Université de Montréal

On Two Sequential Problems: The Load Planning and Sequencing Problem and the Non-normal Recurrent Neural Network

par

Kyle Goyette

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

July 23, 2020

Université de Montréal

Faculté des études supérieures et postdoctorales

Ce mémoire intitulé

On Two Sequential Problems: The Load Planning and Sequencing Problem and the Non-normal Recurrent Neural Network

présenté par

Kyle Goyette

a été évalué par un jury composé des personnes suivantes :

Fabian Bastin

(président-rapporteur)

Emma Frejinger

(directeur de recherche)

Pierre L'Écuyer

(membre du jury)

Abstract

The work in this thesis is separated into two parts. The first part deals with the load planning and sequencing problem for double-stack intermodal railcars, an operational problem found at many rail container terminals. In this problem, containers must be assigned to a platform on which the container will be loaded, and the loading order must be determined. These decisions are made with the objective of minimizing the costs associated with handling the containers, as well as minimizing the cost of containers left behind. The deterministic version of the problem can be cast as a shortest path problem on an ordered graph. This problem is challenging to solve because of the large size of the graph. We propose a two-stage heuristic based on the Iterative Deepening A* algorithm to compute solutions to the load planning and sequencing problem within a five-minute time budget. Next, we also illustrate how a Deep Q-learning algorithm can be used to heuristically solve the same problem.

The second part of this thesis considers sequential models in deep learning. A recent strategy to circumvent the exploding and vanishing gradient problem in recurrent neural networks (RNNs) is to enforce recurrent weight matrices to be orthogonal or unitary. While this ensures stable dynamics during training, it comes at the cost of reduced expressivity due to the limited variety of orthogonal transformations. We propose a parameterization of RNNs, based on the Schur decomposition, that mitigates the exploding and vanishing gradient problem, while allowing for non-orthogonal recurrent weight matrices in the model.

Key words: Intermodal rail terminal, containers, rail, train, double-stack, load planning and sequencing, dynamic programming, deep reinforcement learning, sequential modelling, recurrent neural networks, exploding and vanishing gradient problem

Sommaire

Le travail de cette thèse est divisé en deux parties. La première partie traite du problème de planification et de séquencement des chargements de conteneurs sur des wagons, un problème opérationnel rencontré dans de nombreux terminaux ferroviaires intermodaux. Dans ce problème, les conteneurs doivent être affectés à une plate-forme sur laquelle un ou deux conteneurs seront chargés et l'ordre de chargement doit être déterminé. Ces décisions sont prises dans le but de minimiser les coûts associés à la manutention des conteneurs, ainsi que de minimiser le coût des conteneurs non chargés. La version déterministe du problème peut être formulé comme un problème de plus court chemin sur un graphe ordonné. Ce problème est difficile à résoudre en raison de la grande taille du graphe. Nous proposons une heuristique en deux étapes basée sur l'algorithme Iterative Deepening A* pour calculer des solutions au problème de planification et de séquencement de la charge dans un budget de cinq minutes. Ensuite, nous illustrons également comment un algorithme d'apprentissage Deep Q peut être utilisé pour résoudre heuristiquement le même problème.

La deuxième partie de cette thèse examine les modèles séquentiels en apprentissage profond. Une stratégie récente pour contourner le problème de gradient qui explose et disparaît dans les réseaux de neurones récurrents (RNN) consiste à imposer des matrices de poids récurrentes orthogonales ou unitaires. Bien que cela assure une dynamique stable pendant l'entraînement, cela se fait au prix d'une expressivité réduite en raison de la variété limitée des transformations orthogonales. Nous proposons une paramétrisation des RNN, basée sur la décomposition de Schur, qui atténue les problèmes de gradient, tout en permettant des matrices de poids récurrentes non orthogonales dans le modèle.

Mots-clés: Transport ferroviaire intermodal, conteneurs, planification et séquencement des chargements, programmation dynamique, apprentissage par renforcement profond, modélisation séquentielle, réseaux de neurones récurrents

Contents

List of Tables

List of Figures

List of Abbreviations

AAR	Association of American Railroad
ADP	approximate dynamic programming
ANN	artificial neural network
BFS	breadth-first search
BPC	bits per character
BPTT	backpropagation through time
CNN	convolutional neural network
DFS	depth-first search
DP	dynamic programming
DQN	deep Q-network
DRL	deep reinforcement learning
EURNN	efficient unitary recurrent neural network
EVGP	exploding and vanishing gradient problem
expRNN	exponential recurrent neural network
FMC	Fisher memory curve
GRU	gated recurrent unit
ILP	integer linear programming

LB	lower bound
LDFS	learning depth-first search
LPP	load planning problem
LPSP	load planning and sequencing problem
LSP	load sequencing problem
LR	learning rate
LSTM	long short-term memory
NLP	natural language processing
nnRNN	non-normal recurrent neural network
PTB	Penn Treebank
ReLU	rectified linear unit
RL	reinforcement learning
RNN	recurrent neural network
SDMP	sequential decision-making process
SGD	stochastic gradient descent
SVD	singular value decomposition
SSPP	ship stowage planning problem

Acknowledgements

I'd like to express my thanks to Dr. Emma Frejinger for her excellent guidance and leadership throughout the project. I'd also like to thank Dr. Guillaume Lajoie for his leadership on the nnRNN project. We are grateful to Eric Larsen for his valuable comments that helped improve this manuscript.

Part 1

The Load Planning and Sequencing Problem

Chapter 1

Introduction: The Load Planning and Sequencing Problem

Rail transportation of containerized cargo is a sustainable mode, being both environmentally friendly and cost effective, for long-distance ground transportation of containers inland for the North American market. The number of containers shipped via rail has been growing consistently since 2016, to more than 30 million containers in 2018. Double-stack railcars, introduced in 1984 in North America, allow for twice as many containers to be shipped, making rail an efficient mode of long-distance ground container transportation. Crucial to the continued growth of rail transportation is the efficient operation of rail terminals, wherein containers are loaded and offloaded from railcars.

The load planning and sequencing problem (LPSP) is the focal point of this work. It jointly considers the assignment of containers to platforms and the sequence of operations to place containers on platforms such that the value of containers loaded onto the platforms is maximized, and the cost of handling containers is minimized. The selection of containers must meet constraints defined by the railcar specifications and weight distribution requirements defined by guides from the Association of American Railroads (AAR). This combinatorial optimization problem was first introduced by [67] and modeled as an integer linear program (ILP) and solved with a commercial solver.

In this part of the thesis, our objective is to cast the problem as a shortest path problem and solve it through dynamic programming. We aim to achieve a solution within a limited time budget of 5 minutes, which is much shorter than the time it takes to solve the ILP formulation. The problem is challenging due to the large size of the state and action spaces. We propose several heuristics based on the Iterative Deepening A* (IDA*) algorithm and we derive a lower bound (LB). We also illustrate how deep reinforcement learning (DRL) techniques can be used to solve the LPSP.

Context on the LPSP is presented in Section 1.1, and Section 1.2 provides a high-level background on the methodologies we use. Chapter 2 contains the article reporting on the proposed heuristics and Chapter 3 presents the DRL algorithm.

1.1. Rail Transportation of Containerized Cargo

We present several aspects that are important to rail transportation of containers. We begin by discussing the containers themselves, and follow up with discussion of railcars, platforms and the rules constraining the placement of containers on platforms. Finally, the terminal and handling equipment are discussed.

1.1.1. Containers

Containers are critical to the growth of rail transportation world wide. They are defined by several characteristics, namely height, length, weight and type. When considering the type of container, there exist dry containers, meaning those that are closed, non-refrigerated, and carrying dry materials. Additionally, there are containers which carry liquids, dangerous goods and those that require power sources to refrigerate their contents. Among dry containers, which are most common, there are 6 distinct sizes:

- 20-foot standard (20' - 8' - 8'6"),
- 40-foot standard (40' - 8' - 8'6"),
- 40-foot high cube (40' - 8' - 9'6"),
- 45-foot high cube (45' - 8' - 9'6"),
- 48-foot high cube (48'- 8' - 9'6"),
- 53-foot high cube (53'- 8' - 9'6").

The 48-foot and 53-foot containers are only present in the domestic North American market. Most of these containers can be stacked on top of each other, on ships, in terminals and on railcars. However there do exist some containers with soft walls, or rules regulating how a container carrying dangerous goods can be handled which limit the ability to stack containers.

1.1.2. Railcars, Platforms and Slots

The double-stack intermodal railcar allows for roughly twice as many containers to be carried on a single railcar than a single-stack railcar. This doubling of load is critical to the efficient transportation of containers, and the growth of transportation via rail in the North American market. A train is comprised of several railcars, which are in turn comprised of platforms. Each railcar is characterized by the number and length of its platforms and by the loading patterns that determine the lengths of the containers that can be placed on each platform. A double-stack railcar has platforms on which containers can be placed on the bottom slot, and on the top slot. Platforms are defined by their length, weight, carrying capacity, as well as position in a railcar. An example of a double-stack train, loaded with containers is shown in Figure 1.1.

Figure 1.1. A double-stack train composed of several railcars, loaded with containers

1.1.3. Intermodal Container Terminals

Container terminals exist at the interface of two modes of transportation. At these locations, containers are removed from one mode of transportation, then stored or loaded directly onto another mode of transportation. The two most common types of terminals are maritime terminals, where ships interface with ground transportation, and inland terminals where containers are shifted from one mode of ground transportation to another, for example from truck to rail.

When a container is stored, it is placed in a storage yard, an area of the terminal where containers are often stacked, to some maximum height, for short term storage. A row of stacked containers is called a *lot*, which are generally organized by container length, such that each lot is comprised of uniform stacks. The containers in the storage area are typically organized by container destination, such that each zone of the storage area has containers going to the same destination, and as such can be loaded onto the same group of railcars. Along with the storage area, terminals have designated areas for both unloading and loading of transportation vehicles.

1.1.4. Handling Equipment

Moving containers within a terminal is done by one of two types of equipment: the gantry crane or the reach stacker, shown in Figure 1.2 and Figure 1.3 respectively. Each of these types of handling equipment is characterized by the maximum weight they can carry, as well as which containers can be reached at any given time in the storage area. Briefly, the gantry crane can lift any container off the top of any stack, while the reach stacker can only lift the top container of a stack if it is visible from the side of the storage area from which the container is being reached, and within three rows from the foremost container in that lot.

The gantry crane offers many more options for which containers can be lifted at any time. Indeed, any container that can be lifted by the reach stacker can also be lifted by a gantry crane.

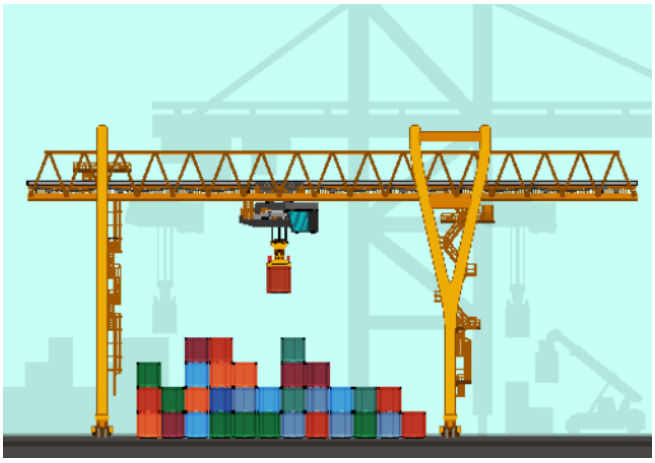


Figure 1.2. A gantry crane loading a container onto a railcar

Figure 1.3. A reach stacker holding a container

1.1.5. The Load Planning and Sequencing Problem

Central to the viability of rail transportation are efficient operations at the terminals. The LPSP, first introduced by [67], is important to the operations, seeking the exact sequence and positions in which the containers will be loaded. The LPSP considers the fine details of the operations, as it addresses the handling cost, accounting for each container movement as well as wear and tear on the equipment. Generally, the objective of the LPSP is to load at minimum cost, for a given destination, as many containers as possible onto a sequence of railcars, until no more containers remain in the storage yard, or there are no more slots available on the railcars. As such, the best solutions to the LPSP are either to leave no slot unfilled while having containers remaining in the storage area, or to have no containers remaining in the storage area. The problem is defined by the layout of containers in the

storage area, the railcars available to be loaded as well as the handling equipment available to perform the loading actions.

One can consider the deterministic LPSP or the stochastic LPSP. In the deterministic case, changes to the layout of the storage area are perfectly known. Here, either containers do not arrive during the loading procedure, or containers do arrive, but the exact time of arrival and placement are known. In the stochastic case, containers can arrive during the loading procedure, and the exact time and placement of the arriving containers in the storage area are not known with certainty when the loading starts. We consider the deterministic LPSP and exclude the arrival of additional containers during loading.

1.2. Methodologies

In this section, we provide high-level background on the methodologies presented in Chapters 2 and 3 to solve the LPSP. We recall that our objective is to reformulate the LPSP that was introduced by [67] as a shortest path problem on an ordered graph. Hence, we change the solution approach from integer programming to dynamic programming. We open with an introduction to dynamic programming and follow with a brief discussion on heuristic solution methods.

1.2.1. Introduction to Dynamic Programming

We define a general deterministic sequential decision-making problem, which corresponds to a shortest path problem [7]. Time is discretized into time intervals or stages indexed by $t = 1, \ldots, T$. At each t, the state of the system $s_t \in S_t$ is observed and an action $a_t \in A(s_t)$ is taken, incurring a cost/negative reward $G_t(s_t,a_t)$. The terminal cost is denoted $G_T(s_T)$. The state transitions from s_t to s_{t+1} according to the transition function through $s_{t+1} = f_t(s_t,a_t)$. For a given initial state s_0, the sequential decision-making problem is defined by

$$\min_{a_0,\ldots,a_{T-1}} G_T(s_T) + \sum_{t=0}^{T-1} G_t(s_t,a_t)$$

subject to $a_t \in A(s_t)$, $s_{t+1} = f_t(s_t,a_t)$, for $t = 1, \ldots, T-1$. We denote an admissible policy with $\pi = \{a_0, a_1, \ldots, a_{T-1}\}$. Further, we can define the recursive dynamic programming equation (also called the Bellman equation) as

$$v_t(s) = \min_{a \in A(s)} \{G_t(s,a) + v_{t+1}(f_t(s,a))\}, \ \forall s \in S_t, t = T-1, T-2, \ldots, 0 \tag{1.1}$$

with $v_T(s) = G_T(s)$, $\forall s \in S_T$. We also define the state-action value, the so-called Q-factor, for a given state s_t and an action a_t following a policy π as

$$q_{t,\pi}(s_t,a_t) = G_t(s_t,a_t) + v_{t+1,\pi}(f_t(s_t,a_t)). \tag{1.2}$$

Due to the curse of dimensionality, it is only possible to solve (1.1) exactly for fairly small size problems. Therefore, there exist a wide range of heuristics that have been proposed in the optimal control (e.g., [7]) and reinforcement learning (e.g., [77]) literatures. While these two fields use different vocabularies, their methods have deep commonalities and a recent textbook is devoted to this topic [8]. Chapter 2 proposes algorithms rooted in the optimal control literature whereas Chapter 3 illustrates the application of an algorithm stemming from the reinforcement learning literature. In the following section, we present background on the latter. We note that we use the reinforcement learning vocabulary, but that similar techniques are described using a slightly different vocabulary in the contexts of approximate dynamic programming [64] or neuro-dynamic programming [9]. Reference [8] provides translations between reinforcement learning and optimal control vocabulary.

1.2.2. Artificial Neural Networks and Deep Reinforcement Learning

We begin by presenting a brief and high-level overview of function approximation in reinforcement learning. Next, we discuss artificial neural networks (ANNs) which are used as function approximators in DRL. Finally, we present some common techniques used in DRL.

ANNs are a common form of nonlinear function approximators. They have the property of being universal function approximators [26], which allows them to approximate any function. We leverage this to have an ANN approximate the action-value function. An ANN is comprised of several interconnected units which are modelled to have properties similar to neurons, a component of the nervous system. These units are organized into layers, which have linear weights, describing the strength of their connections. Additionally, each unit comprises a nonlinear activation function generating its final output. The output of unit i in layer l is defined as

$$a_i(x^{l-1}) = f(W^l_{i,:}x^{l-1} + b^l_i) \tag{1.3}$$

where $W^l_{i,:}$ is a row of the connection weight matrix, b^l_i is the ith bias value in the current layer, x^{l-1} is the input into layer l, and f is the nonlinear activation function. This can be repeated for each unit in the lth layer, which can act as the input into the $(l+1)$th layer.

The outputs of the ANN are compared with the observed labels via a loss function which measures the difference between the predicted and the target outputs. The loss function is selected depending on the type of problem being solved. For example, mean squared error loss is often used when working with scalar regression models, while cross-entropy loss is frequently used for classification models. The choice of loss function is up to the designer of the model, but it is necessary for the loss function to be differentiable, to allow for updates to be made to the weights and biases of the network. We define a loss function, $L(g(x_j;\theta), y_j)$, where $g(x_j;\theta)$ is the output of the network, based on input x_j and current

weights and biases, θ, and y_j is the target output label for input x_j. Consider a dataset $D = \{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$, drawn from some distribution generated by p_{data}. We would like to minimize the expected loss over the underlying distribution, a value called *risk*:

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(x;\theta),y), \tag{1.4}$$

with respect to θ.

However, since we do not have the data generating probability distribution, p_{data}, the true risk cannot be minimized, as we only have the sample D, whose empirical distribution is $\hat{p}_{data}$. Thus, we minimize the expected loss over the available dataset known as the *empirical risk*:

$$\hat{J}(\theta) = \mathbb{E}_{(x,y)\sim \hat{p}_{data}} L(f(x;\theta),y). \tag{1.5}$$

In order to minimize the loss over the dataset, we update the weights and biases of each layer via backpropagation. In backpropagation, the derivative of the empirical risk over the dataset is computed with respect to each weight and bias of the ANN and these derivatives are used to update the values of the weights and biases using gradient descent as shown in (1.6).

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla \hat{J}(\theta_t) \tag{1.6}$$

Generally, this updating procedure is performed over minibatches of the dataset rather than the full dataset, leading to stochastic gradient descent (SGD).

Convolutional neural networks (CNNs) are specialized for working with higher-dimensional data arranged spatially, such as images [41]. While CNNs were initially designed for visible images, they have been extended to handle higher-dimensional data, for instance that generated by magnetic resonance and by computational topography in medicine, with networks based on 3-D convolutional layers. We make use of CNNs in this work to attend to the 3-D aspects of the LPSP.

Having discussed ANNs and their ability to be universal function approximators, we move on to their application in the realm of DRL. They differ in what exactly they are used to approximate, either the value function or the action-value function or both of them. In DRL, the agent interacts with the environment, creating a dataset on which the agent is trained, based on its behaviour. The objective in DRL is learning which actions an agent should take in an environment to maximize rewards. The labels in DRL are not the *correct output* but a model's approximation to the actual output based on its own experience and parameters. Moreover, the algorithms in DRL aim to improve the reward attained by the agent, rather than just better approximate the value functions. This fundamental difference

leads to algorithms designed to maximize the expected reward that an agent will attain during the training procedure.

Learning policies in DRL can be broken into two distinct types: *on-policy* and *off-policy* learning. As described by [77], on-policy learning occurs in methods which attempt to evaluate or improve the policy which makes decisions, whereas off-policy learning occurs in methods which attempt to evaluate or improve a policy which is different from the one used to generate the data. Learning in off-policy based methods allows for the use of trajectories and data that were created following a different policy than the current one. An important characteristic of off-policy learning is sample-efficiency, since it can make use of experiences generated using a different policy from the one currently defined by the network.

In Chapter 3, we consider deep Q-learning, which uses a deep Q-network (DQN) to estimate the action-value function for the LPSP. The DQN is trained using an algorithm called deep Q-learning with experience replay. The idea behind the deep Q-learning algorithm is to estimate the action-value function using the Bellman equation (1.2) as an iterative update (over iterations indexed by i). In the vocabulary and notation of reinforcement learning, this is expressed as follows:

$$Q_{i+1}(s,a) = \mathbb{E}[r(s,a) + \gamma \max_{a' \in A(s')} Q_i(s',a') | s,a], i = \text{iterative update} \tag{1.7}$$

where $Q_i(s,a)$, $r(s,a)$ and γ are respectively the Q-factor at iterate i, the observed reward from taking action a in state s and the discount factor. Moreover, s' and a' are the next stage state and actions. The discount factor controls the importance of future rewards to the model. While the LPSP itself is best represented in an undiscounted environment, the discount factor can help to improve stability during training. We use r and max here rather than the previous notation to match the notation found in DRL, where the outcome from taking an action in an environment is typically called the reward, which one aims to maximize rather than minimize, in contrast to costs. Here, the action-value function is updated each step taken in the environment. Thus, the action-value function Q_{i+1} uses the previous iterations action-value function, Q_i, to estimate of the following state s' and actions, $A(s')$. After each step and action taken in the environment, the action-value function is updated.

To do this, we sample experiences through an experience replay. Experience replay is a technique which makes use of a memory of past states, actions and rewards. Those are sampled randomly to train the network and update the parameters, thereby decorrelating the data used to train the network, and smoothing the training distribution over many past behaviours. The experience replay also discards the need to develop a probabilistic model of the system, but can lead to ovefitting to the history of the agent. Since experiences are sampled from a memory, they are likely to be generated using a model defining a different policy than the current model. As such, this method is an example of off-policy learning.

As a high-level description, we use the Q-network, $Q(s,a;\theta)$, with parameter vector θ, and calculate the loss as follows

$$L(\theta) = \mathbb{E}_{s,a\sim p}[(y - Q(s,a;\theta))^2] \tag{1.8}$$

where p is a probability distribution over the behaviours saved in the experience memory. We draw from the experience memory s_t, a_t, r_t, s_{t+1}, recall that $s_{t+1} = f(s_t, a_t)$, and define the label using (1.7) and (1.9)

$$y = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_{a'} Q(s_{t+1},a';\theta) & \text{otherwise.} \end{cases} \tag{1.9}$$

The following specific challenges must be met in an application of DRL: ensuring adequate exploration of the environment, managing instability during the training process, for example stemming from bootstrapping.

Chapter 2

First Article.

Heuristics for the Load Planning and Sequencing Problem

by

Kyle Goyette[1], Emma Frejinger[1], and Giancarlo Kerg[2]

([1]) Department of Computer Science and Operations Research and CIRRELT, Université de Montréal
([2]) Montréal Institute of Learning Algorithms

This article will be further revised and submitted to a yet undetermined publication.

Author Contributions

My contributions to this paper were as follows:

- formulation of the LPSP as a shortest path problem,
- design, implementation and test of the heuristics,
- definition of a lower bound and its computation,
- analysis of test results, comparisons and presentation, including writing most content of the paper.

Giancarlo Kerg contributed to the proof of the lower bound.

2.1. Introduction

Rail transportation is a critical component of intermodal transportation in North America, providing an efficient yet flexible means to transport goods. In the U.S., intermodal rail volume has grown from 9 million containers in 2000 to 13.7 million containers in 2017. Containers accounted for 92 percent of intermodal transportation volume in 2017 according to [4], partly due to the fact that they can be double-stacked, which increases efficiency of transporting goods. While these large increases in volumes have been gained, efficient terminal operation is an important factor for this growth.

Central to intermodal rail transportation is the loading of containers onto railcars. This paper is devoted to this problem. As defined by [67], given a set of containers in a rail terminal, and a set of railcars to be loaded, one must jointly determine the assignment of containers to railcar platforms as well as a sequence of handling operations placing the containers such that the commercial value of the train is maximized and the cost of handling the containers is minimized.

Typically, the problem is divided into two separate problems, the load planning problem (LPP) and the load sequencing problem (LSP), which are solved sequentially as shown in [63]. In the LPP, a set of containers is chosen to maximize the commercial value of a train and the output is the load plan which indicates the position of each container on the train. The LSP then uses the load plan to determine a sequence of operations which minimizes the costs of handling operations when loading the containers. When solved sequentially, the solution to the joint problem can be poor. This can be alleviated by taking into account that there can be many optimal solutions to the LPP.

Double-stack intermodal railcars have only recently become a focus of research. Reference [53] proposed a model with several container and railcar types which include double-stacking railcars but specifically focus on the LPP. Reference [67] extend this work by introducing the LPSP. They solve the problem using a commercial integer linear programming (ILP) solver, but this solution approach does not scale well to large problem instances, or to problems considering the distance travelled by the handling equipment. They show that solution quality can be improved compared to a sequential solution approach.

In this work we adopt a different solution approach. We cast the LPSP as a shortest path problem on a directed graph with the aim to propose heuristics that can provide high-quality solutions to the real-size LPSP in short computing time.

The contributions of this paper are as follows:

- The deterministic LPSP for double-stack intermodal railcars is formulated as a sequential decision-making problem.
- We propose a two-stage heuristic that combines beam search with depth-first search. It produces high-quality solutions in less than 5 minutes.
- We prove that a lower bound (LB) can be computed by solving a relaxation of the problem following a greedy policy. We leverage the LB in the proposed search heuristics.
- We compare the quality of different distance functions at approximating the true cost. The results show that considering the distance travelled in one direction provides a good approximation to the total cost.

The paper is structured as follows: the next section describes the LPSP problem. Section 2.3 provides an overview of related work. We present the formulation in Section 2.4. The proposed heuristics are outlined in Section 2.5. An extensive numerical study is reported in Section 2.6. Finally, Section 2.7 provides concluding remarks and outlines some directions for further research.

2.2. The Load Planning and Sequencing Problem for Double-Stack Intermodal Trains

The LPSP is found at many container terminals and arises when one must load containers onto a vehicle. Here we focus on the LPSP found at rail terminals as described by [67]. In this problem containers are loaded onto a train and we do not explore additional complexities such as having containers off-loaded from trains or trucks during the same time period. The problem considers the handling equipment used at the terminal, the layout of the container storage in the terminal as well as the loading constraints related to the vehicle and container characteristics. In the problem, a sequence of $T-1$ container movements by a single given handling equipment is selected. At each time step $t < T$ a container is paired with a slot on a platform, or a container is double touched, which removes the container from the storage area but does not place it onto a platform.

Intermodal rail terminals are divided into several distinct areas based on their activities. An overview of the areas in a railroad container terminal is shown in Figure 2.1. Containers can arrive by truck, vessel or rail to the unloading operation area and are transferred to the storage area. Handling equipment then organizes all containers in the storage area into stacks with some maximum height. We denote the set of all containers in a problem as

$C = \{c^1, \ldots, c^{|C|}\}$. A train is composed of several railcars, $R = \{r^1, \ldots, r^{|R|}\}$, which are in turn composed of one to five platforms. We define the sequence of all platforms in the problem as $P = \{p^1, \ldots, p^{|P|}\}$. Each platform has 2 slots, thus we define the set of all slots as $Q = \{q_b^1, q_t^1, \ldots, q_b^{|P|}, q_t^{|P|}\}$, where b refers to the bottom slot, and t to the top slot, and the numerical index refers to the platform number. When a train is ready to be loaded, containers are taken from the storage area, one at a time, by the handling equipment, h, and placed in their assigned location on the train. In the context of this paper, the side of the storage area closest to the unloading operation area is considered the back side of the storage area, while the side closest to the train operation area is considered the front side.

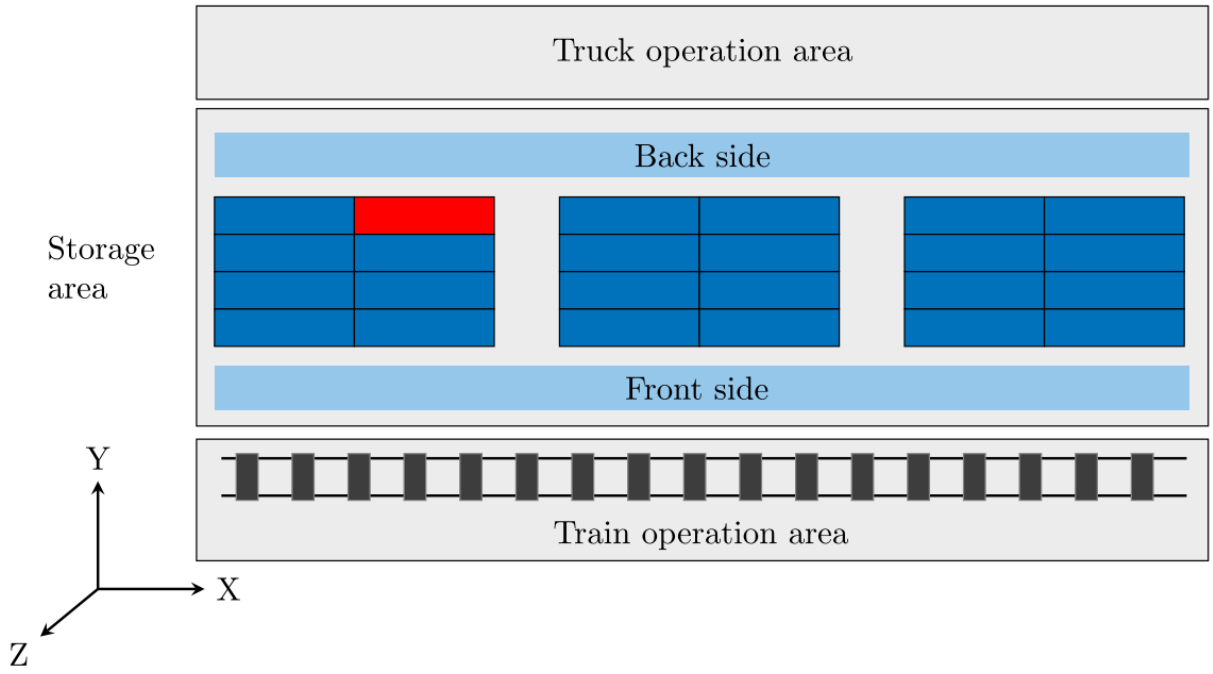


Figure 2.1. An overhead view of the railroad container terminal considered (taken from [67])

The retrieval of containers from the storage area is generally governed by the layout of the storage area and the handling equipment. Container placement is constrained by the placement of previous containers, the rules of placement for a given railcar, and the rules of placement for specific container types.

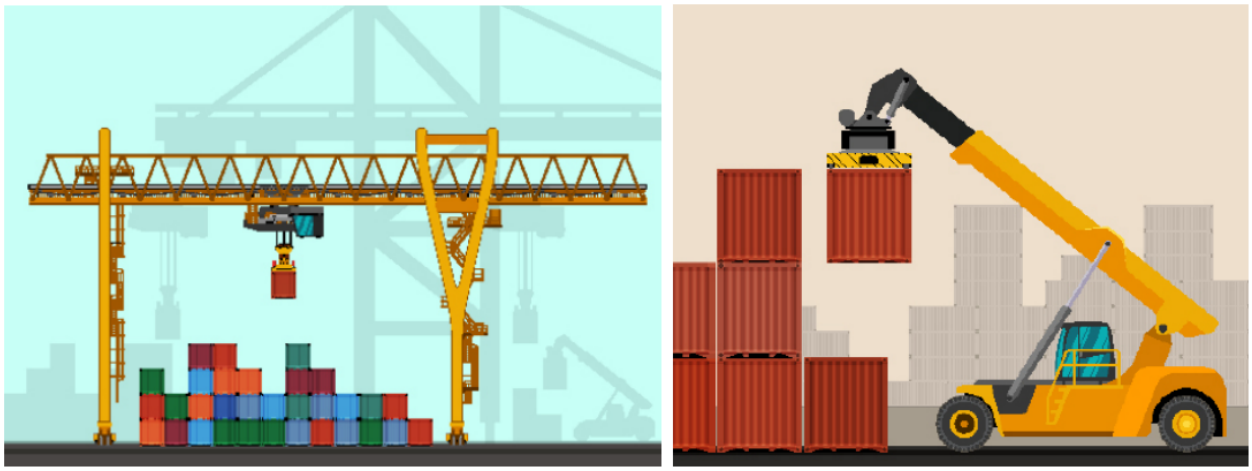
We consider the deterministic form of the LPSP, where the location of all containers is known, and additional containers are not placed in the storage area during the sequence of operations.

2.2.1. Retrieving Containers from the Storage Area

The storage area is a structured environment with each container's position indicated by x, y and z coordinates. The area is organized into lots, where each contains several stacks of

containers, represented by the x coordinate in Figure 2.1. The y coordinate indicates which row the container is in, where $y = 1$ indicates the first row next to the train. The z coordinate indicates the vertical position of a container. Stacks in a lot are limited to a certain height, in our application three containers. We also assume that containers are sorted by their destinations into separate sections of the storage area. Hence, all containers in a single problem instance can be loaded onto the same sequence of railcars which are also assumed to be going to the same destination. We define the position of a container c as (x_c, y_c, z_c). We define the position of platform p as $(x_p, y_p, 0)$. We also define the position of platform p's bottom slot q_b as $(x_{q_b}, y_{q_b}, z_{q_b}) = (x_p, y_p, 1)$ and top slot q_t as $(x_{q_t}, y_{q_t}, z_{q_t}) = (x_p, y_p, 2)$.

In order for a container to be retrieved, it must be accessible by the handling equipment available in the terminal. Handling equipment in railroad container terminals is generally one of two types, a gantry crane or reach stacker, both shown in Figure 2.2. Each type has its own set of rules determining which containers can be reached given a layout of containers in a storage area. For a gantry crane, the rule is simple, if the container is the top container on any given stack, it can be retrieved. However, the rules for a reach stacker are somewhat more involved.



(a) A gantry crane (b) A reach stacker

Figure 2.2. Schematic designs of gantry cranes and reach stackers

A reach stacker retrieves containers from the storage area by lifting a container from the top while facing the container longest side. There are several rules determining when a container can be reached by a reach stacker. It is capable of lifting only a subset of the top containers and a summary of the forbidden loading operations can be found in Figure 2.3. In this figure, the blocking container must be moved prior to the selected container. Case a) is the only relevant case for the gantry crane, while cases a) through e) apply to the reach stacker. As such, the top containers of stacks closest to the back or front side of the storage area are reachable and the reach stacker is only capable of retrieving containers which are

visible from either the front or back side of the storage area. Formally, for a container c with position (x_c, y_c, z_c) there cannot be a container c' with $x_{c'} = x_c$ and $z_{c'} = z_c$ where $y_c < y_{c'}$ for the container to be reached from the front side of the storage area, or $y_{c'} > y_c$ to be reached from the back side of the storage area.

The reach stacker is capable of reaching over one or two rows of containers to access a container, so long as the desired container does not exceed the maximum weight which depends on the number of rows being reached over by the reach stacker. We denote by m_c the weight of container c, θ_1 the maximum container weight when reaching over one row of containers, and θ_2 the maximum container weight when reaching over two rows of containers.

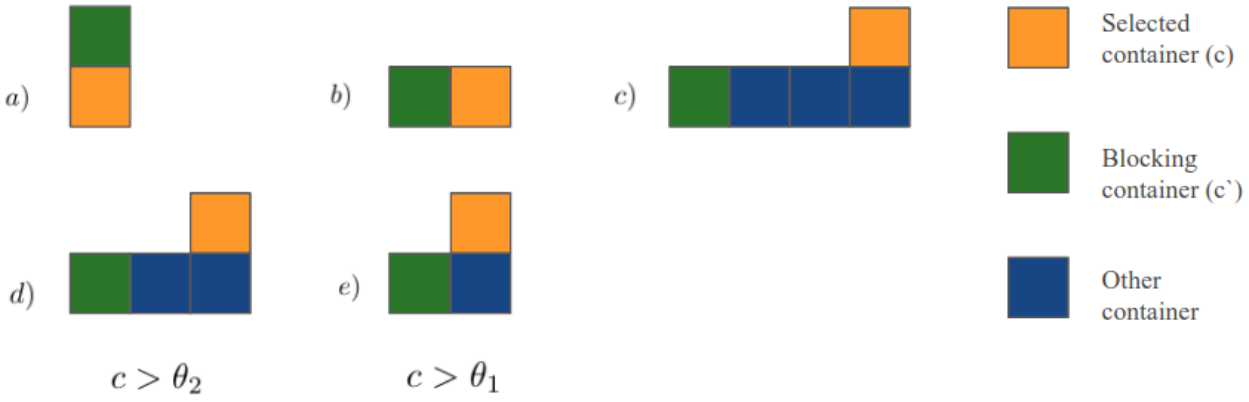


Figure 2.3. Summary of forbidden loading operations for a reach stacker (adapted from [67])

Given the layout of the storage area, it may be that one would like to retrieve a container that cannot be accessed without first moving another container. Moving a container without placing it onto a train is referred to as *double touching* and is often called reshuffling in the literature. The container that is double touched can be placed onto a platform at a later time, or left at the terminal. Since double touches entail additional costs and represent inefficiency, they should be avoided whenever possible. However, double touches are sometimes necessary, and with an appropriate allocation of cost, can be necessary to minimize the overall cost of operations at a railroad terminal.

Handling containers is a source of cost in a container terminal. One of the objectives to be minimized is the cost of handling the containers while loading a train. Handling costs account for the number of containers handled, and the distance the handling equipment must travel. As discussed previously, containers can be accessed from both the front and the back side of the storage area. When accessing containers from the back of the storage area, a reach stacker must perform a detour. In this case, the reach stacker must travel to the sides of the storage region to both lift the container, and place it on a railcar platform.

	Bottom Slot					Top Slot				
Platform number from front	1	2	3	4	5	1	2	3	4	5
	AAR Guide									
Load capabilties	2-20' 1-40'	2-20' 1-40'	2-20' 1-40'	2-20' 1-40'	2-20' 1-40'	1-40' 1-45' 1-48' 1-53'[1]	1-40' 1-45' 1-48'	1-40' 1-45' 1-48' 1-53'[1]	1-40' 1-45' 1-48'	1-40' 1-45' 1-48' 1-53'[1]

Table 2.1. Example of an AAR Guide loading pattern

2.2.2. Assignment of Containers to Railcars

There are several factors determining which platform slots can receive containers in the solution to the LPSP. The railcar platform one may place a container onto can be restricted due to center of gravity constraints, maximum weight capacity of the platform, or loading patterns associated with the railcar as described in [53]. Additionally, the restrictions considered are the same as those considered in [67].

The containers found within the railroad terminal, C, are standardized containers. Each container, $c \in C$, is characterized by its length, L_c (there are five distinct lengths used in the North American market (20-foot, 40-foot, 45-foot, 48-foot, and 53-foot), by its height, h_c and by its weight, m_c. There are six types of containers defined by [53] which can restrict where a container can be placed. However, we only consider two types here, high-cube dry containers, having a height of 9 feet 6 inches and low-cube dry containers, having a height of 8 feet 6 inches, both of which are standard six-sided containers. Finally, it is also assumed that each container has a cost associated with not being loaded on the railcar, π. Note that we do not handle 20-foot containers in our problem instances.

Intermodal trains are made up of a sequence of railcars. Each railcar r has an associated cost of use τ_r. An intermodal railcar is in turn made up of a number of platforms, where the set of all platforms associated with r are denoted P_r. Each platform $p \in P$, is characterized by its length, L_p, weight capacity, g_p and tare weight, m_p. A railcar is also characterized by a set of *loading patterns*, which each define the lengths of possible containers that can be placed in each slot (top or bottom) for each platform on the railcar. A loading pattern can also indicate that a container of a certain length can be placed in a position, if the positions adjacent to that position are filled with containers beneath a certain length. For example, Table 2.1 shows the loading patterns for a 5-platform railcar. The railcar can accept two

[1] 53 ft container in top slot of first, third and fifth platform only when maximum 40 ft container is loaded in the top slots of second and fourth platform.

20-foot containers, or a 40-foot container on the bottom slots of each platform. The top slot of each platform can be filled with any container length up to 53 feet, but a 53-foot container can only be placed on the first, third and fifth platform and still can only be placed there if the adjacent top slots have a container with maximum length of 40 feet. Loading patterns restrict where a container can be placed, and in doing so make the placement of containers interconnected.

The placement of containers is also restricted by center-of-gravity (COG) restrictions imposed on the set of double-stacked containers. As regulated by the AAR, the COG of a fully loaded platform cannot exceed 98 inches at top of rail. Reference [53] provide a formulation for the maximum weight of the container occupying the top slot of a double-stacked platform, knowing the weight and height of the bottom container and the weight of the platform. The COG restriction creates another manner in which the placement of future containers is dependent on the placement of previous containers. Finally, each platform of a railcar has a maximum weight capacity, which cannot be exceeded by the sum of the weights of all containers loaded onto the platform.

In summary, the LPSP for double-stack intermodal railcars is defined with the objective of minimizing the cost of unloaded containers, unused railcars and handling of containers at the terminal. The loading depends on the location of the containers in the terminal and on restrictions governing the assignment of containers to slots on railcars (e.g., COG constraints, weight capacity of platforms and loading patterns of railcars). The solution to the problem is defined as a sequence of handling operations wherein a container is moved either directly to a given slot or is placed aside.

2.3. Related Literature

We begin by discussing the relevant literature to the LPSP for double-stack intermodal railcars and outline the differences between existing works and what is considered in this paper. The literature related to container handling problems in intermodal terminals is abundant. Here, we focus on work closely related to our problem. For a detailed overview of the related work we refer to [76, 75, 12, 13, 67]. Following this, we consider problems similar to the LPSP which are solved through dynamic programming (DP). Next, we discuss the literature addressing stochastic problems similar to the LPSP. Finally, we present studies relevant to the heuristics described in this work.

Closest to our work is that of [67] who introduce the LPSP for double-stack intermodal railcars. They compute solutions with an ILP solver with different distance functions. These distance functions approximate the cost of travel as a constant, consider the distance covered between the container and its target location, and the distance travelled in both directions, from the handling equipments current position to the container, and from the container back to the selected platform. We use the same distance functions here and these are introduced in

Section 2.4.2. As opposed to their exact solution approach, we focus on designing heuristics to solve larger instances in shorter time. Where possible, we compare our solutions with their results.

There is a rich body of literature focused on solving different kinds of operational terminal optimization problems through dynamic programming. Beam search is used by [38] to find the joint solution to the placement and load sequencing of outbound containers at port terminals. They focus on maximizing operational efficiency while satisfying loading constraints. The researchers propose a method using filtered beam search to determine the number of containers to select from each yard-bay and standard beam search to determine the loading sequence of the containers. Their problem differs from ours in that the representation of the ocean terminal is different from that of inland terminals which we consider.

The container load sequencing problem in ocean terminals is approached using a hybrid DP algorithm by [10]. They define the container load sequencing problem as follows: given an initial yard layout as well as a final layout of containers once loaded onto a ship, one must determine the sequence of loading operations which minimizes container relocations. Similarly, [71] consider the same problem as [10]. However they add an additional constraint, based on the crane operator's ability to see where they are placing a container based on previously loaded containers. They propose an alternative solution using a greedy randomized adaptive search procedure. This differs from the problem here in that the cost only considers container relocations and the final layout is given, making the problem closer to the sequential solution of the LPP and the LSP.

The container marshalling problem is formulated as a sequential decision-making problem in [31] and [32]. In this problem, a sequence of crane operations is created which selects containers and moves them to their desired location within a storage area of a container terminal. The problem aims to minimize the cost of moving containers from their initial positions in a storage area of a terminal, to one of many given desired container layouts for the storage area. Both the container load sequencing and the container marshalling problem share similarities with the LPSP: Both must determine an ordering of container movements while minimizing some handling cost and both problems are solved assuming perfect information. However, they differ fundamentally from the LPSP in that in both cases the desired location of containers is known before sequencing operations are selected. As such, they are more similar to solving the LPP and LSP sequentially as in [63].

The Ship Stowage Planning Problem (SSPP) is formulated as a sequential decision-making problem in [72]. The problem consists in optimizing the container loading sequence and assignment of containers to slots on a vessel, while respecting weight constraints. As in our problem, containers can be reshuffled (referred to as a double touch in this work). Additionally, in both the LPSP and the SSPP weight constraints can impact where containers can be placed based on previous container placement. Also, similar to the LPSP proposed

here, the problem is purely deterministic, as the layout of the containers is known, and unchanging throughout the solution. However, the two problems differ in many ways. First, the SSPP focuses purely on minimizing handling costs, which are defined by the number of times a crane must shift from yard to yard and the number of reshuffles required. In the SSPP, contraints are only dictated by the container weights.

Since some heuristics proposed in this paper rely on beam search to find solutions to the LPSP, we discuss beam search and its uses in considering container-terminal problems here. Beam search was first described by [66]. It is now often used in combinatorial optimization problems [85]. Moreover, it has been further expanded to focus specifically on combinatorial optimization problems as in [17].

State space planning is a well-explored topic in optimization and DP, wherein a search algorithm searches among states for solutions. Most state space planning methods are known as heuristic search [77]. Heuristic search allows one to better explore very large state spaces by focusing the search in regions where one expects to find a high-quality solution. Our work is an example of heuristic search, which aims to handle larger search spaces in shorter amounts of time, while dropping guarantees of optimality. Learning depth-first search (LDFS) [11], combines DFS with learning a value function. Learning is used in this sense as the definition from [40] and [5], where state values are updated to make them consistent with their successors. Reference [11] presents a unified approach for heuristic search mechanisms in both deterministic and non-deterministic settings and constructs piecewise policies that behave optimally over distinct regions of the state space. In the deterministic setting, LDFS corresponds to a variant of the A* algorithm, IDA*.

Although a number of problems related to container terminal operations have been formulated as sequential decision-making problems, none attend to the LPSP for double-stack intermodal railcars. We focus on this gap.

2.4. Mathematical Formulation

We begin by presenting the LPSP problem as a sequential decision-making problem that can be solved through dynamic programming. We define the states, actions and state transitions of the system. The time horizon T is discretized into a finite number of container movements indexed by $t = 1, \ldots, T$. Next, we present three cost functions. Finally, we introduce a LB for the LPSP problem.

2.4.1. States and Actions of the LPSP

The state of the LPSP can defined as a vector $s = (s_{\text{cpos}}, s_{\text{d}}, s_{\text{l}}, s_{\text{hpos}})$ where

- s_{cpos} is a vector defined by the position of each container,
  $s_{\text{cpos}} = [(x_c, y_c, z_c)\ \forall c \in C]$.

- s_{d} is a vector indicating whether each container has been double touched, $s_{\mathrm{d}} = [d_c \; \forall c \in C]$
- s_{l} is a vector defined by whether each container has been loaded onto a platform, $s_{\mathrm{l}} = [l_c \; \forall c \in C]$.
- s_{hpos} is a vector defined by the positions of the handling equipment, $s_{\mathrm{hpos}} = [(x_h, y_h)]$.

The number of states grows quickly with the number of containers and platforms in the problem instance, as the number of dimensions which define the problem increase. Of course, the number of states also grows when considering the position of the handling equipment.

At each time step t of the problem, an action a_t can be selected from $A(s_t)$, which includes either moving any reachable container onto any platform that does not violate constraints, or double touching any reachable container. To represent these two types of actions, we consider an action $a_t = (c, p)$ to indicate moving a container c onto a platform p, and $a_t = (c, \emptyset)$ to indicate moving a container aside. Each of these leads to deterministic state transition functions $s_{t+1} = f(s_t, a_t)$. We sometimes refer to s_{t+1} as s'.

First, let us discuss how the state transitions when a container c is placed onto a platform p. The state here is updated by having the container position updated to that of the platform slot, which is the same as the platform, that is: $(x_c,y_c, z_c) = (x_p,y_p, z)$. We set $z = 1$ when occupying the bottom slot, and $z = 2$ when occupying the top slot of the platform. The slot occupied by the container is determined by the current occupancy of the platform. If the bottom slot is unoccupied, the bottom slot is filled, whereas if the bottom slot is occupied, the top slot is filled. When placing a container on a platform, all containers in the storage area previously blocked by this container become reachable by the handling equipment.

Next, let us discuss how the state transitions when a container c is double touched, defined as $a_t = (c, \emptyset)$. Double touching a container updates the position of the container to $(x_c, y_c, z_c) = (x_c,0,0)$, and allows all containers previously unreachable due to container c to be reached. Additionally, we set the double touched indicator for the container, $d_c = 1$. Now that the container has been double touched, it is reachable by the handling equipment from this point onward, until such point that it is placed on a platform.

2.4.2. Cost

The objective of the LPSP is to find a policy $\{a_0, a_1, \ldots, a_{T-1})$ that minimizes the total cost for a given initial state s_0 defined as

$$G^* = \min_{a_1,\ldots,a_{T-1}} \sum_{t=1}^{T-1} G_t(s_t,a_t) + G_T(s_T), \tag{2.1}$$

where $G_T(s_T) = \sum_{c\in C} \pi(1 - s_l^c)$ the terminal cost incurred by unloaded containers (s_l^c equals 1 if c is loaded) and π is the cost per container. Moreover, the general form of the immediate

cost $G_t(s_t,a_t)$ is

$$G_t(s_t,a_t) = \nu + \gamma(a_t)\eta + w(a_t)\tau_r + d(a_t)\kappa + [\zeta_x D_x(s_t,a_t) + \zeta_y D_y(s_t,a_t)] + [\phi_x F_x(s_t,a_t) + \phi_y F_y(s_t,a_t)] \quad (2.2)$$

where:

- ν is the fixed cost of lifting a container.
- κ is the fixed cost of double touching a container.
- $w(a_t)$ is an indicator function which takes the value 1 if a first slot is used on a railcar, and 0 otherwise.
- τ_r is the cost of using railcar r.
- $d(a_t) \in \{0,1\}$ is an indicator function which takes the value 1 if at time step t a container is double touched, and 0 otherwise.
- η is the the cost of a detour to the back side of the storage area for the reach stacker.
- $\gamma(a_t)$ is an indicator variable which takes the value 1 if at time step t a detour is taken by the reach stacker to retrieve a container from the backside of the storage area, and 0 otherwise.
- ζ_x and ζ_y are the costs of travelling with a container in the x and y direction respectively.
- ϕ_x and ϕ_y are the costs of travelling without a container in the x and y direction respectively.
- $D_x(s_t,a_t)$ and $D_y(s_t,a_t)$ are functions which determine the minimal Manhattan distance travelled between the container and its placement location while respecting the ability of the handling equipment to travel through the storage area.
- $F_x(s_t,a_t)$ and $F_y(s_t,a_t)$ are functions which determine the minimal Manhattan distance travelled between handling equipment and the container to be lifted while respecting the ability of the handling equipment to travel through the storage area.

It is important to note that the difficulty of the problem depends on how we incorporate distance in the cost function. Consistent with [67], we therefore define three different cost functions which model the distance cost with increasing levels of accuracy. We present the 0-way distance function, 1-way distance function and 2-way distance function which are all special cases of (2.2). The 0-way distance function ignores distance cost, but has fixed values for double touches and detours. The 1-way distance function considers the Manhattan distance travelled from the container to its placement location, as well as adding a fixed cost for detours. Finally, the 2-way distance function considers the Manhatten distance travelled from the handling equipment's starting position, to the container and from the container to its placement location, but does not include a fixed cost for detours. The 1-way and 2-way

Cost Function	ν	κ	η	ζ_x	ζ_y	ϕ_x	ϕ_y
0-way distance	1	> 0	> 0	0	0	0	0
1-way distance	1	> 0	> 0	> 0	> 0	0	0
2-way distance	1	> 0	0	> 0	> 0	> 0	> 0

Table 2.2. Handling cost definition values for different distance functions

distance functions grow linearly with the distance the handling equipment must travel, while 0-way distance is a constant value depending on the container and placement location for the container. The values for travelling costs of the handling equipment and names of the cost functions are shown in Table 2.2.

The cost of leaving a container behind, π, is larger than the railcar usage cost, τ_r, and is larger than the maximum handling cost when taking any action wherein a container is placed onto a platform slot.

Due to the curse of dimensionality, only small problem instances can be solved with an exact dynamic programming algorithm. We therefore propose heuristics which aim to find good solutions to the LPSP in a reasonable time frame, while sacrificing the guarantee of optimality.

2.5. Heuristics

We begin by introducing the LB for the LPSP in Section 2.5.1. In Section 2.5.2, we propose a two-stage heuristic, using beam search in the first stage to find a good initial solution. A second stage DFS improves this solution. Additionally, we present different variants of the heuristic. The search space can also be limited to a *preferred action space* defined using the problem structure, detailed in Section 2.5.3.

2.5.1. Calculation of a Lower Bound

The algorithms terminate once a maximum running time is reached, the space has been fully searched, or when they find a solution which meets a LB. Additionally, we compare solution values to the LB to assess their quality.

In order to compute a LB we consider a relaxed version of the LPSP that ignores both weight and length constraints of containers and railcars. In other words, any container can be placed onto any slot on any platform, so long as that slot is not occupied.

The minimum cost for the relaxed problem is achieved by minimizing handling costs while loading as many containers as possible. In short, it never reduces the total cost to leave a slot open when a container can be placed in that slot.

We demonstrate that a greedy algorithm, as shown in Algorithm 1, produces an optimal solution for all distance functions to the relaxed problem, in cases where the first lot is aligned with the first platform and lots are filled in order from the first lot along the train.

Algorithm 1 Greedy Algorithm for Calculating LB

Greedy($\sigma(S,A)$)
$G = 0$
while $\sigma(S,A)$ is not terminal **do**
  $g_{\min} = \infty$
  % $C(s)$ is the set of all reachable containers in state s
  % $P(s)$ is the set of all unfilled platforms in state s
  for all $c \in C(s)$ **do**
    for all $p \in P(s)$ **do**
      $g = \phi_x|x_h - x_c| + \phi_y|y_h - y_c| + \zeta_x|x_p - x_c| + \zeta_y|y_p - y_c|$
      if $g <= g_{\min}$ **then**
        $g_{\min} = g$
        $a_{\min} = (c,p)$
      end if
    end for
  end for
  Take action $a_{\min}$ and observe s',r
  $G = G + r$
  $s = s'$
end while
return G

Proposition 1. *Let G^R denote the cost of a relaxed LPSP ignoring constraints related to container and railcar properties. The cost of greedy solution π^G to the relaxed problem computed by Algorithm 1 is a LB on the cost G^* (2.1), $G^R(\pi^G) \leq G^*$.*

The proof can be found in Appendix 2.8.1.

2.5.2. A Two-stage Heuristic

We propose a two-stage hybrid heuristic search method. The first phase searches greedily among $A(s)$ using beam search and a heuristic cost for each action. The second phase attempts to improve the solution by performing a DFS.

2.5.2.1. *Phase 1: Beam Search*

One can develop an order of actions to search based on the cost of retrieving and placing a container on a platform. This idea naturally leads to an implementation of beam search, wherein the cost heuristic is the immediate cost for retrieving and placing a container, and all actions that do not involve a double touch are selected with higher priority than actions wherein a container is double touched. Hence the heuristic cost used in the beam search algorithm is:

$$G_{heuristic} = \begin{cases} G_{\mathrm{t}} + \Delta & \text{if involves a double touch} \\ G_{\mathrm{t}} & \text{otherwise,} \end{cases} \tag{2.3}$$

where Δ is a large positive value such that double touches are avoided, even if the cost of a double touch is less than the cost of placing a container on a platform. This is done to prevent cases wherein containers could be placed onto a platform, but the total handling cost is greater than that of simply performing a double touch on a nearby container. The beam search algorithm is shown in Algorithm 2. Note that we refer to the cost to reach state s as $cost_s$, corresponding to the labels in a label correction algorithm.

Algorithm 2 Beam search **BS**

Beam search(StartState σ (S,A,T,R),β)
$B \leftarrow StartState$
$R \leftarrow \emptyset$
while $B \neq \emptyset$ **do**
  for all $s \in B$ **do**
    for all $a \in A(s)$ **do**
      $c_{s,a} = Heuristic(a) + \text{cost}_s$
      $C \leftarrow C \cup \{c_{s,a}\}$
    end for
  end for
  $T \leftarrow \beta$ state-action pairs from C with lowest heuristic cost
  for all $(s,a) \in T$ **do**
    Take action a from state s, observe r, s'
    $B \leftarrow B \cup \{s'\}$
  end for
  for all $s \in B$ **do**
    if s is terminal **then**
      $R \leftarrow R \cup \{\text{cost}_s\}$
      $B \leftarrow B \setminus \{s\}$
    end if
  end for
end while
return $\min(R)$

2.5.2.2. *Phase 2: IDA* DFS State Space Search*

In the second phase of the heuristic, we attempt to improve the first phase solution by DFS. Performing DFS can be considered a form of A*, as discussed in [25] and [46], IDA* [39]. In this variant of A*, the next nodes to search are stored in a stack, leading to an A* algorithm wherein the nodes are searched in a DFS manner. One can leverage the LB when searching with IDA* to discard all states which could not achieve better performance than the current best solution.

The search over the state space to tabulate the state-action value function is an idea adopted from [11], and corresponds to label correction algorithms (e.g., [7]). We show a small example of how the minimum cost is updated in Figure 2.4. Briefly, one can store the states, actions and costs to reach a given state. Once a terminal state is reached, the

state-action value function for all states and actions in the trajectory can be updated with the costs following that state and action. The search can then continue at the state prior to the terminal state for other solutions. One only needs to update the trajectory as long as the current branch of the trajectory performs better than previous branches. Moreover, once a state has been visited and all children explored, the value for that state is known and that part of the graph does not need to be explored again. This property is useful since the LPSP has many different paths to reach the same state. This is the idea behind the IDA* Algorithm 3. Importantly, this algorithm can receive a sequence of actions, which can be used to begin the search from the output state of the beam search algorithm.

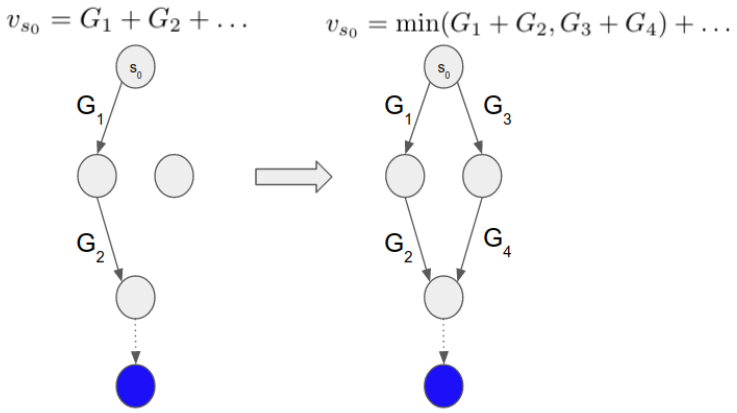


Figure 2.4. IDA* update algorithm

Algorithm 3 IDA* **P-IDA***

1. IDA* Driver($\sigma(S,A,T,R)$,s,Bound, Presearched Moves)

$Q \leftarrow$ HashMap, $N \leftarrow$ HashMap
root $\leftarrow$ Node(s,$a \leftarrow \emptyset$,Parent$\leftarrow \emptyset$,Cost$\leftarrow$ 0,Moves$\leftarrow \emptyset$)
node $\leftarrow$ root
for all $a \in$ Presearched Moves **do**
  Take action a observe s',r
  child $\leftarrow$ Node(s',a,**parent** $\leftarrow$ *node*, **Cost** $\leftarrow$ node.cost $+ r$, **Moves** $\leftarrow$ node.moves $+ a$)
  node $\leftarrow$ child
end for
root $\leftarrow$ node
IDA*(root)
return Tabulated State-Action Value Function Q

2. IDA*(node)

if node.s is Terminal **then**
  BackUp(node.s)
end if
if $A^P(S) \neq \emptyset$ **then**
  actions $= A^P$(node.s)
  sort actions based on Q(node.s,action) for action in actions
  for all $a \in$ actions **do**
    Take action a observe s', r
    if $s' \notin N$ and node.cost $+ r <$ Bound **then**
      child $\leftarrow$ Node(s',a,**parent** $\leftarrow$ node, **Cost** $\leftarrow$ node.cost $+ r$, **Moves** $\leftarrow$ node.moves $+ a$)
      $N[s'] \leftarrow$ child
      IDA*(child)
    else if $s' \in N$ and $r +$ node.cost $> N[s']$.cost **then**
      BackupVisited(s',s)
    end if
  end for
else
  actions $= A$(node.s)
  sort actions based on Q(node.s,action) for action in actions
  for all $a \in$ actions **do**
    Take action a observe s', r
    if $s' \notin N$ and node.cost $+ r <$ Bound **then**
      child $\leftarrow$ Node(s',a,**parent** $\leftarrow$ *node*, **Cost** $\leftarrow$ node.cost $+ r$, **Moves** $\leftarrow$ node.moves $+ a$)
      $N[s'] \leftarrow$ child
      IDA*(child)
    else if $s' \in N$ and $r +$ node.cost $> N[s']$.cost **then**
      BackupVisited(s',s)
    end if
  end for
end if

3. Backup(s)

node $\leftarrow N[s]$
$C_{\text{accum}} \leftarrow$node.cost - node.parent.cost
Q[node.s,node.a] $\leftarrow C_{\text{accum}}$
while node.parent $\neq \emptyset$ **do**
  node = node.parent
  $C_{\text{accum}} \leftarrow C_{\text{accum}} + (\text{node.cost} - \text{node.parent.cost})$
  if $C_{\text{accum}} > Q$[node.s,node.a] **then**
    Q[node.s,node.a] $\leftarrow C_{\text{accum}}$
  else
    return
  end if
end while
if node.s =root **then**
  Bound $\leftarrow \min_a Q$(node.s,a)
end if

4. BackupVisited(s',s)

node $\leftarrow N[s']$
newParent $\leftarrow N[s]$
$C_{\text{accum}} \leftarrow Q$(node.$s$,node.$a$)
node.parent $\leftarrow$ newParent
while node.parent $\neq \emptyset$ **do**
  if $C_{\text{accum}} > Q(\text{node.parent}.s, \text{node}.a)$ **then**
    $Q(s,a) \leftarrow C_{\text{accum}}$
    $C_{\text{accum}} \leftarrow C_{\text{accum}} + (\text{node.cost} - \text{node.parent.cost})$
  else
    return
  end if
end while
if node = root **then**
  Bound $\leftarrow \max_a Q$(node.s,a)
end if

2.5.3. Leveraging Problem Structure to Reduce the Search Space

The LSP naturally provides an order of action quality based on the costs incurred for each action. This is easily seen in the 0-way distance function, where retrieving a container from the front of the storage area and placing it directly on a platform incurs less cost than retrieving from the back of the storage area, or placing the container aside. As such, we can create two groups of actions, preferred actions and the set of all actions. The set of preferred actions does not include the option to double touch a container. Furthermore, in the case of the reach stacker, preferred actions also avoid selecting containers reachable from the back side of the container terminal, while in the case of the gantry crane preferred actions only

consider containers in the three stacks closest to the front side of the storage area for each lot. We define the set of all actions for a given state s as $A(s)$. Similarly, we define the set of preferred actions for a given state s as $A^P(s)$.

2.6. Numerical Study

In this section, we present an extensive numerical study designed to assess the performance of the proposed two-stage heuristic. For this purpose we use problem instances with different characteristics (described in Section 2.6.1) and three cost functions (0-way, 1-way and 2-way distance) as well as two sets of experimental cost values, one giving more and one giving less importance to distance. We compare the performance of different versions of the two-stage heuristic and compare to the baselines that we describe in Section 2.6.3, in addition to LBs computed as detailed in Section 2.5.1.

2.6.1. Problem Instances

The instances used here are the same as those described in [67] and [53]. As reported in Table 2.3, we divide them in four categories depending on their size. The table reports the configuration of the stacks in the yard. There are 50 instances in each category corresponding to a combination of 10 containers and 5 railcar scenarios. For the containers, there are 5 instances with only 40 ft containers and 5 with a mix of all different lengths (40 ft, 45 ft, 48 ft, and 53 ft). Container weights are drawn at random. The 5 railcar scenarios are generated by drawing railcar types at random, conditional on the length of each instance size. Common to the 5 scenarios are the positions of the platforms on the track. More precisely, each platform occupies a single unit of distance in the system and the mth platform occupies position (m,0) on the track.

A note about comparisons to the ILP formulation: there are small discrepancies between the instances in [67] and those presented here. First, platforms here begin at position (1,0) and extend to ($|P|$,0) while platforms there begin at (0,0) and extend to ($|P| - 1, 0$). This difference should only lead to very small changes, and ILP solutions have been tested using their designed platform layout. Second, there are slight differences between the loading patterns, specifically in 3 known cases, wherein ILP formulations allow for loading patterns that are not accepted by our constraints. It is unknown if there are cases here which violate the loading patterns designated by the ILP system. Finally, the heights of containers here are all considered to be HC containers.

2.6.2. Model Parameters

Two sets of experiments were performed, one with lower and equal distance costs in the x and y directions, identical to those in [67], and one with larger costs for travelling in

Instance Size	Number of Instances	Number of Containers	Number of Lots	Maximum Depth
Toy: 200 ft	50	15	2	3
Small: 667 ft	50	50	6	3
Medium: 1000 ft	50	75	3	9
Large: 1500 ft	50	115	5	9
Very Large: 2000 ft	50	150	6	9

Table 2.3. Characteristics of problem instances

the x and y directions, to assess if the algorithms are sensitive to parameter values in the cost function. These experiments have scaled up container leave behind costs, π, double touch costs, κ, and detour costs, η, as well. The second set of tests considers an imbalance in required distance travelled in the x and y directions. The cost values which vary over distance functions are reported in Table 2.4. We additionally present cost parameter values and environment values which remain constant across distance functions in Table 2.5.

	Small Cost Values						
Distance Function	ν	η	κ	ζ_x	ζ_y	ϕ_x	ϕ_y
0-way distance	1	60	80	0	0	0	0
1-way distance	1	60	80	1	1	0	0
2-way distance	1	0	80	1	1	1	1
	Large Cost Values						
Distance Function	ν	η	κ	ζ_x	ζ_y	ϕ_x	ϕ_y
1-way distance	1	3180	4240	53	8	0	0
2-way distance	1	0	4240	53	8	53	8

Table 2.4. Cost parameters for each distance function in all experiments

Parameter	Description	Small Cost Test Values	Large Cost Test Values
π	Cost if container remains unloaded	100	5300
τ	Cost of using a railcar	1	1
θ_1	Reach stacker weight limit when reaching over one row of containers	20,000	20,000
θ_2	Reach stacker weight limit when reaching over two rows of containers	15,000	15,000

Table 2.5. Cost parameters and environment variables which remain constant across distance functions

2.6.3. Baselines

In order to show the impact of leveraging the problem structure using the preferred action space, the effect of leveraging the LB during the IDA* search and that the two-stage hybrid heuristic is beneficial, we compare our two-stage algorithm to algorithms which are

components of the full algorithm as baselines. We compare to standard IDA*, bounded by the accrued cost to a state, called **IDA***. We compare to the same IDA* method leveraging the preferred action space as **P-IDA***. Note, in further cases, the presence of the P indicates leveraging the preferred action space. We also compare to the IDA* and P-IDA* methods which search greedily first (GF) based on the heuristic formulation proposed in Equation 2.3 as **GF-IDA*** and **GF-P-IDA***. Note that this is identical to using a beam width of size 1 in our two-stage algorithm. Additionally, we compare to IDA* methods which leverage the LB to prevent searching nodes which could not outperform the current best solution as **B-IDA***, **BP-IDA***, **GF-B-IDA*** and **GF-BP-IDA***. Finally, we compare our two-stage heuristic to beam search without additional IDA* searching, as **Beam search** $\beta = 5$.

2.6.4. Results

We begin by presenting results for the two-stage hybrid heuristic, as well as for the aforementioned baselines that are outlined in Table 2.6. Sections 2.6.4.1, 2.6.4.2 and 2.6.4.3 present results, in order of decreasing difficulty, for the 2-way distance function, 1-way distance function and 0-way distance function with small cost values respectively. Where possible we compare to the results produced by [67]. Numerical results are shown for medium, large and very large size problems, for each of the distance functions discussed in 2.4.2. Additionally, small size problems are shown for the 1-way distance function to compare to ILP results while toy size problems are shown for the 2-way distance function to compare to ILP results. Sections 2.6.4.4 and 2.6.4.5 present the results of the 2-way distance cost function and 1-way distance cost function for larger cost values for medium to very large size problems.

Abbreviation	**Description**
IDA*	IDA*
P-IDA*	IDA* leveraging preferred action space
B-IDA*	IDA* leveraging LB
BP-IDA*	IDA* leveraging preferred action space and LB
GF-IDA*	IDA* searching greedily
GF-P-IDA*	IDA* searching greedily leveraging preferred action space
GF-B-IDA*	IDA* searching greedily leveraging LB
GF-BP-IDA*	IDA* searching greedily leveraging preferred action space and LB
Beam search $\beta = 5$	beam search using beam width of 5
BS-B-IDA*	2-stage heuristic beam search, with beam width 5, followed by IDA* leveraging the LB
BS-BP-IDA*	2-stage heuristic: beam search, with beam width 5, followed by IDA* leveraging preferred action space and LB

Table 2.6. Heuristics abbreviations and descriptions

The time limit was set to 5 minutes for each instance. When using hybrid methods we limit the IDA* time to 240 seconds, leading to a total search time well below 5 minutes,

as the beam search time is well under a minute. After testing different beam width values, we report results for a width of 5, since this was found to have good performance while producing solutions fast.

For each of the settings we analyze the same performance measures:

- the percentage of instances that was solved to the LB,
- average, minimum and maximum gap to LB,
- average, minimum and maximum of the CPU time for each heuristic,
- average, minimum and maximum CPU time to best found solution,
- the percentage of instances with incomplete loads which did not fill all slots,
- the percentage of instances which had at least one double touch,
- when considering the reach stacker, the percentage of instances which had at least one detour.

Each result is presented with the results for the gantry crane in the upper half of the table, and the reach stacker in the lower half of the table.

2.6.4.1. *Small Costs: 2-way Distance Function*

We begin by presenting results for tests using the 2-way distance function. We show results for toy size problems and compare to our baselines and the ILP solutions. For larger sizes the ILP formulation was unable to converge to a reasonable gap even in hours of computing time.

We present the results to toy size problems in Table 2.7, which we show to compare performance with ILP methods for the reach stacker, as they are are the only ILP results available for the 2-way distance function. We see that a number of instances are solved to meet the LB by all algorithms. No instances are left with an incomplete load by the heuristics, and the average CPU time is around 1.5 minute. Beam search is improved by the addition of IDA*. The best performing heuristic is a simple combination of IDA* while leveraging the preferred action space for both the gantry crane and reach stacker. Large maximum gaps are notable here, this is because as the LB is quite small a single double touch, or inefficient action leads to a notably large gap to the LB. Moreover, since many of these cases have very few slots and containers, double-touches are more often necessary to fill all slots on a platform.

Examining Table 2.8, we see that for medium size instances, beam search coupled with IDA* offers the best performance, with a mean gap to the LB of 0.2%, and are able to solve 34% of instances to meet the LB for the reach stacker. Similar findings emerge when considering the gantry crane, with better results, with 50% of instances solved to meet the LB and a mean gap to the lower bound of 0.1%. GF heuristics are able to solve more instances to meet the LB, but also fail to fill all slots on the railcars for 4% of instances. Moreover, we see that leveraging the LB to bound the search offers slight improvement over the IDA* and

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	78	3.9	0.0	53.5	70.7	0.0	300	18.1	0.0	285.7	0	8
P-IDA*	78	0.8	0.0	32.3	69.8	0.0	300	12.0	0.0	281.8	0	2
B-IDA*	78	3.9	0.0	53.5	70.3	0.0	300	14.0	0.0	222.2	0	8
BP-IDA*	74	3.9	0.0	52.8	79.5	0.1	300	8.0	0.0	126.3	0	8
GF-IDA*	74	6.0	0.0	53.3	79.0	0.0	300	7.0	0.0	289.8	0	12
GF-P-IDA*	78	0.8	0.0	32.1	74.7	0.0	300	10.7	0.0	116.5	0	2
GF-B-IDA*	74	6.0	0.0	53.7	79.0	0.0	300	1.6	0.0	13.8	0	12
GF-BP-IDA*	70	6.0	0.0	52.8	90.5	0.0	300	4.2	0.0	108.8	0	12
Beam search $\beta = 5$	38	9.3	0.0	54.0	0.1	0.0	0.1	0.1	0.0	0.1	0	22
BS-B-IDA*	74	6.0	0.0	52.8	76.5	0.1	240.1	15.2	0.1	217.9	0	12
BS-BP-IDA*	66	6.0	0.0	52.8	83.2	0.1	240.1	2.0	0.1	19.0	0	12

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
ILP	14	4.6	0.0	57.2	553	-	-	-	-	-	26	2	0
IDA*	74	4.0	0.0	53.5	85.4	0.0	300	19.2	0.0	255.1	0	8	12
P-IDA*	78	0.9	0.0	32.1	70.3	0.0	300	12.2	0.0	290.0	0	2	10
B-IDA*	74	3.9	0.0	53.5	82.4	0.0	300	14.5	0.0	227.5	0	8	12
BP-IDA*	74	4.0	0.0	53.1	79.5	0.1	300	7.4	0.0	218.4	0	8	12
GF-IDA*	70	6.1	0.0	53.3	91.3	0.0	300	7.4	0.0	246.0	0	12	18
GF-P-IDA*	78	0.8	0.0	32.1	74.7	0.0	300	11.5	0.0	114.9	0	2	8
GF-B-IDA*	70	6.1	0.0	53.7	91.0	0.0	300	7.9	0.0	135.9	0	12	18
GF-BP-IDA*	70	6.1	0.0	52.8	90.6	0.0	300	2.5	0.0	68.4	0	12	16
Beam search $\beta = 5$	42	9.3	0.0	54.0	0.1	0.0	0.1	0.1	0.0	0.1	0	22	16
BS-B-IDA*	70	6.0	0.0	52.8	75.3	0.1	240.1	7.6	0.1	133.8	0	12	18
BS-BP-IDA*	70	6.1	0.0	52.8	73.8	0.1	240.1	2.1	0.1	25.2	0	12	16

Table 2.7. 2-way results for 50 200-ft instances

P-IDA* methods, but does not improve performance when compared to GF methods, where it offers no improvement for the gantry crane and increases the gap to the LB for the reach stacker. Leveraging the preferred action space leads to slightly better performance, except when considering the reach stacker and GF methods where it finds many more solutions meeting the LB but increases the average gap to the LB.

When considering the larger instances in Table 2.9, we see that the greedy first depth-first search offers the best performance, finding solutions meeting the LB in 42 % of cases for the reach stacker and 40% for the gantry crane, which compares favorably to the BS-B-IDA* methods which find 22% of instances to meet the LB for both the reach stacker and gantry crane. Leveraging the LB to bound the search improves results when considering non GF and non BS methods for the gantry crane, but can be detrimental when used in conjunction with GF, BS, or leveraging the preferred action space for the reach stacker. Leveraging the preferred action space consistently improves results for the gantry crane, and never degrades results for the reach stacker. The average runtime is higher than that of the medium size instances, which is sensible since fewer instances were solved to meet the LB.

Considering the largest instances in Table 2.10, we see that the BS-B-IDA* and BS-BP-IDA* methods perform the best on average, except when considering the gantry crane where GF-IDA* heuristics, both leveraging and not leveraging the LB, perform best. Additionally,

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	4.2	2.2	56.6	300	300	300	77.6	0.3	296.3	2	2
P-IDA*	0	4.1	2.2	56.8	300	300	300	45.7	0.2	289.4	2	2
B-IDA*	0	4.0	2.2	56.8	300	300	300	90	6.4	258.3	2	2
BP-IDA*	0	4.0	2.2	56.8	300	300	300	61.9	6.0	283.6	2	2
GF-IDA*	56	2.3	0.0	56.4	132.2	0.1	300	1.7	0.1	18.4	4	4
GF-P-IDA*	58	2.3	0.0	56.4	126.1	0.1	300	10.2	0.1	270.1	4	4
GF-B-IDA*	56	2.3	0.0	56.4	133.6	1.4	300	4.7	1.4	43.4	4	4
GF-BP-IDA*	58	2.3	0.0	56.6	129.3	2.7	300	7	2.7	37.5	4	4
Beam search $\beta = 5$	46	0.1	0.0	0.7	2.1	1.1	3.6	2.1	1.1	3.6	0	0
BS-B-IDA*	50	0.1	0.0	0.7	123.5	2.4	243.6	5.5	2.4	10.5	0	0
BS-BP-IDA*	50	0.1	0.0	0.7	123.5	2.4	243.6	5.5	2.4	10.6	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	4.6	2.2	56.7	300	300	300	58.2	0.1	293.2	2	10	26
P-IDA*	0	4.5	2.2	56.8	300	300	300	34.2	0.1	292.6	2	10	24
B-IDA*	0	4.5	2.2	56.8	300	300	300	66.9	1.7	292.1	2	10	24
BP-IDA*	0	4.5	2.2	56.8	300	300	300	60.8	3.4	300	2	10	24
GF-IDA*	38	2.4	0.0	56.4	186.1	0.1	300	1.0	0.1	15.4	4	4	6
GF-P-IDA*	58	2.4	0.0	56.4	126.1	0.1	300	20.2	0.1	287.9	4	4	6
GF-B-IDA*	38	2.4	0.0	56.4	186.7	0.9	300	3.6	0.9	39.0	4	4	6
GF-BP-IDA*	58	2.5	0.0	56.6	128.1	1.7	300	4.2	1.7	12.3	4	8	2
Beam search $\beta = 5$	34	0.3	0.0	4.3	1.4	0.8	2.5	1.4	0.8	2.5	0	4	10
BS-B-IDA*	34	0.2	0.0	1.8	160.4	1.6	242.5	3.5	1.6	6.9	0	0	10
BS-BP-IDA*	34	0.2	0.0	1.8	160.4	1.6	242.5	3.4	1.6	6.8	0	0	10

Table 2.8. 2-way results for 50 1000-ft instances

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	4.1	3.4	4.4	300	300	300	122.4	0.3	257.2	0	0
P-IDA*	0	3.8	2.7	4.1	300	300	300	125.0	0.3	296.1	0	0
B-IDA*	0	3.9	3.0	4.4	300	300	300	168.7	12.7	299.4	0	0
BP-IDA*	0	3.8	2.7	4.0	300	300	300	151.4	64.1	295.9	0	0
GF-IDA*	36	0.1	0.0	0.2	192.2	0.4	300	2.8	0.3	48.3	0	0
GF-P-IDA*	40	0.1	0.0	0.3	180.2	0.3	300	7.3	0.3	277.7	0	0
GF-B-IDA*	36	0.1	0.0	0.2	197.2	11.2	300	16.5	11.2	28.9	0	0
GF-BP-IDA*	40	0.1	0.0	0.3	191.4	22.3	300	31.3	22.3	56.0	0	0
Beam search $\beta = 5$	22	0.1	0.0	0.3	6.0	4.3	7.5	6.0	4.3	7.5	0	0
BS-B-IDA*	22	0.1	0.0	0.3	196.6	16.3	247.5	21.8	16.3	27.8	0	0
BS-BP-IDA*	22	0.1	0.0	0.3	196.6	16.3	247.5	21.7	15.4	27.4	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	4.2	3.6	6.4	300	300	300	77.5	0.2	241.7	0	4	6
P-IDA*	0	3.8	3.0	4.7	300	300	300	92.3	4.7	291.9	0	0	6
B-IDA*	0	4.0	3.4	6.4	300	300	300	163.8	6.3	295.4	0	6	4
BP-IDA*	0	3.9	3.0	9.8	300	300	300	151.3	28.8	299.2	0	2	8
GF-IDA*	36	0.1	0.0	0.3	192.1	0.2	300	8.5	0.2	206.2	0	0	4
GF-P-IDA*	42	0.1	0.0	0.3	179.8	0.2	300	12.1	0.2	287.4	0	0	0
GF-B-IDA*	36	0.1	0.0	0.3	194.5	6.2	300	11.4	6.2	66.1	0	0	4
GF-BP-IDA*	40	0.1	0.0	0.3	185.5	12.1	300	25.7	12.1	226.4	0	0	0
Beam search $\beta = 5$	22	0.7	0.0	15.6	3.6	2.7	4.4	3.6	2.7	4.4	0	4	10
BS-B-IDA*	22	0.1	0.0	0.4	192.6	10.8	244.4	14.4	8.6	77.5	0	0	14
BS-BP-IDA*	22	0.1	0.0	0.4	192.6	10.8	244.4	14.4	9.0	80.2	0	0	10

Table 2.9. 2-way results for 50 1500-ft instances

GF methods find the most solutions meeting the LB, but perform worse on average when considering the reach stacker. Leveraging the preferred action space seems to be detrimental when considering GF and BS methods. Overall, all GF and BS methods are able to find an average gap to the LB under 0.5%. We see that beam search paired IDA* is able to find solutions to every problem without a single double touch across all instances and all sizes. Runtimes continue to rise, which correlates with the decreasing percentage of instances solved to meet the LB.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	5.4	5.1	5.6	300	300	300	31.3	10.2	169.9	0	0
P-IDA*	0	5.1	4.9	5.3	300	300	300	144.1	29.1	299.7	0	0
B-IDA*	0	5.3	5.1	5.5	300	300	300	165.3	85.7	283.2	0	0
BP-IDA*	0	5.1	4.9	5.2	300	300	300	213.4	119.6	299.4	0	0
GF-IDA*	48	0.0	0.0	0.2	156.9	0.7	300	7.1	0.7	73.2	0	0
GF-P-IDA*	44	0.1	0.0	1.0	168.5	0.7	300	13.5	0.7	191.8	0	8
GF-B-IDA*	48	0.0	0.0	0.1	176.4	34.2	300	54.9	34.2	182.2	0	0
GF-BP-IDA*	44	0.1	0.0	1.0	205.1	67.2	300	94.2	67.2	207.0	0	8
Beam search $\beta = 5$	16	0.1	0.0	0.2	11.1	8.1	13.7	11.1	8.1	13.7	0	0
BS-B-IDA*	16	0.1	0.0	0.2	218.5	39.0	253.7	51.3	39.0	61.6	0	0
BS-BP-IDA*	16	0.1	0.0	0.2	218.5	39.0	253.7	50.8	39.0	61.7	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	5.3	5.1	5.7	300	300	300	113.9	20.0	299.8	0	0	4
P-IDA*	0	5.1	4.9	5.2	300	300	300	139.1	14.0	296.2	0	0	0
B-IDA*	0	5.1	4.9	5.2	300	300	300	168.8	50.0	297.0	0	0	0
BP-IDA*	0	5.0	4.9	5.2	300	300	300	198.6	49.9	299.6	0	0	0
GF-IDA*	38	0.4	0.0	4.2	186.3	0.4	300	25.4	0.4	274.1	0	8	12
GF-P-IDA*	44	0.3	0.0	3.8	168.2	0.3	300	22.6	0.3	298.4	0	8	8
GF-B-IDA*	38	0.4	0.0	4.3	193.2	14.8	300	36.7	14.8	271.0	0	8	12
GF-BP-IDA*	44	0.4	0.0	4.3	184.5	29.4	300	53.3	29.4	243.3	0	8	8
Beam search $\beta = 5$	14	0.2	0.0	3.6	6.3	4.1	7.8	6.3	4.1	7.8	0	4	4
BS-B-IDA*	14	0.1	0.0	0.3	215.3	21.2	247.8	29.7	17.1	121.0	0	0	8
BS-BP-IDA*	14	0.1	0.0	1.2	215.3	21.2	247.8	36.1	17.3	245.8	0	4	4

Table 2.10. 2-way results for 50 2000-ft instances

2.6.4.2. *Small Costs: 1-way Distance Function*

We present the results for the 1-way distance function for all tests. We begin with small size problems in Table 2.11. The ILP formulation exhibits the best overall performance on the gantry crane, and finds the highest percentage of instances to meet the LB for the reach stacker, but has a higher average gap to the LB than GF methods which leverage the LB to reduce the search space. Among the heuristics, the combination of beam search with IDA* methods perform best on the gantry crane, while GF-BP-IDA* performs best when using the reach stacker. Leveraging the preferred action space is detrimental when applied with GF methods and BS methods using the gantry crane, but beneficial in all cases for the reach stacker. GF methods improve upon non GF heuristics when considering the gantry crane. However, when considering the reach stacker GF methods leveraging the preferred

action space outperform GF methods which do not. Leveraging the LB to reduce the search space has a slight positive effect for all methods. The ILP formulation requires the longest CPU time, with all heuristics running on average for less than a quarter of the time spent on the ILP.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
ILP	70	0.0	0.0	0.2	1423.0	-	-	-	-	-	0	100
IDA*	0	2.4	1.7	2.6	300	300	300	141.7	2.6	299.5	0	0
P-IDA*	0	2.1	1.7	2.5	300	300	300	152.8	0.5	276.4	0	0
B-IDA*	0	2.3	1.7	2.5	300	300	300	174.5	3.4	298.4	0	0
BP-IDA*	0	2.1	1.7	2.5	300	300	300	161.2	3.0	295.1	0	0
GF-IDA*	34	0.1	0.0	0.8	198.4	0.0	300	9.8	0.0	223.0	0	0
GF-P-IDA*	12	0.3	0.0	1.0	264.0	0.0	300	8.8	0.0	186.7	0	0
GF-B-IDA*	34	0.1	0.0	0.8	198.6	0.6	300	14.6	0.6	220.2	0	0
GF-BP-IDA*	16	0.3	0.0	1.0	257.9	1.3	300	12.7	1.2	161.8	0	0
Beam search $\beta = 5$	34	1.3	0.0	18.5	0.8	0.6	1.1	0.8	0.6	1.1	0	8
BS-B-IDA*	34	0.1	0.0	0.3	159.5	1.1	241.0	6.0	1.1	59.5	0	0
BS-BP-IDA*	34	0.1	0.0	0.4	159.4	1.1	241.0	6.2	1.1	159.6	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
ILP	28	1.5	0.0	46.2	1388	-	-	-	-	-	0	4	4
IDA*	0	2.3	1.8	2.7	300	300	300	141.4	0.1	299.0	0	0	0
P-IDA*	0	2.1	1.8	2.7	300	300	300	150.9	0.1	286.1	0	0	0
B-IDA*	0	2.2	1.8	2.7	300	300	300	193.6	1.3	297.8	0	0	0
BP-IDA*	0	2.1	1.8	2.7	300	300	300	194.6	2.3	299.7	0	0	0
GF-IDA*	8	3.0	0.0	16.8	276.0	0.0	300	11.0	0.0	195.5	0	0	36
GF-P-IDA*	12	1.4	0.0	19.6	264.0	0.0	300	10.5	0.0	194.7	0	0	10
GF-B-IDA*	8	2.8	0.0	16.8	276.1	0.7	300	15.5	0.7	247.6	0	0	34
GF-BP-IDA*	16	1.4	0.0	19.6	258.8	1.4	300	13.7	1.3	185.9	0	0	10
Beam search $\beta = 5$	8	3.4	0.0	17.2	0.8	0.6	1.1	0.8	0.6	1.1	0	8	44
BS-B-IDA*	8	2.2	0.0	10.6	221.6	1.7	241.1	6.1	1.2	78.3	0	0	40
BS-BP-IDA*	8	2.0	0.0	10.6	221.7	1.7	241.1	5.5	1.2	87.5	0	0	38

Table 2.11. 1-way results for 50 667-ft instances

Examining the medium size problems in Table 2.12, we see that GF methods which do not leverage the preferred action space perform best overall for the gantry crane. Meanwhile, beam search, BS-B-IDA* and and BS-BP-IDA* perform best for the reach stacker, as other methods are prone to failing to fill all slots on the railcars for at least 4% of instances. GF methods solve most instances to meet the LB for both the reach stacker and gantry crane, but BS methods solve a large percentage to the LB for both types of handling equipment as well. The use of the LB to reduce the search space is beneficial to IDA* and P-IDA* methods, but not GF or BS based methods, wherein there is no change in result. Average runtimes fall well under the 300 seconds time limit, as a high fraction of cases are solved to meet the LB. BS heuristics are the only algorithms able to find solutions which fill all slots across all instances for both gantry crane and reach stacker.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	3.0	1.4	59.0	300	300	300	79.4	0.3	298.5	2	2
P-IDA*	0	2.9	1.5	59.0	300	300	300	39.3	0.2	293.8	2	2
B-IDA*	0	3.0	1.4	59.0	300	300	300	88.5	5.4	240.5	2	2
BP-IDA*	0	2.9	1.5	59.0	300	300	300	53.4	5.9	298.3	2	2
GF-IDA*	68	0.0	0.0	0.3	96.2	0.1	300	0.2	0.1	2.0	0	0
GF-P-IDA*	56	2.4	0.0	58.9	132.1	0.1	300	0.2	0.1	1.2	4	4
GF-B-IDA*	68	0.0	0.0	0.3	98.1	1.3	300	3.3	1.3	7.9	0	0
GF-BP-IDA*	56	2.4	0.0	58.9	135.1	2.6	300	6.4	2.6	16.2	4	4
Beam search $\beta = 5$	54	0.1	0.0	0.3	2.1	1.1	3.6	2.1	1.1	3.6	0	0
BS-B-IDA*	58	0.1	0.0	0.3	104.6	2.4	243.3	5.4	2.4	10.1	0	0
BS-BP-IDA*	58	0.1	0.0	0.3	106.6	2.4	243.3	7.4	2.4	54.7	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	5.3	1.6	59.0	300	300	300	49.1	0.1	251.1	2	10	24
P-IDA*	0	5.3	1.5	59.0	300	300	300	28.4	0.1	166.3	2	10	24
B-IDA*	0	5.2	1.5	59.0	300	300	300	55.7	1.8	270.6	2	10	24
BP-IDA*	0	5.3	1.5	59.0	300	300	300	37.6	3.5	240.8	2	10	24
GF-IDA*	34	5.1	0.0	58.9	198	0.1	300	7.5	0.1	154.7	6	6	12
GF-P-IDA*	56	3.4	0.0	58.9	132.1	0.1	300	0.1	0.1	0.7	4	4	10
GF-B-IDA*	34	5.1	0.0	58.9	198.6	0.9	300	15.7	0.9	286.5	6	6	12
GF-BP-IDA*	56	3.4	0.0	58.9	134	1.8	300	4.1	1.8	9.7	4	4	10
Beam search $\beta = 5$	48	1.5	0.0	16.2	1.4	0.7	2.6	1.4	0.7	2.6	0	4	10
BS-B-IDA*	52	1.3	0.0	14.2	117.7	1.5	242.6	3.5	1.5	7.1	0	0	10
BS-BP-IDA*	52	1.3	0.0	14.2	117.7	1.5	242.6	3.5	1.5	7.1	0	0	10

Table 2.12. 1-way results for 50 1000-ft instances

Large instances are considered in Table 2.13 and the best overall performance is found in the coupling of beam search with IDA* methods for the gantry crane, and using GF methods with the preferred action space for the reach stacker. Leveraging the preferred action space has positive effects when using the reach stacker. However, GF methods reduce the percentage of instances that are optimally solved when using the gantry crane. Leveraging the LB to reduce the search space has little impact, only reducing the mean gap to the LB for the IDA* algorithms. Runtimes continue to rise as the percentage of cases solved to meet the LB decreases.

Finally, when we examine the largest size instances, with 150 containers in Table 2.14, we find that GF methods improve drastically over non GF search methods. BS methods have the smallest average gap to the LB when considering the gantry crane, while GF-P-IDA* and GF-BP-IDA* have the smallest average gap to the LB when considering the reach stacker. Interestingy, IDA* provides no benefit to beam search here. Leveraging the preferred action space is beneficial when not searching GF. Finally, bounding search using the LB is beneficial only for non GF methods, and increases the gap to the LB when considering GF methods for the reach stacker.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	2.5	2.0	2.6	300	300	300	95.4	0.4	284.2	0	0
P-IDA*	0	2.4	1.7	2.5	300	300	300	121.1	0.4	255.4	0	0
B-IDA*	0	2.5	1.9	2.7	300	300	300	150.7	15.2	280.7	0	0
BP-IDA*	0	2.3	1.7	2.5	300	300	300	153.6	30.6	296.7	0	0
GF-IDA*	40	0.0	0.0	0.2	180.2	0.4	300	0.6	0.3	3.1	0	0
GF-P-IDA*	32	0.1	0.0	0.4	204.6	0.4	300	8.1	0.3	169.1	0	0
GF-B-IDA*	40	0.0	0.0	0.2	186.1	12.4	300	24.6	10.9	237.8	0	0
GF-BP-IDA*	32	0.1	0.0	0.4	214.5	23.9	300	33.2	21.9	70.1	0	0
Beam search $\beta = 5$	44	0.0	0.0	0.1	5.8	4.3	7.4	5.8	4.3	7.4	0	0
BS-B-IDA*	44	0.0	0.0	0.1	146.6	16.4	247.4	21.0	15.3	26.7	0	0
BS-BP-IDA*	44	0.0	0.0	0.1	146.6	16.4	247.4	20.9	16.2	26.4	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	2.7	2.2	8.9	300	300	300	117.3	0.3	289.8	0	4	6
P-IDA*	0	2.5	2.2	8.9	300	300	300	83.2	1.2	291.9	0	0	6
B-IDA*	0	2.6	2.2	8.9	300	300	300	146.7	6.6	293.9	0	4	6
BP-IDA*	0	2.5	2.2	8.9	300	300	300	120.9	17.6	298.7	0	0	6
GF-IDA*	26	1.1	0.0	12.3	227.2	0.2	300	17.9	0.2	277.8	0	0	16
GF-P-IDA*	32	0.2	0.0	1.9	205.4	0.2	300	3.8	0.2	35.6	0	6	0
GF-B-IDA*	28	1.0	0.0	12.3	225.3	5.4	300	20.9	5.4	219.9	0	0	14
GF-BP-IDA*	32	0.2	0.0	1.9	212.0	10.5	300	24.6	10.4	124.0	0	6	0
Beam search $\beta = 5$	20	1.1	0.0	15.0	3.6	2.6	4.4	3.6	2.6	4.4	0	4	10
BS-B-IDA*	20	0.5	0.0	8.4	197.0	9.0	244.4	13.1	8.7	48.0	0	0	10
BS-BP-IDA*	20	0.5	0.0	8.4	197.0	8.9	244.4	13.0	8.7	49.6	0	0	10

Table 2.13. 1-way results for 50 1500-ft instances

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	3.3	3.2	3.4	300	300	300	181.4	1.8	299.0	0	0
P-IDA*	0	3.2	3.0	3.3	300	300	300	150.6	19.1	299.7	0	0
B-IDA*	0	3.3	3.2	3.4	300	300	300	183.6	42.8	298.8	0	0
BP-IDA*	0	3.2	3.0	3.3	300	300	300	195.4	103.2	299.1	0	0
GF-IDA*	44	0.0	0.0	0.1	169.4	0.9	300	11.3	0.9	128.6	0	0
GF-P-IDA*	30	0.1	0.0	0.3	210.3	0.8	300	2.8	0.7	14.6	0	0
GF-B-IDA*	48	0.1	0.0	2.0	179.7	35.6	300	77.0	34.6	289.3	0	2
GF-BP-IDA*	30	0.1	0.0	0.3	235.3	71.9	300	88.4	62.6	116.7	0	0
Beam search $\beta = 5$	48	0.0	0.0	0.0	10.9	8.1	13.5	10.9	8.1	13.5	0	0
BS-B-IDA*	48	0.0	0.0	0.0	154.2	37.6	252.7	49.7	37.6	60.2	0	0
BS-BP-IDA*	48	0.0	0.0	0.0	154.3	37.4	252.7	49.8	37.4	60.1	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	3.3	3.1	3.3	300	300	300	107.5	34.7	280.0	0	0	0
P-IDA*	0	3.1	2.9	3.2	300	300	300	153.1	12.4	292.3	0	0	0
B-IDA*	0	3.2	3.0	3.3	300	300	300	178.0	47.0	272.7	0	0	0
BP-IDA*	0	3.2	3.0	3.2	300	300	300	191.0	68.4	294.9	0	0	0
GF-IDA*	24	0.4	0.0	3.7	228.2	0.4	300	22.7	0.4	289.9	0	0	20
GF-P-IDA*	30	0.1	0.0	0.3	210.1	0.4	300	21.9	0.4	235.1	0	0	0
GF-B-IDA*	24	0.4	0.0	3.7	232.5	14.9	300	29.8	14.9	235.5	0	0	20
GF-BP-IDA*	30	0.1	0.0	1.1	221.0	30.3	300	62.9	29.4	282.8	0	4	0
Beam search $\beta = 5$	32	0.2	0.0	2.9	6.5	4.9	7.7	6.5	4.9	7.7	0	0	8
BS-B-IDA*	32	0.2	0.0	2.9	176.1	21.0	247.7	27.0	20.9	32.0	0	0	8
BS-BP-IDA*	32	0.2	0.0	2.9	176.0	20.6	247.7	26.9	20.6	31.7	0	0	8

Table 2.14. 1-way distance functions for 50 2000-ft instances

2.6.4.3. *Small Costs: 0-way Distance Function*

We present 0-way distance function results for small, medium, large and very large size problems. We begin with small instances in Table 2.15. We see that nearly all heuristics are able to achieve performances meeting the LB on all instances, in most cases with low average computing time. Unlike when we considered the 1-way distance function, the ILP no longer outperforms all heuristics.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
ILP	100	0.0	0.0	0.0	651.0	-	-	-	-	-	-	-
IDA*	100	0.0	0.0	0.0	0.1	0.0	0.2	0.1	0.0	0.2	0	0
P-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.1	0.1	0.0	0.1	0	0
B-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.2	0.1	0.0	0.2	0	0
BP-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.2	0.1	0.0	0.2	0	0
GF-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.4	0.1	0.0	0.4	0	0
GF-P-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.1	0.1	0.0	0.1	0	0
GF-B-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.4	0.1	0.0	0.4	0	0
GF-BP-IDA*	100	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0	0
Beam search $\beta = 5$	100	0.0	0.0	0.0	0.7	0.6	1.0	0.7	0.6	1.0	0	0
BS-B-IDA*	100	0.0	0.0	0.0	0.8	0.6	1.1	0.8	0.6	1.1	0	0
BS-BP-IDA*	100	0.0	0.0	0.0	0.8	0.6	1.1	0.8	0.6	1.1	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
ILP	100	0.0	0.0	0.0	1210.0	-	-	-	-	-	-	-	-
IDA*	92	0.0	0.0	0.0	24.0	0.0	300	0.0	0.0	0.2	0	0	0
P-IDA*	100	0.0	0.0	0.0	0.1	0.0	0.2	0.1	0.0	0.2	0	0	0
B-IDA*	92	0.0	0.0	0.0	24.0	0.0	300	0.0	0.0	0.2	0	0	0
BP-IDA*	100	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.2	0	0	0
GF-IDA*	100	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0	0	0
GF-P-IDA*	100	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0	0	0
GF-B-IDA*	98	0.0	0.0	0.0	6.0	0.0	300	0.0	0.0	0.1	0	0	0
GF-BP-IDA*	98	0.0	0.0	0.0	6.0	0.0	300	0.0	0.0	0.1	0	0	0
Beam search $\beta = 5$	100	0.0	0.0	0.0	0.9	0.5	10.8	0.9	0.5	10.8	0	0	0
BS-B-IDA*	100	0.0	0.0	0.0	0.9	0.6	10.8	0.9	0.6	10.8	0	0	0
BS-BP-IDA*	100	0.0	0.0	0.0	0.9	0.6	10.8	0.9	0.6	10.8	0	0	0

Table 2.15. 0-way results for 50 667-ft instances

Next, we consider the medium size instances in Table 2.16. When considering the gantry crane, the ILP formulation outperforms all heuristics. However, this is not the case for the reach stacker. We see that almost all heuristics have instances with double touch actions, as well as incomplete loads. It is interesting that the beam search combined with depth-first search methods do not manage to find a solution without double touching here, but can do so on both the 1-way and 2-way distance function. We believe the relatively similar quality across most algorithms is related to the symmetry amongst costs for most actions, paired with the few number of lots available in this instance size. That is, all algorithms search in a similar order.

Next, we consider the large instances in Table 2.17. We see a return to most heuristics finding solutions which meet the LB on all instances for the gantry crane. However, some

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
ILP	100	0.0	0.0	0.0	2688.0	-	-	-	-	-	0	0
IDA*	98	1.3	0.0	65.9	6.1	0.1	300	0.1	0.1	0.2	2	2
P-IDA*	98	1.3	0.0	65.9	6.2	0.1	300	0.2	0.1	1.0	2	2
B-IDA*	98	1.3	0.0	65.9	6.1	0.1	300	0.1	0.1	0.2	2	2
BP-IDA*	98	1.3	0.0	65.9	6.2	0.1	300	0.2	0.1	1.0	2	2
GF-IDA*	96	2.6	0.0	65.9	12.1	0.1	300	0.1	0.1	0.3	4	4
GF-P-IDA*	96	2.6	0.0	65.9	12.1	0.1	300	0.1	0.1	0.3	4	4
GF-B-IDA*	96	2.6	0.0	65.9	12.1	0.1	300	0.1	0.1	0.3	4	4
GF-BP-IDA*	96	2.6	0.0	65.9	12.1	0.1	300	0.1	0.1	0.3	4	4
Beam search $\beta = 5$	98	2.0	0.0	100.0	1.9	1.1	3.5	1.9	1.1	3.5	2	2
BS-B-IDA*	98	1.3	0.0	65.0	6.8	1.1	242.6	2.0	1.1	3.8	2	2
BS-BP-IDA*	98	1.3	0.0	65.0	6.8	1.1	242.6	2.0	1.1	3.8	2	2

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
ILP	14	5.9	0.0	49.7	17450.0	-	-	-	-	-	0	14	22
IDA*	70	3.8	0.0	65.9	90.1	0.0	300	14.1	0.0	203.8	2	16	18
P-IDA*	70	3.8	0.0	65.9	90.1	0.0	300	4.9	0.0	60.6	2	16	18
B-IDA*	70	3.8	0.0	65.9	90.1	0.0	300	14.2	0.0	205.0	2	16	18
BP-IDA*	70	3.8	0.0	65.9	90.1	0.0	300	4.9	0.0	59.1	2	16	18
GF-IDA*	90	3.3	0.0	65.9	30.1	0.1	300	9.5	0.1	255.5	4	8	2
GF-P-IDA*	92	2.0	0.0	65.9	27.7	0.1	300	8.8	0.1	181.7	2	6	2
GF-B-IDA*	90	3.3	0.0	65.9	30.1	0.0	300	5.7	0.0	151.0	4	8	2
GF-BP-IDA*	92	2.0	0.0	65.9	27.7	0.0	300	7.0	0.0	181.8	2	6	2
Beam search $\beta = 5$	88	3.0	0.0	100.0	1.5	0.8	11.0	1.5	0.8	11.0	2	2	10
BS-B-IDA*	88	2.2	0.0	65.0	30.3	0.8	242.3	3.0	0.8	41.5	2	6	6
BS-BP-IDA*	92	1.7	0.0	65.0	26	0.8	242.3	12.4	0.8	149.5	2	6	2

Table 2.16. 0-way results for 50 1000-ft instances

double touches occur with several heuristics when considering the reach stacker. The preferred action space is beneficial to the beam search coupled with IDA* algorithms, as well as GF algorithms for the reach stacker, but the opposite is seen for the gantry crane. Overall, heuristics now outperform the ILP formulation in a fraction of the time allocated.

Finally, we present the very large size problems in Table 2.18. Here, all IDA* methods are able to find optimal solutions in all cases, as are almost all methods, except beam search, which fails to find optimal solutions to 2 % of cases.

2.6.4.4. *Large Costs: 2-way Distance Function*

We begin by presenting results for the 2-way distance function. First, examining Table 2.19 we see that for medium size instances, BS coupled with IDA* offer the best performance. Moreover, BS-B-IDA* is able to solve 50% of instances to meet the LB for the gantry crane. GF methods fail critically on 4% of instances, leaving slots open in these cases for both the gantry crane and the reach stacker. Leveraging the preferred action space is detrimental when comparing P-IDA* and BP-IDA*, and is not consistently beneficial or detrimental otherwise.

When considering the larger instances in Table 2.20, performance is better on average than that achieved over the medium size instances. Here, the GF methods which do not leverage the preferred action space outperform BS methods for the gantry crane. BS methods

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
ILP	30	14.6	0.0	44.2	22777.0	-	-	-	-	-	0	70
IDA*	100	0.0	0.0	0.0	1.1	0.3	12.9	1.1	0.3	12.9	0	0
P-IDA*	100	0.0	0.0	0.0	1.2	0.3	14.4	1.2	0.3	14.4	0	0
B-IDA*	100	0.0	0.0	0.0	1.1	0.3	12.9	1.1	0.3	12.9	0	0
BP-IDA*	100	0.0	0.0	0.0	1.2	0.3	14.5	1.2	0.3	14.5	0	0
GF-IDA*	100	0.0	0.0	0.0	1.5	0.3	9.1	1.5	0.3	9.1	0	0
GF-P-IDA*	96	0.1	0.0	2.0	13.6	0.4	300	2.0	0.4	19.7	0	4
GF-B-IDA*	100	0.0	0.0	0.0	1.5	0.3	9.0	1.5	0.3	9.0	0	0
GF-BP-IDA*	96	0.1	0.0	2.0	13.6	0.3	300	1.9	0.3	19.4	0	4
Beam search $\beta = 5$	94	0.6	0.0	13.5	5.4	3.9	7.2	5.4	3.9	7.2	0	6
BS-B-IDA*	100	0.0	0.0	0.0	6.0	4.2	10.4	6.0	4.2	10.4	0	0
BS-BP-IDA*	100	0.0	0.0	0.0	6.0	4.2	10.5	6.0	4.2	10.5	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
ILP	0	68.5	60.7	73.2	35910.0	-	-	-	-	-	0	98	100
IDA*	78	0.1	0.0	2.0	67.0	0.2	300	3.9	0.2	42.8	0	6	0
P-IDA*	96	0.0	0.0	2.0	13.5	0.2	300	1.9	0.2	30.0	0	2	0
B-IDA*	78	0.1	0.0	2.0	67.0	0.2	300	3.9	0.2	42.4	0	6	0
BP-IDA*	96	0.0	0.0	2.0	13.5	0.2	300	1.8	0.2	28.9	0	2	0
GF-IDA*	96	0.1	0.0	2.0	23.5	0.2	300	18.4	0.2	281.6	0	4	0
GF-P-IDA*	100	0.0	0.0	0.0	15.7	0.2	299.0	15.7	0.2	299.0	0	0	0
GF-B-IDA*	96	0.1	0.0	2.0	19.0	0.2	300	11.1	0.2	167.6	0	4	0
GF-BP-IDA*	100	0.0	0.0	0.0	8.0	0.2	136.1	8.0	0.2	136.1	0	0	0
Beam search $\beta = 5$	94	2.1	0.0	100.0	2.7	2.3	3.7	2.7	2.3	3.7	2	6	0
BS-B-IDA*	94	0.5	0.0	11.9	17.3	2.5	242.7	11.2	2.5	228.9	0	4	4
BS-BP-IDA*	100	0.0	0.0	0.0	2.9	2.5	4.4	2.9	2.5	4.4	0	0	0

Table 2.17. 0-way results for 50 1500-ft instances

are able to solve 26% of cases to meet LB and GF-B-IDA* solves 52% of cases to meet the LB. Leveraging the preferred action space is inconsistently beneficial, reducing the gap to the LB for IDA* heuristics and B-IDA* when considering the gantry crane, but having a detrimental effect on almost all other heuristics. The best overall performance for the gantry crane is achieved by GF-B-IDA* when considering the mean gap to the LB and the number of instances solved to meet the LB, while for the reach stacker the best performance in terms of the mean gap to the LB is achieved by BS-BP-IDA* whereas GF-BP-IDA* and GF-B-IDA* find more examples to meet the LB.

Considering the largest instances in Table 2.21 we see that beam search methods generally perform the best on average in terms of the range of the gaps to the LB. It is notable that the performances of BS-B-IDA* and BS-BP-IDA* are better on average than those of GF-B-IDA* and GF-IDA*, despite finding fewer solutions meeting the LB. Leveraging the preferred action space is beneficial across all heuristics for the reach stacker, but not for GF heuristics for the gantry crane. Additionally, leveraging the LB is beneficial for the gantry crane, but detrimental for the reach stacker.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
ILP	0	32.7	23.7	39.9	35995.0	-	-	-	-	-	0	100
IDA*	100	0.0	0.0	0.0	0.9	0.7	1.9	0.9	0.7	1.9	0	0
P-IDA*	100	0.0	0.0	0.0	0.9	0.7	1.7	0.9	0.7	1.7	0	0
B-IDA*	100	0.0	0.0	0.0	0.9	0.6	1.9	0.9	0.6	1.9	0	0
BP-IDA*	100	0.0	0.0	0.0	0.9	0.6	1.8	0.9	0.6	1.8	0	0
GF-IDA*	100	0.0	0.0	0.0	1.0	0.7	7.3	1.0	0.7	7.3	0	0
GF-P-IDA*	100	0.0	0.0	0.0	1.0	0.7	7.3	1.0	0.7	7.3	0	0
GF-B-IDA*	100	0.0	0.0	0.0	1.0	0.7	7.2	1.0	0.7	7.2	0	0
GF-BP-IDA*	100	0.0	0.0	0.0	1.0	0.7	7.3	1.0	0.7	7.3	0	0
Beam search $\beta = 5$	98	0.0	0.0	1.2	10.7	8.1	14.1	10.7	8.1	14.1	0	2
BS-B-IDA*	100	0.0	0.0	0.0	11.6	8.8	15.2	11.6	8.8	15.2	0	0
BS-BP-IDA*	100	0.0	0.0	0.0	11.6	8.8	15.2	11.6	8.8	15.2	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
ILP	0	67.8	62.3	70.1	35891.0	-	-	-	-	-	0	100	100
IDA*	96	0.0	0.0	0.0	13.2	0.3	300	1.2	0.3	19.6	0	0	0
P-IDA*	100	0.0	0.0	0.0	0.6	0.3	3.9	0.6	0.3	3.9	0	0	0
B-IDA*	96	0.0	0.0	0.0	13.2	0.3	300	1.3	0.3	20.1	0	0	0
BP-IDA*	100	0.0	0.0	0.0	0.6	0.3	3.9	0.6	0.3	3.9	0	0	0
GF-IDA*	100	0.0	0.0	0.0	0.6	0.3	9.1	0.6	0.3	9.1	0	0	0
GF-P-IDA*	100	0.0	0.0	0.0	0.5	0.3	3.1	0.5	0.3	3.1	0	0	0
GF-B-IDA*	100	0.0	0.0	0.0	0.5	0.3	3.5	0.5	0.3	3.5	0	0	0
GF-BP-IDA*	100	0.0	0.0	0.0	0.5	0.3	2.1	0.5	0.3	2.1	0	0	0
Beam search $\beta = 5$	98	0.0	0.0	2.4	5.6	4.1	6.9	5.6	4.1	6.9	0	2	0
BS-B-IDA*	100	0.0	0.0	0.0	6.0	4.4	11.0	6.0	4.4	11.0	0	0	0
BS-BP-IDA*	100	0.0	0.0	0.0	6.0	4.4	8.4	6.0	4.4	8.4	0	0	0

Table 2.18. 0-way results for 50 2000-ft instances

2.6.4.5. *Large Costs: 1-way Distance Function*

First, we report the results for medium size problems using the 1-way distance function in Table 2.22. Overall, BS-B-IDA* and BS-BP-IDA* have the best performances. We see that leveraging the preferred action space is detrimental, except for the methods based on beam search. Leveraging the LB to reduce the search space shows inconsistent results, sometimes helping and sometimes producing very poor results, as is the case of BP-IDA* when compared with P-IDA*. However, bounding helps very little when it does. Greedy first methods do not consistently outperform their corresponding baselines, having a negative impact for the gantry crane, and a positive impact for the reach stacker, only when not leveraging the preferred action space.

Next, we consider Table 2.23, where the BS-B-IDA* and BS-BP-IDA* methods perform best overall for both reach stacker and gantry crane. BS-B-IDA* is able to find 50 % of cases to meet the LB for the gantry crane while maintaining a mean gap to the LB of less than 0.1%. The BS-B-IDA* and BS-BP-IDA* methods are unable to solve any instances to meet the LB for the reach stacker, but maintain a mean gap to the LB of 0.7%. Leveraging the preferred action space is beneficial for most non GF methods, except when leveraging the LB when using the reach stacker. Leveraging the LB is beneficial for GF methods when

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	3.0	1.4	56.8	300	300	300	76.1	0.2	295.0	2	2
P-IDA*	0	2.9	1.4	57.0	300	300	300	78.3	0.2	273.4	2	2
B-IDA*	0	2.9	1.4	57.0	300	300	300	122.0	4.9	293.5	2	2
BP-IDA*	0	12.1	0.1	57.4	300	300	300	156.7	4.6	300	20	0
GF-IDA*	52	2.3	0.0	56.5	147.6	0.1	300	19.5	0.1	207.5	4	4
GF-P-IDA*	18	2.5	0.0	56.7	253.4	0.1	300	19.0	0.1	208.8	4	8
GF-B-IDA*	58	2.3	0.0	56.5	131.4	1.5	300	19.6	1.5	112.7	4	4
GF-BP-IDA*	18	2.2	0.0	50.4	252.8	2.9	300	37.8	2.9	300	4	4
Beam search $\beta = 5$	42	0.1	0.0	0.4	2.0	1.0	5.6	2.0	1.0	5.6	0	0
BS-B-IDA*	50	0.1	0.0	0.4	124.5	2.4	244.5	7.5	2.3	35.1	0	0
BS-BP-IDA*	46	0.1	0.0	0.4	135.1	2.4	244.5	9.9	2.3	89.5	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	3.3	1.4	56.8	300	300	300	67.2	0.1	259.2	2	10	44
P-IDA*	0	3.2	1.4	57.0	300	300	300	60.2	0.1	295.3	2	10	26
B-IDA*	0	3.2	1.4	57.0	300	300	300	91.3	1.9	296.8	2	10	30
BP-IDA*	0	15.2	1.4	57.1	300	300	300	172.6	4.9	300	26	0	0
GF-IDA*	14	2.9	0.0	56.5	258.0	0.1	300	23.4	0.1	145.9	4	16	52
GF-P-IDA*	18	2.8	0.0	56.7	251.7	0.1	300	35.2	0.1	265.2	4	20	32
GF-B-IDA*	14	2.8	0.0	56.5	258.2	0.8	300	25.8	0.8	213.9	4	14	44
GF-BP-IDA*	18	3.1	0.0	56.7	251.0	1.5	300	30.7	1.5	218.9	4	26	34
Beam search $\beta = 5$	0	0.9	0.0	5.8	1.3	0.7	2.4	1.3	0.7	2.4	0	16	58
BS-B-IDA*	0	0.5	0.0	4.8	241.3	240.7	242.4	39.1	1.7	167.0	0	8	38
BS-BP-IDA*	0	0.5	0.0	4.9	241.3	240.7	242.4	34.7	1.7	210.9	0	10	34

Table 2.19. 2-way results for 50 1000-ft instances using large cost values

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	3.0	2.6	3.1	300	300	300	179.1	0.8	289.5	0	0
P-IDA*	0	2.8	2.4	3.0	300	300	300	126.7	0.8	296.8	0	0
B-IDA*	0	2.9	2.5	3.0	300	300	300	186.1	17.1	296.7	0	0
BP-IDA*	0	2.8	2.4	3.0	300	300	300	133.2	33.1	286.9	0	0
GF-IDA*	28	0.0	0.0	0.1	221.3	0.3	300	28.1	0.3	143.4	0	0
GF-P-IDA*	24	0.1	0.0	0.4	230.7	0.3	300	19.1	0.3	298.5	0	0
GF-B-IDA*	52	0.0	0.0	0.1	172.4	10.5	300	49.0	10.5	172.3	0	0
GF-BP-IDA*	30	0.1	0.0	0.4	223.7	20.7	300	53.3	20.6	294.6	0	0
Beam search $\beta = 5$	26	0.0	0.0	0.2	6.0	3.5	14.2	6.0	3.5	14.2	0	0
BS-B-IDA*	26	0.0	0.0	0.2	188.3	13.7	253.4	22.4	13.7	52.7	0	0
BS-BP-IDA*	26	0.0	0.0	0.2	188.4	13.6	253.4	22.4	13.4	53.8	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	3.0	2.7	4.5	300	300	300	166.6	0.4	296.7	0	4	18
P-IDA*	0	2.8	2.5	3.1	300	300	300	104.2	6.1	249.6	0	0	8
B-IDA*	0	2.9	2.7	4.5	300	300	300	163.0	7.5	299.0	0	4	20
BP-IDA*	0	3.7	2.7	51.2	300	300	300	156.9	14.2	300	2	0	0
GF-IDA*	12	0.6	0.0	8.2	264.1	0.2	300	40.6	0.2	261.9	0	12	82
GF-P-IDA*	30	0.6	0.0	8.6	215.6	0.2	300	33.2	0.2	287.9	0	12	18
GF-B-IDA*	14	0.6	0.0	8.2	261.1	6.8	300	49.6	5.8	265.2	0	12	82
GF-BP-IDA*	30	0.8	0.0	10.1	216.4	13.1	300	58.9	11.5	293.2	0	16	20
Beam search $\beta = 5$	0	0.5	0.0	9.8	3.6	2.3	8.5	3.6	2.3	8.5	0	4	62
BS-B-IDA*	0	0.4	0.0	8.7	243.6	242.3	248.5	30.8	6.7	186.6	0	4	68
BS-BP-IDA*	0	0.4	0.0	7.3	243.6	242.3	248.5	36.2	6.8	225.0	0	4	60

Table 2.20. 2-way results for 50 1500-ft instances using large cost values

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	4.0	3.9	4.0	300	300	300	35.2	10.2	182.0	0	0
P-IDA*	0	3.8	3.7	3.9	300	300	300	163.9	29.1	297.3	0	0
B-IDA*	0	3.9	3.8	4.0	300	300	300	224.1	99.0	299.9	0	0
BP-IDA*	0	3.7	3.7	3.8	300	300	300	255.7	119.6	299.9	0	0
GF-IDA*	22	0.0	0.0	0.1	234.2	0.7	300	23.4	0.7	297.0	0	0
GF-P-IDA*	24	0.1	0.0	0.2	229.1	0.7	300	5.7	0.7	140.3	0	0
GF-B-IDA*	34	0.0	0.0	0.1	217.0	28.9	300	53.0	28.9	267.9	0	0
GF-BP-IDA*	24	0.1	0.0	0.2	247.4	56.9	300	94.6	56.9	287.8	0	0
Beam search $\beta = 5$	16	0.0	0.0	0.1	11.5	7.4	27.6	11.5	7.4	27.6	0	0
BS-B-IDA*	16	0.0	0.0	0.1	219.1	37.8	267.6	59.2	36.1	226.1	0	0
BS-BP-IDA*	16	0.0	0.0	0.1	219.1	38.1	267.6	55.5	36.2	183.4	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	4	0.5	0.0	11.4	288.8	20.3	300	65.1	0.3	293.8	0	4	66
P-IDA*	24	0.3	0.0	10.2	228.7	0.4	300	10.1	0.3	264.3	0	4	4
B-IDA*	4	0.5	0.0	12.7	288.5	12.8	300	46.8	12.8	292.6	0	4	66
BP-IDA*	24	0.3	0.0	11.6	235.3	23.1	300	41.3	23.1	298.9	0	4	4
GF-IDA*	4	0.5	0.0	12.1	288.9	22.0	300	64.2	0.4	274.2	0	4	66
GF-P-IDA*	24	0.3	0.0	10.2	228.8	0.4	300	9.8	0.4	231.3	0	4	4
GF-B-IDA*	4	0.6	0.0	13.4	288.7	15.0	300	41.7	14.0	281.9	0	4	66
GF-BP-IDA*	24	0.3	0.0	11.6	236.3	27.3	300	43.8	27.3	298.7	0	4	4
Beam search $\beta = 5$	0	0.3	0.0	9.6	6.4	3.9	15.7	6.4	3.9	15.7	0	4	68
BS-B-IDA*	0	0.3	0.0	9.6	246.4	243.9	255.7	64.3	18.5	253.5	0	2	64
BS-BP-IDA*	0	0.3	0.0	9.6	246.4	243.9	255.7	58.2	18.7	249.1	0	2	64

Table 2.21. 2-way results for 50 2000-ft instances using large cost values

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	2.3	0.9	59.6	300	300	300	74.4	0.2	280.9	2	2
P-IDA*	0	2.3	0.9	59.6	300	300	300	74.4	0.2	289	2	2
B-IDA*	0	2.3	0.9	59.6	300	300	300	103.5	5.0	297.1	2	2
BP-IDA*	0	12.2	0.1	61.2	300	300	300	155.3	12.2	300	20	0
GF-IDA*	50	2.4	0.0	59.3	152.6	0.1	300	10.9	0.1	282.8	4	4
GF-P-IDA*	22	2.5	0.0	60.6	238.1	0.1	300	17.0	0.1	111.6	4	4
GF-B-IDA*	60	2.4	0.0	59.3	135.6	1.4	300	18.6	1.4	283.8	4	4
GF-BP-IDA*	28	2.3	0.0	54.9	230.5	3.6	300	34.4	2.6	300	4	0
Beam search $\beta = 5$	60	0.0	0.0	0.2	1.7	1.0	3.2	1.7	1.0	3.2	0	0
BS-B-IDA*	68	0.0	0.0	0.2	80.9	2.2	243.1	5.0	2.2	15.8	0	0
BS-BP-IDA*	72	0.0	0.0	0.2	72.2	2.2	243.1	5.9	2.2	26.6	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	3.0	1.0	65.4	300	300	300	80.1	0.1	220.6	2	10	30
P-IDA*	0	3.0	1.0	65.4	300	300	300	64.3	0.1	286.9	2	10	24
B-IDA*	0	3.0	1.0	65.4	300	300	300	121.1	1.9	295.0	2	10	26
BP-IDA*	0	13	1.0	60.1	300	300	300	159.1	4.5	300	22	0	0
GF-IDA*	0	0.7	0.0	10.4	300	300	300	40.2	0.1	276.3	0	6	56
GF-P-IDA*	28	3.9	0.0	66.6	230.5	0.1	300	32.5	0.1	288.2	4	18	22
GF-B-IDA*	0	0.7	0.0	10.4	300	300	300	31.0	0.8	293.6	0	6	54
GF-BP-IDA*	24	3.7	0.0	54.9	235.1	1.5	300	41.2	1.4	300	4	14	18
Beam search $\beta = 5$	4	1.8	0.0	26.5	1.2	0.8	2.2	1.2	0.8	2.2	0	12	54
BS-B-IDA*	4	0.5	0.0	10.1	231.7	2.6	242.2	24.2	1.5	105.0	0	4	32
BS-BP-IDA*	4	0.5	0.0	10.2	231.7	2.5	242.2	37.1	1.5	237.7	0	4	28

Table 2.22. 1-way results for 50 1000-ft instances using large cost values

considering the gantry crane, but not the reach stacker. Leveraging the LB is beneficial for most non GF methods, except when paired with the preferred action space for the reach stacker as mentioned before. GF methods outperform their non greedy counterparts for the gantry crane, but can lead to relatively poor solutions when considering the reach stacker.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	1.8	1.6	1.9	300	300	300	125.2	0.7	297.0	0	0
P-IDA*	0	1.7	1.5	1.8	300	300	300	129.5	0.7	297.9	0	0
B-IDA*	0	1.7	1.5	1.8	300	300	300	204.7	14.4	295.2	0	0
BP-IDA*	0	1.7	1.5	1.8	300	300	300	142.8	28.6	298.2	0	0
GF-IDA*	20	0.0	0.0	0.1	240.4	0.4	300	13.5	0.3	269.5	0	0
GF-P-IDA*	0	0.1	0.0	0.3	300	300	300	27.0	0.3	263.5	0	0
GF-B-IDA*	32	0.0	0.0	0.1	216.2	12.4	300	49.9	10.3	264.6	0	0
GF-BP-IDA*	0	0.1	0.0	0.3	300	300	300	56.3	20.0	297.1	0	0
Beam search $\beta = 5$	44	0.0	0.0	0.1	5.5	3.5	14.2	5.5	3.5	14.2	0	0
BS-B-IDA*	50	0.0	0.0	0.1	137.8	14.3	254.2	25.3	13.1	130.8	0	0
BS-BP-IDA*	46	0.0	0.0	0.1	141.5	14.4	254.2	19.9	13.1	56.7	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	1.9	1.6	4.6	300	300	300	150.3	0.3	288.4	0	4	12
P-IDA*	0	1.7	1.5	1.9	300	300	300	78.7	4.0	295.7	0	0	8
B-IDA*	0	1.8	1.6	4.6	300	300	300	153.4	5.9	286.4	0	4	12
BP-IDA*	0	2.7	1.6	51.4	300	300	300	127.6	12.1	300	2	0	0
GF-IDA*	0	1.3	0.0	19.1	300	300	300	31.7	0.2	295.8	0	12	90
GF-P-IDA*	0	0.9	0.0	17.2	300	300	300	29.1	0.2	280.4	0	8	10
GF-B-IDA*	0	1.3	0.0	19.1	300	300	300	45.5	4.7	285.5	0	10	90
GF-BP-IDA*	2	1.0	0.0	21.0	299.2	258.5	300	44.1	9.6	258.5	0	8	10
Beam search $\beta = 5$	0	0.9	0.0	20.6	3.4	2.2	8.5	3.4	2.2	8.5	0	6	60
BS-B-IDA*	0	0.7	0.0	15.7	243.4	242.2	248.5	24.8	6.5	110.6	0	6	64
BS-BP-IDA*	0	0.7	0.0	14.8	243.4	242.2	248.5	27.1	6.6	161.5	0	6	60

Table 2.23. 1-way results for 50 1500-ft instances using large cost values

Finally, we consider the largest problem instances in Table 2.24. The best performing heuristics for both the reach stacker and gantry crane is BS-BP-IDA*, which solves 22 % of cases to meet the LB when considering the gantry crane. Moreover, this method maintains a gap to the LB of 0.1% for the reach stacker and less than 0.1 % for the gantry crane. GF methods outperform their non GF counterparts on average, but can lead to a small number of poor solutions for the reach stacker. Leveraging the preferred action space is beneficial when considering non GF methods, but leads to worse performance overall when considering GF methods. Using the LB during search is beneficial for non GF methods, but has no impact when used in conjunction with GF search for the gantry crane, and is detrimental when paired with GF search for the reach stacker.

2.6.5. Comparison of Results for Different Distance Functions

We are interested in judging the relative accuracy of the 1-way distance function and 0-way distance function. We compare solutions which meet the LB on their respective distance function to the gap to the LB when that solution is tested with the 2-way distance function.

Gantry Crane

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max		
IDA*	0	2.4	2.4	2.4	300	300	300	217.8	18.4	299.1	0	0
P-IDA*	0	2.3	2.3	2.4	300	300	300	165.8	30.9	299.7	0	0
B-IDA*	0	2.4	2.3	2.4	300	300	300	230.0	134.8	299.2	0	0
BP-IDA*	0	2.3	2.2	2.4	300	300	300	243.6	100.3	298.0	0	0
GF-IDA*	12	0.0	0.0	0.0	264.1	0.7	300	7.7	0.7	135.1	0	0
GF-P-IDA*	2	0.1	0.0	0.2	294.0	0.9	300	3.6	0.6	58.2	0	0
GF-B-IDA*	14	0.0	0.0	0.0	263.8	31.5	300	43.6	29.9	75.6	0	0
GF-BP-IDA*	2	0.1	0.0	0.2	295.5	75.5	300	92.7	56.5	296.9	0	0
Beam search $\beta = 5$	22	0.0	0.0	0.0	11.4	7.1	29.8	11.4	7.1	29.8	0	0
BS-B-IDA*	22	0.0	0.0	0.0	207.5	34.1	269.8	51.9	33.9	138.8	0	0
BS-BP-IDA*	22	0.0	0.0	0.0	207.5	34.1	269.8	59.7	34.1	159.4	0	0

Reach Stacker

Algorithm	Solved to LB [%]	Gap to LB [%]			CPU [s]			Time to Solution[s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max			
IDA*	0	2.4	2.3	2.4	300	300	300	155.0	46.2	300	0	0	0
P-IDA*	0	2.3	2.3	2.4	300	300	300	171.9	16.5	298.6	0	0	0
B-IDA*	0	2.3	2.3	2.4	300	300	300	241.5	71.5	299.9	0	0	0
BP-IDA*	0	2.3	2.2	2.4	300	300	300	237.4	57.6	299.3	0	0	0
GF-IDA*	2	0.5	0.0	15.9	294.0	0.5	300	28.1	0.3	256.0	0	4	98
GF-P-IDA*	2	1.0	0.0	15.6	294.0	0.5	300	19.4	0.3	295.0	0	6	2
GF-B-IDA*	2	0.5	0.0	15.9	294.4	19.4	300	49.3	11.6	265.6	0	4	98
GF-BP-IDA*	2	1.0	0.0	15.8	294.8	38.3	300	56.3	22.7	288.1	0	6	0
Beam search $\beta = 5$	0	0.2	0.0	4.0	6.2	4.0	16.0	6.2	4.0	16.0	0	4	64
BS-B-IDA*	0	0.1	0.0	2.1	246.2	244.0	256.0	49.9	18.4	188.2	0	2	60
BS-BP-IDA*	0	0.1	0.0	0.2	246.2	244.0	256.0	59.8	18.0	199.2	0	0	60

Table 2.24. 1-way results for 50 2000-ft instances using large cost values

We present these results in Tables 2.25 and 2.26. The gaps presented in Table 2.25 do not include container cost, while Table 2.26 reports the gap to the LB including container cost.

Notably, solutions which meet the LB on the 0-way distance function perform poorly when considered in a 2-way distance setting, having average gaps between 9.4% and 20.9% for the gantry crane and between 4.0% and 18.5 % for the gantry crane. Meanwhile, the 1-way distance function offers a much better approximation to the 2-way distance function, leading to a much smaller gap. Nonetheless, this gap exists and is larger than the gap to the LB for several heuristics applied to the 2-way distance function.

We make the same comparison with solutions which meet the LB on the 1-way distance function when using large values. We present these results in Tables 2.27 and 2.28. Notably, the distance gap shrinks for the 1-way distance function, for both the reach stacker and the gantry crane. Neither large nor very large size instances results are available for the reach stacker, since none of these instances were solved to reach the LB by the BS algorithms.

2.6.6. Discussion

The results presented above indicate that beam search paired with state space depth-first search (**BS-B-IDA***) is capable of consistently providing high-quality solutions up to very large sizes of LPSP problems in less than 5 minutes, while considering costs associated with

Gantry Crane

Length	0-Way Distance Function			1-Way Distance Function		
	Avg.	Min	Max	Avg.	Min	Max
667	20.9	10.4	29.6	6.5	4.0	8.9
1000	13.8	3.7	22.2	0.7	0.2	1.4
1500	9.4	6.7	12.4	0.8	0.5	1.2
2000	10.6	9.0	13.1	0.7	0.4	1.0

Reach Stacker

Length	0-Way Distance Function			1-Way Distance Function		
	Avg.	Min	Max	Avg.	Min	Max
667	18.5	8.0	23.9	5.7	5.4	6.0
1000	3.9	0.8	8.6	0.7	0.2	1.0
1500	4.0	1.9	7.5	0.9	0.5	1.1
2000	4.2	1.9	6.6	0.8	0.5	1.0

Table 2.25. Gap to LB of distance travelled using solutions meeting LB from each distance function on the 2-way distance function for small cost values

Gantry Crane

Length	0-way Distance Function			1-way Distance Function		
	Avg.	Min	Max	Avg.	Min	Max
667	4.6	3.4	6.4	1.3	0.8	1.6
1000	4.2	2.8	5.3	0.2	0.0	0.4
1500	3.7	3.0	4.7	0.3	0.2	0.5
2000	4.2	3.6	5.0	0.3	0.2	0.4

Reach Stacker

Length	0-way Distance Function			1-way Distance Function		
	Avg.	Min	Max	Avg.	Min	Max
667	4.0	2.3	5.8	0.9	0.8	0.9
1000	1.1	0.3	2.7	0.2	0.1	0.3
1500	1.5	0.7	2.7	0.3	0.2	0.4
2000	1.6	0.8	2.5	0.3	0.2	0.4

Table 2.26. Gap to LB using solutions meeting LB from each distance function on the 2-way distance function for small cost values

	Gantry Crane			Reach Stacker		
Length	1-way Distance Function			1-way Distance Function		
	Avg.	Min	Max	Avg.	Min	Max
1000	0.9	0.4	1.5	1.0	1.0	1.0
1500	1.0	0.6	1.2	N/A	N/A	N/A
2000	0.7	0.6	0.9	N/A	N/A	N/A

Table 2.27. Gap to LB of distance travelled using solutions meeting LB from each distance function on the 2-way distance function for small cost values

Length	Gantry Crane 1-way Distance Function			Reach Stacker 1-way Distance Function		
	Avg.	Min	Max	Avg.	Min	Max
1000	0.2	0.1	0.3	0.3	0.3	0.3
1500	0.3	0.2	0.4	N/A	N/A	N/A
2000	0.2	0.2	0.3	N/A	N/A	N/A

Table 2.28. Gap to LB using solutions meeting LB from each distance function on the 2-way distance function for large cost values

both handling and container loading. The heuristics proposed here are able to find solutions meeting the LB when using the 0-way cost function for all instance sizes, as well as solutions meeting the LB for both the 1-way distance function and 2-way distance function. While depth-first search methods employing a greedy search in the first stage are sometimes able to outperform the combination of beam search and depth-first search, finding more instances whose solutions meet the LB, it was found that depth-first search alone is susceptible to poorer performance on average than beam search coupled with IDA*. We also note that beam search alone, while very fast, can lead to poor solutions.

The LB can be used in conjunction with search to discard nodes which are not promising, by calculating the cost of the optimal solution from the current state before choosing to explore the node further. This led to mixed results throughout testing, yielding sometimes better and sometimes worse solutions within the time limit. The primary drawback of using the bound is the time to calculate the LB from each state. Indeed, as frequently seen in relation with the GF methods, using the LB to bound the search requires more time to find the same quality of solution. This is especially true when considering larger instances. The time to calculate the LB is longer on larger instances, which leads to poorer performance within the time budget. Finding a faster method of determining the LB could lead to improvements.

Since optimal solutions were not known for all of the problems presented here, we compared to the LB. We found that as the problem size is increased, except when compared to medium size problems, the gap between the best found solution and the LB grows, but remains less than 1% for even the largest instances.

The IDA* heuristic performs quite well in spite of its simplicity. Though it shows the largest gap on average, it maintains an average gap to the LB of 5.4% on the largest problem sizes using small cost values, and an average gap of 4% using larger cost values. Notably, it performs better on average on the reach stacker than on the gantry crane. Any solution for the reach stacker is valid for the gantry crane. The reach stacker has a smaller state space than the gantry crane whence a larger fraction can be searched within the time limit, leading to cases where the IDA* method performs better on the reach stacker than on the gantry crane.

Switching to large cost values deteriorates the quality of the reach stacker solutions for all heuristics using a greedy first search method, including the beam search methods. The large distance cost values induce a greedy behaviour where containers are selected lot by lot. Improving those results is a topic for future research.

Among all instances, the instances of medium size consistently show the largest gap to the LB for the reach stacker. This can be attributed to the layout of the containers in the storage area, instances of this size having fewer lots, which leads to a smaller number of containers being reachable during the loading procedure, and to worse overall performance.

2.7. Conclusion

In this paper we proposed a two-stage heuristic based on a formulation of the LPSP as a sequential decision-making problem that can be solved through dynamic programming. We extensively analyzed different heuristics, comparing to an exact solution method (ILP formulation) whenever possible.

When comparing the accuracy of the 0-way and 1-way distance functions, we see that the solutions according to the 0-way distance function lead to an average gap to the LB of up to 4.6% when evaluated in a 2-way distance function environment, while solutions to the 1-way distance function lead to a gap up to 1.3%. It is obvious that the 1-way distance function is a better approximation of the true distance cost, but notable that these gaps are larger than the gaps found using the heuristics equipped with the 2-way distance function. We conclude that the 2-way distance function is beneficial and that approximating the distance without the return distance, while close, can lead to worse solutions to the LPSP.

We leave for future work explorations into the impact of different cost values within the LPSP system, including exploration of heuristics that work with the large cost values presented here that do not induce a lot by lot selection when paired with greedy search. Additionally, we leave for future work explorations into stochastic systems, for example examination of the stochastic LPSP or of systems with uncertainty in container weights.

Acknowledgements

We gratefully acknowledge the close collaboration with personnel from the Canadian National Railway Company (CN), the funding through the CN Chair in Optimization of Railway Operations at Université de Montréal and funding from the National Sciences and Engineering Council of Canada (NSERC) through a Collaborative Research and Development grant.

2.8. Appendix

2.8.1. Proof of Propostion 1

2.8.1.1. *Notation and Introduction*

Let us adopt the notation where $q = (q_x,q_y)$ designates the position of a slot, and $c = (c_x,c_y)$ designates the position of a container. More than one slot can occupy the same x,y position. $\mathcal{C}$ is the set of all positions of containers and $\mathcal{Q}$ the set of all positions of slots.

A policy π consists of a sequence $\{(c_t,q_t)\}_{t=0}^{T-1}$ where $T-1$ it the time such that no additional containers can be loaded, whether there are no containers left to place, no more slots in which to place containers or none of the remaining containers can be placed in the remaining slots. We denote $\mathcal{C}_t$ and $\mathcal{Q}_t$ the set of positions of containers and slots still to be selected at time t, respectively.

We define the greedy policy π_G as follows:

$$(c_{t+1},q_{t+1}) = \text{argmin}_{c\in\mathcal{C}_{t+1},p\in\mathcal{Q}_{t+1}} \left[d(q_t,c) + d(c,q)\right]$$

where

$$d(a,b) = \zeta_x|a_x - b_x| + \zeta_y|a_y - b_y|$$

is the scaled distance. Moreover, we assume that the initial position of the handling equipment is (0,0), the position of the first lot is aligned with the first platform and lots are filled in order, moving along the train.

Our goal is to minimize the total immediate costs $G_{T-1}(\pi)$ as a function of the policy π, where

$$G_{T-1}(\pi) = \sum_{t=0}^{T-1} \left[d(q_t,c_{t+1}) + d(c_{t+1},q_{t+1})\right].$$

Proposition 2. *Let $G_{T-1}(\pi)$ be the total immediate costs as a function of the policy π and π_G be the greedy policy.*

$$\pi_G = argmin_\pi G_{T-1}(\pi).$$

2.8.1.2. *Proof for Fixed q*

Let us for now assume that the sequence $\{q_t\}_{t=0}^{T-1}$ is fixed, and define $\tilde{\pi}_G$ to be the greedy policy with respect to c for $\{q_t\}_{t=0}^{T-1}$ fixed.

Since $\{q_t\}_{t=0}^{T-1}$ is given, we are optimizing directly for $\{c_t\}_{t=0}^{T-1}$, and we define

$$F_t(\pi) = d(q_t, c_{t+1}^{(\pi)}) + d(c_{t+1}^{(\pi)}, q_{t+1})$$

and hence for all $t \geq 0$, we get the recursive expression:

$$G_{t+1}(\pi) = G_t(\pi) + F_{t+1}(\pi)$$

and thus if we define

$$\pi_{t+1}^\star = \text{argmin}_\pi G_{t+1}(\pi) = \text{argmin}_\pi \left[G_t(\pi) + F_{t+1}(\pi) \right].$$

We get

$$G_t(\pi_{t+1}^\star) + F_{t+1}(\pi_{t+1}^\star) \leq G_t(\tilde{\pi}_G) + F_{t+1}(\tilde{\pi}_G)$$

We will now prove by induction that $\tilde{\pi}_G = \text{argmin}_\pi G_{T-1}(\pi)$ with the induction hypothesis

$$\mathcal{H}_t : \begin{cases} \pi_t^\star = \tilde{\pi}_G \\ \tilde{\pi}_G = \text{argmin}_\pi F_t(\pi) \end{cases}$$

Let us first verify the induction hypothesis $\mathcal{H}_0$ for $t = 0$. Let us observe that

$$G_0(\pi) = d(q_0, c_1^{(\pi)}) + d(c_1^{(\pi)}, q_1) = F_0(\pi)$$

and thus by definition of π_G, we have that $c_1^{(\tilde{\pi}_G)} = c_1^{(\pi_0^\star)}$, and thus

$$\pi_G = \text{argmin}_\pi G_0(\pi) = \text{argmin}_\pi F_0(\pi).$$

Hence the proposition $\mathcal{H}_0$ is true.

Let us now assume that $\mathcal{H}_k$ is true for all $k \leq t$. Assuming $\pi_t^\star = \tilde{\pi}_G$, we have

$$\underbrace{G_t(\pi_{t+1}^\star) - G_t(\pi_t^\star)}_{\geq 0} \leq F_{t+1}(\pi_G) - F_{t+1}(\pi_{t+1}^\star)$$

Further assuming that $\tilde{\pi}_G = \text{argmin}_\pi F_t(\pi)$, then we must have (by the definition of $\tilde{\pi}_G$) that

$$\tilde{\pi}_G = \text{argmin}_\pi F_{t+1}(\pi)$$

Thus

$$0 \leq G_t(\pi_{t+1}^\star) - G_t(\tilde{\pi}_G) \leq F_{t+1}(\tilde{\pi}_G) - F_{t+1}(\pi_{t+1}^\star) \leq 0$$

or in other words

$$\tilde{\pi}_G = \text{argmin}_\pi G_{t+1}(\pi),$$

completing the proof by induction. Thus Proposition 2 is holds true.

N.B.: Note this result holds for any fixed sequence $\{q_t\}_{t=0}^{T-1}$.

2.8.1.3. *Final Step of the Proof*

Let us now reconsider the the initial greedy policy π_G. We would like to prove that

$$\pi_G = \text{argmin}_\pi G_{T-1}(\pi).$$

Note that π_G induces a sequence $\{q'_t\}_{t=0}^{T-1}$. Now observe that π_G is indeed the same policy as $\tilde{\pi}_G$ over $\{q'_t\}_{t=0}^{T-1}$.

Now we just need to prove that $\{q'_t\}_{t=0}^{T-1}$ is indeed optimal. Note that once we have $\{q_t\}_{t=0}^{T-1}$ fixed, we know which $\{c_t\}_{t=0}^{T-1}$ is optimal namely the one of the greedy policy. More precisely, once we have chosen q_t, we know which c_{t+1} to pick for optimality, thus we can assume that it is fixed once p_t is chosen. Thus let us define:

$$F_t(\pi) = d(q_t,c_{t+1}) + d(c_{t+1},q_{t+1}^{(\pi)})$$

where we assume that at time t, q_t and c_{t+1} are already fixed, and we are to optimize the choice of q_{t+1}.

Recalling the recursive expression:

$$G_{t+1}(\pi) = G_t(\pi) + F_{t+1}(\pi)$$

let us now prove by induction that $\{q'_t\}_{t=0}^{T}$ is indeed optimal, with the induction hypothesis $\mathcal{H}'_t$ that π_G at time t consisting of the sequence $\{q_0,c_1,q_1,\ldots,c_{t+1},p_{t+1}\}$ corresponds to $\pi_t^\star$.

Let us first verify the induction hypothesis $\mathcal{H}'_0$ for $t = 0$. Let us observe that

$$G_0(\pi) = d(q_0,c_1) + d(c_1,q_1^{(\pi)}) = F_0(\pi)$$

and thus by definition of π_G, we have that $q_1^{(\pi_G)} = q_1^{(\pi_0^\star)}$, and thus

$$\pi_G = \text{argmin}_\pi G_0(\pi) = \text{argmin}_\pi F_0(\pi).$$

Hence the proposition $\mathcal{H}'_0$ is true.

Let us now assume that $\mathcal{H}'_k$ is true for all $k \leq t$. Then we have $G_t(\pi_G) \leq G_t(\pi)$ for all π by the induction hypothesis, while and $F_t(\pi_G) \leq F_t(\pi)$ for all π, by the definition of π_G. Thus

$$G_{t+1}(\pi_G) = G_t(\pi_G) + F_{t+1}(\pi_G) \leq G_t(\pi) + F_{t+1}(\pi) = G_{t+1}(\pi)$$

for all π. Hence $\pi_G = \pi_{t+1}^\star$, completing the proof by induction.

Chapter 3

Second Article.

Load Planning and Sequencing with Deep Q-Networks

by

Kyle Goyette[1], and Emma Frejinger[1]

([1]) Department of Computer Science and Operations Research and CIRRELT, Université de Montréal

This article will be further revised and submitted to a yet undetermined publication.

Author Contributions

My contribution to this paper spans all components:

- design of state representation, action representation for the LPSP,
- implementation of deep Q-learning algorithm, DQN model, as well as the design and implementation of an algorithm pairing of DQN with beam search,

- hyperparameter search for all networks and cost settings,
- detailed analysis of results as well as writing of the paper.

3.1. Introduction

Rail transportation is a crucial component of supply chains in North America and world wide. The number of transported intermodal units has been rising steadily since 2016, reaching 30 million units in 2018. Double-stack intermodal railcars are efficient means of transporting goods over rails, doubling the amount of containers which can be shipped over single stack railcars. Central to the performance of rail transportation are efficient operations of intermodal rail terminals where containers are sorted, temporarily stored and loaded onto railcars.

Deep reinforcement learning (DRL) and reinforcement learning (RL) have seen a surge in research and successful applications since the seminal contribution of [56], which introduced deep Q-networks (DQN). More recently, DRL algorithms have been able to rise to the performance of the world's top chess players [73], and beat the world's best Go players [74]. Due to the sequential nature of loading containers onto railcars, DRL may be a good candidate for solving this problem. If successful, DRL models would be able to provide solutions in very short computing time which could potentially be leveraged in future work to solve a dynamic and stochastic version of the problem.

The load planning and sequencing problem (LPSP) for double-stack railcars is a problem found at North American rail container terminals. The objective is to maximize the value of containers loaded onto a sequence of railcars, while minimizing the handling costs associated with placing these containers. Here, we approach the problem for double-stack intermodal railcars. The loading of such railcars must respect a variety of constraints, e.g., pertaining to the center of gravity (COG), maximum carrying weight and the loading patterns associated with a railcar [53]. We refer to Section 2.2 for more details. It is notable that in this chapter we mostly use an optimal control vocabulary [8], despite the solution method lying closer to reinforcement learning.

The objective of this work is to illustrate how the LPSP can be approached using DRL techniques, that an agent is able to learn how to perform well in the environment, and propose acceptable solutions to the LPSP in short computing time. Challenging in this context is the representation of the state of the system and constraints on the action space at each state. For this purpose we use a deep neural network.

The contributions of this paper are as follows:

- We present a representation of the LPSP state that accounts for the LPSP's several modalities and 3-dimensional structure.
- At test time, we pair the trained model with a beam search in order to improve the solutions.

- We measure the performance of the model by comparing the performance to a lower bound (LB).
- We compare the performance of the model to the two-stage heuristic proposed in Section 2.5. While the proposed RL model does not outperform the two-stage heuristic on average, it reaches similar performance for one type of handling equipment (gantry crane).

The remainder of the paper is structured as follows: In Section 3.2 we discuss work relevant to the LPSP and the methods we use to approach the LPSP. Section 3.3 briefly presents the LPSP, how the LPSP is modified for deep reinforcement training, the representation of the LPSP for the model and the model architecture. Section 3.4 outlines our experimental study and its results and in Section 3.5 we provide a detailed discussion of the latter. Finally, Section 3.6 concludes and defines directions for further research.

3.2. Literature Review

In this section we begin by presenting the literature related to the LPSP for double-stack intermodal railcars. Next, we present applications of deep reinforcement learning to problems closely related to the LPSP, focusing on problems wherein container movements in terminals are considered. We then discuss common methods of solving combinatorial optimization problems using deep reinforcement learning. Finally, we present the literature related to deep Q-learning Networks, as our model, and training procedure, are based upon the DQN algorithm.

The load planning problem (LPP) for double-stack intermodal railcars is introduced by [53], and addresses the pairing between containers and platform slots. They present loading constraints that occur in practice along with an integer linear programming (ILP) formulation that can be solved by a commercial solver. The load sequencing problem (LSP) defines the sequence of actions taken to place containers onto their respective platform slots. Typically, the LPP and LSP are solved sequentially, with the result of the load planning problem being used to define the containers to be loaded in the load sequencing problem as in [63]. Reference [67] introduces the LPSP which integrates the solution of the two problems and also introduce distance functions which we present here. They solve the problem by means of integer programming. In Chapter 2 we proposed to formulate the LPSP as a shortest path problem on an ordered graph and to solve it through dynamic programming.

In the literature, the operational decision problems that occur at container terminals have been approached by dynamic programming (see Section 2.3) as well as reinforcement learning. Reference [72] uses a DQN to to find solutions to the ship stowage planning problem. In that problem, containers are loaded from the loading bay onto a ship, the slots to be filled have a relative loading sequence such that the furthest slots must be loaded first. Moreover, the problem considers weight limits for each slot as well as placing heavier containers on the

bottom layers of the ship. They construct a feature vector using the statistics and positions of the containers in the yard. Reference [69] uses a deep reinforcement learning algorithm as well as evolutionary strategies to approach the container loading problem at a ship terminal. The objective is to place containers on the ship while limiting the number of reshuffles required. They do not consider the distance travelled by the handling equipment, nor is their problem 3-dimensional, because they only consider one plane of containers running alongside a ship. This would be most similar to solving the problem using the 0-way distance formulation as presented by [67] and Section 2.4.2. Reference [79] compares reinforcement and evolutionary algorithms for the container loading problem, defined as a space utilization problem, which considers maximizing the number of containers that can be loaded into a yard block. They approach the problem using the tabular reinforcement learning technique Q-learning. There are currently no studies that use DRL to approach the LPSP when considering the costs associated with the distance travelled by the handling equipment in a 3-dimensional environment. Moreover, we could not find any studies using DRL to approach the LPSP for double-stack intermodal railcars. We fill these gaps here.

We train the model using an algorithm called deep Q-learning as proposed by [56], wherein a model is trained to play Atari games. Deep Q-learning trains a model through interaction with the environment, using a technique called *experience replay* [50], which keeps previous states, actions, rewards and transitions in a memory to be sampled randomly. The purpose of the random sampling is to smooth the training distribution over past behaviours. During training, the model takes actions in the environment, and at each step a mini-batch is drawn from the replay memory to update the network. Additionally, there is a second network, the *target network*, which is only updated infrequently, improving the stability of the model. In order to improve stability further we scale the rewards, as used in [18] and [23], and discussed in [29], by a constant factor to compress the space of expected returns which the DQN must output. This leads to smaller gradients and more stability during training. Another method of stabilizing training is gradient clipping [60, 22].

DQNs have been employed in deterministic settings since their inception as explained in [24], as many of the Atari games are deterministic except for the initial hidden state. They have also been employed in molecular optimization, a deterministic setting with an accurate model of the environment in [86]. DQN was also used successfully in the deterministic maze environment in [47]. Finally, DRL methods have been employed to solve a deterministic bin packing problem [43].

3.3. Methodology

In this section, we present the components which define the DRL agent we will use to approach the LPSP. The problem is defined using negative rewards, referred to as costs in the context of this paper (see Section 3.3.1). An episode is defined as the sequence of states,

actions and rewards between the initial state, at $t = 0$, to the terminal state, at $t = T$. For the LPSP, this involves the loading of containers onto railcar platforms until no more containers can be loaded. During each episode, states are represented to the model through a state representation vector, which describes the state of the LPSP in a manner that can be fed into a feed-forward deep neural network, as we show in Section 3.3.2. The representation of actions is presented in Section 3.3.3. The neural network architecture is presented in Section 3.3.4. We outline how the network is trained in Section 3.3.5 and how the trained network can be integrated with beam search to improve results in Section 3.3.6.

3.3.1. The Load Planning and Sequencing Problem

For a detailed description of the LPSP we refer the reader to Section 2.2. Here we consider the same problem and notation, however we make one modification: During training we modify the terminal cost for the LPSP. We introduce the indicator variable $f_q \in \{0,1\}$ that equals 1 if the slot is occupied and 0 otherwise. We modify the cost function at the end of an episode to add a penalty to the model only if at least one slot remain open, as shown below

$$G_T(s_T) = \begin{cases} \pi \sum_{c \in C}(1 - l_c) & \text{if } \sum_{q \in Q}(1 - f_q) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

This modification improves the system by removing the large terminal penalty when a solution is found. Instead, the agent is penalized for leaving slots empty while having unloaded containers. Since all the containers have identical leave-behind cost, π, during a successful sequence of loading operations, wherein all slots are occupied, or all containers are placed, the modification does not lead to a change to the overall cost which we aim to be minimize.

3.3.2. State Representation

We define a feature representation which indicates container and platform information to the neural network. The state representation has been defined such that it can represent any problem instance within the distribution we consider. We propose a representation for each of the components of the LPSP. Section 3.3.2.1 presents the feature representation of the containers in the storage area. Section 3.3.2.2 presents the feature representation for the containers which have been double touched and placed aside, and Section 3.3.2.3 presents the representation for the railcar platforms to be loaded, the containers that have been loaded onto each platform, as well as the position of the handling equipment.

First, we introduce two statistics calculated over the instance being represented. We determine the maximum weight of a container in an instance, $m_{\max} = \max_{c \in C} m_c$. Furthermore, we define the average container weight as the average weight of all containers in the instance, $\bar{m}_C = \frac{\sum_{c \in C} m_c}{|C|}$, and the standard deviation of container weights as $\hat{\sigma}_{m_C}$. We use

these values to determine the normalized weight of each container as follows:

$$\tilde{m}_c = \frac{m_c - \bar{m}_C}{\hat{\sigma}_{m_C}}. \tag{3.2}$$

3.3.2.1. *Storage Area*

We represent the storage area to the DQN as a 4-D tensor of shape $(k, \max(x_{SA}), \max(y_{SA}), \max(z_{SA}))$ where $\max(x_{SA}), \max(y_{SA}), \max(z_{SA})$ are the maximum x, y and z positions of containers in the storage area among all problems we consider. We refer to the first dimension of the representation as the channels, where each channel $k = 1, \ldots, 11$ holds information ψ_k about the containers in the storage yard,

$$\psi^1_{SA}(x,y,z) = \begin{cases} 1 & \text{if } (x_c, y_c, z_c) = (x,y,z) \qquad \exists c \in C \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

$$\psi^2_{SA}(x,y,z) = \begin{cases} \frac{m_c}{m_{\max}} & \text{if } (x_c, y_c, z_c) = (x,y,z) \qquad \exists c \in C \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

$$\psi^3_{SA}(x,y,z) = \begin{cases} \tilde{m}_c & \text{if } (x_c, y_c, z_c) = (x,y,z) \qquad \exists c \in C \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

The first channel (3.3) represents the occupancy of the position (x,y,z). The second channel (3.4) represents the weight of the container at (x,y,z), divided by the maximum weight of all containers in the instance. The third channel (3.5) represents the normalized weight of the container at position (x,y,z).

Next, we consider the length of each container. This is done by category, since lengths are discrete values. We show these in (3.6) which indicates the length of a container at each location in the storage area. Container length, L_c, can be 40 ft, 45 ft, 48 ft or 53 ft. Defining $\bar{L}_1 = 40, \ldots, \bar{L}_4 = 53$, we have

$$\psi^k_{SA}(x,y,z) = \begin{cases} 1 & \text{if } (x_c, y_c, z_c) = (x,y,z) \text{ and } L_c = \bar{L}_{k-3} \qquad \exists c \in C \\ 0 & \text{otherwise.} \end{cases} \qquad k = 4, \ldots, 7. \tag{3.6}$$

We next consider the container height. Containers can be either low-cube (LC), having height 8 feet 6 inches, or high-cube (HC), having height 9 feet 6 inches. We define the set of all LC containers as C^{LC} and the set of all HC containers as C^{HC} Since the height is a discrete variable, we represent it using 2 one-hot inputs, one for each height. Both are 0 if a container is not present. The first (3.7) indicates the container at (x,y,z) is a LC container,

while the second (3.8) indicates the container at (x,y,z) is a HC container:

$$\psi^8_{SA}(x,y,z) = \begin{cases} 1 & \text{if } (x_c, y_c, z_c) = (x,y,z) \text{ and } c \in C^{\text{LC}} \qquad \exists c \in C \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

$$\psi^9_{SA}(x,y,z) = \begin{cases} 1 & \text{if } (x_c, y_c, z_c) = (x,y,z) \text{ and } c \in C^{\text{HC}} \qquad \exists c \in C \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

We next indicate whether a container is reachable by the handling equipment:

$$\psi^{10}_{SA}(x,y,z) = \begin{cases} 1 & \text{if } (x_c, y_c, z_c) = (x,y,z) \text{ is reachable} \qquad \exists c \in C \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

Finally, we introduce an additional channel for the reach stacker, to identify whether the container requires a detour to retrieve:

$$\psi^{11}_{SA}(x,y,z) = \begin{cases} 1 & \text{if } (x_c, y_c, z_c) = (x,y,z) \text{ and } c \text{ requires a detour} \qquad \exists c \in C \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

3.3.2.2. *Double-touched Containers*

We define the feature tensor for containers which have already been double touched similarly to those in the storage area. Features ψ^1_{SA} through ψ^{10}_{SA} are defined to represent containers that have been double touched but these features do not include tensors indicating reachability, since all containers that have been double touched are reachable. Moreover, the double touched containers feature tensor maintains the initial position of the container in the storage area to define the containers location. We do this so that there is no limit to the number of containers which can be placed aside, and if several containers are double touched they do not interfere with the feature representation.

3.3.2.3. *Platforms*

Platforms are represented with a 1-D vector containing information about all individual platform weights, carrying capacities, lengths, sets of loading patterns, and loaded containers. Let N_P denote the maximum number of platforms over all instances. The vector is of size $w * |N_P|$ where $w = 32$, as there are 32 channels of information to be represented for each platform, such that each w block of the input vector describes one platform. Platforms are always considered in order of increasing x to give the system a consistent input relative to the positions of the platforms. In instances where there are fewer than $|N_p|$ platforms the remaining blocks are populated by zeros. We define each of the state features for $p \in P$ below:

$$\psi^1_P(p) = \frac{m_p}{m_{\max}} \tag{3.11}$$

$$\psi_P^2(p) = \frac{g_p}{m_{\max}} \tag{3.12}$$

where (3.11) represents the mass of the platform, p divided by the maximum mass of containers. The weight capacity of the platform p divided by the maximum mass of containers is represented by (3.12).

We consider a total of 9 different sets of loading patterns $o \in O$ across all instances, these are represented categorically, as shown in (3.13).

$$\psi_P^k(p) = \begin{cases} 1 & \text{if } p \text{ belongs to a railcar with (k-2)th set of loading patterns} \\ 0 & \text{otherwise.} \end{cases} \quad k = 3, \ldots, 11 \tag{3.13}$$

Next, we consider the position of the platform on the railcar. There are 1, 3 or 5 platforms on any railcar and loading pattern definitions lead to symmetry between platforms in the first, third and fifth positions, as well as between the second and fourth. Thus we indicate whether a platform belongs in either of these 2 groups in (3.14).

$$\psi_P^{12}(p) = \begin{cases} 1 & \text{if } p \text{ is the first, third or fifth platform in the railcar} \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

Next, we consider the information regarding the containers loaded onto the platform. We first consider whether a container can be placed in each platform bottom or top slots, shown in (3.15) and (3.16) respectively. Next, we indicate whether each platform's bottom or top slot is occupied in (3.17) and (3.18) respectively.

$$\psi_P^{13}(p) = \begin{cases} 1 & \text{if a container can be placed on } p\text{'s bottom slot} \\ 0 & \text{otherwise.} \end{cases} \tag{3.15}$$

$$\psi_P^{14}(p) = \begin{cases} 1 & \text{if a container can be placed on } p\text{'s top slot} \\ 0 & \text{otherwise.} \end{cases} \tag{3.16}$$

$$\psi_P^{15}(p) = \begin{cases} 1 & \text{if a container occupies } p\text{'s bottom slot} \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

$$\psi_P^{16}(p) = \begin{cases} 1 & \text{if a container occupies } p\text{'s top slot} \\ 0 & \text{otherwise.} \end{cases} \tag{3.18}$$

Next, we consider the length of each container loaded onto the platform top and bottom slots. This is done identically to the storage area as shown in (3.6) above. (3.19) indicates the length of the container occupying the bottom slot of a platform and (3.20) indicates the same information for the top slot.

$$\psi_P^k(p) = \begin{cases} 1 & \text{if container } c \text{ occupying } p\text{'s bottom slot has } L_c = \bar{L}_{k-16} \\ 0 & \text{otherwise.} \end{cases} \quad k = 17, \ldots, 20 \tag{3.19}$$

$$\psi_P^k(p) = \begin{cases} 1 & \text{if container } c \text{ occupying } p\text{'s top slot has } L_c = \bar{L}_{k-20} \\ 0 & \text{otherwise.} \end{cases} \quad k = 21, \ldots, 24 \tag{3.20}$$

Next, we consider the height of each container loaded onto the railcar, again using categorical values. First, (3.21) and (3.22) indicate whether the container occupying the bottom slot of the platform is LC or HC respectively. Second, (3.23) and (3.24) indicate the height of the container occupying the platform's top slot.

$$\psi_P^{25}(p) = \begin{cases} 1 & \text{if container occupying } p\text{'s bottom slot } \in C^{LC} \\ 0 & \text{otherwise.} \end{cases} \tag{3.21}$$

$$\psi_P^{26}(p) = \begin{cases} 1 & \text{if container occupying } p\text{'s bottom slot } \in C^{HC} \\ 0 & \text{otherwise.} \end{cases} \tag{3.22}$$

$$\psi_P^{27}(p) = \begin{cases} 1 & \text{if container occupying } p\text{'s top slot } \in C^{LC} \\ 0 & \text{otherwise.} \end{cases} \tag{3.23}$$

$$\psi_P^{28}(p) = \begin{cases} 1 & \text{if container occupying } p\text{'s top slot } \in C^{HC} \\ 0 & \text{otherwise.} \end{cases} \tag{3.24}$$

Next, (3.25) indicates the maximum weight which can be loaded onto the platform while respecting the platform's capacity constraints. We define the set of containers placed on a platform p as p_C.

$$\psi_P^{29}(p) = \begin{cases} \frac{g_p - \sum_{c \in p_C} m_c}{m_{\max}} & \text{if } p \text{ does not have top and bottom slots occupied} \\ 0 & \text{otherwise.} \end{cases} \tag{3.25}$$

Next, (3.26) and (3.27) indicate the maximum weight of containers which could be placed on the platform while respecting COG constraints for LC and HC containers respectively. We introduce the notation of the maximum weight of a container which could be placed while respecting COG constraints, $m_{\text{LC COG}}$ and $m_{\text{HC COG}}$.

$$\psi_P^{30}(p) = \begin{cases} 1 & \text{if } p \text{ has no container in bottom slot} \\ 0 & \text{if } p \text{ has containers in both top and bottom slot} \\ m_{\text{LC COG}} & \text{if } p \text{ has a container in the bottom slot.} \end{cases} \tag{3.26}$$

$$\psi_P^{31}(p) = \begin{cases} 1 & \text{if } p \text{ has no container in bottom slot} \\ 0 & \text{if } p \text{ has containers in both top and bottom slot} \\ m_{\text{HC COG}} & \text{if } p \text{ has a container in the bottom slot.} \end{cases} \tag{3.27}$$

Finally, we represent the position of the handling equipment in (3.28)

$$\psi_P^{32}(p) = \begin{cases} 1 & \text{if } (x_h, y_h) = (x_p, y_p) \\ 0 & \text{otherwise.} \end{cases} \tag{3.28}$$

3.3.3. Action Representation

We represent the actions in the LPSP using a 4-D tensor of shape $(2 * N_P + 1, \max(x_{SA}), \max(y_{SA}), \max(z_{SA}))$ where, as before, N_P is the maximum number of platforms among all problems in the distribution we consider. We refer to the first dimension of the tensor as the channel, while the other dimensions refer to the 3-D space of the storage area.

The first channel of the tensor is used to indicate double touching a container, the following N_P channels are used to indicate moving a container from the storage area onto a platform. For example, channel k is associated with actions placing containers onto the $(k-1)$th platform. The remaining channels are used to indicate placing a container that has been double touched onto a platform.

The 3-D space maps the (x,y,z) positions of each container in the storage area. In this manner, to select a container from position (x,y,z) and place it on the kth platform, the resulting action index is $(k-1,x,y,z)$. Notably, we use the same 3-D space when selecting containers which have been double touched. Here, we indicate the original (x,y,z) position of the container rather than the position in which the container was placed when it was double touched.

3.3.4. Neural Network Architecture

Here, we present the architecture of the neural network used as an agent in the LPSP. The design of the network required identifying the so-called feature representation modes of the LPSP, and finding an effective manner of combining and extracting their information. Additionally, special care was needed in the design to account for the 3-D shape of the problem. The architecture has three main inputs, one for each mode. These modes are created through concatenation of the problem features. They are then mixed through addition and the results are fed into more abstract layers of the network where the information is supposed to be usefully combined. We discuss how inputs are passed into the network, the specialized network layers which transform each of the different data types, how different feature modes

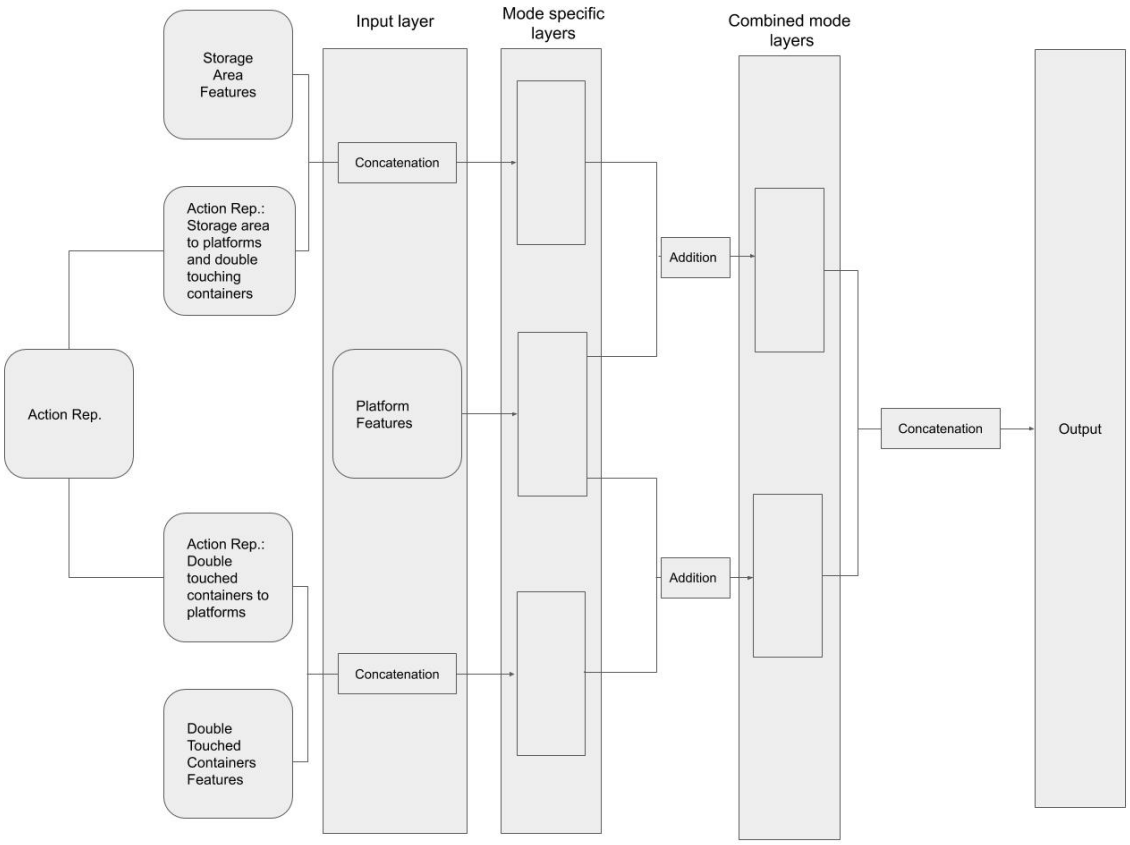


Figure 3.1. Overview of architecture of neural network model

are combined and finally the output of the network. An overview of the architecture can be seen in Figure 3.1.

The neural network receives as initial inputs each of the features defined in Section 3.3.2 as well as the mapping of feasible actions. The mapping of feasible actions is the 4-D mapping given in Section 3.3.3 where indices associated with feasible actions have values of 1, and all other indices have values of 0.

To begin, the mapping of feasible actions is split into actions moving a container from the storage and actions moving a container which has been double touched. Each of these actions is concatenated with its respective feature representation. We will continue to refer to the concatenated representations as storage area features and aside features.

Storage area features, and aside features are each fed through two 3-D convolutional layers, with rectified linear units (ReLU) as nonlinearities outputting a shape of $(k, \max(x_{SA}), \max(y_{SA}), \max(z_{SA}))$. Meanwhile, the platform features are passed through a two-layer fully connected feed-forward neural network, with output size equal to $N_P * \max(x_{SA}) * \max(y_{SA})$, which is then reshaped to be size $(N_P, \max(x_{SA}), \max(y_{SA}))$ and finally repeated along an expanded last dimension such that the final shape is $(N_P, \max(x_{SA}), \max(y_{SA}), \max(z_{SA}))$. This new feature representation is fed through two

more 3-D convolutional layers. We thus create three distinct representations, one for the storage area, one for the aside area and one for the platforms. These representations are more abstract than the initial layers, and can be more usefully mixed as mentioned in [42]. We represent each of these abstract spaces as $\tilde{\psi}_{SA}$, $\tilde{\psi}_{A}$, and $\tilde{\psi}_{P}$ for the storage area features, aside features and platform features respectively.

Next, the platform representation is added to the storage area and aside representations separately as follows:

$$\tilde{\psi}_{SA-P} = \tilde{\psi}_{SA} + \tilde{\psi}_{P} \tag{3.29}$$

$$\tilde{\psi}_{A-P} = \tilde{\psi}_{A} + \tilde{\psi}_{P}. \tag{3.30}$$

Afterwards, each combined representation is further fed through five 3-D convolutional layers separately.

Notice that we use a 3-D output shape which allows for any number of containers to be placed aside while not creating a very large fully connected output layer. With a 3-D output shape, the number of containers which could be placed aside did not need to be artificially limited to have a fixed output size of the network.

3.3.5. Training

Due to the different characteristics of the reach stacker and the gantry crane, we train a different model for each of these two types of handling equipment.

The models were trained following the standard DQN algorithm as presented by [56]. Since the costs at each time step can be quite large, it was found that scaling the cost, multiplying it by a constant α, improved the networks performance. Furthermore, we replace the mean squared error loss used in the DQN paper with the Huber loss (sometimes referred to as smooth-L1 loss). This reduces the size of the gradient, and leads to further numerical stability. Training is performed using the look ahead optimization as presented in [84], using the rectified adam (RAdam) [51] as the optimizer.

Instances are split into training, validation and test for each problem size. Once every 20 episodes, the current network is tested against the set of the largest validation instances. We use the model with the best overall performance on the validation set to produce our final results. We present the resulting performance curves in Section 3.4.3.

3.3.6. Beam Search with DQN

At test time, we apply beam search to the trained model. We denote beam width with β. Beam search paired with a DQN model is performed by expanding the β nodes with the best accumulated score as predicted by the DQN. During the first time step, the top β state-action values predicted by the DQN are selected, actions associated with these values are taken, and the predicted state-action values are each retained in a *cumulant* to be used

in later time steps. Next, each of the new β states are input into the DQN, and outputs are added with the state's associated cumulant. From these values, the top β values are selected, and actions associated with these values are expanded for the next stage. This process continues until each trajectory created by the search reaches a terminal state. This algorithm is shown in Algorithm 4. We indicate the total cost incurred to reach a state, s, as cost_s. Furthermore, we define the transition function $s' = g(s,a)$ where s' is the state resulting from taking action a in state s.

Algorithm 4 Beam-search Rollout Policy with DQN

DRL Beam-Search$(Q, \sigma(S,A), s)$
$B \leftarrow$ s
$R \leftarrow \emptyset$
while $B \neq \emptyset$ **do**
  for all $s \in B$ **do**
    $c_{s,a} \leftarrow Q(s,a)$
    $C \leftarrow C \cup \{c_{s,a}\}$
  end for
  $T \leftarrow \beta$ best (s,a) pairs from C
  $B \leftarrow \emptyset$
  for all $(s,a) \in T$ **do**
    Take action a from s, observe s'
    $B \leftarrow B \cup \{s'\}$
  end for
  for all $s \in B$ **do**
    if s is terminal **then**
      $R \leftarrow \text{cost}_s$
      $B \leftarrow B \setminus \{s\}$
    end if
  end for
end while
return $\min(R)$

3.4. Experiments

3.4.1. Instance Generation and Environment Cost Values

We use the same instances as in Section 2.6.1 and [67]. We test two sets of cost values, a small set of cost values and a set of large cost values as presented in Section 2.6.2. The small cost values are also identical to those presented in [67]. Values describing each of these environment costs are indicated in Table 3.1. Notably, we use a different set of costs for training and testing in the large distance cost test values. This was done as it was found to be more stable than increasing the scaling value to account for higher distance costs.

Parameter	Small Distance Cost Train and Test Values	Large Distance Cost Train Values	Large Distance Cost Test Values
π_c	100	100	5300
τ_r	1	1	1
η	0	0	0
κ	80	80	4240
ζ_x	1	1	53
ζ_y	1	0.15	8
ϕ_x	1	1	53
ϕ_y	1	0.15	8

Table 3.1. Environment costs

As stated in Section 2.6.1, we neglect 20-foot containers, and all containers are considered high-cube (HC) containers.

3.4.2. Hyperparameters

During training and testing, there are several hyperparameters for the model, as well as values used to describe the environment. The hyperparameters for training are presented in Table 3.2. Note that the hyperparameters remain unchanged between the two networks, and across sets of environment cost values.

Hyperparameter	Gantry Crane	Reach Stacker
γ	0.8	0.9
Learning Rate	0.0001	0.0001
Grad Clipping Value	1	1
ϵ_{start}	0.95	0.95
ϵ_{end}	0.05	0.05
$\epsilon_{\text{decay rate}}$	500	500
Cost scale α	0.02	0.02
Target Net Update Frequency	100	100
Batch size	16	16
RAdam β_1	0.95	0.95
RAdam β_2	0.999	0.999
Lookahead Synchronization Period	5	5

Table 3.2. Training hyperparameters

Models were trained for two days on Compute Canada Beluga cluster. Each model was trained with 8 GB of memory, on nodes using one Nvidia V100SXM2 GPU and on two cores of an Intel Gold 6148 Skylake CPU.

3.4.3. Numerical Results

We begin by presenting numerical results for the problems using the small cost test values. Next, we explore the performance of large cost test values. In each set of results, we report the evolution of the performance on the validation set during training, as well as the performance of a trained network with different beam widths on the train, validation and test set. For each one of test settings we analyze the following performance measures:

- percentage of instances solved to the LB,
- average, minimum and maximum values of the gap to the LB,
- average, minimum and maximum computing time,
- percentage of instances which fail to fill all slots while having containers unloaded,
- percentage of instances with double touches,
- for the instances with a reach stacker only, the percentage of instances with detours.

We present the results of experiments using the small distance costs in Section 3.4.3.1. Section 3.4.3.2 presents the results to experiments using the large value distance costs, which are more challenging.

3.4.3.1. *Small Distance Cost Results*

We show the performance curves for both the gantry crane and reach stacker when trained on the small distance cost values in Figure 3.2. Next, we show how the models perform on the training, validation and test instances for small, medium, large, and largest size problem instances.

The performance curves on the training set in Figure 3.2 indicate that both networks improve during training, but that the improvement is noisy, especially in the case of the reach stacker. Nonetheless, both models improve as training continues, reaching new higher peaks than previously seen. The maximum performance for the gantry crane occurs after episode 3600, and for the reach stacker performance peaks at around 4000.

Tables 3.3, 3.4 and 3.5 show the performance on the train, validation and test sets respectively for medium size problem instances. Interestingly, increasing beam size does not consistently improve performance for both the gantry crane and the reach stacker on both the validation and test sets. The model performs similarly to BS-B-IDA* (from Chapter 2) in terms of average gap to LB on the training set for the gantry crane, but this is rarely seen in the validation or test sets. The model performs poorly with the reach stacker, being unable to successfully fill all slots on the railcar for any beam size on the test set. Overall, the average computing time is lower than BS-B-IDA*, but it is unclear if increasing beam size to match the average computing time would improve results.

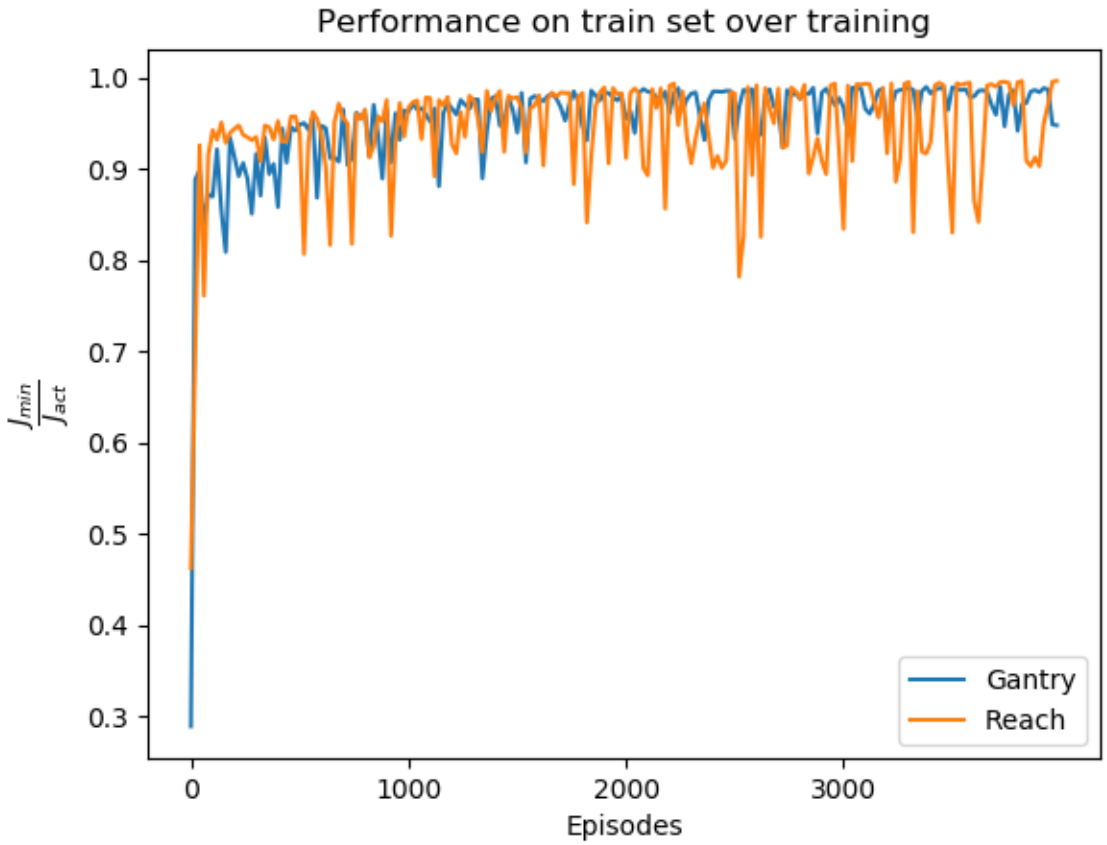


Figure 3.2. Performance of model on training set when training on small cost values

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max		
BS-B-IDA*	25	0.1	0.0	0.2	184.0	6.0	243.6	0	0
DRL β 1	0	0.2	0.1	0.3	1.2	0.9	1.8	0	0
DRL β 5	0	0.1	0.0	0.2	6.1	4.6	9.2	0	0
DRL β 10	0	0.1	0.0	0.2	12.1	9.0	18.0	0	0
DRL β 15	12.5	0.1	0.0	0.2	18.9	14.0	28.0	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max			
BS-B-IDA*	12.5	0.2	0.0	0.4	212.3	6.7	242.5	0	0	25
DRL β 1	0	5.2	1.4	14.2	0.7	0.5	1.0	0	25	100
DRL β 5	0	2.5	1.3	3.0	3.1	2.3	4.6	0	0	100
DRL β 10	0	2.0	1.3	2.9	6.2	4.6	9.3	0	0	100
DRL β 15	0	1.6	0.9	2.5	15.4	11.5	22.4	0	0	100

Table 3.3. Medium size training problems

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	60	0.1	0.0	0.7	96.2	2.4	242.6	0	0
DRL β 1	0	4.2	0.0	16.5	1.2	0.9	1.8	0	26.7
DRL β 5	6.7	0.2	0.0	1.7	6.1	4.6	9.2	0	0
DRL β 10	33.3	0.2	0.0	1.1	12.1	9.0	18.0	0	0
DRL β 15	6.7	2.0	0.0	13.6	18.9	14.0	28.0	0	13.3

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	53.3	0.2	0.0	1.8	121.8	1.6	241.8	0	0	6.7
DRL β 1	0	1.9	0.8	3.9	0.7	0.5	1.0	0	0	100
DRL β 5	0	1.5	0.1	4.0	3.1	2.3	4.6	0	0	86.7
DRL β 10	0	4.8	0.0	54.0	6.2	4.6	9.3	6.7	6.7	80
DRL β 15	0	1.2	0.0	3.2	15.4	11.5	22.4	0	0	93.3

Table 3.4. Medium size validation problems

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	50	0.1	0.0	0.5	123.9	2.8	243.5	0	0
DRL β 1	0	2.3	0.0	20.4	1.2	0.9	1.8	0	12.6
DRL β 5	4.2	0.1	0.0	0.5	6.1	4.6	9.2	0	0
DRL β 10	4.2	0.2	0.0	0.5	12.1	9.0	18.0	0	0
DRL β 15	4.2	0.3	0.0	1.9	18.9	14.0	28.0	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	29.2	0.2	0.0	1.2	172.1	2.1	242.5	0	0	8.3
DRL β 1	0	6.8	0.5	53.7	0.7	0.5	1.0	9.1	13.6	100
DRL β 5	0	7.0	0.5	55.5	3.1	2.3	4.6	9.1	18.2	100
DRL β 10	0	7.2	0.0	56.0	6.2	4.6	9.3	9.1	18.2	95.8
DRL β 15	0	7.7	0.0	54.5	15.4	11.5	22.4	9.1	27.3	95.8

Table 3.5. Medium size test problems

We present the results for the large size training problem instances in Table 3.6, and the results of the validation and test instances in Tables 3.7 and 3.8 respectively. Here we see an improvement in performance for both reach stacker and gantry crane over the medium size instances. As expected, the results on test and validation sets are worse than the training set for the gantry crane, but the validation set has the best performance with the largest beam size for the reach stacker. Increasing beam size consistently improves the performance

across all sets, except when considering beam sizes of 1 and 5 for the reach stacker on the test set. Nonetheless, further increasing the beam size closes the gap to the LB. Overall, the results are quite good, with gaps to the LB being less than 1% for the largest beam size, and for the gantry crane, the LB is achieved in 4.5% of the test cases. The trained model is able to achieve a smaller average gap to the LB than the BS-B-IDA*, but only on the training set and for the gantry crane.

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max		
BS-B-IDA*	0	0.1	0.0	0.1	245.9	245.0	246.4	0	0
DRL β 1	0	1.3	0.0	4.9	2.6	2.4	2.9	0	25
DRL β 5	12.5	0.1	0.0	0.1	13.0	12.0	14.8	0	0
DRL β 10	25	0.0	0.0	0.1	25.8	23.6	29.5	0	0
DRL β 15	12.5	0.0	0.0	0.1	40.1	36.6	45.7	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max			
BS-B-IDA*	0	0.1	0.0	0.1	243.4	242.7	243.9	0	0	0
DRL β 1	0	1.4	1.2	1.8	1.4	1.3	1.6	0	0	100
DRL β 5	0	1.0	0.7	1.4	6.2	5.7	7.3	0	0	100
DRL β 10	0	0.8	0.5	1.5	12.3	11.5	14.5	0	0	100
DRL β 15	0	0.7	0.6	1.1	32.4	30.4	36.4	0	0	100

Table 3.6. Large size training problems

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max		
BS-B-IDA*	33.3	0.1	0.0	0.2	171.5	21.9	247.4	0	0
DRL β 1	33.3	1.9	0.0	13.8	2.6	2.4	2.9	0	13.3
DRL β 5	6.7	0.1	0.0	0.3	13.0	12.0	14.8	0	0
DRL β 10	20	0.1	0.0	0.4	25.8	23.6	29.5	0	0
DRL β 15	6.7	0.1	0.0	0.2	40.1	36.6	45.7	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max			
BS-B-IDA*	20	0.1	0.0	0.3	192.2	11.0	244.4	0	0	20
DRL β 1	0	8.8	1.3	54.4	1.4	1.3	1.6	13.3	20	100
DRL β 5	0	1.9	0.6	6.2	6.2	5.7	7.3	0	13.3	100
DRL β 10	0	2.0	0.6	7.4	12.3	11.5	14.5	0	13.3	100
DRL β 15	0	0.7	0.3	1.0	32.4	30.4	36.4	0	0	100

Table 3.7. Large size validation problems

Finally, Tables 3.9, 3.10 and 3.11 show the performance on the train, validation and test sets respectively for the largest-sized problem instances. Once again, larger beam sizes

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	9.1	0.1	0.0	0.2	234.1	41.9	253.7	0	0
DRL β 1	0	0.2	0.0	0.6	4.3	4.0	4.6	0	0
DRL β 5	0	0.1	0.0	0.4	21.7	20.5	23.7	0	0
DRL β 10	0	0.1	0.0	0.3	42.7	39.7	46.4	0	0
DRL β 15	4.5	0.1	0.0	0.3	67.2	63.0	72.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	13.6	0.1	0.0	0.2	218.6	21.5	247.8	0	0	8.3
DRL β 1	0	1.8	0.9	6.9	2.1	1.9	2.3	0	9.1	90.9
DRL β 5	0	1.9	0.6	11.8	9.7	9.3	10.8	0	9.1	100
DRL β 10	0	0.9	0.4	1.6	19.2	18.1	20.7	0	0	90.9
DRL β 15	0	0.9	0.2	1.5	53.5	50.8	58.4	0	0	81.8

Table 3.8. Large size test problems

generally improve the results, but increase the computing times noticeably. The model outperforms BS-B-IDA* on the training set and meets its performance on the test set when using the gantry crane, but does not meet the performance in other cases. The average computing time is much lower for the DRL model than BS-B-IDA*, but it remains unclear if larger beam sizes could meet the performance of the heuristic. The DRL model is able to solve 37.5% of training instances and 4.5% of test instances to meet the LB for the gantry crane, but none are found to meet the LB for the reach stacker.

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	0	0.1	0.0	0.1	250.9	249.4	253.4	0	0
DRL β 1	12.5	0.0	0.0	0.1	4.3	4.0	4.6	0	0
DRL β 5	0	0.1	0.0	0.3	21.7	20.5	23.7	0	0
DRL β 10	0	0.0	0.0	0.0	42.7	39.7	46.4	0	0
DRL β 15	37.5	0.0	0.0	0.1	67.2	63.0	72.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.1	0.0	0.1	246.4	245.0	247.0	0	0	25
DRL β 1	0	1.4	1.0	1.9	2.1	1.9	2.3	0	0	100
DRL β 5	0	1.0	0.7	1.6	9.7	9.3	10.8	0	0	100
DRL β 10	0	0.9	0.7	1.6	19.2	18.1	20.7	0	0	100
DRL β 15	0	0.9	0.6	1.4	53.5	50.8	58.4	0	0	100

Table 3.9. Largest size training problems

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	33.3	0.0	0.0	0.2	183.2	39.0	253.0	0	0
DRL β 1	0	0.2	0.0	0.4	4.3	4.0	4.6	0	0
DRL β 5	0	0.2	0.0	0.5	21.7	20.5	23.7	0	0
DRL β 10	0	0.2	0.0	0.4	42.7	39.7	46.4	0	0
DRL β 15	0	0.1	0.0	0.4	67.2	63.0	72.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	20	0.1	0.0	0.3	197.0	21.2	247.6	0	0	0
DRL β 1	0	1.4	0.7	2.1	2.1	1.9	2.3	0	0	100
DRL β 5	0	1.2	0.7	1.8	9.7	9.3	10.8	0	13.3	100
DRL β 10	0	0.8	0.5	1.3	19.2	18.1	20.7	0	0	86.7
DRL β 15	0	0.7	0.3	1.1	53.5	50.8	58.4	0	0	86.7

Table 3.10. Largest size validation problems

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	9.1	0.1	0.0	0.2	234.1	41.9	253.7	0	0
DRL β 1	0	0.2	0.0	0.6	4.3	4.0	4.6	0	0
DRL β 5	0	0.1	0.0	0.4	21.7	20.5	23.7	0	0
DRL β 10	0	0.1	0.0	0.3	42.7	39.7	46.4	0	0
DRL β 15	4.5	0.1	0.0	0.3	67.2	63.0	72.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	13.6	0.1	0.0	0.2	218.6	21.5	247.8	0	0	8.3
DRL β 1	0	1.8	0.9	6.9	2.1	1.9	2.3	0	9.1	90.9
DRL β 5	0	1.9	0.6	11.8	9.7	9.3	10.8	0	9.1	100
DRL β 10	0	0.9	0.4	1.6	19.2	18.1	20.7	0	0	90.9
DRL β 15	0	0.9	0.2	1.5	53.5	50.8	58.4	0	0	81.8

Table 3.11. Largest size test problems

3.4.3.2. *Large Cost Value Results*

In this section we present the results of the large cost values. We begin by reporting the performance curves, showing that the model improves with training. Next, we report results between all instances, the training, validation and testing sets for the medium, large and largest problem sizes.

First, consider the performance curves in Figure 3.3. We see that both training curves are noisier, as well as reaching lower peak values than their small cost counterparts. Nonetheless, both curves improve as training continues, reaching peak values after the 3000th training episode.

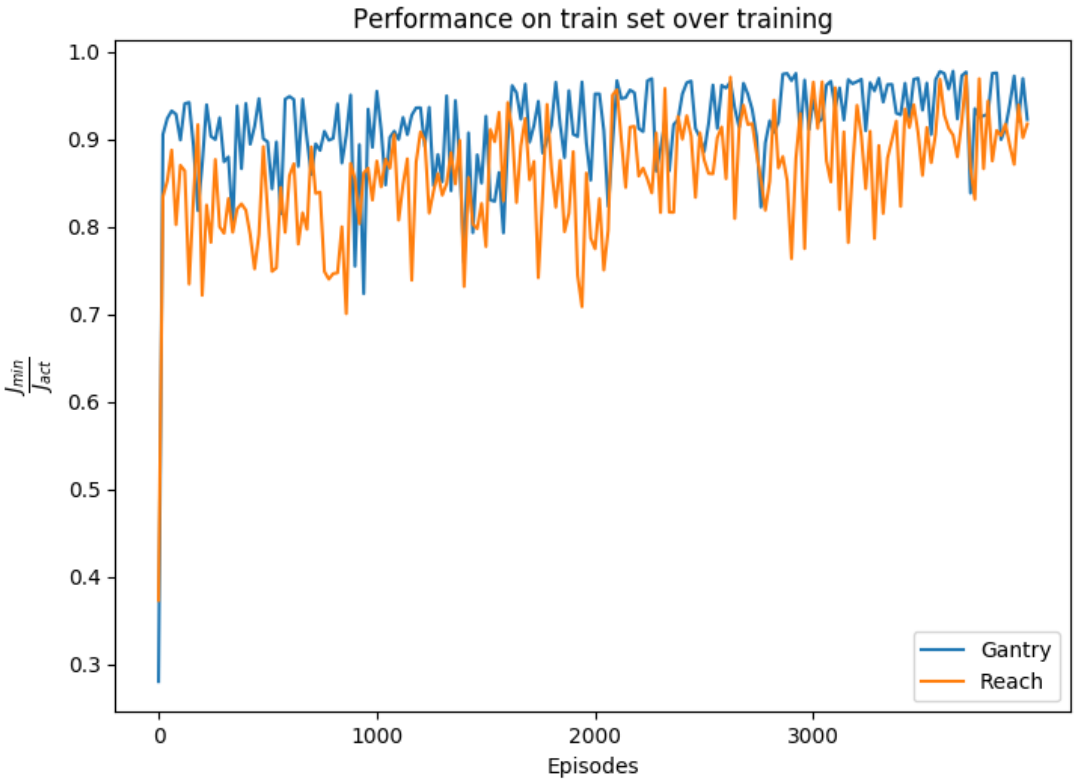


Figure 3.3. Performance of model on training set when training on uneven small cost values

We present the results of medium size training problems in Table 3.12, and compare with the results on the validation and test sets in Table 3.13 and Table 3.14. The validation set has better results with the gantry crane than both the training and testing set. Increasing beam size does not consistently improve results, especially for the reach stacker. In fact, the largest beam size has the worst result for the reach stacker on the test set. This could be due to the beam search algorithm finding overly optimistic solutions for a given state that remains unexplored with smaller beam sizes. In that case, the search would tend to discard trajectories that would have provided a better solution. This could also be due to the nonlinearities introduced by the neural network. Overall, performance is unable to approach that found by BS-B-IDA*, but the average computing time is only a fraction of the heuristic for even the largest beam sizes.

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	37.5	0.0	0.0	0.1	155.9	6.5	243.2	0	0
DRL β 1	0	0.2	0.1	0.3	1.2	0.9	1.9	0	0
DRL β 5	0	0.2	0.1	0.4	6.0	4.4	8.9	0	0
DRL β 10	0	0.2	0.1	0.3	12.0	8.8	18.3	0	0
DRL β 15	0	0.2	0.1	0.4	17.6	13.0	26.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.2	0.1	0.3	241.5	240.9	242.4	0	0	50
DRL β 1	0	0.4	0.2	0.5	1.0	0.7	1.5	0	0	100
DRL β 5	0	13.7	0.2	52.3	4.7	3.4	6.8	25	37.5	100
DRL β 10	0	2.3	0.1	6.8	9.5	7.1	14.3	0	37.5	100
DRL β 15	0	0.7	0.2	3.5	14.6	11.0	21.6	0	12.5	100

Table 3.12. Medium size training problems using large cost values

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	66.7	0.0	0.0	0.3	83.3	2.4	242.1	0	0
DRL β 1	0	0.7	0.1	2.0	1.2	0.9	1.9	0	26.7
DRL β 5	6.7	0.1	0.0	0.4	6.0	4.4	8.9	0	0
DRL β 10	6.7	0.1	0.0	0.3	12.0	8.8	18.3	0	0
DRL β 15	0	0.1	0.0	0.3	17.6	13.0	26.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.5	0.0	3.2	241.1	240.7	241.7	0	13.3	33.3
DRL β 1	0	1.3	0.1	6.1	1.0	0.7	1.5	0	26.7	86.7
DRL β 5	0	0.7	0.1	3.2	4.7	3.4	6.8	0	20	73.3
DRL β 10	0	0.5	0.1	3.2	9.5	7.1	14.3	0	6.7	86.7
DRL β 15	0	4.1	0.1	51.6	14.6	11.0	21.6	6.7	20	100

Table 3.13. Medium size validation problems using large cost values

Next, we present the results on large instance sizes. Results on training instances are shown in Table 3.15, while validation and test set results are shown in Tables 3.16 and 3.17 respectively. The model performs well on all sets when using the gantry crane. However, when using the reach stacker the model performs well on the training set but poorly on both the validation and test sets, with average gaps to the LB in the order of 7% for the validation set and 5% for the test set. Several validation and test set instances fail to fill all slots on the train for the reach stacker, leading to large gaps to the LB. Once again, increasing beam

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	40.9	0.1	0.0	0.4	144.8	2.4	244.5	0	0
DRL β 1	0	5.2	0.1	54.6	1.2	0.9	1.9	9.1	9.1
DRL β 5	4.5	7.5	0.0	54.6	6.0	4.4	8.9	13.6	13.6
DRL β 10	9.1	5.1	0.0	54.6	12.0	8.8	18.3	9.1	9.1
DRL β 15	4.5	5.1	0.0	54.6	17.6	13.0	26.6	9.1	9.1

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.5	0.0	4.8	241.4	240.7	242.4	0	9.1	36.3
DRL β 1	0	3.9	0.1	54.8	1.0	0.7	1.5	4.5	36.4	100
DRL β 5	0	1.1	0.1	6.2	4.7	3.4	6.8	0	18.2	100
DRL β 10	0	3.7	0.1	53.7	9.5	7.1	14.3	4.5	22.7	100
DRL β 15	0	8.6	0.1	54.1	14.6	11.0	21.6	13.6	27.3	100

Table 3.14. Medium size test problems using large cost values

size does not consistently improve results, though a beam size of one tends to have the worst performance for the reach stacker. The model fails to fill all slots when considering the reach stacker for both the validation and test sets. Notably, the gantry crane performs well across all sets, and increasing beam size leads to better results. Performance does not meet that of BS-B-IDA*, but comes substantially closer for the gantry crane. Once again, the average computing time is lower than BS-B-IDA*, but it remains unclear whether increasing beam size would further improve the performance of the DRL model.

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	0	0.0	0.0	0.0	246.1	243.5	252.5	0	0
DRL β 1	0	0.1	0.0	0.1	2.7	2.4	3.1	0	0
DRL β 5	0	0.1	0.0	0.2	12.9	11.9	14.9	0	0
DRL β 10	0	0.1	0.0	0.1	26.4	24.2	30.4	0	0
DRL β 15	0	0.1	0.0	0.1	38.4	35.3	44.2	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.1	0.0	0.1	243.1	242.5	243.6	0	0	50
DRL β 1	0	0.4	0.1	0.6	1.8	1.7	2.0	0	0	100
DRL β 5	0	0.4	0.1	0.7	9.0	8.5	10.1	0	0	100
DRL β 10	0	0.4	0.1	0.7	18.2	17.1	21.4	0	0	100
DRL β 15	0	0.4	0.1	0.8	27.3	25.7	30.6	0	0	100

Table 3.15. Large size training problems using large cost values

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	26.7	0.0	0.0	0.1	183.3	17.6	253.4	0	0
DRL β 1	0	0.1	0.0	0.2	2.7	2.4	3.1	0	0
DRL β 5	0	0.1	0.0	0.2	12.9	11.9	14.9	0	0
DRL β 10	0	0.1	0.0	0.1	26.4	24.2	30.4	0	0
DRL β 15	0	0.1	0.0	0.2	38.4	35.3	44.2	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.6	0.0	8.7	243.9	242.3	248.5	0	6.7	80
DRL β 1	0	7.6	0.1	52.3	1.8	1.7	2.0	13.3	33.3	100
DRL β 5	0	7.3	0.1	53.3	9.0	8.5	10.1	13.3	13.3	100
DRL β 10	0	7.2	0.1	53.3	18.2	17.1	21.4	13.3	13.3	100
DRL β 15	0	7.2	0.1	52.5	27.3	25.7	30.6	13.3	13.3	100

Table 3.16. Large size validation problems using large cost values

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]
BS-B-IDA*	31.8	0.0	0.0	0.2	172.8	13.7	247.0	0	0
DRL β 1	0	0.1	0.0	0.3	2.7	2.4	3.1	0	0
DRL β 5	0	0.1	0.0	0.2	12.9	11.9	14.9	0	0
DRL β 10	0	0.1	0.0	0.2	26.4	24.2	30.4	0	0
DRL β 15	0	0.1	0.0	0.2	38.4	35.3	44.2	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%] Avg.	Min	Max	CPU [s] Avg.	Min	Max	Incomplete Load [%]	Double Touches [%]	Detours [%]
BS-B-IDA*	0	0.4	0.0	7.3	243.5	242.3	248.4	0	4.5	68.2
DRL β 1	0	5.3	0.1	52.9	1.8	1.7	2.0	9.1	18.2	100.0
DRL β 5	0	2.8	0.1	53.2	9.0	8.5	10.1	4.5	4.5	100
DRL β 10	0	3.0	0.1	51.4	18.2	17.1	21.4	4.5	13.6	100
DRL β 15	0	5.0	0.0	53.9	27.3	25.7	30.6	9.1	9.1	100

Table 3.17. Large size test problems using large cost values

Finally, we present the results for the problems of largest size. Table 3.18 shows the results for the training instances, Table 3.19 shows the results for the validation instances and Table 3.20 shows the results for the test instances. Here, we see that increasing beam size generally leads to better performance, but not in all cases. In fact, when considering the validation set for the reach stacker a beam size of 10 leads to incomplete loads on 13.3% of instances, whereas a beam size of 5 achieves complete loads on all instances. However, increasing beam size to 15 did remove all cases wherein all slots were not filled for the reach

stacker. Overall, both the reach stacker and gantry crane are able to maintain gaps to the LB of less than 1% across all instances provided beam size is sufficiently large. Performance once again does not approach that of BS-B-IDA* but computing times remain smaller.

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max		
BS-B-IDA*	0	0.0	0.0	0.0	250.6	247.8	252.2	0	0
DRL β 1	0	0.3	0.3	0.4	4.6	4.3	4.9	0	0
DRL β 5	0	0.3	0.2	0.3	22.2	20.5	24.7	0	0
DRL β 10	0	0.3	0.2	0.3	44.8	41.7	49.0	0	0
DRL β 15	0	0.3	0.2	0.4	64.9	60.9	70.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max			
BS-B-IDA*	0	0.1	0.0	0.1	245.7	244.3	246.6	0	0	75
DRL β 1	0	0.8	0.4	1.0	2.8	2.7	3.1	0	0	100
DRL β 5	0.0	1.0	0.7	1.7	13.3	12.2	14.5	0	25	100
DRL β 10	0	0.8	0.7	0.9	27.2	25.4	30.0	0	0	100
DRL β 15	0	0.7	0.7	0.9	40.7	38.4	43.3	0	0	100

Table 3.18. Largest size training problems using large cost values

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max		
BS-B-IDA*	26.7	0.0	0.0	0.0	206.5	37.8	266.9	0	0
DRL β 1	0	0.4	0.2	0.6	4.6	4.3	4.9	0	0
DRL β 5	0	0.3	0.2	0.5	22.2	20.5	24.7	0	0
DRL β 10	0	0.3	0.2	0.4	44.8	41.7	49.0	0	0
DRL β 15	0	0.3	0.2	0.5	64.9	60.9	70.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max			
BS-B-IDA*	0	0.6	0.0	9.6	246.6	244.1	255.7	0	6.7	66.7
DRL β 1	0	4.2	0.5	53.2	2.8	2.7	3.1	6.7	6.7	100
DRL β 5	0	0.9	0.5	2.5	13.3	12.2	14.5	0	6.7	100
DRL β 10	0	7.8	0.6	53.6	27.2	25.4	30.0	13.3	13.3	100
DRL β 15	0	0.9	0.6	1.2	40.7	38.4	43.3	0	0	100

Table 3.19. Largest size validation problems using large cost values

3.5. Discussion

This section offers a discussion of the previously reported results. We start by discussing the noisy training curves, and a hypothesis for their cause. Next, we discuss the benefit of

Gantry Crane

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]
		Avg.	Min	Max	Avg.	Min	Max		
BS-B-IDA*	18.2	0.0	0.0	0.1	218.2	47.5	267.6	0	0
DRL β 1	0	1.7	0.2	15.7	4.6	4.3	4.9	0	9.1
DRL β 5	0	2.3	0.2	22.1	22.2	20.5	24.7	0	9.1
DRL β 10	0	0.3	0.2	0.5	44.8	41.7	49.0	0	0
DRL β 15	0	0.3	0.2	0.5	64.9	60.9	70.6	0	0

Reach Stacker

Model	Solved to LB [%]	Gap to Lower Bound [%]			CPU [s]			Incomplete Load [%]	Double Touches [%]	Detours [%]
		Avg.	Min	Max	Avg.	Min	Max			
BS-B-IDA*	0	0.1	0.0	0.2	246.6	243.9	255.6	0	0	68.2
DRL β 1	0	3.2	0.4	52.3	2.8	2.7	3.1	4.5	4.5	100
DRL β 5	0	5.6	0.5	52.8	13.3	12.2	14.5	9.1	9.1	100
DRL β 10	0	0.9	0.5	1.7	27.2	25.4	30.0	0	0	100
DRL β 15	0	0.8	0.5	1.6	40.7	38.4	43.3	0	0	100

Table 3.20. Largest size test problems using large cost values

beam search in this environment and follow with remarks on the difference in performance between the two distance cost values. Finally, we discuss the challenges faced in our work.

First, consider the performance curves in Figures 3.2 and 3.3. Both curves vary wildly over episodes leading to unpredictable behaviour. We hypothesize that this behaviour is exhibited as the model becomes more greedy, placing nearer containers as it is rewarded for minimizing travel cost, until such time that the model neglects the constraints imposed by the problem, and begins finding it difficult to fill all slots of the railcars. Deep reinforcement learning does not yet have a strong set of tools for dealing with hard constraints such as the COG constraint, or loading pattern constraints seen here. It is hoped that these constraints will be learned, but hard constraints such as these produce a challenging learning environment.

We believe that beam search allows for an alternative mechanism for dealing with hard constraints. The beam search allows the model to expand several options at each time, so should it select an action which leads to poor solutions, the model has other trajectories to choose from. Reference [57] presents an algorithm wherein DQNs learn to use beam search during training via imitation learning. We believe this would be a good candidate here.

There is an important difference in performance between the sets of values used to define the environment costs for the reach stacker. In the large values set, travel in the x direction is heavily penalized relative to the y direction. This promotes the model selecting containers lot by lot, leaving fewer and fewer reachable containers for the reach stacker as more lots are removed. The model currently does not appear to learn to avoid this behaviour, and it can lead to poor performance as a result.

There were many obstacles for designing a reinforcement learning agent for this environment. The system has an extremely large action space, which is constantly changing based on the state, several input modalities, and, as discussed, hard constraints which are difficult to impose while training a model. While we have succeeded in training a network which performs well when considering the gantry crane, we have had less success with the reach stacker.

3.6. Conclusion

We showed that a neural network trained using deep Q-learning is able to learn to move containers effectively in the LPSP environment. While the DQN did not perform at the same level as the dynamic programming heuristic, there exist many opportunities for improvement not presented here. First, deep Q-learning is typically used in environments with relatively small action spaces, for example in Atari games, but existing research results could help improve the ability of deep Q-learning to be applied to large action spaces, as described in [19]. Moreover, DQNs can use beam search during training rather than only at test time as was performed here, as discussed here [57]. We leave exploration of these alternatives for future work.

Acknowledgements

We gratefully acknowledge the close collaboration with personnel from the Canadian National Railway Company (CN), the funding through the CN Chair in Optimization of Railway Operations at Université de Montréal and funding from the National Sciences and Engineering Council of Canada (NSERC) through a Collaborative Research and Development grant.

Part 2

Learning Long-term Dependencies While Increasing Expressivity in Sequential Models

Chapter 4

Introduction: Recurrent Neural Networks and the Exploding and Vanishing Gradient Problem

Problems involving sequential data arise when working with natural language processing (NLP), speech recognition and predictive analytics, for example with stock prices. Recurrent neural networks (RNNs) are a common and powerful tool for working with these types of data, having the ability to store information in their state, to use this information at later time steps, as well as being able to handle input sequences of variable length.

It is a well-known phenomenon that RNNs suffer from the exploding and vanishing gradient problem (EVGP) during training, which leads to challenges in training long-term dependencies that are robust to noise. The EVGP stems from applying the backpropagation algorithm over long sequences (>150), and causes the gradient in these sequences to explode or vanish, depending on the eigenvalues of the recurrent weight matrix. A common method for dealing with the EVGP is to enforce normality onto the recurrent weight matrix, but this comes at the cost of reduced expressivity of the model. We propose a non-normal recurrent neural network (nnRNN) architecture which is able to control the EVGP, while allowing short-term complex interactions, that increases expressivity of the model over imposing normality on the recurrent neural network, through the Schur decomposition [34].

This chapter will provide an overview of the RNNs, the EVGP and the methods used to approach the EVGP problem for RNNs. Furthermore, it will introduce required mathematical background for the nnRNN presented in Chapter 5. Section 4.1 outlines RNNs, their architecture, and the backpropagation algorithm for RNNs. Section 4.2 outlines the root cause of the EVGP. Section 4.3 presents the real Schur decomposition which is used to define the nnRNN.

4.1. Recurrent Neural Networks

RNNs, introduced by [68], are neural networks designed to process sequential data, often with variable length. These models use *parameter sharing*, wherein the same parameters are

used across different time steps, which improves generalization to sequence lengths not seen during training. RNNs also form a discrete dynamical system, as hidden states are iterated forward in time according to the parametrized connectivity.

Consider the sequence of inputs $x_1, x_2, ..., x_n$ where $x_i \in \mathbb{R}^n$, and the output sequence $y_1, y_2, ..., y_n$ with $y_i \in \mathbb{R}^k$ (both deterministic). The RNN model applies the equations:

$$h_t = f(Ux_t + Vh_{t-1} + b) \tag{4.1}$$

$$o_t = Wh_t + c \tag{4.2}$$

$$\hat{y}_t = g(o_t) \tag{4.3}$$

where:

- $U \in \mathbb{R}^{nxm}$ contains the connections from input to the hidden state.
- $x_t \in \mathbb{R}^n$ is the input at time t.
- $V \in \mathbb{R}^{mxm}$ is called the recurrent weight matrix and contains the weights from hidden state to hidden state.
- $h_t \in \mathbb{R}^m$ is the hidden state at time t.
- $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^k$ are biases.
- $W \in \mathbb{R}^{mxk}$ contains the connections from hidden state to the output.
- f is a differentiable nonlinear activation function.
- o_t is the output at time t.
- g is some differentiable function, converting the output of the RNN to a format directly comparable to the target, for example a softmax function.

4.2. The Exploding and Vanishing Gradient Problem

Backpropagation in RNNs is performed with an algorithm called *back propagation through time* (BPTT), which propagates errors from each time step to all previous time steps. For brevity, we will not derive the gradient calculation here. However, it can be shown that the gradient of the loss function, L, with respect to the hidden state h_t, through an output of time τ is proportional to the product of the transposed recurrent weight matrix V^T times the diagonal matrix D_s (containing derivatives of the activation functions of each hidden unit):

$$\nabla_{h_t} L \propto \prod_{s=t+1}^{\tau} (V^T \cdot D_s). \tag{4.4}$$

If we consider a linear RNN, wherein the activation functions have derivative 1, and $D_s = \mathbb{I}$, then the gradient is proportional to the self-product of the transposed recurrent weight matrix V^T. We can factorize V with the eigendecomposition $V = Q\Lambda Q^T$, where Q is

the orthogonal matrix whose columns are eigenvectors of V and Λ is a diagonal matrix whose elements are the eigenvalues of V. From this factorization we can see that the gradient with respect to h_t is proportional to the diagonal eigenvalue matrix Λ raised to the power of $\tau - t$:

$$\nabla_{h_t} L \propto Q^T \Lambda^{\tau - t} Q. \tag{4.5}$$

Thus, for $\tau - t$ sufficiently large, one of three cases can occur. For eigenvalues greater than one, $\Lambda^{\tau-t}$ will explode, for eigenvalues less than one, $\Lambda^{\tau-t}$ will decay to zero, and with eigenvalues equal to one, $\Lambda^{\tau-t}$ will neither vanish nor explode. Several mechanisms have been suggested to control this behaviour, including gating mechanisms, and controlling the eigenspectrum of the recurrent weight matrix.

4.3. The Real Schur Decomposition

The real Schur decomposition is used to decompose any real matrix A into an orthogonal matrix Q, and a block upper triangular matrix T, with diagonal block sizes of at most two, as

$$A = QTQ^T.$$

Here, T can be further decomposed into a block diagonal matrix R, and an upper triangular matrix U as

$$A = Q(R + U)Q^T$$

where each two-by-two block on the diagonal of R contains a complex conjugate pair of eigenvalues for the matrix A. These complex conjugate eigenvalues can be expressed with real numbers as follows:

$$R = \begin{bmatrix} r_1 & 0 & \dots & 0 \\ 0 & r_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & r_{N/2} \end{bmatrix}$$

$$r_k = \gamma_i \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix}$$

where γ_i represents the norm of the complex conjugate pair of eigenvalue i, and θ_i represents the angle in the complex plane. One can see that if γ_i is maintained at 1, the eigenvalue has norm of 1.

This decomposition is central to the idea of the nnRNN, and is used to mitigate the EVGP, while allowing for non-normal matrices to be exploited as recurrent weight matrices.

Chapter 5

Third Article.

Non-normal Recurrent Neural Network (nnRNN): learning long time dependencies while improving expressivity with transient dynamics

by

Giancarlo Kerg[1,2], Kyle Goyette[1,3,4], Maximilian Puelma Touzel[2,5], Gauthier Gidel[2,3], Eugene Vorontsov[2,6], Yoshua Bengio[2,3,7], and Guillaume Lajoie[2,8]

1: Indicates first authors. Ordering determined by coin flip.
2: Mila - Quebec AI Institute, Canada
3: Université de Montréal, Département d'informatique et de recherche opérationelle, Montreal, Canada
4: Université de Montréal, CIRRELT, Montreal, Canada
5: IVADO post-doctoral fellow
6: Ecole Polytechnique de Montréal, Montreal, Canada
7: CIFAR senior fellow
8: Université de Montréal, Département de mathématiques et de statistique, Montreal, Canada

Submitted and published in NeurIPS 2019.

Author Contributions

My contributions to this paper are as follows:

- involvement in discussion of the use of the Schur Decomposition to solve the EVGP,
- selection of regularization methods for lower triangular matrix,
- implementation of the nnRNN, as well as all large scale experiments used to benchmark the quality of the nnRNN,
- hyperparameter search and analysis of results, presentation of results which lead to deeper understanding of the nnRNN,
- writing parts of the published paper related to experimental results and experimental setup.

5.1. Introduction

Training recurrent neural networks (RNN) to process temporal inputs over long timescales is notoriously difficult. A central factor is the exploding and vanishing gradient problem (EVGP) [33, 6, 61], which stems from the compounding effects of propagating signals over many iterates of recurrent interactions. Several approaches have been developed to mitigate this issue, including the introduction of gating mechanisms (e.g. [36, 33]), purposely using non-saturating activation functions [14], and manipulating the propagation path of gradients [3]. Another way is to constrain connectivity matrices to be orthogonal (and more generally, unitary) leading to a class of models we refer to as *orthogonal RNNs* [55, 52, 48, 81, 37, 80, 27, 2]. Orthogonal RNNs have eigen- and singular-spectra with unit norm, therefore helping to prevent exponential growth or decay in long products of Jacobians associated with EVGP. They perform exceptionally well on tasks requiring memorization of inputs over long timescales [28] (outperforming gated networks) but struggle on tasks involving continued computations across timescales. A contributing factor to this limitation is the mutually orthogonal nature of connectivity eigendirections which substantially limits the space of solutions available to orthogonal RNNs.

In this paper, we propose a first step toward a solution to this expressivity problem in orthogonal RNNs by allowing non-orthogonal eigenbases while retaining control of eigenvalues norms. We achieve this by leveraging the *Schur* decomposition of the connectivity matrix, though we avoid the need to compute this costly factorization explicitly. This provides a separation into "diagonal" and "feed-forward" parts, with their own optimization constraints.

Mathematically, our contribution amounts to adding "non-normal" connectivity, and we call our novel architecture *non-normal RNN* (nnRNN). In linear algebra, a matrix is called *normal* if its eigenbasis is orthogonal, and *non-normal* if not. Orthogonal matrices are normal, with eigenvalues of norm 1 (i.e. on the unit circle). In recurrent networks, normal connectivity produces dynamics solely characterized by the eigenspectrum while non-normal connectivity allows transient expansion and compression. Transient dynamics have known computational advantages [30, 20, 21], but orthogonal RNNs cannot produce them. The added flexiblity in nnRNN allows such transients, and we show analytically how they afford additional expressivity to better encode complex inputs, while at the same time retaining efficient signal propagation to learn long-term dependencies. Through a series of numerical experiments, we show that the nnRNN provides two main advantages:

(1) On tasks well-suited for orthogonal RNNs, nnRNN learns orthogonal (normal) connectivity and matches state-of-the art performance while training as fast as orthogonal RNNs.

(2) On tasks requiring additional expressivity, non-normal connectivity emerges from training and nnRNN outperforms orthogonal RNNs.

From a parametric standpoint, this advantage can be attributed to the fact that the nnRNN has access to *all* matrices with unit-norm eigenspectra, of which orthogonal ones are only a subset.

5.2. Background

5.2.1. Unitary RNNs and constrained optimization

First outlined in [2] and inspired by [70, 82, 45], RNNs whose recurrent connectivity is determined by an orthogonal, or unitary matrix are a direct answer to the EVGP since their eigenspectra and singular spectra exactly lie on the complex unit circle. The same mechanism was invoked in a series of theoretical studies for deep and recurrent networks in the large size limit, showing that ideal regimes for effective network performance are those initialized with such spectral attributes [65, 62, 16]. By construction, orthogonal matrices and their complex-valued counterparts, unitary matrices, are isometric operators and do not expand or contract space, which helps to mitigate the EVGP. A central challenge to train unitary RNNs is to ensure that parameter updates are restricted to the manifolds satisfying orthogonality constraints known as *Stiefel* manifolds (see review in [35]). This is an active area of optimization research, and several techniques have been used for orthogonal or unitary RNN training. In [2], the authors construct connectivity matrices with long products of rotation matrices leveraging fast Fourier transforms. In [81, 80, 27], the Cayley transform is used, which parametrizes weight matrices using skew-symmetric matrices that need to be inverted (cf. [15] for an RNN implementation directly using skew-symmetric matrices).

Another approach uses Householder reflections [55]. Recent studies also adapt some of these methods to the quaternion domain [59]. The methods listed above have their advantages by either being fast, or memory efficient, but suffer from only parametrizing a subset of all orthogonal (unitary) matrices. A novel approach considering the group of unitary matrices as a Lie group and leveraging a parametrization via the exponential map applied to its Lie algebra addresses this problem and currently outperforms the rest on many tasks [48]. Still, of all matrices with unit-norm eigenvalues, unitary matrices are only a small subset and remain limited in their expressivity since they are restricted to isometric transformations [28]. This is why orthogonal RNNs, while performing better than a conventional RNN or LSTM at some tasks (e.g. copy task [33], or sequential MNIST [45]), struggle at more complex tasks requiring computations across multiple timescales.

5.2.2. Non-normal connectivity

Any diagonalizable matrix V can be expressed as $V = P\Theta P^{-1}$ where P's columns are V's eigenvectors and Θ is a diagonal matrix containing its eigenvalues. V is said to be *normal* if its eigenbasis is orthogonal and thus, $P^{-1} = P^\top$ and $V = P\Theta P^\top$. Orthogonal matrices are normal matrices with eigenvalues on the unit circle. When a matrix is *non-normal*, it is diagonalized with a non-orthogonal basis. However, it is still possible to express it using an orthogonal basis at the cost of adding (lower) triangular structure to Θ. This is known as the *Schur* decomposition: for any matrix V, we have $V = P(\Lambda + T)P^\top$ with P an orthogonal matrix, Λ a diagonal matrix containing the eigenvalues, and T a strictly lower triangular matrix.[1] In short, T contains the interactions between the orthogonal column vectors of P (called *Schur modes*). P and T are obtained from orthogonalizing the non-orthogonal eigenbasis of V, and do not affect the eigenspectrum. As a recurrent connectivity matrix, T represents a purely feed-forward structure that produces strictly transient dynamics impossible to produce in normal (orthogonal) matrices. In other words, if normal and non-normal matrices share exactly the same eigenspectrum, the iterative propagation of an input will be equivalent in the long-term, but can differ greatly in the short-term. We revisit this distinction in §5.3. This was exploited by [21, 30] to analyze the decomposition of the activity of recurrent networks (in continuous time) into a normal part responsible for slow fluctuations, and a non-normal part producing fast, transient ones. How this mechanism propagates information was studied in [20] for stochastic linear dynamics. The authors show analytically that non-normal dynamics can lead to extensive memory traces, as measured by the Fisher information of the distribution of hidden state trajectories parametrized by the input signal. To the best of our knowledge, an explicit demonstration and explanation of the benefits of

[1] When eigenvalues and eigenvectors are complex, P is unitary and $P^\top$ corresponds to conjugate transposition. However for any real V, it is possible to find an orthogonal (real) P with Λ being block-diagonal with 2×2 blocks instead of complex-conjugate eigenvalues.

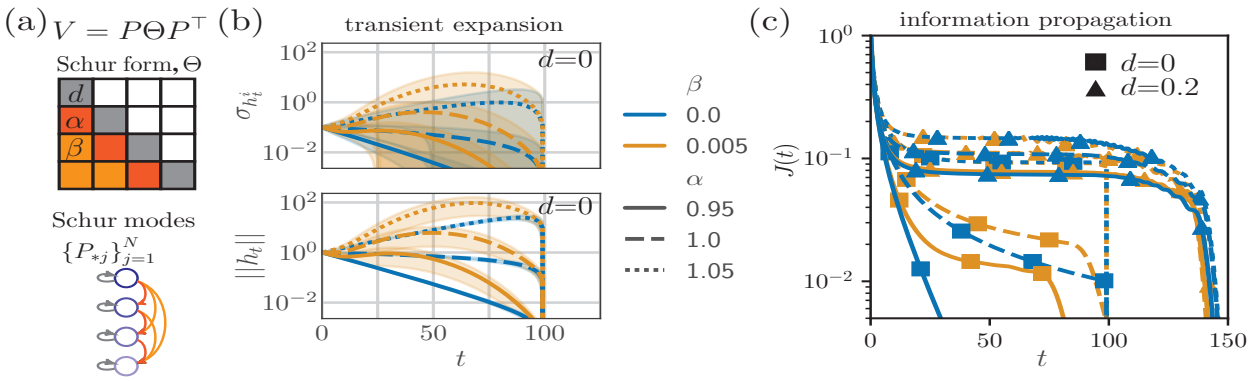


Figure 5.1. Benefits of non-normal dynamics

non-normal dynamics for learning in RNNs is lacking, though see [58] for similar ideas used for initialization.

5.3. Non-normal matrices are more expressive and propagate information more robustly than orthogonal matrices

We now outline the role of non-normal connectivity we exploit for recurrent network parametrization. To provide mathematically grounded intuition for the benefit it provides for learning, we first consider generic RNN dynamics,

$$\begin{aligned} h_{t+1} &= \phi(Vh_t + Ux_{t+1} + b) \\ V &= P\Theta P^\top,\ \Theta = \Lambda + T \end{aligned} \tag{5.1}$$

where $h_t \in \mathbb{R}^N$ is the time-varying hidden state vector, ϕ is a nonlinear function, x_t is the input sequence projected into the dynamics via matrix U, and b is a bias (we omit the output for brevity). V is the matrix of recurrent weights, which in line with §5.2.2 we decompose into its lower triangular Schur form Θ in Eq. (5.1), with P orthogonal and Θ lower triangular. Θ has two parts: a (block) diagonal part Λ, and a strictly lower triangular part T.[2] The Schur decomposition maps the hard problem of controlling the directions of a non-orthogonal basis to the easier problem of specifying interactions between fixed orthogonal modes. It is important to highlight the fact that an orthonormalization of the eigenbasis is just a change in representation and thus has no effect on the spectrum of V, which still lies on the diagonal of Λ. The triangular part T can thus be modified independently from the constraint (employed in orthogonal RNN approaches) that the spectrum have norms equal or near 1.

The ability to encode complex signals and then selectively recall past inputs is a basic requirement needed to solve many sequence-based tasks. Intuitively, the two features that allow systems to perform well in such tasks are:

(1) High-dimensional activity to better encode complex input,

(2) Efficient signal propagation, to better learn long-term dependencies.

[2]We use the real Schur decomposition but a similar treatment can be derived for the complex case.

To illustrate how non-normal dynamics controlled by the entries in the lower triangle of T contribute to these two features, we consider a simplified linear case where $\phi(h_t) = h_t$ and Θ is parametrized as follows and illustrated in Fig. 5.1(a)):

$$(\Theta)_{i,j} = d\delta_{i,j} + \alpha\delta_{i,j+1} + \beta \sum_{2\leq k\leq i} \delta_{i,j+k}\,. \tag{5.2}$$

Fig. 5.1(a) shows that the Schur decomposition provides the lower triangular Schur form. A feed-forward interaction coupling among Schur modes underlies non-normal dynamics.

Here, diagonal entries are set to d, sub-diagonal entries to α, and the remaining entries in the lower triangle to β. By varying α and β we will show how the lower triangle in T enhances expressivity and information propagation.

5.3.1. Non-normality drives expressive transients

RNNs can be made more expressive with stronger fluctuations of hidden state dynamics. The dependence of hidden state variance on the values of Θ was studied in depth in [30]. Here, we present experiments where the RNN parametrized by Eqs. (5.1), (5.2) exemplifies some of those results. We numerically compute a set of trajectories over a sampled ensemble of inputs with $x_t > 0$ for $t = 0$ and 0 otherwise. Without loss of generality we assume a form of U and distribution of x_0 that leads to input-dependent initial conditions on the unit hypersphere in the space of h_t. For $\alpha = 0.95$,1.0,1.05, $\beta = 0, 0.005$ and $d = 0$, we see that trajectories of single units exhibit increasing large transients with increasing α and β, that abruptly end at $t = N$ (Fig. 5.1(b)). Fig. 5.1 shows that the lower triangle generates stronger transients. Trajectories of standard deviation across hidden units (top) and norm of hidden state vector (bottom) are obtained from the dynamics of Eq. (5.1). Lines and shading are average and standard deviation, respectively, over 10^3 initial conditions uniformly distributed on the unit hypersphere. The latter is a result of the nilpotent property of a strictly triangular matrix: each iteration removes the top entries in each column until $\Theta^N = 0$. Computing ensemble statistics, we find that α contributes significantly to the strength of the exponential amplification, while β structures the shape of the transient. This ability of T to both exhibit amplification, and to control its shape, is what endows the Schur form Θ with expressivity (see §5.5.3 for empirical evidence in trained nnRNNs).

5.3.2. Non-normality allows for efficient information propagation

Propagation of information in a network requires feed-forward interactions. Perhaps the most simple example of a feed-forward structure is the local feed-forward chain (also called *delay-line* [20]), where each mode feeds its signal only to the next mode in the chain ($\alpha > 0$, $\beta = 0$, $d = 0$; see Fig. 5.1(a)). In this case, we denote Θ by Θ_{delay}. As a consequence, signals feeding the first entry of Θ_{delay} propagate down the chain and are amplified or attenuated

according to the values of these non-zero entries. Moreover, inputs from different time steps do not interact with each other thanks to this ordered propagation down the line. In contrast, the signal is not propagated across modes for dynamics given by a purely (block) diagonal Θ. It instead simply decays within the mode into which it was injected on the timescale intrinsic to that mode, which can be much less than the $O(N)$ timescale of the chain.

To quantify the efficiency with which a RNN can store inputs, we follow and extend the approach of [20]. For a given scalar-valued input sequence, $x_t = s_t + \xi_t$, $t \in \mathbb{N}$, composed of signal s_t and injected noise ξ_t, the noise ensemble induces the conditional distribution, $P(h_{:t}|s_{:t})$, over trajectories of hidden states, $h_{:t}$, given the received input, $s_{:t}$, where the $:t$ subscript is short hand for $(k : k \leq t)$. Taking the signal sequence $s_{:t}$ as a set of parameters of a model, and $P(h_{:t}|s_{:t})$ as this model's likelihood, provided that the information identity property holds, the corresponding Fisher information matrix that captures how $P(h_{:t}|s_{:t})$ changes with the input $s_{:t}$ is,

$$\mathbf{J}_{k,l}(s_{:t}) = \left\langle -\frac{\partial^2}{\partial s_k \partial s_l} \log P(h_{:t}|s_{:t}) \right\rangle_{P(h_{:t}|s_{:t})} \qquad k,l \leq t\,. \tag{5.3}$$

The diagonal of this matrix, $J(t) := \mathbf{J}_{t,t}$ is called the Fisher memory curve (FMC) and has a simple interpretation: if a single signal s_0 is injected into the network at time 0, then $J(t)$ is the Fisher information that h_t retains about this single signal.

In [20], the authors proved that the delay line Θ_{delay} achieves the highest possible values for the FMC when $k \leq N$: $J(k) = \alpha^k \frac{\alpha-1}{\alpha^{k+1}-1}$. However, we show (proof in SM§5.7.2) that any strictly lower triangular matrix may approach the performance of a delay line:

Proposition 3. *Let $\Theta \in \mathbb{R}^{N\times N}$ be any strictly lower triangular matrix with $\sqrt{\alpha}$ on the lower diagonal and let $T_{Gram} \in \mathbb{R}^{N\times N}$ be the triangular matrix associated with the Gram–Schmidt orthogonalization process of the columns of Θ (thus with only 1 on the diagonal). Then,*

$$J(k) \geq \frac{\alpha^k}{\sigma_{\max}^{2(N-1)}} \frac{\alpha-1}{\alpha^{k+1}-1}\,, \tag{5.4}$$

where $\sigma_{\max}$ is the maximum singular value of T_{Gram}.

Note that $\sigma_{\max} \geq 1$ and is equal to 1 for a delay line and close to 1 when Θ is close to a delay line. In Fig. 5.1(a) we present a class of matrices providing feed-forward interaction and compute the FMC of some matrices of this class in Fig. 5.1(c) which shows the Fisher memory curves across α and β. The delay line from [20] with $\alpha > 1$ (shown to be optimal for $t \leq N$) retains the most Fisher information across time up to time step N, when the nilpotency of Θ erases all information. As expected from Prop. 3, non-zero β, which endows the dynamics with expressivity (Fig. 5.1(b)), does not significantly degrade the information propagation of the delay line. Interestingly, the addition of diagonal terms ($d > 0$), i.e. Λ non-zero, helps to maintain almost optimal values of the FMC for $t < N$, while extending

the memory beyond $t = N$, and thus outperforming the delay line with regard to the area under the FMC (see Table 5.3 in the supplemental materials (SM)).

Together with the last section, these results demonstrate that non-normal dynamics, as parametrized through the entries in the lower triangle of Θ, provide significant benefits to expressivity and information propagation. What remains to show is how these benefits translate into enhanced performance of our nnRNN on actual tasks.

5.3.3. Non-normal matrix spectra and gradient propagation

While eigenvalues control the exponential growth and decay of matrix iterates, the spectral norm of these iterates may behave differently [6]. This norm is dominated by the modulus of the largest singular value of the matrix, and can thus differ from the eigenvalues' moduli. This is a subtle difference influencing gradient growth rates, and is explicitly revealed by different spectral constraints on RNNs. For comparison, a singular value decomposition (SVD) is presented in [83] with the same motivation as our Schur-decomposition: to maintain expressivity, whilst controlling a sprectrum (both using regularization). First note that, while constraining the eigenspectrum to the unit circle, non-normality implies having the largest singular value (and thus the spectral norm of the Jacobian) greater than 1. Hence, our approach mitigates gradient vanishing, but not necessarily gradient explosion. In this case however, gradients explode polynomially in time rather than exponentially [61, 2]. We provide a theorem (proof in SM§5.7.3) to establish this for triangular matrices.

Proposition 4. *Let $A \in \mathbb{R}^{n\times n}$ be a matrix such that $A_{ii} = 1$, $A_{ij} = x$ for $i < j$, and $A_{ij} = 0$ otherwise. Then for all integer $t \geq 1$ and $j > i$, we have $(A^t)_{ij} = p^{(t)}_{j-i}(x)$ is polynomial in x of degree at most $j - i$, where the coefficient of x^0 is zero and the coefficient of x^l is $O(\binom{t}{l})$ for $l = 1,2,\ldots,j - i$ (which is polynomial in t of degree at most l).*

This reveals that gradient explosion in nnRNN with unit-norm eigenspectrum, if present, is polynomial and thus not as severe as the case where eigenvalues are larger than one (in which case the gradient explosion is exponential). In §5.5.3, we illustrate that relaxing unit-norm requirements for eigenvalues using regularization allows the optimizer to find a task-dependent trade-off, thus balancing control over exponential vanishing and polynomial exploding gradients respectively. See also SM§5.7.6 for gradient propagation measurements.

5.4. Implementing a non-normal RNN

The nnRNN is a standard RNN model where we parametrize the recurrent connectivity matrix V using its real Schur decomposition[3] as in Eq. (5.1), yielding the form:

$$V = P \left(\begin{bmatrix} \mathcal{R}_1 & 0 & \dots & 0 \\ 0 & \mathcal{R}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathcal{R}_{N/2} \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & 0 \\ t_{2,1} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ t_{N,1} & t_{N,2} & \dots & 0 \end{bmatrix} \right) P^\top \tag{5.5}$$

with

$$\mathcal{R}_i(\gamma_i,\theta_i) \stackrel{\text{def}}{=} \gamma_i \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix},$$

where P is constrained to be an $N \times N$ orthogonal matrix. Each parameter above (including entries in P) is subject to optimization, as well as specific constraints outlined below. We note that although this parametrization uses the Schur form, we never explicitly compute Schur decompositions, which would be expensive and has stability issues.[4] Note that Eq. 5.5 can express any matrix V with a set of complex-conjugate pairs of eigenvalues.

During training, the orthogonal matrix P is optimized using the expRNN algorithm [48], a Riemannian gradient descent-like algorithm operating inside the Stiefel manifold of orthogonal matrices. We note that other suitable orthogonality-preserving algorithms could be used here (see §5.2) but we found expRNN to be the fastest and most stable. Instead of rigidly enforcing that eigenvalues be of unit norm, we found relaxing this constraint to be helpful. We therefore allow γ_i to be optimized but add a strong L2 regularization constraint $\delta \| |1-\gamma_i| \|_2^2$ to encourage them to be to close to 1. The hyperparameter δ is tuned differently for each task (see SM§5.7.1) but remains high overall, indicating only mild departure from unit-norm eigenvalues. Both θ_i and t_{ij} are freely optimized via automatic differentiation. The non-linearity we use is *modReLU*, as defined in [2, 27]. We initialize P as in [48] using Henaff or Cayley initialization scheme [28], θ_i from a uniform distribution between 0 and 2π, and γ_i's are initialized at 1.

We reiterate that the set of orthogonal matrices is a subset of all the connectivity matrices covered by nnRNN, by setting all γ's to 1, and $T = 0$. Consequently, the connectivity matrix in nnRNN has more parameters than an orthogonal matrix: $N(N-1)/2$ for T, and $N/2$ γ_i's, which in total gives roughly $N^2/2$ more parameters than orthogonal RNNs.

The forward pass of the nnRNN has the same complexity as that of a vanilla RNN, that is $O(Tn^2)$, for a hidden state of size n and a sequence of length T. The backward pass is similarly $O(Tn^2)$ plus the update cost of P, in addition to a once-per-update cost of $O(n^3)$ to combine the Schur parametrization via matrix multiplication. Importantly, the

[3] See discussion for more details about complex-valued implementations.
[4] See SM§5.7.4 for a discussion.

nnRNN leverages any orthogonal/unitary optimizer for P which have complexities ranging from $O(n \log n)$ to $O(n^3)$ at each update, with their own advantages and caveats (see §5.2.1). We chose the expRNN scheme, which is $O(n^3)$ in the worst case, but has fast run-time in practice.

5.5. Numerical experiments

In this section, we test the performance of our nnRNN on various sequential processing tasks. We have two goals:

(1) Establish the nnRNN's ability to perform as well as orthogonal RNNs on tasks with pathologically long-term dependencies: the copy task and the permuted sequential MNIST task.

(2) Demonstrate improved performance over orthogonal RNNs on a more realistic task requiring ongoing computation and output: the Penn Treebank character-level benchmark.

We compare our nnRNN model to the following architectures: vanilla RNN (RNN), the orthogonally initialized RNN (RNN-orth) [28], the Efficient Unitary RNN (EURNN) [55], and the Exponential RNN (expRNN) [48]. Our goal is to establish performance for non-gated models, but we include LSTM [33] for reference. For comparison, models are separately matched in the number of hidden units and number of parameters. Every training run was tuned with a thorough optimization hyperparameter search. Model training and task setup are detailed in SM§5.7.1.

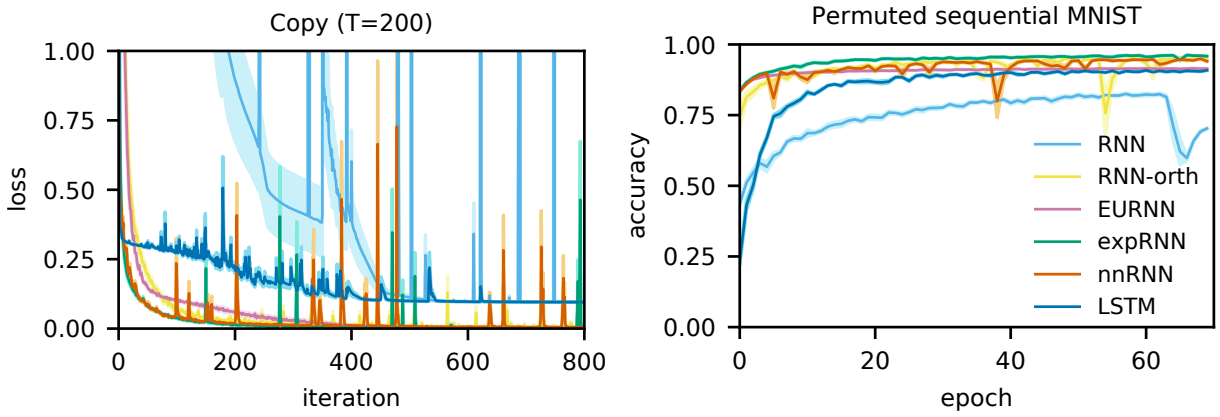


Figure 5.2. Cross entropy of each model on copy task (left) and permuted sMNIST (right)

5.5.1. Copy task & permuted sequential MNIST

The copy task, introduced in [33], requires that a model reads a sequence of inputs, waits for some delay T (here we use $T = 200$), and then outputs the same sequence. Fig. 5.2 shows the cross entropy of each tested with $N = 128$ hidden units. Holding the number

N of hidden units constant, model performance is plotted for the copy task (T=200, left; cross-entropy loss; $N \sim 128$) and for the permuted sequential MNIST task (right; accuracy; $N \sim 512$). Shading indicates one standard error of the mean. We see little difference if we match the number of parameters with $\sim$ 18.9K (see Fig. 5.4 in SM§5.7.1.2). For reference, a model that simply predicts a constant set of output tokens for every input sequence is expected to achieve a *baseline* loss of 0.095. As shown in [28], an orthogonal RNN is an optimal solution for the copy task. Indeed, the LSTM struggled to solve the task and RNN failed completely, unlike all orthogonal RNNs who learn to solve at very high performance very quickly. The proposed nnRNN matched the performance of orthogonal RNNs, as well as best training timescales.

Sequential MNIST [45] requires a model to classify an MNIST digit after reading the digit image one pixel at a time. The pixels are permuted in order to increase the time delay between inter-dependent pixels, making the task harder. Fig. 5.2 shows mean validation accuracy of each tested model with with $N = 512$ hidden units (see Fig. 5.4 in SM§5.7.1.2 for parameter match). As with the copy task, the nnRNN matches orthogonal RNNs in performance, whereas RNN and LSTM show lesser performances.

5.5.2. Penn Treebank (PTB) character-level prediction

Character level language modelling with the Penn Treebank corpus (PTB) [54] consists of predicting the next character at each character in a sequence of text (see SM§5.7.1.3 for test accuracy). We compare the performance of different models on this task in Table 5.1 in terms of test mean bits per character (BPC), where lower BPC indicates better performance. We compare truncated backpropagation through time over 150 time steps and over 300 time steps. The table shows two comparisons across models. Left a fixed number of parameters, and right fixed number of hidden units. Error range indicates standard error of the mean.

	Test Bit per Character (BPC)			
	Fixed # params ($\sim$1.32M)		Fixed # hidden units ($N = 1024$)	
Model	$T_{PTB} = 150$	$T_{PTB} = 300$	$T_{PTB} = 150$	$T_{PTB} = 300$
RNN	2.89 $\pm$ 0.002	2.90 $\pm$ 0.0016	2.89 $\pm$ 0.002	2.90 $\pm$ 0.002
RNN-orth	1.62 $\pm$ 0.004	1.66 $\pm$ 0.006	1.62 $\pm$ 0.004	1.66 $\pm$ 0.006
EURNN	1.61 $\pm$ 0.001	1.62 $\pm$ 0.001	1.69 $\pm$ 0.001	1.68 $\pm$ 0.001
expRNN	1.49 $\pm$ 0.008	1.52 $\pm$ 0.001	1.51 $\pm$ 0.005	1.55 $\pm$ 0.001
nnRNN	**1.47 $\pm$ 0.003**	**1.49 $\pm$ 0.002**	**1.47 $\pm$ 0.003**	**1.49 $\pm$ 0.002**

Table 5.1. PTB test performance bit per character (BPC) for sequence lengths $T_{PTB} = 150, 300$

In contrast to the copy and psMNIST tasks (see §5.5.1), the PTB task requires online computation across several inputs received in the past. Furthermore, it is a task that demands an output from the network at each time step, as opposed to a prompted one. These ingredients are not particularly well-suited for orthogonal transformations since it is not

enough to simply keep inputs in memory or integrate input paths to a classification outcome, the network must transform past inputs to compute a probability distribution. Gated networks are well-suited for such tasks, and we could get an LSTM with $N = 1024$ hidden units to achieve 1.37 ± 0.003 BPC (see §5.6 for a discussion).

Importantly, without the use of gating mechanisms, our nnRNN outperformed all other models we tested. To our knowledge, it also surpasses all reported performances for other non-gated models of comparable size (see also [49]). While the performance gap to expRNN (the state-of-the-art orthogonal RNN) is modest for equal number of parameters and shorter time scale ($T_{PTB} = 150$), it significantly improves for $T_{PTB} = 300$. Where the nnRNN shines is for equal numbers of hidden units, where the performance gap to expRNN is much greater. This suggests two things: (i) the nnRNN improves propagation of *meaningful* signals over longer time scales, and (ii) its connectivity structure provides superior expressivity for a fixed number of neurons, a desirable feature for efficient model deployment. In the next section, we explore the structure of trained nnRNN weights to illustrate that the mechanisms responsible for this performance gain are consistent with the arguments presented in §5.3.

5.5.3. Analysis of learned connectivity structure

To validate the theoretical arguments in favor of non-normal dynamics presented in §5.3, we take a look at the connectivity structure that emerges from our training procedure (see §5.4). Fig. 5.3 (and 5.5 in SM§5.7.5) shows the triangular Schur form $\Theta = \Lambda + T$ of the recurrent connectivity matrix $V = P\Theta P^\top$, at the end of training. Fig. 5.3 shows the elements of learned Θ matrix entries for copy task in (a) are concentrated on the diagonal, and distributed in the lower triangle for the PTB task in (b). Insets in (a) and (b) show the distribution of eigenvalues angles θ_i (cf. Eq. 5.4). (c) The mean magnitude of entries along the kth sub-diagonal of the lower triangle in (b) shows both a delay-line and lower triangle component. Inset: the distribution of entry magnitudes along the delay line is bimodal from its two contributions: the cosine of uniformily distributed angles, and the relatively small, but significant pure delay line entries.

For the copy task, Θ is practically composed of 2×2 rotation blocks along its diagonal (i.e. $T = 0$). This indicates that the learned dynamics are normal, and orthogonal. In contrast, for the PTB task we find that the lower triangular part T shows a lot of structure, indicating that non-normal transient dynamics are used to solve the prediction task. The distributions of elements of T away from the diagonal highlights the nature of the tasks. The network distributes the angles roughly uniformly in the case of the copy task, consistent with the explicit optimal solution that involves such a distribution of rotations [28]. For the PTB task however, the angles strongly align, promoting the delay-line motif in Θ, shown in §5.3 to be optimal for the information propagation useful for character prediction. This is

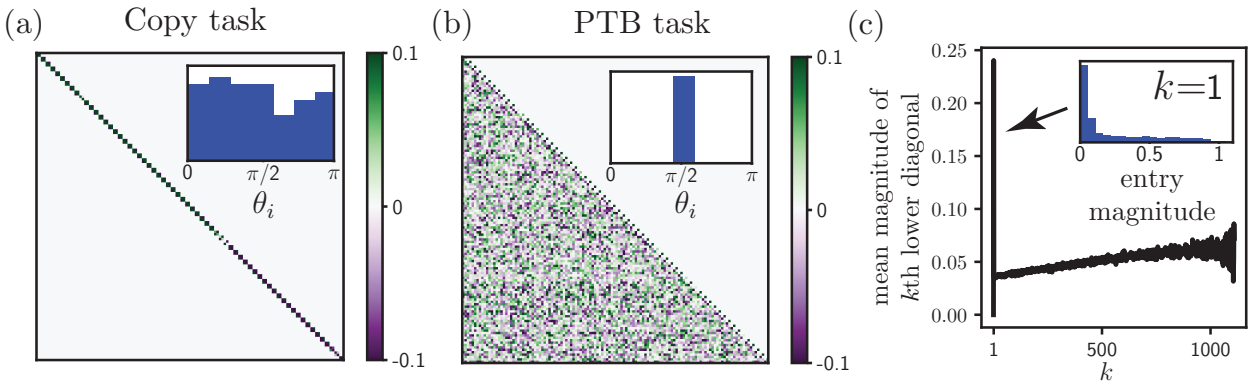


Figure 5.3. Learned Θs show decomposition into Λ and T

more clearly demonstrated by the mean absolute value of entries away from the diagonal, shown in Fig. 5.3. The rest of the triangle also shows structure, consistent with our proof that the lower triangle and delay line can jointly contribute to information propagation.

In summary, these findings indicate that when tasks are well-suited for isometric transformations (e.g. storing things in memory for later recall) the nnRNN easily learns to eliminate non-normal dynamics and restricts itself to the set of orthogonal matrices. Moreover, it does so without any penalty on learning speed, as shown in Fig. 5.2. However, when tasks require online computations, non-normal dynamics come into play and enable transient activity to be shaped for computations.

Lastly, as already discussed in §5.3.3, the expressivity afforded by non-normality must come with a trade-off between maintaining the eigen and singular spectra "close" to the unit circle, balancing control over exponential vanishing and polynomial exploding gradients respectively. This fact remains true for any parametrization of non-normal matrices, including the SVD used in spectral RNN [83]. The nnRNN is naturally suited to target this balance by explicitly allowing regularization over normal and non-normal parts of a matrix, and enabling the optimizer to find that trade-off. This explains why we find that allowing eigenvalues to deviate slightly from the unit circle throughout training (regularization on γ), along with weight decay for the non-normal part, yields the best results with most stable training. Further evidence of this balancing mechanism is found in trained matrices (see Fig. 5.3). For the PTB task, non-normal structure emerges and the mean eigenvalue norm is balanced at $\bar{\gamma} \sim 0.958$. In contrast for the copy task, matrices remain normal and $\bar{\gamma} \sim 1$. See SM§5.7.6 for additional experiments with fixed γ further outlining their role in this trade-off.

5.6. Discussion

With the nnRNN, we showed that augmenting orthogonal recurrent connectivity matrices with non-normal terms increases the flexibility of a recurrent network. We compared the nnRNN's performance to several other recurrent models on distinct tasks; some that are

well-suited for orthogonal RNNs, and another that targets their limitations. We find that the non-normal structure affords two distinct improvements for nnRNNs:

(1) *Preservation of advantages from purely orthogonal RNNs* (long-term gradient propagation; fast learning on tasks involving long-term memory),

(2) *Compared to orthogonal RNNs, increased expressivity on tasks requiring online computations thanks to transient dynamics.*

To better understand why this is, we derived analytical expressions that outline the role of non-normal dynamics that were corroborated by an analysis of nnRNN connectivity structure after training. Importantly, the nnRNN leverages existing optimization algorithms for orthogonal matrices with increased scope, all the while retaining learning speed.

The principal contribution of this paper is not to report major gains in performance as measured by tests, but rather to convincingly outline a promising novel direction for spectrally constrained RNNs. This spans the expressivity and ability to handle long-term dependencies of orthogonal RNNs on one hand, and completely unconstrained RNNs on the other. The nnRNN is a first step toward a trainable RNN parametrization where regularization over the eigenspectrum is readily available while conserving the flexibility of arbitrary eigenbases. This allows explicit control over quantities with direct impact on gradient propagation and expressivity, providing a promising RNN toolbox. Unlike the orthogonal RNNs present in our tests, which have benefited over the years from a series of algorithmic improvements, our nnRNN is basic in its implementation, and presents a number of areas for direct improvement. These include (i) using a complex-valued parametrization as in [2], (ii) exploring better initializations, and (iii) identifying helpful regularization schemes for the non-normal part. Beyond these, we should mention that the Schur decomposition presents implicit instabilities which can jeopardize training when eigenbases become degenerate (see SM§5.7.4). Simple perturbation schemes to prevent this should greatly improve performance.

Finally, we acknowledge that on a number of time-dependent tasks, gated recurrent networks such as the LSTM or the GRU [36] have clear advantages (see also [78] for a derivation of gated dynamics from first principles). Building on these, there is promising evidence that combining orthogonal connectivity with gates can greatly help learning [36]. This further motivates the development of spectrally constrained recurrent architectures to be combined with gating, thereby optimizing the efficiency of gradient propagation and expressivity with both explicit mechanisms, and implicit structure. Ongoing work in this direction is under way, leveraging our nnRNN findings.

Acknowledgements

We would like to thank Tim Cooijmans, Sarath Chandar, Jonathan Binas, Anirudh Goyal, César Laurent and Tianyu Li for useful discussions. YB ackowledges support from CIFAR, Microsoft and NSERC. MPT acknowledges IVADO support. GL is funded by an

NSERC Discovery Grant (RGPIN-2018-04821), an FRQNT Young Investigator Startup Program (2019-NC-253251), and an FRQS Research Scholar Award, Junior 1 (LAJGU0401-253188).

5.7. Appendix

5.7.1. Task setup and training details

Please refer to `https://github.com/nnRNN/nnRNN_release` for available code and ongoing work.

5.7.1.1. *Copy task*

For the copy task, networks are presented with an input sequence x_t of length $10 + T_c$. For $t = 1, \ldots, 10$, x_t can take one of 8 distinct values $\{a_i\}_{i=1}^{8}$. For the following $T_c - 1$ time steps, x_t takes the same value a_9. At $t = T_c$, a cue symbol $x_t = a_{10}$ prompts the model to recall the first 10 symbols and output them sequentially in the same order they were presented. Models are trained to minimize the average cross entropy loss of symbol recalls. A model that simply predicts a constant set of output tokens for every input sequence would achieve a *baseline* loss of $\frac{10\log(8)}{T+20}$. All models were trained using a mini batch size of 10. All non-gated models except "RNN" were initialized such that the recurrent network was orthogonal. The non-normal RNN had it's orthogonal weight matrix initialized as in expRNN with the log weights initialized using Henaff intialization. Importantly, all non-gated models used the *modReLU* activation function for state-to-state transitions. This is critical for the copy task since a nonlinearity makes the task very difficult to solve [80] and *modReLU* acts as identity at initialization. There were 6 evaluation runs for each model. Fig. 5.4 (left) shows cross entropy loss for all models throughout training when the number of parameters is held constant. Model and training hyperparameters are summarized in Table 5.2. Here, "hid" is hidden state size, "LR" is learning rate, "LR orth" is the learning rate of the orthogonal transition matrix (its skew symmetric matrix), α is the smoothing parameter of RMSprop, δ is the weight of L2 penalty applied to the rotation blocks' moduli γ_i defined in equation 5.5, T decay is the weight of the L2 penalty applied on T in equation 5.5, and "V init" is the initialization scheme for the state transition matrix.

Model	hid	LR	LR orth	α	δ	T decay	V init
nnRNN	128	0.0005	10^{-6}	0.99	0.0001	10^{-6}	Henaff
expRNN	128	0.001	0.0001	0.99			Henaff
expRNN	176	0.001	0.0001	0.99			Henaff
LSTM	128	0.0005		0.99			Glorot Normal
LSTM	63	0.001		0.99			Glorot Normal
RNN Orth	128	0.0002		0.99			Random orth
EURNN	128	0.001		0.5			
EURNN	256	0.001		0.5			
RNN	128	0.001		0.9			Glorot Normal

Table 5.2. Hyperparameters for the copy task

5.7.1.2. *Sequential MNIST classification task*

The sequential MNIST task [45] measures the ability of an RNN to model complex long-term dependencies. In this task, each pixel is fed into the network one at a time, after which the network must classify the digit. Permutation increases the difficulty of the problem by applying a fixed permutation to the sequence of the pixels, which creates longer term dependencies between the pixels. We train this task for all networks using mini batch sizes of 128. All non-gated networks except "RNN" were initialized with orthogonal recurrent weight matrices using Cayley initialization[27]. The non-normal RNN has its orthogonal weight matrix initialized as in [48] with the log weights initialized using Cayley initialization. Fig. 5.4 (right) shows validation accuracy for all models throughout training when the number of parameters is held constant. There were 3 evaluation runs for each model. Model and training hyperparameters are summarized in Table 5.3. Here, "hid" is hidden state size, "LR" is learning rate, "LR orth" is the learning rate of the orthogonal transition matrix (its skew symmetric matrix), α is the smoothing parameter of RMSprop, δ is the weight of L2 penalty applied to the rotation blocks' moduli γ_i defined in equation 5.5, T decay is the weight of the L2 penalty applied on T in equation 5.5, and "V init" is the initialization scheme for the state transition matrix.

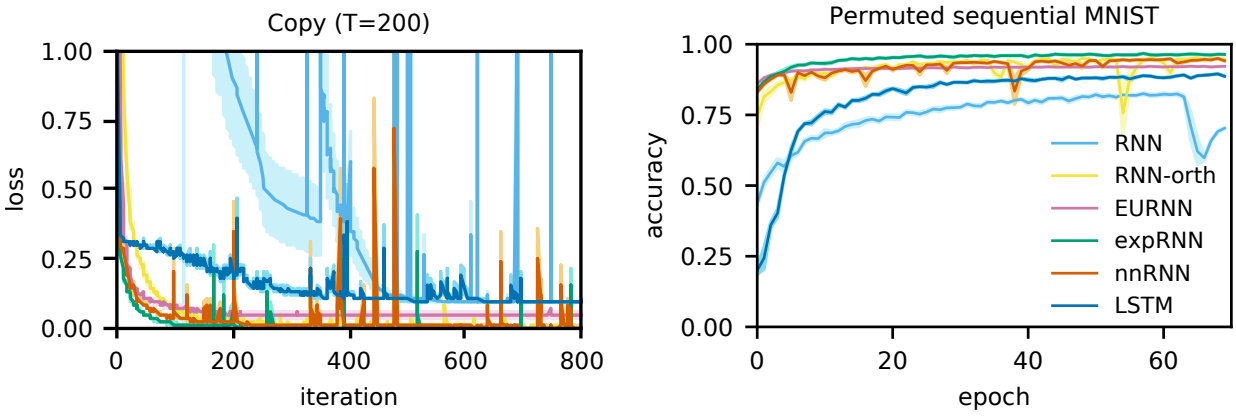


Figure 5.4. Model performance on copy task (left) and permuted sequential MNIST (right) with same number of trainable parameters

5.7.1.3. *Penn Treebank character prediction task*

The Penn Treebank character prediction task is that of predicting the next character in a text corpus at every character position, given all previous text. We trained all models sequentially on the entire corpus, splitting it into sequences of length 150 or 300 for truncated backpropagation through time. Consequently, the initial hidden state for a sequence is the last hidden state produced from its preceding sequence. All models were trained for 100 epochs with a mini batch size of 128. Following training, for each model,

Model	hid	LR	LR orth	α	δ	T decay	V init
nnRNN	512	0.0002	$2 * 10^{-5}$	0.99	0.1	0.0001	Cayley
expRNN	512	0.0005	$5 * 10^{-5}$	0.99			Cayley
expRNN	722	0.0005	$5 * 10^{-5}$	0.99			Cayley
LSTM	512	0.0005		0.9			Glorot Normal
LSTM	257	0.0005		0.9			Glorot Normal
RNN Orth	512	$5 * 10^{-5}$		0.99			Random orth
EURNN	512	0.0001		0.9			
EURNN	1024	0.0001		0.9			
RNN	512	0.0001		0.9			Glorot Normal

Table 5.3. Hyperparameters for the permuted sequential MNIST task

the state which yielded the best performance on the validation data was evaluated on the test data. Table 2 reports the same performance for the same model states as in Table 1 in the main text but presents test accuracy instead of BPC. There were 5 evaluation runs for each model. Model and training hyperparameters are summarized in Table 5.5. Here, "hid" is hidden state size, "LR" is learning rate, "LR orth" is the learning rate of the orthogonal transition matrix (its skew symmetric matrix), α is the smoothing parameter of RMSprop, δ is the weight of L2 penalty applied to the rotation blocks' moduli γ_i defined in equation 5.5, T decay is the weight of the L2 penalty applied on T in equation 5.5, and "V init" is the initialization scheme for the state transition matrix.

	Test Accuracy			
	Fixed # params (∼1.32M)		Fixed # hidden units ($N = 1024$)	
Model	$T_{PTB} = 150$	$T_{PTB} = 300$	$T_{PTB} = 150$	$T_{PTB} = 300$
RNN	40.01 ± 0.026	39.97 ± 0.025	40.01 ± 0.026	39.97 ± 0.025
RNN-orth	66.29 ± 0.07	65.53 ± 0.09	66.29 ± 0.07	65.53 ± 0.09
EURNN	65.68 ± 0.002	65.55 ± 0.002	64.01 ± 0.002	64.20 ± 0.003
expRNN	68.07 ± 0.15	67.58 ± 0.04	67.51 ± 0.11	66.89 ± 0.024
nnRNN	**68.78 ± 0.0006**	**68.52 ± 0.0004**	**68.78 ± 0.0006**	**68.52 ± 0.0004**

Table 5.4. PTB test performance: Test Accuracy

5.7.1.4. *Hyperparameter search*

For all models with a state transition matrix that is initialized as orthogonal (nnRNN, expRNN, RNN-orth), three orthogonal initialization schemes were tested: (1) random, (2) Cayley, and (3) Henaff. Random initialization is achieved by sampling a random matrix whose QR decomposition yields an orthogonal matrix with positive determinant 1 and then mapping this orthogonal matrix via a matrix logarithm to the skew symmetric parameter matrix used in expRNN. Cayley and Henaff initializations initialize this skew symmetric matrix as described in [48]. The vanilla RNN is also tested with a Glorot Normal initialization, with the model then referred to as simply "RNN".

Model	hid	LR	LR orth	α	δ	T decay	V init
Length 150							
nnRNN	1024	0.0008	$8 * 10^{-5}$	0.9	1	0.0001	Cayley
expRNN	1024	0.005	0.0001	0.9			Cayley
expRNN	1386	0.005	0.0001	0.9			Cayley
LSTM	1024	0.008		0.9			Glorot Normal
LSTM	475	0.001		0.99			Glorot Normal
RNN Orth	1024	0.0001		0.9			Random orth
EURNN	1024	0.001		0.9			
EURNN	2048	0.001		0.9			
RNN	1024	10^{-5}		0.9			Glorot Normal
Length 300							
nnRNN	1024	0.0008	$6 * 10^{-5}$	0.9	0.0001	0.0001	Cayley
expRNN	1024	0.005	0.0001	0.9			Cayley
expRNN	1386	0.005	0.0001	0.9			Cayley
LSTM	1024	0.008		0.9			Glorot Normal
LSTM	475	0.003		0.9			Glorot Normal
RNN Orth	1024	0.0001		0.9			Cayley
EURNN	1024	0.001		0.9			
EURNN	2048	0.001		0.9			
RNN	1024	$1 * 10^{-5}$		0.9			Glorot Normal

Table 5.5. Hyperparameters for the Penn Treebank task (at 150 and 300 time step truncation for gradient backpropagation)

For training, learning rates were searched between 0.005 and 10^{-5} in increments of 0.0001 and 0.0002 or either $5\times$ or $10\times$; the learning rate for the orthogonal matrix tested between 10^{-5} and 10^{-4}; and RMSprop was used as the optimizer with smoothing parameter α as 0.5, 0.9, or 0.99. In Equation 5.5, δ was searched in 0, 0.0001, 0.001, 0.01, 0.1, 0.15, 1.0, 10; the L2 decay on the strictly lower triangular part of the transition matrix T was searched in 0, 10^{-6}, 10^{-5}, 10^{-4}.

5.7.2. Fisher memory curves for strictly lower triangular matrices

Let, Θ be a strictly lower triangular matrix such that $[\Theta]_{i+1,i} = \sqrt{\alpha}$ for $1 \leq i \leq N-1$ and A be the associated lower triangular Gram-Schmidt orthogonalization matrix. We have that,

$$\Theta = DA \tag{5.6}$$

where D is the delay line, $D_{i+1,i} = \sqrt{\alpha}$ and $A_{i,i} = 1$ for $1 \leq i \leq N$. Let us recall the expression of $J(k)$ for independent Gaussian noise derived by [20, Eq. 3],

$$J(k) = U^T(\Theta^k)^\top C_n^{-1}\Theta^k U\,, \quad \text{where} \quad C_n = \epsilon \sum_{k=0}^{\infty} \Theta^k(\Theta^k)^\top\,, \tag{5.7}$$

and $U = [1,0,\ldots,0]$ is the source. We have that for any vector u,

$$u^\top C_n u = \epsilon \sum_{k=0}^{\infty} ((D^k)^\top u)^\top A A^T ((D^k)^\top u) \tag{5.8}$$

$$= \epsilon \sum_{k=0}^{N-1} ((D^k)^\top u)^\top A A^T ((D^k)^\top u) \tag{5.9}$$

$$\leq \epsilon \sigma_{\max}^{2(N-1)}(A) \sum_{k=0}^{N-1} u^\top D^k (D^k)^\top u \tag{5.10}$$

where for the first equality we used the fact that Θ is nilpotent and for the last inequality the fact that $\sigma_{\max}(A) \geq 1$. Recall that for two symmetric matrices we define: $A \succeq B$ if and only if $B - A$ is positive semidefinite. By definition we have,

$$C_n \preceq \epsilon \sigma_{\max}^{2(N-1)}(A) \sum_{k=0}^{\infty} D^k (D^k)^\top = \epsilon \sigma_{\max}^2(A) \left(\mathrm{diag}(1, \tfrac{1-\alpha^2}{1-\alpha}, \ldots, \tfrac{1-\alpha^N}{1-\alpha}) \right) \tag{5.11}$$

where the last equality is due to $[D^k (D^k)^\top]_{i,j} = \alpha^k$ if $i = j \geq k+1$ and 0 otherwise. Thus using [44, Theorem 2 P. 146] we can take the inverse to get

$$C_n^{-1} \succeq \frac{1}{\epsilon \sigma_{\max}^{2(N-1)}(A)} \left(\mathrm{diag}(1, \tfrac{1-\alpha^2}{1-\alpha}, \ldots, \tfrac{1-\alpha^N}{1-\alpha}) \right)^{-1} = \frac{1}{\epsilon \sigma_{\max}^{2(N-1)}(A)} \mathrm{diag}(1, \tfrac{1-\alpha}{1-\alpha^2}, \ldots, \tfrac{1-\alpha}{1-\alpha^N}).$$

Finally, using that $\Theta^k U = [\underbrace{0, \ldots, 0}_{k}, \sqrt{\alpha}^k, *, \ldots, *]$, we have that for $0 \leq k \leq N-1$,

$$J(k) = U^T (\Theta^k)^\top C_n^{-1} \Theta^k U \tag{5.12}$$

$$\geq \frac{1}{\epsilon \sigma_{\max}^{2(N-1)}(A)} \alpha^k \frac{\alpha - 1}{\alpha^{k+1} - 1}\,. \tag{5.13}$$

α	β	d	$J_{\mathrm{tot}} = \sum_{t=0}^{\infty} J(t)$
0.95	0.0	0.0	3.03
1.00	0.0	0.0	5.19
1.05	0.0	0.0	12.1
0.95	0.005	0.0	3.18
1.00	0.005	0.0	5.30
1.05	0.005	0.0	12.1
0.95	0.0	0.2	12.0
1.00	0.0	0.2	16.2
1.05	0.0	0.2	20.5
0.95	0.005	0.2	12.1
1.00	0.005	0.2	16.3
1.05	0.005	0.2	20.4

Table 5.6. Fisher memory curve performance: Shown is the sum of the FMC for the models considered in section 5.3.

5.7.3. Proof of proposition 4

Let us prove this claim by induction on t. The case $t = 1$ is trivial, so let us assume the claim to be true for $t = r$, then for each $k = 1, \dots n-1$, we expand A^{r+1} as $A^r \cdot A$ and get

$$p_k^{(r+1)}(x) = x \cdot \left[1 + \sum_{s=1}^{k-1} p_s^{(r)}(x)\right] + p_k^{(r)}(x)$$

which again is a polynomial of degree at most k, where the coefficient of x^0 in $p_k^{(r+1)}$ is zero and the coefficient of x^l in $p_k^{(r+1)}$ is $O\left(\binom{r}{l-1}\right) + O\left(\binom{r}{l}\right) = O\left(\binom{r+1}{l}\right)$ for all $l = 1,2,\dots,k$, concluding the induction.

5.7.4. Numerical instabilities of the Schur decomposition

The Schur decomposition is computed via multiple iterations of the QR algorithm. The QR algorithm is known to be *backward stable*, which gives accurate answers as long as the eigenvalues of the matrix at hand are well-conditioned, as is explained in [1].

Eigenvalue sensitivity is measured by the angle formed between the left and right eigenvectors of the same eigenvalues. Normal matrices have coinciding left and right eigenvectors but non-normal matrices do not, and thus certain non-normal matrices such as the Grcar matrix have very high eigenvalue sensitivity, and thus gives rise to inaccuracies in the Schur decomposition.

This motivates training the connectivity matrix in the Schur decomposition directly instead of applying the Schur decomposition in a separate step.

5.7.5. Learned connectivity structure on psMNIST

For completeness, let us take a look at the Schur matrix after training on psMNIST in Fig. 5.5. We can see that the distribution of learned angles in the rotation blocks is rather flat, and thus is very different from the distribution learned in the PTB task, as can be seen in Fig. 5.3. The flatness in distribution comes somewhat close to the flatness of the learned angle distribution in the copy task. In other words, the angle distribution in the PTB task is highly structured, while in the copy task and psMNIST task, it seems to be close to uniform.

Furthermore, we can also observe that the connectivity structure learned in the lower triangle is significantly weaker in the psMNIST task than in the PTB task, while not being completely absent as in the copy task.

Thus it seems that we can spot a spectrum of connectivity structure:

- The copy task has no connectivity structure in the lower triangle, close to uniform angle distribution and the absence of a delay line, on the one end.
- The PTB task has a lot of connectivity structure in the lower triangle, a very narrow angle distribution and the presence of a delay line, on the other end.

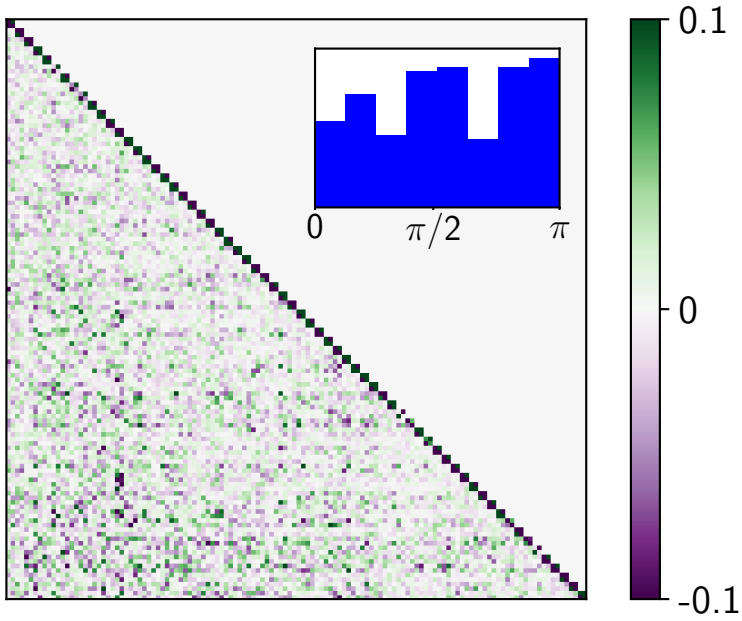


Figure 5.5. *Learned* Θ *on psMNIST task.* Inset: angles θ_i distribution of block diagonal rotations. (cf. Eq.5.4).

For the psMNIST task, it appears that we are located somewhere in the middle of that spectrum.

5.7.6. Gradient propagation analysis

In the results from the main text, the parameter γ (scaling factor for eigenvalues) is allowed to be changed by the optimizer, but is heavily regularized to force it to stay close to one. As argued, a mean value of $\bar{\gamma} \sim 0.958$ found for the best solutions on the PTB task is indicative of a trade-off between eigenvalues and singular values to allow stable propagation and good expressivity. To further elucidate the effect, we train nnRNNs with clamped values of γ at 1, and at 0.958.

Results are found in Table 5.4 and complement those of Table 5.1 in the main text ($\sim$1.32M params, $N = 1024$ units). As expected for $\gamma = 1$, some run did not converge (asterisks indicate number out of 5) as the emergence of non-normal structure pushes singular values above one. Despite this, on runs that did converge we found the best performance out of all methods (including regularized γ nnRNN), strongly indicating that non-normality does indeed provide more expressivity. For γ clamped at 0.958 the performance was virtually identical to that of nnRNN with regularized γ, indicating non-normal connectivity learning appears robust and independent of γ learning.

Table 5.7 shows the performance of each model on the PTB task. The bits per character (BPC), for sequence lengths $T_{PTB} = 150, 300$ are both shown. Three version of nnRNN shown: nnRNN (reproduced from main text), γ clamped at 1, γ clamped at 0.958. All

Model	$T_{PTB} = 150$	$T_{PTB} = 300$
nnRNN	1.47 ± 0.003	1.49 ± 0.002
nnRNN-$\gamma = 1$	1.46 ± 0.005*	1.49 ± 0.022**
nnRNN-$\gamma = 0.958$	1.47 ± 0.005	1.49 ± 0.008

Table 5.7. PTB test performance: bits per character (BPC)

models have ∼1.32M parameters and $N = 1024$. Error range indicates standard error of the mean. Asterisks indicate number of failed runs out of 5.

Fig. 5.6 shows example gradient norms for nnRNN on PTB task, with eigenvalues clamped or regularized. The plot on the left shows the gradient magnitude while back-propagating from time step to time step (growing polynomially). The plot on the right shows the gradient magnitude during training. In this example, all runs converged, and we can observe that gradient norms behave nicely during backpropagation and throughout training. This is indicative that although γ plays an important stabilizing role, gradient explosion leading to diverging training runs appears to be an all-or-nothing event.

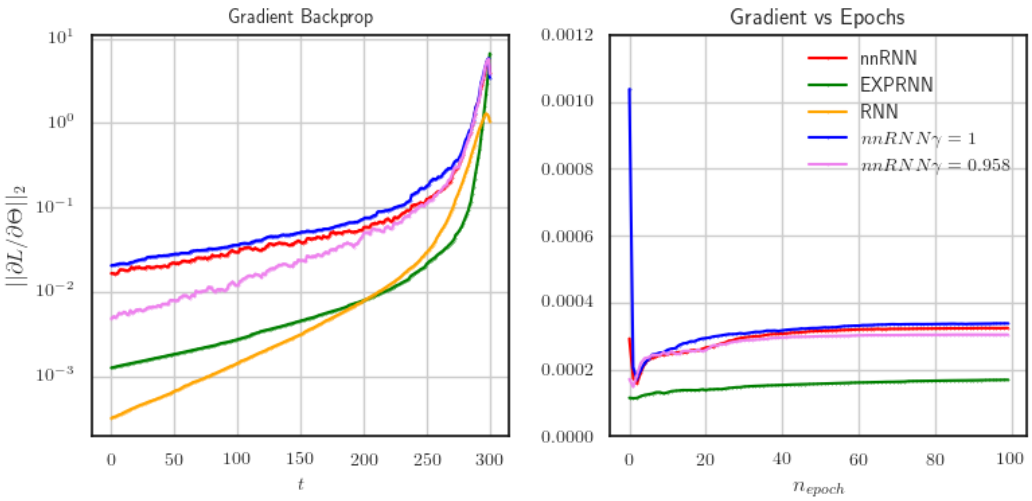


Figure 5.6. Gradient propagation for each model across time steps

Part 3

Concluding Remarks

Chapter 6

Conclusion

This thesis presented several methods addressing the LPSP for double-stack intermodal railcars, as well as a model capable of learning long-term dependencies in non-normal recurrent neural networks, while maintaining expressivity in sequential models. We begin by presenting concluding remarks to Part 1. Here we discuss the objectives of the papers, conclusions from the work performed and future research opportunities. Next, we present concluding remarks about Part 2, discussing the objectives, conclusions and future research opportunities for the nnRNN.

6.1. The Load Planning and Sequencing Problem

The objective of Part 1 of the thesis was to solve the LPSP through dynamic programming. It corresponds to a shortest path problem on an ordered graph and the proposed heuristics are rooted in the well-known A* algorithm. Chapter 2 shows that the LPSP is a challenging problem but it can be solved using carefully constructed heuristics. The definition of a lower bound computed by greedily solving a relaxation of the problem provides a consistent metric with which one can compare the quality of the solutions. The results indicate that greedy depth-first search approaches frequently provide high-quality solutions, and that they can however be prone to finding poor solutions, by for example leaving slots empty. The combination of beam search and depth-first search was found to be best on average over all proposed heuristics, for all distance functions considered. The difference in initial container storage area layouts between the medium size, and other sizes indicates that overall performance is coupled closely with the initial layout of the containers. Moreover, the heuristics are sensitive to the distance costs. Overall, heuristics are capable of providing high-quality solutions to the LPSP, and scaling to practical sizes with minimal degradation of solution quality.

Chapter 3 aimed to explore the LPSP using DRL techniques in an effort to show that presenting the problem to a learning model can lead to high-quality solutions as well. The results indicate that DRL models are capable of learning the LPSP environment through

the use of deep Q-learning with feature representations which can be quickly constructed to represent the environment. The primary focus of the chapter relates to a representation of the problem, including modalities of the storage area, platforms and containers which have been double touched, in the model. The results show that while the model is generally able to learn to perform well, it is prone to failure, leaving some slots open. However, this can be mitigated by using beam search, increasing the set of actions explored, and reducing the number of failed searches. The paper shows that the model is prone to overfitting, as it performed better on the training set than on the test set, which could be mitigated through a more thorough hyperparameter search involving higher weight decay values.

There are several directions for further research which can be followed to handle the LPSP. The most significant opportunities are listed below:

- Improve the heuristic methods per se. For example, beam search could provide a learned tabular action-state value function to depth-first search methods, which would lead to better exploration of the underlying problem.
- Have heuristics consider container weights, and platform flexibility when considering the cost of placing a container in a platform slot.
- Explore the stochastic and dynamic LPSP, wherein containers arrive during the loading procedure. In this context, one could leverage the proposed heuristics as base policies in a rollout algorithm [8].
- Consider a multi-agent problem, wherein there is more than one piece of handling equipment performing loading operations.
- Apply more complex DRL techniques for solving the LPSP. As of now we have only tried deep Q-learning successfully, but the problem lends itself to improvements such as applying techniques to handle the large action space.

6.2. Learning Long-term Dependencies While Increasing Expressivity in Sequential Models

The objective of Part 2 was to improve expressivity of models, while eliminating the EVGP. Chapter 5 indicates that the nnRNN provides two major advantages over conventional RNNs. First, it preserves the advantages of purely orthogonal RNNs that are characterized by stability of long-term gradient propagation and fast learning on tasks involving long-term memory. Second, the nnRNN has improved expressivity over orthogonal RNNs in tasks which require short-term complex interactions.

While the numerical results presented do not show major gains over orthogonal networks, the hope is to identify a new set of architectures capable of handling the EVGP. The paper aims to propose a novel research direction for RNNs with constrained eigenspectrums. These

new models could be capable of improving the RNN architecture's ability to learn long-term sequences while maintaining the ability to model short-term interactions between states.

We consider several directions for further research:

- Using a complex Schur decomposition for the recurrent weight matrix which would no longer impose that eigenvalues come in complex conjugate pairs.
- Pairing the nnRNN parametrization with gated models like the LSTM and GRU.
- Improving regularization schemes for the non-normal part of the nnRNN recurrent weight matrix.

Bibliography

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J., Dongarra, J. J., Du Croz, J., Hammarling, S., Greenbaum, A., McKenney, A. and Sorensen, D. [1999]. *LAPACK Users' Guide (Third Ed.)*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[2] Arjovsky, M., Shah, A. and Bengio, Y. [2016]. Unitary evolution recurrent neural networks, *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, JMLR.org, pp. 1120–1128.
URL: *http://dl.acm.org/citation.cfm?id=3045390.3045509*

[3] Arpit, D., Kanuparthi, B., Kerg, G., Ke, N. R., Mitliagkas, I. and Bengio, Y. [2019]. h-detach: Modifying the LSTM Gradient Towards Better Optimization, *ICLR* .

[4] Association of American Railroads [2018]. Rail intermodal keeps america moving.
URL: *https://www.aar.org/wp-content/uploads/2018/07/AAR-Rail-Intermodal.pdf*

[5] Barto, A. G., Bradtke, S. J. and Singh, S. P. [1995]. Learning to act using real-time dynamic programming, *Artificial intelligence* **72**(1-2): 81–138.

[6] Bengio, Y., Simard, P. and Frasconi, P. [1994]. Learning long-term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks* **5**(2): 157–166.

[7] Bertsekas, D. P. [1995]. *Dynamic programming and optimal control*, Vol. 1, Athena scientific Belmont, MA.

[8] Bertsekas, D. P. [2019]. *Reinforcement learning and optimal control*, Athena scientific Belmont, MA.

[9] Bertsekas, D. P. and Tsitsiklis, J. [1996]. *Neuro-dynamic programming*, Athena scientific Belmont, MA.

[10] Bian, Z., Shao, Q. and Jin, Z. [2016]. Optimization on the container loading sequence based on hybrid dynamic programming, *Transport* **31**(4): 440–449.

[11] Bonet, B. and Geffner, H. [2006]. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to mdps, *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'06, AAAI Press, pp. 142–151.
URL: *http://dl.acm.org/citation.cfm?id=3037104.3037124*

[12] Carlo, H. J., Vis, I. F. and Roodbergen, K. J. [2014a]. Storage yard operations in container terminals: Literature overview, trends, and research directions, *European journal of operational research* **235**(2): 412–430.

[13] Carlo, H. J., Vis, I. F. and Roodbergen, K. J. [2014b]. Transport operations in container terminals: Literature overview, trends, research directions and classification scheme, *European journal of operational research* **236**(1): 1–13.

[14] Chandar, S., Sankar, C., Vorontsov, E., Kahou, S. E. and Bengio, Y. [2019]. Towards Non-saturating Recurrent Units for Modelling Long-term Dependencies, *AAAI* .

[15] Chang, B., Chen, M., Haber, E. and Chi, E. H. [2019]. AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks, *ICLR* .

[16] Chen, M., Pennington, J. and Schoenholz, S. S. [2018]. Dynamical Isometry and a Mean Field Theory of RNNs: Gating Enables Signal Propagation in Recurrent Neural Networks, *ICML* .

[17] Della Croce, F., Ghirardi, M. and Tadei, R. [2004]. Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems, *Journal of Heuristics* **10**(1): 89–104.

[18] Duan, Y., Chen, X., Houthooft, R., Schulman, J. and Abbeel, P. [2016]. Benchmarking deep reinforcement learning for continuous control, *International Conference on Machine Learning*, pp. 1329–1338.

[19] Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T. and Coppin, B. [2015]. Deep reinforcement learning in large discrete action spaces.

[20] Ganguli, S., Huh, D. and Sompolinsky, H. [2008]. Memory traces in dynamical systems, *Proceedings of the National Academy of Sciences of the United States of America* **105**(48): 18970–18975.

[21] Goldman, M. S. [2009]. Memory without Feedback in a Neural Network, *Neuron* **61**(4): 621–634.

[22] Graves, A. [2013]. Generating sequences with recurrent neural networks, *arXiv preprint arXiv:1308.0850* .

[23] Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E. and Levine, S. [2016]. Q-prop: Sample-efficient policy gradient with an off-policy critic, *arXiv preprint arXiv:1611.02247* .

[24] Guo, X., Singh, S., Lee, H., Lewis, R. L. and Wang, X. [2014]. Deep learning for real-time atari game play using offline monte-carlo tree search planning, *Advances in neural information processing systems*, pp. 3338–3346.

[25] Hart, P. E., Nilsson, N. J. and Raphael, B. [1968]. A formal basis for the heuristic determination of minimum cost paths, *IEEE transactions on Systems Science and Cybernetics* **4**(2): 100–107.

[26] Hecht-Nielsen, R. [1992]. Theory of the backpropagation neural network, *Neural networks for perception*, Elsevier, pp. 65–93.

[27] Helfrich, K., Willmott, D. and Ye, Q. [2018]. Orthogonal Recurrent Neural Networks with Scaled Cayley Transform, *ICML* .

[28] Henaff, M., Szlam, A. and LeCun, Y. [2016]. Recurrent orthogonal networks and long-memory tasks, *in* M. F. Balcan and K. Q. Weinberger (eds), *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, pp. 2034–2042.
URL: *http://proceedings.mlr.press/v48/henaff16.html*

[29] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D. [2017]. Deep reinforcement learning that matters.

[30] Hennequin, G., Vogels, T. P. and Gerstner, W. [2012]. Non-normal amplification in random balanced neuronal networks, *Phys. Rev. E* **86**: 011909.
URL: *https://link.aps.org/doi/10.1103/PhysRevE.86.011909*

[31] Hirashima, Y. [2009a]. A q-learning system for container marshalling with group-based learning model at container yard terminals, *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 (IMECS 2009)*, Vol. 1.

[32] Hirashima, Y. [2009b]. A q-learning system for group-based container marshalling with a-priori knowledge for ship loading, *ICCAS-SICE, 2009*, IEEE, pp. 1728–1733.

[33] Hochreiter, S. and Schmidhuber, J. [1997]. Long short-term memory, *Neural Comput.* **9**(8): 1735–1780.
URL: *http://dx.doi.org/10.1162/neco.1997.9.8.1735*

[34] Horn, R. A. and Johnson, C. R. [2012]. *Matrix analysis*, Cambridge university press.

[35] Jiang, B. and Dai, Y.-H. [2015]. A framework of constraint preserving update schemes for optimization on Stiefel manifold, *Mathematical Programming* **153**(2): 1–41.

[36] Jing, L., Gulcehre, C., Peurifoy, J., Shen, Y., Neural, M. T. and 2019 [2019]. Gated orthogonal recurrent units: On learning to forget, *Neural Computations* .

[37] Jing, L., Shen, Y., Peurifoy, J., Skirlo, S., LeCun, Y. and Tegmark, M. [2017]. Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs, *ICML* .

[38] Kim, K. H., Kang, J. S. and Ryu, K. R. [2004]. A beam search algorithm for the load sequencing of outbound containers in port container terminals, *OR spectrum* **26**(1): 93–116.

[39] Korf, R. E. [1985]. Depth-first iterative-deepening: An optimal admissible tree search, *Artificial intelligence* **27**(1): 97–109.

[40] Korf, R. E. [1990]. Real-time heuristic search, *Artificial intelligence* **42**(2-3): 189–211.

[41] Krizhevsky, A., Sutskever, I. and Hinton, G. E. [2012]. Imagenet classification with deep convolutional neural networks, *Advances in neural information processing systems*, pp. 1097–1105.

[42] Lai, M. [2015]. Giraffe: Using deep reinforcement learning to play chess, *CoRR* **abs/1509.01549**.
URL: *http://arxiv.org/abs/1509.01549*

[43] Laterre, A., Fu, Y., Jabri, M. K., Cohen, A.-S., Kas, D., Hajjar, K., Dahl, T. S., Kerkeni, A. and Beguir, K. [2018]. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization.

[44] Lax, P. D. [2007]. *Linear Algebra and Its Applications*, John Wiley & Sons.

[45] Le, Q. V., Jaitly, N. and Hinton, G. E. [2015]. A simple way to initialize recurrent networks of rectified linear units, *arXiv* .

[46] L'Écuyer, P. [2020]. Modèles déterministes et plus court chemin. Lecture notes, IFT 6521, DIRO, Université de Montréal.
URL: *http://www.iro.umontreal.ca/ lecuyer/ift6521/deterministe1.pdf*

[47] Lee, S. Y., Sungik, C. and Chung, S.-Y. [2019]. Sample-efficient deep reinforcement learning via episodic backward update, *Advances in Neural Information Processing Systems*, pp. 2110–2119.

[48] Lezcano-Casado, M. and Martínez-Rubio, D. [2019]. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group, *ICML* .

[49] Li, S., Li, W., Cook, C., Zhu, C. and Gao, Y. [2018]. Independently recurrent neural network (indrnn): Building a longer and deeper rnn, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[50] Lin, L.-J. [1993]. Reinforcement learning for robots using neural networks, *Technical report*, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

[51] Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J. and Han, J. [2019]. On the variance of the adaptive learning rate and beyond, *arXiv preprint arXiv:1908.03265* .

[52] Maduranga, K. D. G., Helfrich, K. E. and Ye, Q. [2019]. Complex Unitary Recurrent Neural Networks using Scaled Cayley Transform, *AAAI* .

[53] Mantovani, S., Morganti, G., Umang, N., Crainic, T. G., Frejinger, E. and Larsen, E. [2018]. The load planning problem for double-stack intermodal trains, *European Journal of Operational Research* **267**(1): 107 – 119.
URL: *http://www.sciencedirect.com/science/article/pii/S0377221717310275*

[54] Marcus, M. P., Marcinkiewicz, M. A. and Santorini, B. [1993]. Building a large annotated corpus of english: The penn treebank, *Comput. Linguist.* **19**(2): 313–330.
URL: *http://dl.acm.org/citation.cfm?id=972470.972475*

[55] Mhammedi, Z., Hellicar, A., Rahman, A. and Bailey, J. [2017]. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections, *ICML* .

[56] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. [2015]. Human-level

control through deep reinforcement learning, *Nature* **518**(7540): 529.

[57] Negrinho, R., Gormley, M. R. and Gordon, G. J. [2018]. Learning beam search policies via imitation learning.

[58] Orhan, A. E. and Pitkow, X. [2019]. Improved memory in recurrent neural networks with sequential non-normal dynamics, *arXiv.org* .

[59] Parcollet, T., Ravanelli, M., Morchid, M., Linarès, G., Trabelsi, C., De Mori, R. and Bengio, Y. [2019]. Quaternion Recurrent Neural Networks, *ICLR* .

[60] Pascanu, R., Mikolov, T. and Bengio, Y. [2012]. On the difficulty of training recurrent neural networks.

[61] Pascanu, R., Mikolov, T. and Bengio, Y. [2013]. On the difficulty of training recurrent neural networks., *ICML* .

[62] Pennington, J., Schoenholz, S. S. and Ganguli, S. [2017]. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice, *Advances in Neural Information Processing Systems* .

[63] Perrault, W. [2019]. *Load sequencing for double-stack trains*, Master's thesis, Université de Montréal.

[64] Powell, W. B. [2007]. *Approximate Dynamic Programming: Solving the curses of dimensionality*, Vol. 703, John Wiley & Sons.

[65] Raghu, M., Poole, B., Kleinberg, J., Ganguli, S. and Dickstein, J. S. [2017]. On the expressive power of deep neural networks, *ICML*.

[66] Rubin, S. M. and Reddy, R. [1977]. The locus model of search and its use in image interpretation., *IJCAI*, Vol. 2, pp. 590–595.

[67] Ruf, M., Cordeau, J.-F. and Frejinger, E. [2019]. The load planning and load sequencing problem for double-stack intermodal trains. Technical Report (under review after first revision for the European Journal of Operations Research).

[68] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. [1986]. Learning representations by back-propagating errors, *nature* **323**(6088): 533.

[69] Saikia, S., Verma, R., Agarwal, P., Shroff, G., Vig, L. and Srinivasan, A. [2018]. Evolutionary rl for container loading, *arXiv preprint arXiv:1805.06664* .

[70] Saxe, A. M., McClelland, J. L. and Ganguli, S. [2014]. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, *ICLR* .

[71] Shen, Y. and Zhang, C. [2015]. Loading sequencing with consideration of container rehandling, *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, IEEE, pp. 1237–1241.

[72] Shen, Y., Zhao, N., Xia, M. and Du, X. [2017]. A deep Q-learning network for ship stowage planning problem, *Polish Maritime Research* **24**(S3): 102–109.

[73] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D. [2017].

Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

[74] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. [2017]. Mastering the game of go without human knowledge, *Nature* **550**(7676): 354.

[75] Stahlbock, R. and Voß, S. [2008]. Operations research at container terminals: a literature update, *OR spectrum* **30**(1): 1–52.

[76] Steenken, D., Voß, S. and Stahlbock, R. [2004]. Container terminal operation and operations research-a classification and literature review, *OR spectrum* **26**(1): 3–49.

[77] Sutton, R. S. and Barto, A. G. [2018]. *Reinforcement learning: An introduction*, MIT press.

[78] Tallec, C. and Ollivier, Y. [2018]. Can recurrent neural networks warp time?, *ICLR* .

[79] Tijjani, S. and Ozkaya, A. [2014]. A comparison of reinforcement learning and evolutionary algorithms for container loading problem, *2nd International Symposium on Engineering, Artificial Intelligence and Applications*.

[80] Vorontsov, E., Trabelsi, C., Kadoury, S. and Pal, C. [2017]. On orthogonality and learning rnn with long term dependencies, *ICML* .

[81] Wisdom, S., Powers, T., Hershey, J. R., Le Roux, J. and Atlas, L. [2016]. Full-Capacity Unitary Recurrent Neural Networks, *Advances in Neural Information Processing Systems* .

[82] Yang, Z., Moczulski, M., Denil, M., d. Freitas, N., Smola, A., Song, L. and Wang, Z. [2015]. Deep fried convnets, *ICCV*.

[83] Zhang, J., Lei, Q. and Dhillon, I. [2018]. Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization, *in* J. Dy and A. Krause (eds), *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Stockholmsmässan, Stockholm Sweden, pp. 5806–5814.

[84] Zhang, M., Lucas, J., Ba, J. and Hinton, G. E. [2019]. Lookahead optimizer: k steps forward, 1 step back, *Advances in Neural Information Processing Systems*, pp. 9593–9604.

[85] Zhang, W. [1999]. *Algorithms for Combinatorial Optimization*, Springer New York, New York, NY, pp. 13–33.
URL: *https://doi.org/10.1007/978-1-4612-1538-7_2*

[86] Zhou, Z., Kearnes, S., Li, L., Zare, R. N. and Riley, P. [2019]. Optimization of molecules via deep reinforcement learning, *Scientific reports* **9**(1): 1–10.