

Speeding up stowage planning with deep reinforcement learning

Christian Mølholt Jensen

`chrj@di.ku.dk`

Axel Højmark

`axho@di.ku.dk`

Abstract

For small instances, the MPSP problem has largely been solved using traditional MIP methods. However, these approaches are not scalable to real-life-sized scenarios due to their need for heavy compute and slow inference speed. To our knowledge, only one piece of prior work exists that investigates the use of deep RL on stowage planning. However they tackle a related, but different problem called the master bay planning problem. To further explore the possibility of using deep RL for stowage planning, we introduce a novel deep RL model that is able to learn placement policies that outperform MIP models on large instances with a fraction of the inference time. We show that a single model can generalize across many vessel sizes at some trade-off in performance. For larger instances we achieve up to a 90% reduction in the number of reshuffles compared to MIP models. Given these results, we believe that more research into applying deep RL for MPSP could be important in optimizing and reaching fully automated stowage planning.

1 Introduction

Over 80% of the volume of international trade in goods is carried by sea, and the percentage is even higher for most developing countries (UNCTAD, 2021). This sets transportation by sea, as a cornerstone in international trading. The drive for efficient transportation of goods has resulted in larger and larger container ships. As of June 2022, the world’s largest water vessel has a capacity of more than 24,000 twenty-foot equivalent units¹. Considering these large volumes, maximizing efficiency in container handling is crucial to the global economy, as well as to the environment since reducing unnecessary port operations results in a significantly reduced carbon footprint.

These vessels sail from port to port loading and unloading thousands of containers. Some containers should be unloaded later than others and should therefore be accessible earlier. The containers are stored in stacks so they can be removed only from the top. If a given container must be unloaded in a port, but it is not directly accessible, then the containers above it must be removed first. This movement is called reshuffling.

Minimizing the time spent reshuffling requires a plan of action, referred to as a stowage plan. This plan describes how the containers should be unloaded and loaded. It has to satisfy a wide range of constraints such as ship stability, hull strength, crane limitations, special container requirements, varying container sizes, and many more. This makes creating an effective stowage plan an incredibly complex task. The problem has been shown to be NP-complete, so finding strong heuristic measures are critical (Avriel et al., 2000).

In this project, we will study the multi-port stowage planning problem (MPSP) considering the whole route of the ship and the different sets of containers that must be loaded and unloaded at various ports along the route. For small instances, the MPSP problem has largely been solved using traditional heuristic methods (Parreño-Torres et al., 2019) however to perform this search extensive computational resources had to be utilized. Moreover, as the paper mentions to solve real-life problems features such as stability, container type, and many other features have to be considered. Although these features can be included in the integer linear model presented by (Parreño-Torres et al., 2019), the results obtained clearly indicate that the extended model will not be able to solve real-world instances.

We see the following possibilities in using reinforcement learning for Stowage Planning:

1. **Performance.** Deep reinforcement learning has shown state-of-the-art results in combinatorial problems like chip placement (Mirhoseini et al., 2020; Lai et al., 2022), matrix multiplication (Fawzi et al., 2022), and many more.
2. **Scalability.** Deep Reinforcement learning has the potential to scale to massive ship sizes whilst providing stowage plans quickly at inference time.

¹A twenty-foot equivalent unit (TEU) is a unit of measurement used to determine cargo capacity for container ships and terminals. This measurement is derived from the dimensions of a 20ft standardized shipping container.

3. **Reusability.** A pretrained model can be used as a starting point for later iterations. This allows for faster and cheaper domain adaptations.
4. **Applicability.** Real-world constraints can be introduced into the model, allowing it to produce true stowage plans.

2 Related Work

The rich literature on the stowage planning problem can primarily be classified into two main categories: single-port and multi-port.

In the single-port problem, the container vessel is loaded once at the initial port, and exclusively unloaded at subsequent ports (Ambrosino et al., 2004). The majority of the literature related to stowage planning belongs to this category and is often also referred to as the Master Bay Planning Problem (MBPP). For a more detailed description together with a thorough description of relevant real-world constraints see Larsen and Pacino (2021). The most popular approaches to MBPP have generally been genetic algorithms (Bazzazi et al., 2009; Weiss et al., 2017) and multi-objective mixed integer programming (MIP) (Ambrosino et al., 2004; Moura et al., 2013; Imai and Miki, 1989). However, as the problem size increases, MIP approaches usually leads to infeasible run times.

In the multi-port problem, the containers can also be added to the vessel at subsequent ports. Due to the complexity of this problem, heuristics are often utilized. In Parreño-Torres et al. (2019, 2021) various heuristics algorithms are presented together with an open-sourced dataset allowing for direct model comparison. In Junqueira et al. (2022) they tackle the problem utilizing a genetic algorithm.

To our knowledge, only one piece of work exists that tackle stowage planning with Deep RL (Shen et al., 2017). They research MBPP including a subset of real-world constraints such yard crane shifting and weight constraints into their optimization objective. They train a Deep Q-learning agent (Mnih et al., 2013) capable of designing stowage plans and testing them on production data. However, they only test their model on three small instances, in which human port planners systematically outperform the agent. Their main advantage point is speed. The evaluated plans take two to three minutes on average for human planners, whilst their model completes them in less than one second.

We see several possible points of improvement. Firstly, the input features are very condensed approximations of stowage states. This leaves the model with an unclear picture of the current stowage plan. Secondly, they allow the agent to perform illegal actions with large penalties, instead of masking the actions. It has been shown that the common technique of giving a negative reward when an invalid action is issued often fails to scale (Huang and Ontañón, 2022). Thirdly, it is hard to assess the potential of the model, when tested on so few instances.

Due to the lack of research on deep RL for MPSP, we looked elsewhere for inspiration. In Mirhoseini et al. (2020) they present a learning-based approach to chip placement, one of the most complex and time-consuming stages of the chip design process. They utilize Proximal Policy Optimization (Schulman et al., 2017), a deep RL algorithm, to generate placements that are superhuman or comparable, whereas existing baselines require human experts in the loop and take several weeks. We see several similarities between our MPSP problem formulation and chip placement:

- They are both sequential decision-making problems, where current actions may have great implications for all future moves.
- They both operate on an action space with many invalid actions.
- Both states have elements that are interpretable as graphs.

- Both state spaces are enormous in magnitude.

Due to their relatedness, we apply many of the ideas from [Mirhoseini et al. \(2020\)](#) throughout this paper.

3 Problem Description

We will be using a similar notation to [Avriel et al. \(1998\)](#) and [Parreño-Torres et al. \(2019\)](#). We consider a ship traveling along a route with N ports. We assume that all containers are the same size and that no additional loading constraints like the stability of the ship are considered. We avoid detailed models of stacks and vessel stability constraints for two reasons. First, these do not change the theoretical difficulty of the problem and, second, the loading and unloading of containers apply to both ports and vessels, meaning stability constraints are not always relevant ([Tierney et al., 2014](#)). We, therefore, consider the ship only containing a single bay B consisting of R rows and C columns. This is without loss of generality, as this structure can easily be mapped to 3-dimensions. The position of each container on the ship can therefore be described with the pair (r, c) where $r \in \{1, \dots, R\}$ and $c \in \{1, \dots, C\}$. This gives a maximum capacity of $R \cdot C$.

The loading/unloading operations along the route are defined by an $N \times N$ transportation matrix T whose input (i, j) is the number of containers originating at port i with destination port j . T is therefore a non-negative upper triangular matrix, so $T_{ij} = 0$ for all $i \geq j$. The set of containers to be added at a port is referred to as a load list denoted by L .

The ship starts at port 1 and sequentially visits port 2, 3, \dots , N . At each port $i = 1, \dots, N - 1$, containers with destination i are unloaded and containers bound for destinations $j = i + 1, \dots, N$ are loaded. We denote a container whose destination port is j as a j -container.

If there is a j -container in position (r, c) and there is a k -container, $k < j$, in some position (r', c) with $r' < r$, the first container is said to be a blocking container, and will always result in a reshuffle. The objective of MPSP is to minimize the number of reshuffles along the route.

A transportation matrix is said to be feasible if and only if ([Parreño-Torres et al., 2019](#)):

$$\sum_{k=1}^i \sum_{j=i+1}^N T_{kj} \leq R \cdot C \quad \forall i \in \mathcal{N} : i < N$$

Meaning that the ship will never be required to load a number of containers larger than its capacity. The feasibility of a given transportation matrix can be easily checked and thus we assume throughout this paper that T is feasible.

4 Modelling the problem

We will take a deep reinforcement learning approach to the stowage problem where an RL agent will load and unload the containers sequentially.

Most RL problems can be formulated as Markov Decision Processes (MDPs). MDPs consist of four key elements

- *States*: The set of possible world configurations of the problem.
- *Actions*: The set of actions that can be taken by the agent in a given state.
- *Transitions*: Given a state and an action, the transitions describe the probability distribution over the next possible states.
- *Rewards*: The reward measures the quality of an action, which can both be negative and positive.

The following subsections will go into detail about each of these key elements for our problem formulation.

4.1 State

In our case, the states correspond to every joint pair of transportation matrix T of size $N \times N$ and bay matrix B of size $R \times C$. Note that the size of T shrinks throughout the route. At each port i , the transportation matrix will be of size $(N - i + 1) \times (N - i + 1)$. This is practical as no information is needed for offloaded containers. To match the new transportation matrix, all j -containers are now $(j - i + 1)$ -containers. From the agents point of view, this means it never leaves the initial port.

Since we do not consider stability constraints, the ordering of the columns can be swapped arbitrarily. To further reduce the state space we swap the columns of the bay matrix so they are at all times lexicographically ordered from bottom to top. That being said, any arbitrary ordering would suffice in making the same state space reduction. See appendix B for derivations regarding this reduction.

An episode ends when the terminal state is met. In our case, this corresponds to all containers having been offloaded, leaving B and T completely empty.

4.2 Action

For an MDP the action space encompasses the possible choices of the agent. In our case the agent has, depending on the state, between C and $2C$ actions to choose from. For every column, the agent can choose either to load or unload a container. Exceptions are empty columns, where containers cannot be removed, and full columns, where containers cannot be added. As one can only access the top of the stack, it is unambiguous, given a column, where the new container is to be added, or which container is to be removed. The destination port j of the added container is also unambiguous. The containers are stowed in decreasing order so that those going to ports visited later are stowed first.

4.3 Transition

The transitions of an MDP describe the probability distribution over the next states given an action and a current state. The MDP studied is deterministic and all transitions have a probability of 1. This means the MDP can essentially be viewed as a graph, with states (nodes) connected by actions (edges).

Below we will denote the transportation matrix and bay matrix obtained after the transition by T' and B' . When describing alterations, it is implicit that T' and B' are initialized as T and B respectively. We have three kinds of transitions:

1. **A container is removed.**

B' is updated, by removing the upper most container in the respective column. The container is then reinserted into the transportation matrix by increasing T'_{1j} by one, where j was the destination port of the container.

2. **A container is added and $|L| > 1$.**

B' is updated by adding a j -container top down in the relevant column. The value of T'_{1j} is decreased by one. Since containers are added in decreasing order by their destination port, j is the last column of T with a non-zero entry in the first row (the current port).

3. **A new container is added and $|L| = 1$.**

We start out by altering B' and T' as described in transition 2. The load list is now empty which is equivalent to every value of the first row in T' being zero. The ship must now sail along to the next port. The transportation matrix shrinks, and the container destinations are renumbered as described in 4.1. All containers with destination 1 (containers destined to the current port) together with any blocking containers are removed from B' . The 1-containers are discarded and every blocking container is reinserted into T' as described in transition 1. This procedure is done recursively, until reaching a terminal state or a port where $|L| > 0$.

From these transitions it becomes clear that our MDP has a potentially infinite time horizon. An agent choosing to add and remove containers repeatedly would never reach a terminal state.

4.4 Reward

The reward in an MDP is the measure that the agent is trying to maximize. We are interested in minimizing the total number of reshuffles. The reward is modeled, so that the cumulative negative reward over an episode, matches the exact number of reshuffles. This means the sum total reward is bounded between $(-\infty, 0]$. In order to make the reward signal as dense as possible, we punish the model the moment it places a blocking container, rather than delaying it to the moment it is offloaded (this may be many ports down the line). Our rewards are:

- A reward of 0 for every non-blocking container added.
- A reward of -1 for every blocking container added.
- A reward of -1 when a non-blocking container is removed

We omit any negative reward when removing blocking containers, as the model was already punished at its placement.

4.5 Example Episode

We refer to appendix C for an illustrative episode showcasing the interactions described above.

5 Deep Reinforcement Learning Model

We propose a novel neural architecture enabling us to train placement policies for stowage planning, whilst allowing transfer learning between varying-sized bays and number of ports. We use Proximal Policy Optimization, which is an online actor-critic reinforcement learning algorithm (Schulman et al., 2017). Since its formulation in 2017 it has shown remarkable performance on many problem domains (Mirhoseini et al., 2020; Lai et al., 2022; Schulman, 2022). As an actor-critic algorithm, the critic evaluates the current policy and the result is used in the policy training. The actor implements the policy and it is trained using Policy Gradient with estimations from the critic. PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring that the deviation from the previous policy is relatively small. For a well-tested and documented implementation of PPO, we utilized the Stable-Baselines3 python package (Raffin et al., 2021).

5.1 Embedding

Training a policy network is a challenging task for a state space this large. To address this challenge, we first focused on learning rich representations of the state space.

Firstly we tried to use a graph convolutional network to encode the transportation matrix as done in Mirhoseini et al. (2020). We would represent each port as a node N_i and connect nodes N_i and N_j with an edge of weight T_{ij} if $T_{ij} > 0$. This is a very natural representation since the transportation matrix is in essence an adjacency matrix. It also has a nice property, as opposed to the other methods proposed, that it is compatible with arbitrarily large transportation matrices. See appendix D for more details on this.

We also experimented with alternate representations of the transportation matrix. One idea was to transform the transportation matrix into one long loading list. After applying a learned embedding, we would then encode the sequence using an LSTM (Hochreiter and Schmidhuber, 1997). See appendix E for more details on this.

We also tried out a basic feedforward network. We would extract the upper triangular part of T , flatten it, and feed it through the network resulting in a representation of size d .

For all of the above, we tried out many hyperparameter combinations². However, none of these techniques seemed to outperform the following architecture.

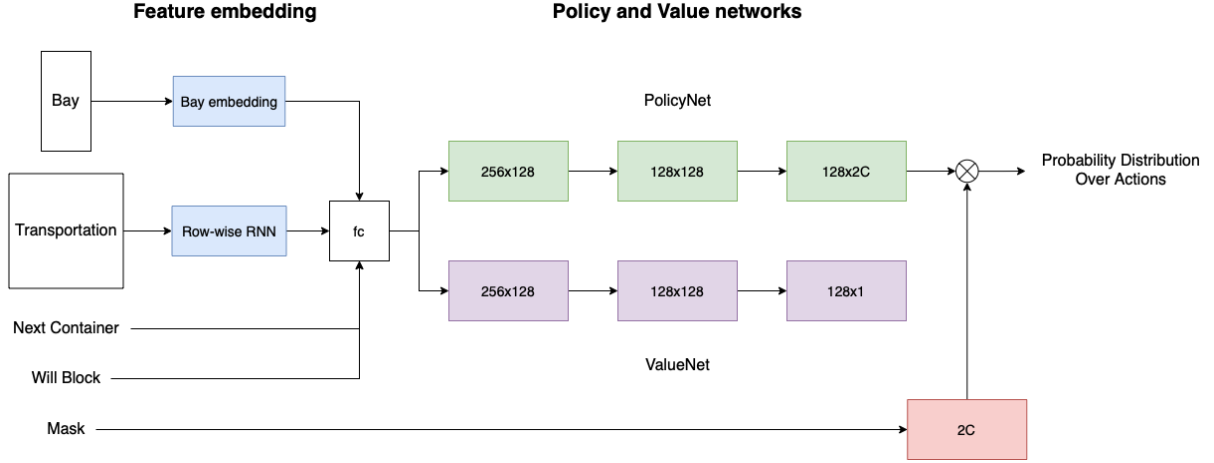


Figure 1: Policy and value network architecture. An embedding layer encodes information about B , T , R and C . The policy and value networks then output a probability distribution over the action space and an estimate of the expected reward for the current placement, respectively.

We start out with the bay matrix B of size $R \times C$. The matrix is then transposed such that B is now $C \times B$. Afterward, a fully connected layer is applied across the columns obtaining $C \times D_{fc1}$ where $D_{fc1} = 128$. The idea here is to encode B column-wise at the beginning of the network and combine these representations deeper into the model. We flatten the tensor and apply the tanh non-linearity, giving the final bay embedding of length $C \cdot D_{fc1}$.

For the transportation matrix we apply a single-layered RNN (Elman, 1990) with a hidden size of $D_{RNN} = 128$ across the rows. This means the transportation matrix is interpreted as a sequence of N elements (rows), each of which is a vector of size N . By doing this we hope to capture the sequential nature of sailing from port to port, where the loading procedure is in many ways the same each time. Extracting the last hidden state of the RNN gives an encoding of the transportation matrix.

Additionally, we included two pieces of meta-information as input features. The first was the next j -container to be placed. This would simply be an integer in the range 2 to N . The second was a binary vector of length C . An entry expressed whether the next container placed would be blocking if inserted in the corresponding column. Both pieces of information can be inferred solely from B and T , and should as a result be learnable by the agent without being given explicitly. We found however that including these lead to faster convergence.

Finally, to combine all of the above, we used a fully connected layer to condense the state to $D_{fc2} = 256$. This is a shared feature representation used by both the policy and value network. The policy and value network both have two hidden layers with $D_{PV} = 128$ neurons.

The mask is a sequence of zeros and ones of length $2C$, where one corresponds to a valid action, and zero corresponds to an invalid action. We interpret the first C values as addition of containers, and the last C values as removal of containers in the corresponding columns. To clarify, the validity of `ADD[5]` is expressed in the 5th value of the mask, whilst the validity of `REMOVE[5]` would correspond to the $(C + 5)$ th spot³. The action probabilities are interpreted in the same manner. To obtain valid action probabilities, the probability distribution is therefore multiplied element-wise with the mask and the entire

²See appendix G for an overview of the parameters varied.

³See appendix C for more context on the ADD and REMOVE notation.

distribution is re-normalized.

5.2 Transfer learning

It would be most desirable to have a single model which could generate high-quality stowage plans across differently sized water vessels and port numbers, instead of having to train separate models for each individual case. We achieve this by manipulating the bay as described below.

Training			Evaluation		
$R = 3, C = 3, N = 5$			$R = 2, C = 2, N = 5$		
				4	5
		4	2	4	5
	2	4	5	5	5

Figure 2: Example of utilizing dummy containers in the bay matrix to allow evaluation on smaller instances. The containers destined for port 5 are inserted to decrease the number of rows and columns by one.

Specifically, the model is initialized to match the largest evaluation/training instance in input and output size, and can then be utilized on all smaller instances. In the case where the current evaluation instance has fewer columns than the max capacity of the model, we fill columns from right to left with dummy containers going to port N . This is done until the number of open columns matches the smaller instance. The same logic is applied to the rows. See figure 2 for an example.

The mask is also manipulated, such that the model is disallowed to remove any of the dummy containers. The transportation matrix has had no alterations in these cases. It is simply sparser than normally seen during training.

6 Training

In order to match the test data from [Parreño-Torres et al. \(2019\)](#) we will limit ourselves to:

$$N \in \{4, \dots, 16\}$$

$$R \in \{6, \dots, 12\}$$

$$C \in \{2, \dots, 12\}$$

Whilst these instances are small compared to real-world examples, they prove to have enormous state spaces. For the largest instance with $N = 16$, $R = 12$, $C = 12$ we find that 10^{160} lower bounds the total number of states⁴. This is orders of magnitude larger than games like Chess and Poker ([Perolat et al., 2022](#)). See appendix I for more details on the test data composition.

We will be training several different agents. Our overall model is cross-trained on all the test configurations of N , R , and C and afterward evaluated on the whole test set. This is done to test the generalization ability of a single agent across many problem sizes. We also train a couple of "singular" models focused on a single choice of N , R , and C to see how the generalization across sizes impacts performance.

For the overall model, we randomly draw a combination of R , C , and N at the start of each episode of training. We then generate T as described in appendix F, and, if needed, adapt B according to 5.2. We use the same transportation matrix generation algorithm for the singular models.

⁴See appendix B for more details on this.

We found that the agents had a hard time learning to employ the remove action without additional guidance. During the early parts of exploration, the agent would experience that using remove almost always leads to more reshuffles and would consequently hardly explore this option further. This makes sense, as removing a container can be very effective, but only in highly specific cases. Randomly using remove, will almost never result in anything interesting. To circumvent this we altered the masking of the remove actions during training. We would only allow removal, in columns for which the next container, if placed, would be blocking. This helps the agent recognize cases, where placing a whole stack of blocking containers in a column, is far more costly than starting off by removing the bottommost containers. We found empirically that this restriction decreased reshuffles significantly.

The hyperparameters used for the models were an entropy coefficient of 0, a learning rate of 0.0001, a batch size of 64, and the discount factor set to 0.99. Moreover, the number of steps to run for each environment per update was set to 2048, and the number of epochs when optimizing the surrogate loss to 3. Lastly, the advantage was normalized during training⁵. The hyperparameters used were chosen on the basis of a small hyperparameter search. However, it was difficult to make decisive conclusions based on the shorter hyperparameter search, as we found the models generally had to be trained for several hours before converging. For more information on the hyperparameter search see appendix G.

All models were trained for 70 million time steps. This takes approximately 10 hours on Apple’s M1 Chip.

7 Results

The average reward during training against training time for the overall model can be seen in figure 3. It seems the model has mostly converged within 5 hours with slight gains when training beyond that. As reference, a random policy achieves an average reward of around -140 on this setup.

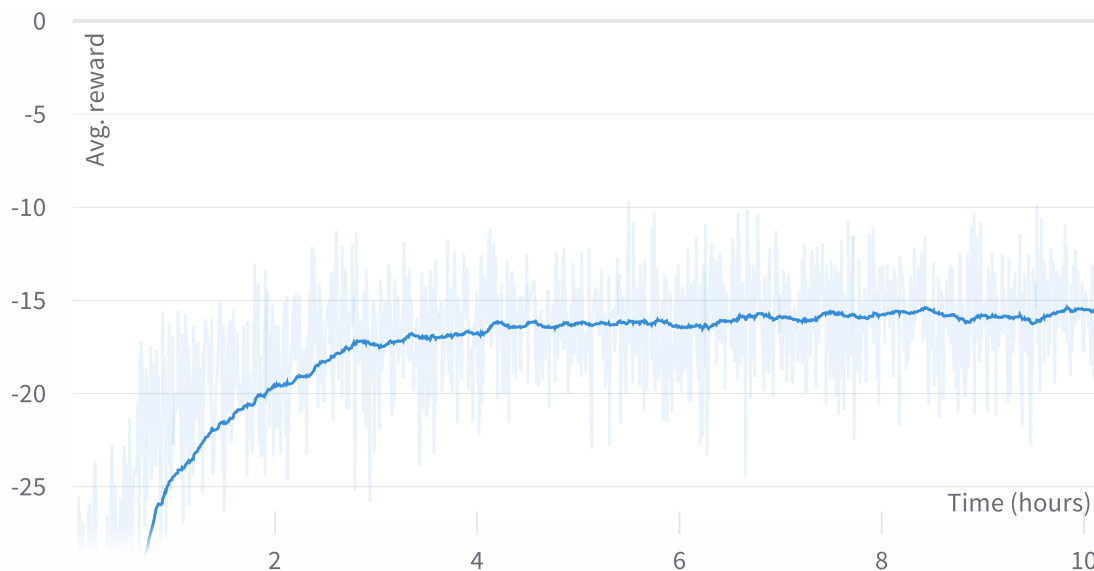


Figure 3: The average training reward over the 100 last episodes over time for the overall model. The dark blue line shows the exponential average with a smoothing of 0.9. To see an unzoomed version of the plot we refer to appendix H

After training the overall model we evaluated it on the aforementioned test data. During evaluation, all the

⁵For more information on these parameters see the [Stable-Baselines3 documentation for PPO](#).

agents would deterministically pick the most probable action rather than sampling actions according to the probability distribution as done in training.

The results from the overall model can be seen in table 1. It should be noted that the models from (Avriel et al., 2000), (Ding and Chou, 2015) and (Parreño-Torres et al., 2019) are run on 576 cores with 3 TBytes of RAM whereas our model is run on a single core on Apple’s M1 chip. Furthermore, we did not focus our efforts much on optimizing the model and environment, and so we expect that one could achieve significantly faster inference and training times.

N	Avriel et al.		Ding and Chou		Parreño-Torres		Our model	
	Res.	CPU	Res.	CPU	Res.	CPU	Res.	CPU
4	0.1	0.2	0.1	0.2	0.1	0.1	3.2	0.1
6	0.6	5.7	0.6	22.7	0.6	1.1	13.6	0.1
8	1.2	241.8	1.2	311.8	1.2	116.7	23.7	0.2
10	2.5	1342.2	3.1	1927.6	2.5	1038.4	49.1	0.2
12	6.3	2516.1	26.8	2873.1	7.4	2172.6	43.4	0.3
14	64.1	3001.7	116.7	3178.4	19.1	2743.8	51.4	0.3
16	269.7	3359.8	276.5	3422.7	68.0	3266.6	61.5	0.3
Total	49.2	1495.3	60.7	1676.6	14.1	1334.2	35.1	0.2

Table 1: Performance of the models from Avriel et al. (2000), Ding and Chou (2015) and Parreño-Torres et al. (2019) compared to our model on the evaluation data grouped by the number of ports (N). Each group has been averaged over 120 instances. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

We see that for most problem sizes our model falls short of Parreño-Torres et al. (2019). This is somewhat to be expected. On small instances, well-optimized search algorithms are able to explore large parts of the state space, and find an optimal solution in a reasonable timeframe. Our model does not have the same inbuilt search capabilities, and so missing a single optimal move will result in additional reshuffles.

However, when the problem size increases, we see that the number of reshuffles rapidly grow for the MIP models. This happens as the MIP models run out the allotted one hour of CPU time per instance. For $N = 16$ almost all models are interrupted with an average compute times of well over 3000 seconds across the board. When $N = 14$ we are able to outperform Avriel et al. (2000), and Ding and Chou (2015), and when $N = 16$ we beat all three algorithms by at least 7.5 reshuffles on average.

In Fig 4 we see the average amount of reshuffles of Parreño-Torres et al. (2019) and our model plotted against N . It seems as though the results from Parreño-Torres et al. (2019) tend towards exponential growth whereas our model looks to be linear. This is an interesting property if proved to be true. Although the pattern fits, we deem this too few data points to be conclusive.

To test how well the overall model performed compared to singular models we investigated two settings. The first was a moderately challenging case with $R = 10$, $C = 4$, and $N = 10$ and the second was the hardest configuration of the evaluation data with $R = 12$, $C = 12$ and $N = 16$.

$R = 10, C = 4, N = 10$			$R = 12, C = 12, N = 16$		
Singular	Overall	Parreño-Torres	Singular	Overall	Parreño-Torres
11.3	24.2	4.7	74.4	141.2	753.0

Table 2: Average number of shifts over 5 instances, for the Singular, Overall and Parreño-Torres et al. (2019) model.

From table 2 we see that training on a single configuration instead of cross-training roughly halves the number of reshuffles needed. This indicates, that the model is somewhat impeded when required to

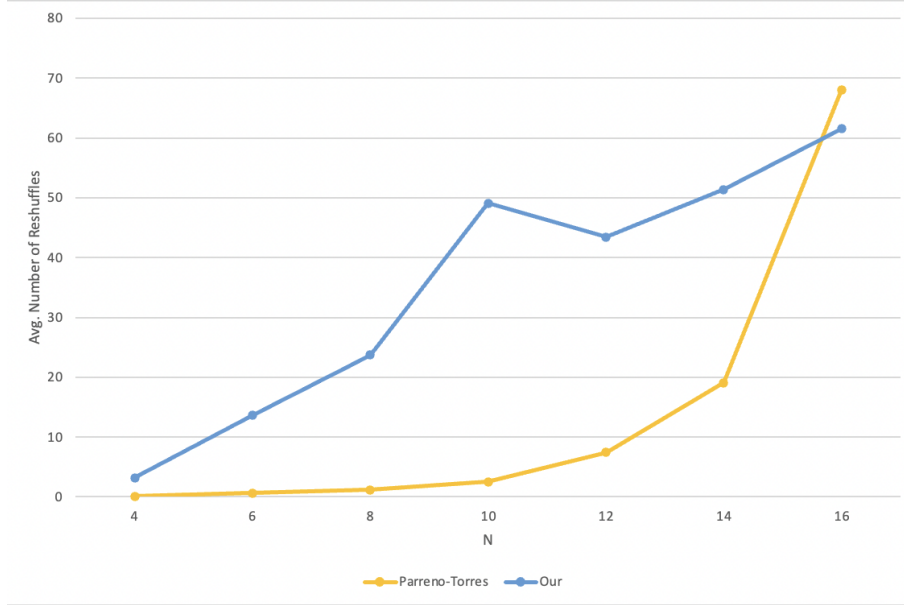


Figure 4: The average amount of reshuffles on the evaluation data grouped by the number of ports against the number of ports. Each group has been averaged over 120 instances.

generalize across multiple settings. Furthermore, we manage to generate stowage plans with less than one tenth of the reshuffles of [Parreño-Torres et al. \(2019\)](#) on the hardest settings.

8 Discussion of Results

Our results seemed to show, that cross-training significantly under-performed the singular models. It may be that the overall model had too few parameters to generalize across sizes, or that it requires more training examples to learn high quality stowage planning across settings. Although the reward over time graph indicated that the model had converged, we noticed that performance on the evaluation benchmark could continue rising slightly even after the graph had plateaued.

The difference in performance could also lie in our representation of the bay matrix. Instead of encoding the container as a integer between 1 and N , one idea could be to simply represent the containers as continous values between 0 and 1. Each j -container could be represented by the value j/N , indicating its relative destination. As a consequence, the container destined for the last port would for example always be represented by a one. Hopefully this could allow the model to more easily exploit facts like containers going to the first and last destination, should be treated much the same no matter the number of ports.

9 Future work

For future work, we see utilizing current solvers to help guide the training process as likely to be beneficial. One way this could be achieved could be in the form of imitation learning ([Ho and Ermon, 2016](#)). Here the goal would be for the model to extract domain knowledge from expert trajectories. Another way current solvers could be utilized is by initially restating the online decision problem into a supervised learning problem ([Mirhoseini et al., 2020](#)). One could create a large dataset of transportation matrices for various N , R , C and their (near)optimal number of shifts. You could then train various networks and architectures on this dataset with two immediate upsides. Firstly, if your model performs well on the supervised learning task, then you can expect that your chosen architecture is well suited for encoding the transportation matrix. And secondly, one can then reuse the model as a warm state for the feature encoder of the policy and value network.

Regarding future modeling approaches to MPSP, we see PPO as a strong candidate. Even though we did not see any improvement in utilizing a graph convolutional network, we still believe exploring this approach further could be valuable due to its natural connection to the transportation matrix. Ideally one would also want a way to represent arbitrary-sized bay matrices. Vision transformers have shown state-of-the-art results on several image classification tasks and could be a possible future direction for size-agnostic representations of the bay matrix (Dosovitskiy et al., 2020).

It could also be interesting to see how AlphaZero would fare on MPSP (Silver et al., 2017). The algorithm was used to beat Go, Chess, Shogi and also recently used to optimize matrix multiplication (Fawzi et al., 2022). This could be an interesting direction since all these problems like MPSP requires careful long term planning.

Investigating the zero-shot performance of these models could also be of great value. For example, how well does training on N ports, transfer to $2N$ ports? If the transfer is strong, then this could be one possible way of reducing training time and effectively scaling these models to real-life scenarios. It would also be fascinating to see, whether the observed linear scaling would hold for larger problem sizes, and how this compares to the scaling of other stowage planning algorithms.

10 Conclusion

Our contributions are threefold. We firstly introduce a novel MDP formulation of the multi-port stowage problem. Secondly we propose a deep RL architecture, utilizing rich representations of the state space. And thirdly we show that the proposed model can produce high quality stowage plans on larger problem sizes whilst outperforming three different MIP algorithms. On one instance we achieve over 90% reduction in the number of reshuffles when compared to the best of the three MIP algorithms. That being said, our models is still far inferior on small instances, where the MIP models are able to find optimal solutions within reasonable time frames. We also observe that generalizing across vessel sizes comes at a cost. Our overall model shows about half the performance of models trained on a singular setting. We believe our work, showcases the strengths of reinforcement learning for stowage planning, and the possible directions that can be taken to further investigate its role in reaching fully automated stowage planning.

References

- Daniela Ambrosino, Anna Sciomachen, and Elena Tanfani. 2004. [Stowing a containership: the master bay plan problem](#). *Transportation Research Part A: Policy and Practice*, 38(2):81–99.
- Mordecai Avriel, Michal Penn, and Naomi Shpirer. 2000. [Container ship stowage problem: complexity and connection to the coloring of circle graphs](#). *Discrete Applied Mathematics*, 103(1):271–279.
- Mordecai Avriel, Michal Penn, Naomi Shpirer Belfer, and Smadar Witteboon. 1998. [Stowage planning for container ships to reduce the number of shifts](#). *Annals of Operations Research - Annals OR*, 76:55–71.
- Mohammad Bazzazi, Nima Safaei, and Nikbakhsh Javadian. 2009. [A genetic algorithm to solve the storage space allocation problem in a container terminal](#). *Computers Industrial Engineering*, 56(1):44–52.
- Ding Ding and Mabel C. Chou. 2015. [Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts](#). *European Journal of Operational Research*, 246(1):242–249.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. [An image is worth 16x16 words: Transformers for image recognition at scale](#).
- Jeffrey L. Elman. 1990. [Finding structure in time](#). *Cognitive Science*, 14(2):179–211.
- Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. 2022. [Discovering faster matrix multiplication algorithms with reinforcement learning](#). *Nature*, 610:47–53.

- Jonathan Ho and Stefano Ermon. 2016. [Generative adversarial imitation learning](#).
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. [Long short-term memory](#). *Neural computation*, 9:1735–80.
- Shengyi Huang and Santiago Ontañón. 2022. [A closer look at invalid action masking in policy gradient algorithms](#). *The International FLAIRS Conference Proceedings*, 35.
- Akio Imai and Tatehiko Miki. 1989. A heuristic algorithm with expected utility for an optimal sequence of loading containers into a containerized ship. *The Journal of Japan Institute of Navigation*, 80:117–124.
- Catarina Junqueira, Anibal Tavares de Azevedo, and Takaaki Ohishi. 2022. [Solving the integrated multi-port stowage planning and container relocation problems with a genetic algorithm and simulation](#). *Applied Sciences*, 12(16):8191.
- Thomas N. Kipf and Max Welling. 2016. [Semi-supervised classification with graph convolutional networks](#).
- Yao Lai, Yao Mu, and Ping Luo. 2022. [Maskplace: Fast chip placement via reinforced visual representation learning](#).
- Rune Larsen and Dario Pacino. 2021. [A heuristic and a benchmark for the stowage planning problem](#). *Maritime Economics and Logistics*, 23:94–122.
- Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2020. [Chip placement with deep reinforcement learning](#).
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. [Playing atari with deep reinforcement learning](#).
- Ana Moura, Jorge Oliveira, and Carina Pimentel. 2013. [A mathematical model for the container stowage and ship routing problem](#). *Journal of Mathematical Modelling and Algorithms*, 12.
- Consuelo Parreño-Torres, Ramon Alvarez-Valdes, and Francisco Parreño. 2019. [Solution strategies for a multiport container ship stowage problem](#). *Mathematical Problems in Engineering*, 2019:1–12.
- Consuelo Parreño-Torres, Hatice Çalık, Ramon Alvarez-Valdes, and Rubén Ruiz. 2021. [Solving the generalized multi-port container stowage planning problem by a matheuristic algorithm](#). *Computers Operations Research*, 133:105383.
- Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. 2022. [Mastering stratego, the classic game of imperfect information](#).
- Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. [Stable-baselines3: Reliable reinforcement learning implementations](#). *Journal of Machine Learning Research*, 22(268):1–8.
- Christina Kim Jacob Hilton Jacob Menick Jiayi Weng Juan Felipe Ceron Uribe Liam Fedus Luke Metz Michael Pokorný Rapha Gontijo Lopes Shengjia Zhao Arun Vijayvergiya Eric Sigler Adam Perelman Chelsea Voss Mike Heaton Joel Parish Dave Cummings Rajeev Nayak Valerie Balcom David Schnurr Tomer Kaftan Chris Hallacy Nicholas Turley Noah Deutsch Vik Goel Jonathan Ward Aris Konstantinidis Wojciech Zaremba Long Ouyang Leonard Bogdonoff Joshua Gross David Medina Sarah Yoo Teddy Lee Ryan Lowe Dan Mossing Joost Huizinga Roger Jiang Carroll Wainwright Diogo Almeida Steph Lin Marvin Zhang Kai Xiao Katarina Slama Steven Bills Alex Gray Jan Leike Jakub Pachocki Phil Tillet Shantanu Jain Greg Brockman Nick Ryder Schulman, Barret Zoph. 2022. [Chatgpt: Optimizing language models for dialogue](#).
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. [Proximal policy optimization algorithms](#).
- Yifan Shen, Ning Zhao, Mengjue Xia, and Xueqiang Du. 2017. [A deep q-learning network for ship stowage planning problem](#). *Polish Maritime Research*, 24.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. [Mastering chess and shogi by self-play with a general reinforcement learning algorithm](#).

Kevin Tierney, Dario Pacino, and Rune Møller Jensen. 2014. [On the complexity of container stowage planning problems](#). *Discrete Applied Mathematics*, 169:225–230.

UNCTAD. 2021. [Review of maritime transport](#).

Miri Weiss, Vitor Coelho, Adi Dahan, and Izzik Kaspi. 2017. [Container vessel stowage planning system using genetic algorithm](#). pages 557–572.

11 Appendix

A Source Code and Evaluation Data

All the source code can be found at <https://github.com/hojmax/RL-MPSP>

The benchmarking data is obtained from the authors of [Parreño-Torres et al. \(2019\)](#). They are available from the author (consuelo.parreno@uv.es) upon request.

B Magnitude of State Space

In order to obtain a lower bound on the size of the state space, we count the total number of bay matrices. As a single bay matrix has numerous possible transportation matrices, this results in a very loose bound. Nonetheless, we observe that the number of states grows exponentially no matter the parameter increased.

First, let us examine the number of bay configurations possible using the naive representation. We can utilize three facts for this calculation:

1. Containers are always stacked bottom up.
2. For an MDP with N ports there exists $N - 1$ unique containers.
3. A column can at most be filled with R containers.

We will start by considering a column in isolation. There is 1 way in which a column can be loaded with 0 containers, $N - 1$ ways in which a column can be loaded with 1 container, $(N - 1)^2$ ways in which a column can be loaded with 2 containers, and so on. We name the total amount of configurations for a column M and derive:

$$M = \sum_{i=0}^R (N - 1)^i = \frac{(N - 1)^{R+1} - 1}{N - 2}$$

This follows from recognizing it as a partial sum of a geometric series. Since the columns are independent, the total number of naive bay configurations becomes:

$$M^C = \left(\frac{(N - 1)^{R+1} - 1}{N - 2} \right)^C$$

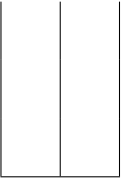
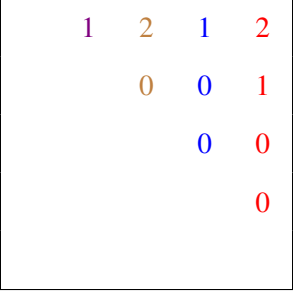
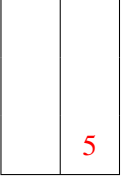
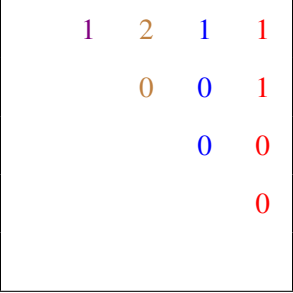
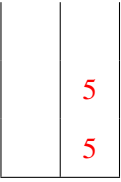
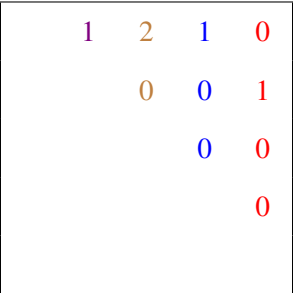
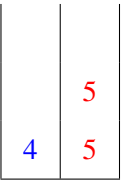
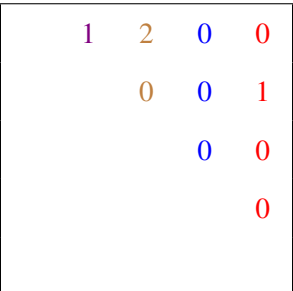
We will now investigate the condensed bay representation. For this representation, we have some arbitrary ordering of all possible column configurations. Since the columns are always swapped to match the ordering, it gives the following number of bay configurations:

$$\binom{M + C - 1}{C} = \binom{\frac{(N-1)^{R+1}-1}{N-2} + C - 1}{C}$$

For an explanation of the above, imagine that all the M possible variations of a column are placed along a line sorted according to the ordering. Choosing C of these, allowing repeats, and inserting them into the bay matrix in the same order that they appeared in the sequence, will produce a unique bay matrix in which the columns are guaranteed to be in the correct order.

C Example Episode

The example below spans an entire episode with $N = 5$, $R = 3$, $C = 2$. Each step will show the following information: time step, bay matrix, transportation matrix, action, and cumulative reward. The colors indicate the destination port, where red corresponds to port 5, blue corresponds to port 4, and so forth. The actions shown will be the ones performed prior to the current state. Hence the initial state has no associated action. Underlined containers denote that they are blocking. All the zero entries in the matrices have been hidden for clarity.

	B	T	Action	Sum Reward
0			%	0
1			ADD [1]	0
2			ADD [2]	0
3			ADD [1]	0

	<i>B</i>	<i>T</i>	Action	Sum Reward
4	<div> <div></div> <div></div> <div></div> <div></div> <div>4</div> <div>5</div> </div>	<div> <div>1</div> <div>2</div> <div>0</div> <div>1</div> <div></div> <div></div> <div>0</div> <div>0</div> <div>1</div> <div></div> <div>0</div> <div>0</div> </div>	REMOVE [2]	−1

	<i>B</i>	<i>T</i>	Action	Sum Reward
5	<div> <div></div> <div></div> <div></div> <div></div> <div><u>5</u></div> <div>4</div> <div>5</div> </div>	<div> <div>1</div> <div>2</div> <div>0</div> <div>0</div> <div></div> <div></div> <div>0</div> <div>0</div> <div>1</div> <div></div> <div>0</div> <div>0</div> </div>	ADD [1]	−2

	<i>B</i>	<i>T</i>	Action	Sum Reward
6	<div> <div></div> <div></div> <div></div> <div></div> <div><u>5</u></div> <div>4</div> <div>3</div> <div>5</div> </div>	<div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div></div> <div></div> <div>0</div> <div>0</div> <div>1</div> <div></div> <div>0</div> <div>0</div> </div>	ADD [2]	−2

	<i>B</i>	<i>T</i>	Action	Sum Reward
7	<div> <div></div> <div></div> <div></div> <div></div> <div><u>5</u></div> <div>4</div> <div>3</div> <div>3</div> <div>5</div> </div>	<div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div></div> <div></div> <div>0</div> <div>0</div> <div>1</div> <div></div> <div>0</div> <div>0</div> </div>	ADD [2]	−2

	<i>B</i>	<i>T</i>	Action	Sum Reward
8	<div> <div></div> <div></div> <div></div> <div></div> <div><u>4</u></div> <div>3</div> <div>2</div> <div>2</div> <div>4</div> </div>	<div> <div>0</div> <div>0</div> <div>1</div> <div></div> <div>0</div> <div>0</div> <div></div> <div>0</div> <div></div> </div>	ADD [1]	−2

	<i>B</i>	<i>T</i>	Action	Sum Reward
9	<div><div></div><div>2</div></div>	<div><div>2</div></div>	ADD [1]	-3

	<i>B</i>	<i>T</i>	Action	Sum Reward
10	<div><div>2</div><div>2</div></div>	<div><div>1</div></div>	ADD [1]	-3

	<i>B</i>	<i>T</i>	Action	Sum Reward
11	<div><div></div><div></div></div>	<div><div></div></div>	ADD [1]	-3

D Embedding T using a Graph Convolutional Network

We used the PyTorch Geometric implementation of the graph convolutional operator described in [Kipf and Welling \(2016\)](#). The node input features were simply the port numbers from 1 to N , and the transportation matrix was passed as the edge weights. We applied two convolutions with the tanh non-linearity in between and aggregated all the node representations with a max and average pooling operation. Appending these aggregations together gave the final output vector.

E Embedding T using a LSTM

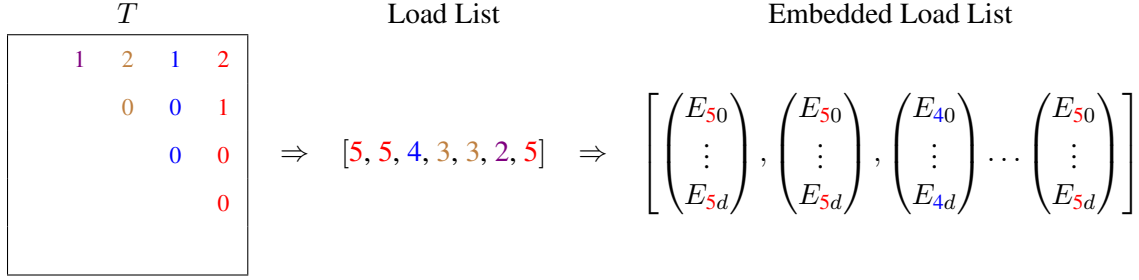


Figure 5: Example of transforming a transportation matrix into a long-winded load list

In figure 6 we see an example of how to transform the transportation matrix into a load list. It is done by unrolling the matrix in the order right to left and then top to bottom ignoring all zeros. E.g. since $T_{15} = 2$ that means 2 containers are going from port 1 to port 5. Since these will be placed first, we add the two 5-containers to the beginning of the load list. This way the list is ordered in the exact way the agent will be placing the containers (ignoring removals and reintroduced blocking containers). Following the unrolling, we use a learned embedding (in essence a lookup table) that projects the $N - 1$ possible containers to d dimensional space. This sequence is then fed through the LSTM. Extracting the final hidden state of the LSTM yields an encoding of the transportation matrix.

One problem with this encoding, is that it is very long-winded. The load list can be as long as $R \cdot C \cdot (N - 1)$. This would be the case if the entire bay matrix was filled and emptied at every port, equivalent to having all the superdiagonal entries equal to $R \cdot C$ and all other entries zero. This is not only slow, but long sequences can also prove to be difficult for LSTMs to encode.

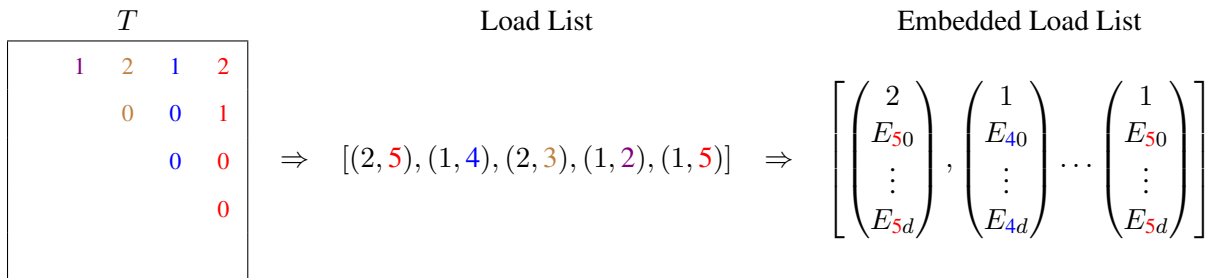


Figure 6: Example of transforming a transportation matrix into a condensed load list

Therefore we also tested a slightly different encoding. We would unroll the transportation matrix in much the same way, except now the load list is filled with tuples of (T_{ij}, j) for all $T_{ij} > 0$. After applying the embedding to j we append T_{ij} to the top of the vector. This represents the exact same information. However, the maximum load list length is now the size of the upper triangular matrix e.g. $\frac{N(N-1)}{2}$. This has the nice property that it only scales with respect to the number of ports, which is most often orders of magnitude smaller than the capacity of the vessel.

F Randomly Generating Transportation Matrices

We trained on mixed distance transportation matrices as described [Avriel et al. \(1998\)](#). However, the dataset provided by [Parreño-Torres et al. \(2019\)](#) was generated using the Authentic Matrices Generation (AMG) Algorithm proposed by [Ding and Chou \(2015\)](#). The main differences are that AMG guarantees that a water vessel is fully loaded when leaving a port and that AMG matrices are generally less sparse. Training on one type and testing on another may introduce some covariate shift, however, all matrices generated by AMG are also possible outputs of the mixed distance method. We, therefore, figured that mastering mixed distance should transfer to mastering AMG-based matrices.

G Hyperparameter Search

For each of the model architectures, the amount of trained and tested models varied. As a broad overview, we inspected:

- Entropy coefficients in the range 0 to $1e-3$
- Learning rates in the range $1e-2$ to $1e-5$
- Number of epochs in the range 3 to 10
- Number of steps in the range 128 to 4096
- Sizes for the hidden layers in the feature encoder from 16 to 128
- Size of the output vector from the feature encoder from 64 to 1024
- Number of final layers in the feature encoder from 1 to 3.
- Different value and policy networks such as [64,64], [64,64,64], [128,128]. These lists are to be interpreted as network architectures where each number represents a certain amount of neurons in a hidden layer.
- Amount of hidden layers in the LSTM and RNN from 1 to 2
- Hidden layer sizes in the LSTM and RNN from 64 to 256.

More information on many of these parameters can be found at the [Stable-Baselines3 documentation for PPO](#).

H Unzoomed Reward Plot

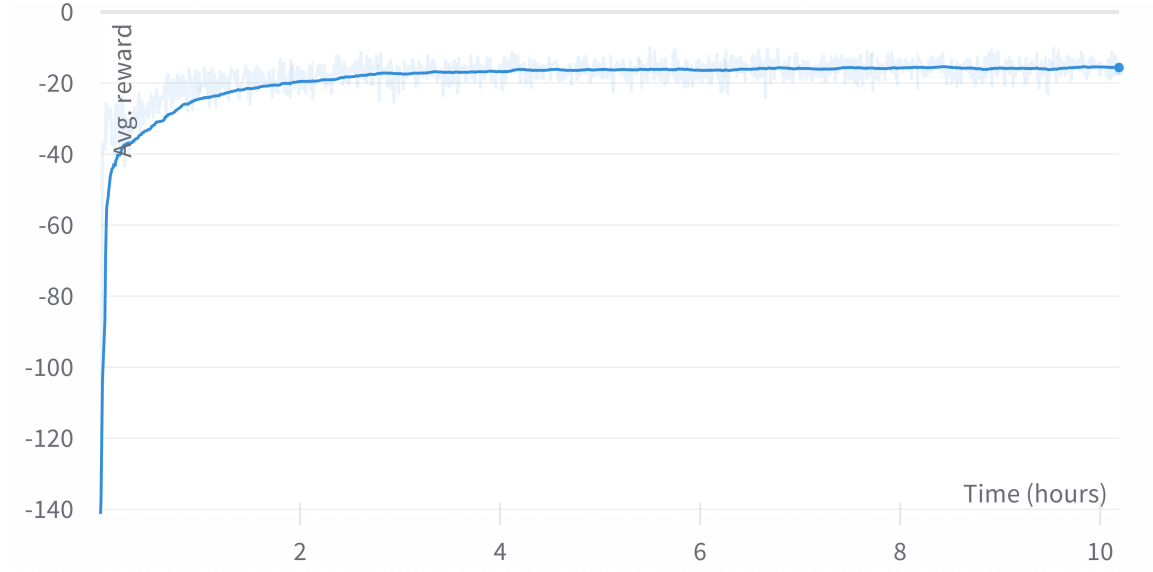


Figure 7: The average training reward over the 100 last episodes over time for the overall model. The dark blue line shows the exponential average with a smoothing of 0.9.

I Test Data

We tested on Set II introduced in [Parreño-Torres et al. \(2019\)](#). It contains 840 instances spanning over the following settings:

$$N \in \{4, 6, \dots, 16\}$$

$$R \in \{6, 8, \dots, 12\}$$

$$C \in \{2, 4, \dots, 12\}$$

For each setting they generated 5 random transportation matrices using the Authentic Matrices Generation (AMG) Algorithm proposed by [Ding and Chou \(2015\)](#).