A Reinforcement Learning Framework for Container Selection and Ship Load Sequencing in Ports

Extended Abstract

Richa Verma, Sarmimala Saikia,
Harshad Khadilkar, Puneet Agarwal,
Gautam Shroff
TCS Research, Delhi 201309, India
richa.verma4@tcs.com

Ashwin Srinivasan
Birla Institute of Technology and Science
Goa 403726, India
ashwin@goa.bits-pilani.ac.in

ABSTRACT

We describe a reinforcement learning (RL) framework for selecting and sequencing containers to load onto ships in ports. The goal is to minimize an approximation of the number of crane movements require to load a given ship, known as the *shuffle count*. It can be viewed as a version of the assignment problem in which the sequence of assignment is of importance and the task rewards are order dependent. The proposed methodology is developed specifically to be usable on ship and yard layouts of arbitrary scale, by dividing the full problem into fixed future horizon segments and through a redefinition of the action space into a binary choice framework. Using data from real-world yard and ship layouts, we show that our approach solves the single crane version of the loading problem for entire ships with better objective values than those computed using standard metaheuristics.

KEYWORDS

Reinforcement learning; Single agent planning & scheduling

ACM Reference Format:
Richa Verma, Sarmimala Saikia,, Harshad Khadilkar, Puneet Agarwal,, Gautam Shroff, and Ashwin Srinivasan. 2019. A Reinforcement Learning Framework for Container Selection and Ship Load Sequencing in Ports. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

Busy ports can handle more than 30 million containers per year, or over 80 thousand containers per day [4]. Container loading and unloading operations in the storage yard are among the most complex in the industry [1], and reducing the time it takes to load a ship can have a significant effect on the yard efficiency. Containers are stored in the form of vertical pillars in the yard, with each pillar possibly a dozen containers high. Whenever a container in a lower position in a stack needs to be accessed, the containers above it have to be moved away first, increasing non-productive moves referred to as *shuffles*. The algorithm described in this paper aims to minimise the total number of shuffles required to load an outbound ship, given a known layout of containers in the yard. This is a combinatorial optimisation problem, which is difficult to solve using traditional approaches. The key contributions are: (1) formulation of the container loading task as an RL problem, (2) keeping the size of the state-action space independent of the problem instance for scalability, (3) reward shaping for fast convergence, and (4) demonstration of the generalisation and transfer learning ability of the trained policy network on real-world problem instances.

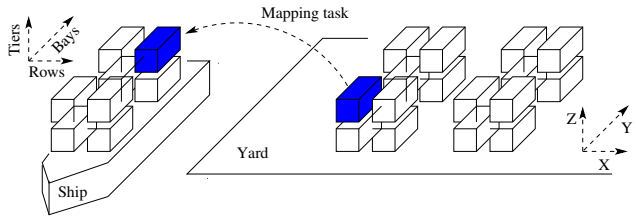

Figure 1: Graphical illustration of the problem. Each slot is associated with a bay, row, and tier.

2 PROBLEM DESCRIPTION

An illustration of the problem is shown in Figure 1. The goal is to minimize the total number of rearrangements (shuffles) required for loading the whole ship. While the problem structure and linear objective function are similar to the classical assignment problem [2], the number of agents (containers) and tasks (slots) is not equal, and the cost is dependent on the loading order. We describe a simple version of the loading problem where there is a single loading crane and a fixed set of required container characteristics for each slot on the ship. Let ψ be the set of all slots on a ship, with $|\psi| = N$. The slots have to be filled in a predefined order $\{\psi_1, \ldots, \psi_N\}$ by exactly one container per slot. The requirements for each slot are known, and we assume that there are $M_\psi \leq N$ unique combinations (which we call mask IDs) among the slots in ψ. The set of all containers in the yard is denoted by κ_0 with $|\kappa_0| = K \geq N$. Based on physical characteristics and the cargo carried, each container $C_j \in \kappa_0$ ($j \in [0, K]$) is also mapped to a mask ID in $\{1, \ldots, M_\psi, \ldots, M_\kappa\}$.

We define the operator μ, which returns the mask ID of a slot or a container as a scalar integer value. A container C_j is eligible for a slot ψ_i if and only if $1 \leq \mu(C_j) = \mu(\psi_i) \leq M_\psi$. The position of a container C_j in the yard is given by $P(C_j) = \{x_j, y_j, z_j\}$, with the axes marked in Figure 1. Instead of physical dimensions, the values of x, y, and z are the indices of containers along the relevant axes. A set of containers with the same x and y coordinates but different z coordinates are known as a *pillar*. The load planning problem starts with the initial yard state κ_0, where the position $P(C_j)$ of each

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

container is given. The initial state of the ship is empty, and the first slot to be filled is ψ_1. We denote the combined ship and yard initial state by *loading step* s_0. One or more containers may need to be moved away by the loading equipment in order to access the chosen container for slot ψ_ℓ, depending on its position in the pillar. These unproductive operations are known as shuffles. The number of shuffles on choosing C_j at loading step s_ℓ is denoted by $Q(C_j, \ell)$.

When loading step s_ℓ is completed, we assume that all containers C_j^* that were moved away are returned to their original pillar, with updated z coordinates $z_j^* \leftarrow z_j^* - 1$. This reflects the departure of the loaded container from the yard. The new yard layout with one fewer container is designated $\kappa_{\ell+1}$ and the algorithm proceeds to loading step $s_{\ell+1}$, terminating at step s_N. The objective of load planning is to minimise the total shuffle count, $J = \sum_{\ell=0}^{N-1} Q(C_j, \ell)$.

3 METHODOLOGY

We model the problem as a Markov Decision Process ($\mathcal{S}$,$\mathcal{A}$,$\mathcal{R}$,$\mathcal{P}$,γ) [3] where $\mathcal{S}$ and $\mathcal{A}$ are the sets of states and actions, respectively, $\mathcal{P}$ represents the transition function, $\mathcal{R}$ denotes the reward function and γ is the discount factor. A standard reinforcement learning setup consists of an agent interacting with its environment through a series of actions. At every time-step t, the agent receives the current state s_t of a fully-observed environment, takes an action a_t based on a policy $\pi : s \rightarrow a$, and receives a scalar reward $r_t = \mathcal{R}(s_t, a_t)$. The goal of learning is to maximize the expected cumulative discounted reward, $E\left[\sum_{i=t}^{\infty} \gamma^t r_t\right]$. We represent the policy π by a multilayer perceptron (MLP) network with parameters θ_P. We use policy gradients to train the RL agent.

We first divide the problem of computing the entire load plan into single loading steps s_ℓ. Each loading step is composed of multiple time steps s_t in which suggested containers are rejected, until a container is selected for loading into slot ψ_ℓ. The state space is restricted to look ahead to a fixed number of slots on the ship, and to a fixed number of containers in the yard. This standardises the size of the state space for all ship and yard layouts, facilitating scalability. The actual mask IDs of slots and containers are represented in the state as relative numbers, based on how imminently a given mask ID needs to be loaded. This makes the RL formulation agnostic towards actual mask IDs, facilitating generalisability. Finally, the reward does not depend on absolute shuffle counts, but on improvement over previous episodes. This property allows the RL to work in instances with different 'optimal' shuffle counts.

The environment uses a scouting procedure to identify pillars in the yard that hold at least one eligible container for the current slot, and present them to the RL agent one at a time. The environment chooses one of the eligible containers in this set as the current suggestion to the RL agent. The agent computes a binary decision based on the current suggestion and the remaining pillars: whether to *pick* the presented container for loading in the current slot, or to *move* to the next option ($\mathcal{A}$ = {pick, move}). The decision is based on the spatio-temporal context provided to the container, as encoded in the state. For each container loading decision (slot ψ_i, yard state κ_{i-1}, loading step s_{i-1}), we only include the current slot and a fixed horizon length h of upcoming slots in a look-up table which is used to encode a portion of the state. The mask IDs associated with this slot sequence are mapped to *relative* mask IDs for each container selection decision. Through the look-up table, the current slot's mask ID is mapped to a value of 0, indicating imminent loading ($\mu(\psi_i) \rightarrow 0$). The mask ID of the next slot gets a value of 1 ($\mu(\psi_{i+1}) \rightarrow 1$) and so on, up to h upcoming slots. The pillars chosen by the environment are also encoded using relative mask IDs and presented to the RL agent as part of the state.

We use a variable threshold τ as the target shuffle count to be achieved by the RL agent at the end of an episode. Prior to training, τ is set to an arbitrarily large value. A terminal (final) reward $+R_{\text{fin}}$ is provided if the total shuffle count $J \leq \tau$. If the latest J is better than the previous best observed value, a further reward of $+R_{\text{fin}}$ is given. If $J > \tau$, a reward of $-R_{\text{fin}}$ is given. The value of τ is updated to be equal the average shuffle count after every E episodes during training, if this is less than its previous value. Thus the reward adapts to the desired value for a specific problem. We train the policy network in an episodic setting. The RL agent is trained on multiple instances, allowing it to generalize its knowledge. In each training iteration, we play $\mathcal{E}$ episodes for each ship to explore the probabilistic space of possible actions using the current policy.

4 EXPERIMENTS AND RESULTS

We compare the performance of the proposed RL methodology with two metaheuristic approaches, based on simulated annealing and genetic algorithms. Three independent datasets are used for training, testing, and comparison. These are obtained from real-world operational data for three ships and their respective yard layouts. The number of slots vary from 130 to 1391, the mask IDs from 25 to 97, and the number of containers in the yard from 25,000 to 7,00,000 across the three instances. The results are compiled in Table 1. Apart from simulated annealing (SA) and genetic algorithms (GA), we also use a greedy heuristic (always pick topmost matching container) as a baseline. Among RL approaches, RL-S1 is trained only on Ship 1 data, while RL-S2 is trained only on Ship 2 data. RL-MUL is trained on Ship 1 followed by Ship 2. None of the algorithms are trained on Ship 3. In the test data, we note that the RL approaches outperform the baselines on all three sets. RL-MUL performs best in 2 of the 3 cases, demonstrating generalisation. Finally, all three RL approaches perform well on the unseen Ship 3 data, which demonstrates transfer learning.

REFERENCES

[1] Héctor J Carlo, Iris FA Vis, and Kees Jan Roodbergen. 2014. Storage yard operations in container terminals: Literature overview, trends, and research directions.

Algo.	Ship 1	Ship 2	Ship 3
	Test Run	Test Run	Test Run
Greedy	273 (0.17)	282 (0.25)	43 (0.02)
SA	243, 261.0 (177)	269, 287.7 (296)	31, 39.4 (2.1)
GA	283 (398)	278 (557)	30 (6.2)
RL-S1	**235**, 252.5 (515)	269, 274.4 (377)	30, 35.1 (6.7)
RL-S2	238, 252.0 (515)	266, 273.6 (377)	29, 35.3 (6.7)
RL-MUL	236, 251.5 (515)	**264**, 271.3 (377)	**28**, 35.0 (6.7)

Table 1: Shuffle counts of algorithms. Numbers in parentheses are computation times in seconds. The first value is the best value, while the second value (if any) is the average.

European Journal of Operational Research 235, 2 (2014), 412–430.
[2] J Munkres. 1957. Algorithms for the Assignment and Transportation Problems. *J. Soc. Indust. Appl. Math.* 5, 1 (March 1957), 32–38.
[3] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.
[4] World Shipping Council. 2018. Ports: About the Industry. (Accessed: March 2018). http://www.worldshipping.org/about-the-industry/global-trade/ports.