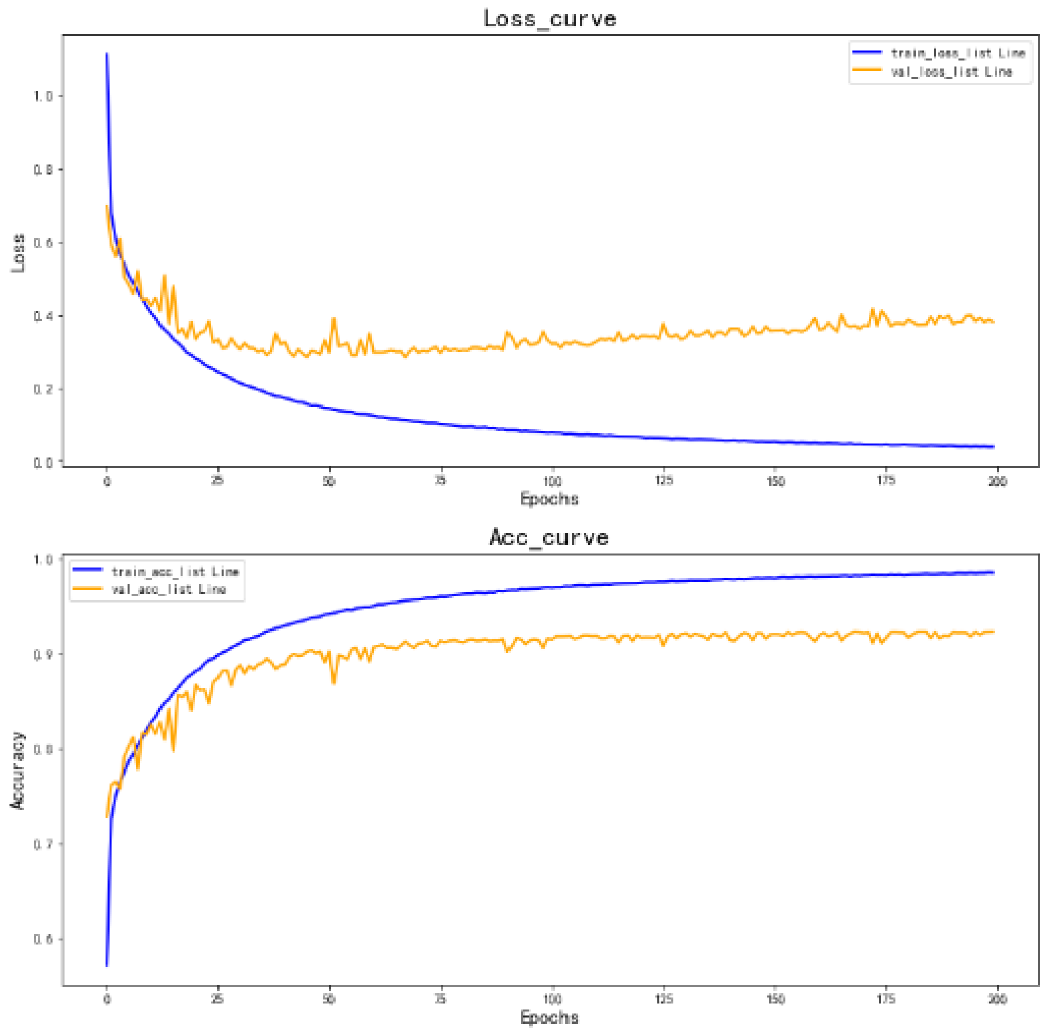

Figure 7 Loss and Accuracy Curve for Classification Model

In this module, we conducted the following experiments: 1. Trained and tested our MLP-CNN model using the same dataset used for training the regression model, and tested the model on the CV and ZSS datasets. 2. Trained and tested our model using the ZSS dataset to compare our model and the features used with those of the ZSS model and features. We also tested the ZSS model on our feature dataset.

We first trained and tested our MLP-CNN model on the same dataset used for training the regression model. The loss and accuracy curves during training are shown in Figure 7. We can see that our model achieved an accuracy of 92.49% on this dataset during testing. To validate the generalization ability of our model, we tested the trained model on the CV dataset. In practical applications, we sometimes need a highly accurate prediction of the container tipping count, while at other times we only need a rough estimate. Therefore, our model calculates the Top-1, Top-2, and Top-3 accuracy rates, and these three standards can help us better evaluate the performance of the model. Table 3 shows the test results of the trained model on the three datasets. The Top-1 accuracy rate is not very high on the CV dataset, and we believe this is because the number of larger configurations in our dataset is relatively small, so the network cannot effectively learn their features. However, the Top-3 accuracy rate can reach more than 91%.

Table 4 Performance Evaluation of the Classification Model on the Different Test Sets

problem category	Datasets	Top-1 Accuracy	Top-2 Accuracy	Top-3 Accuracy
CPMP	CV-like	92.49%	96.84%	99.50%
	CV	58.80%	80.70%	91.34%

Table 5 shows the test results of different models using different input features. MLP-CNN is the model we proposed, and RF, ET, SVM, and LR are the models mentioned in Ye et al. (2022). In the "Features" column, "New" represents the input features of our proposed model, which are the features mentioned in Table 1, and "Old" represents the input features used Ye et al. (2022). Since we did not obtain the specific parameter settings of the models in Ye et al. (2022), we improved and optimized these four models, and their performance using Old features was significantly improved compared to the data provided in Ye et al. (2022). The best accuracy achieved by their model for solving the UCRP problem was only 64%. Additionally, the models using New features obtained slightly better performance compared to those using Old features. We believe this is because the features we used removed some information that could not accurately represent the unique characteristics of the configuration, while the new features can not only represent the unique information of different configurations but also have an implicit relationship with the relocation times. Among them, the SVM model had the highest accuracy, reaching 88.32%, with F1-score, precision, and recall reaching 88%. For our MLP-CNN model, due to the limitation of the input format of the model, that is, the container configuration is input as matrix data, we only tested the effect of using New Features, and its accuracy was significantly improved, reaching 91%, and F1-score, precision, and recall also showed significant improvement. We believe that the reason for such a large improvement is that the CNN module in our model can extract the spatial implicit information of the two-dimensional container configuration well, while other models flatten the two-dimensional container configuration into one-dimensional information, which cannot extract effective spatial information.

Table 5 Performance Evaluation of Different Models and Features on ZSS Sets

problem category	Models	Features	ACC	F1	Precision	Recall
Unrestricted CRP	MLP-CNN(CPMP)	New	0.9124	0.91	0.91	0.91
	RF	New	0.8658	0.87	0.87	0.87
		Old	0.8555	0.86	0.86	0.86
	ET	New	0.8627	0.86	0.86	0.86
		Old	0.8596	0.86	0.86	0.86
	SVM	New	0.8832	0.88	0.88	0.88
		Old	0.8730	0.87	0.87	0.87
	LR	New	0.7275	0.72	0.72	0.73
		Old	0.5758	0.57	0.57	0.58

5.0 Conclusion and Future Research

In this thesis we presented CPMP-MCTS, a monte-carlo tree search method to solve container pre-marshalling problem. To our best knowledge, CPMP-MCTS is the first MCTS method ever applied in CPMP. Though the performance is slightly worse than state-of-the-art DLTS, CPMP-MCTS takes less time to find a solution. Moreover, CPMP-MCTS as a machine learning based method does not require training network, unlike DLTS. We also introduced MLP-CNN, a combined deep neural network to estimate the number of moves to find optimal solutions. Estimation of relocations helps us understand the potential workload without going through a sequence of actions. It can also serve as a tool to develop new algorithms. Compared to the model in Ye et al., our model achieves better performance in term of classification accuracy.

In the future, we would like to keep improving CPMP-MCTS. Since LPFH works well with MCTS, one obvious path is to explore other heuristic methods combined with MCTS. Another path is to substitute the heuristic with our MLP-CNN network. Because all we need for simulation is the number of relocations, MLP-CNN has the potential to further boost CPMP-MCTS without spending time finding all the optimal actions. Lastly we can apply CPMP-MCTS to other routing problems.

Bibliography

[1] UNCTAD. 2022. Container port throughput, annual, 2010-2020. https://unctadstat.unctad.org/wds/TableViewer/tableView.aspx?ReportId=13321.

[2] Świeboda, J., & Zając, M. (2016). Analysis of reshuffling cost at a container terminal. In Dependability engineering and complex systems, Vol. 470 (pp. 491–503). Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-39639-2_43.

[3] Hottung, A., K. Tierney. (2016). A biased random-key genetic algorithm for the container pre-marshalling problem. Computers & Operations Research 75 83 – 102.

[4] Hottung, A., S. Tanaka, and K. Tierney. (2020). Deep Learning Assisted Heuristic Tree Search for the Container Pre-Marshalling Problem. Computers & Operations Research, Vol.113, 2020, p. 104781.

[5] Caserta, M., Schwarze, S., Voß, S. (2011). Container Rehandling at Maritime Container Terminals. In: Böse, J. (eds) Handbook of Terminal Planning. Operations Research/Computer Science Interfaces Series, vol 49. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-8408-1_13

[6] Lee, Y., S-L. Chao. (2009). A neighborhood search heuristic for pre-marshalling export containers. European Journal of Operational Research 196(2) 468 – 475.

[7] Rendl, A., M. Prandtstetter. (2013). Constraint models for the container pre-marshaling problem. G. Kat-sirelos, C.-G. Quimper, eds., ModRef 2013: 12th International Workshop on Constraint Modelling and Reformulation. 44–56.

[8] van Brink, M., R. van der Zwaan. (2014). A branch and price procedure for the container premarshalling problem. A. Schulz, D. Wagner, eds., Algorithms – ESA 2014 , Lecture Notes in Computer Science, vol. 8737. Springer Berlin Heidelberg, 798–809.

[9] Tierney, K., D. Pacino, S. Voß. (2016). Solving the pre-marshalling problem to optimality with A* and IDA*. Flexible Services and Manufacturing Journal 1–37.

[10]Tanaka, S., K. Tierney. (2018). Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. European Journal of Operational Research 264(1) 165 – 180. https://doi.org/10.1016/j.ejor.2017.05.046.

[11]Caserta, M., S. Voß. (2009). A corridor method-based algorithm for the pre-marshalling problem. M. Giacobini et al., ed., Applications of Evolutionary Computing, Lecture Notes in Computer Science, vol. 5484. Springer, Berlin, 788–797.

[12]Exp´osito-Izquierdo, C., B. Meli´an-Batista, M. Moreno-Vega. (2012). Pre-marshalling problem: Heuristic solution method and instances generator. Expert Systems with Applications 39(9) 8337–8349.

[13]Wang, N., B. Jin, A. Lim. (2015). Target-guided algorithms for the container pre-marshalling problem. Omega 53 67–77.

[14]Jovanovic, R., M. Tuba, S. Voß. (2017). A multi-heuristic approach for solving the pre-marshalling problem. Central European Journal of Operations Research 25 1–28.

[15]R. Coulom (2006), "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in Proc. 5th Int. Conf. Comput. And Games, Turin, Italy, pp. 72–83.

[16]Jiang, T., B. Zeng, Y. Wang, and W. Yan. (2021). A New Heuristic Reinforcement Learning for Container Relocation Problem. Journal of Physics: Conference Series, IOP Publishing, Vol. 1873, No. 1, p. 012050.

[17]Wei, L., F. Wei, S. Schmitz, and K. Kunal. (2021). Optimization of Container Relocation Problem via Reinforcement Learning. Logistics Journal: Proceedings, Vol. 2021, No. 17, pp. 1–8.

[18]Ye, R., Ye, R., & Zheng, S. (2023). Machine Learning Guides the Solution of Blocks Relocation Problem in Container Terminals. Transportation Research Record, 2677(3), 721–737. https://doi.org/10.1177/03611981221117157

[19]Schadd, M.P.D., Winands, M.H.M., van den Herik, H.J., Chaslot, G.M.J.B., Uiterwijk, J.W.H.M. (2008). Single-Player Monte-Carlo Tree Search. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds) Computers and Games. CG 2008. Lecture Notes in Computer Science, vol 5131. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-87608-3_1

[20]Klein, S.C. (2015). Attacking SameGame using Monte-Carlo Tree Search : Using randomness as guidance in puzzles.