

RÉF: 2020/II3/

Ministry of Higher Education and Scientific Research  
Manouba University  
National School of Computer Science



**Engineer Diploma Thesis**  
presented in order to graduate as a:  
**Computer Science Engineer**  
Topic

---

**Development of a Reinforcement Learning Algorithm for the Bin Packing Problem**

---

Carried Out By  
**Ahmed Gharbi**  
At



Academic Supervisor : **Dr. Wided Chaari**

Company Supervisor : **Mrs. Eya Rhouma & Mr. Seif Dassi**

Host Organismr: : **InstaDeep**

Address: : **Immeuble ICC3 bloc D 4ème étage. Centre Urbain Nord, Tunis 1082**

Phone: : **+216 50 656 117**

Academic Year : 2019/2020

---

# Abstract

This research work was carried out as part of the end of studies project for obtaining the engineer degree in the National School of Computer Science. It consists of developing a reinforcement learning based approach to solve the three-dimensional bin packing problem. The algorithm results are benchmarked against state-of-the-art heuristics and other reinforcement learning approaches.

**Keywords:** reinforcement learning, bin packing, heuristics, meta-heuristics

# Résumé

Ce travail de recherche a été réalisé dans le cadre du projet de fin d'études pour l'obtention du diplôme d'ingénieur de l'Ecole Nationale des Sciences de l'Informatique. Ce travail consiste à développer une approche basée sur l'apprentissage par renforcement pour résoudre le problème de bin packing à trois dimensions. Les résultats de l'algorithme sont comparés à des heuristiques de pointe et à d'autres approches d'apprentissage par renforcement.

**Mots Clés:** apprentissage par renforcement, bin packing, heuristiques, meta-heuristiques

## ملخص


يندرج هذا العمل في إطار مشروع ختم الدراسة للحصول على شهادة الهندسة من المدرسة الوطنية لعلوم الإعلامية. يتكون هذا العمل من تطوير منهج قائم على التعلم المعزز لحل مشكلة صندوق التعبئة في ثلاثة أبعاد. تتم مقارنة نتائج الخوارزمية بمساعدات على الكشف وأساليب التعلم المعززة الأخرى. الكلمات الرئيسية: التعلم المعزز، مشكلة صندوق التعبئة، مساعدات على الكشف، مساعدات على الكشف التلوي.

---

## Company Supervisor Appreciation

Mrs. Eya Rhouma

Ahmed has shown great seriousness and autonomy in this internship. he was able to successfully tackle different difficulties and obtain great results. His development and research will be beneficial to this project as well as to other engineers working on different projects at InstaDeep.

DocuSigned by:  
  
DCAD582E0CC945A...

Mr. Seif Dassi

Your dedication and hard work have left us in utter amazement in the company! I wish to see you progress in all your endeavors.

  
Dassi Seif

## Academic Supervisor Appreciation

Dr. Wided Chaari

# Acknowledgement

IN conducting this work, I have received meaningful assistance from many people which I would like to put on record here with deep gratitude and great pleasure.

First and foremost, I express my sincere gratitude to my supervisors *Mrs Eya Rhouma* and *Mr Seif Dassi*, who extended their complete support and helped to make me deliver my best. This project wouldn't have been a success without their knowledge, insightful conversations and helpful comments.

I would also like to thank the rest of InstaDeep team especially *Mrs Sinda Ben Salem* for her continuous support and encouragement and *Mr Khalil Gorsan Mestiri*, *Miss Rihab Gorsane* and *Miss Sabrina Krichen* for their guidance and advices. I am honored to have worked with such a competent and dedicated people.

Last but not least, I would like to thank my academic supervisor, *Dr Wided Chaari* who not only was of a great help in achieving this project, but also, whose treasurable training and immense knowledge, during my academic journey, had played a major role in my developement.

Finally, I acknowledge with much appreciation the jury members for having honoured me by agreeing to evaluate this work.

# Contents

<b>General Introduction</b>	<b>2</b>
<b>1 General Context</b>	<b>4</b>
1.1 Context of the Internship . . . . .	4
1.2 Presentation of the Host Company . . . . .	4
1.2.1 About InstaDeep . . . . .	5
1.2.2 Main services . . . . .	5
1.2.3 Values . . . . .	5
1.3 Context of the Project . . . . .	6
1.3.1 Expected Results . . . . .	6
1.4 Project Management Methodology . . . . .	6
<b>2 Preliminary Study</b>	<b>8</b>
2.1 Problem Definition . . . . .	8
2.1.1 Problem Formulation . . . . .	9
2.1.2 Constraints . . . . .	9
2.2 Meta-heuristics . . . . .	10
2.2.1 Simulated Annealing . . . . .	10
2.3 Reinforcement Learning . . . . .	12

2.3.1	Markov Decision Processes . . . . .	12
2.3.1.1	The Agent–Environment Interface . . . . .	12
2.3.1.2	MDP Formal Definition . . . . .	13
2.4	Placement Strategy . . . . .	14
2.4.1	Empty Maximal Spaces . . . . .	14
2.4.2	Placement Heuristics . . . . .	16
2.4.2.1	Distance to the Front-Top-Right Corner-1 (DFTRC-1) . . . . .	16
2.4.2.2	Deepest-Bottom-Left Heuristic (DBL) . . . . .	18
2.4.2.3	Distance to the Front-Top-Right Corner-2 (DFTRC-2) . . . . .	19
<b>3</b>	<b>State of The Art</b>	<b>21</b>
3.1	Existing Solutions . . . . .	21
3.1.1	Heuristics . . . . .	21
3.1.1.1	Wall Building . . . . .	21
3.1.2	Placement Heuristics . . . . .	22
3.1.2.1	Empty Maximal Spaces based Approaches . . . . .	23
3.1.2.2	Extreme Points based approaches . . . . .	23
3.1.3	Meta Heuristics . . . . .	23
3.1.3.1	Genetic Algorithms . . . . .	24
3.1.3.2	Tabu Search . . . . .	24
3.1.4	Reinforcement Learning Methods . . . . .	25
3.2	Critique of the Existing Solutions . . . . .	25
3.3	Proposed Solution . . . . .	26
<b>4</b>	<b>Implementation Details</b>	<b>28</b>
4.1	Work Environment . . . . .	28

4.1.1	Hardware Tools . . . . .	28
4.1.2	Software Environment . . . . .	29
4.2	Reinforcement Learning Driven Heuristic Optimization (RLHO) Overview . . .	29
4.3	The Proposed Solutions . . . . .	31
4.3.1	RLHO for the 3D Bin Packing Problem . . . . .	31
4.3.1.1	Environment Settings . . . . .	32
4.3.1.2	RLHO Environment Changes . . . . .	33
4.3.1.3	Combining RL and SA . . . . .	34
4.3.1.4	Challenges Facing RLHO . . . . .	35
4.3.2	Combining Reinforcement Learning and Placement Strategies . . . . .	35
4.3.3	RLPS Environment Settings . . . . .	36
4.3.4	Environment Improvements . . . . .	37
4.3.4.1	Empty Maximal Spaces Correction . . . . .	37
4.3.4.2	Long Short-Term Memory For Partially Observable MDPs . . .	38
<b>5</b>	<b>Results and Discussion</b>	<b>40</b>
5.1	Data set Overview . . . . .	40
5.2	Experimental Results . . . . .	41
5.2.1	RLHO Results . . . . .	41
5.2.2	RLPS Results . . . . .	42
5.2.3	Environment Improvements Impact . . . . .	42
5.2.3.1	EMS Correction Step Impact . . . . .	43
5.2.3.2	LSTM Cell Use Impact . . . . .	43
5.2.4	RLPS on Unseen Data . . . . .	44
5.2.5	Container Visualization . . . . .	45

5.3 Discussion . . . . .	47
<b>General Conclusion</b>	<b>49</b>
<b>Bibliography</b>	<b>51</b>
<b>Netography</b>	<b>52</b>



# List of Figures

1.1	Test and Learn Methodology Process . . . . .	7
2.1	Simulated Annealing pseudo-code . . . . .	11
2.2	The agent–environment interaction in a Markov decision process . . . . .	14
2.3	The generated Empty Maximal Spaces after placing one element in the bin . . .	15
2.4	DFTRC-1 heuristic . . . . .	16
2.5	Placed box . . . . .	17
2.6	Orientation heuristic in a 2 dimensional space . . . . .	17
2.7	Example of applying DFTRC-2 heuristic . . . . .	19
3.1	Wall Building . . . . .	22
3.2	Extreme Points . . . . .	23
3.3	Flowchart of a Standard Tabu Search algorithm . . . . .	24
4.1	The RLHO framework . . . . .	30
4.2	Proposed Solution Components . . . . .	32
4.3	The environment dynamics . . . . .	33
4.4	Comparison between the performance of an agent before (left) and after (right) the environment changes . . . . .	34
4.5	EMS before (left) and after (right) the EMS correction step . . . . .	38
4.6	Neural Network Architecture . . . . .	39

5.1	The first container of Instance_1 . . . . .	45
5.2	The first container of Instance_2 . . . . .	46
5.3	The first container of Instance_3 . . . . .	46
5.4	The first container of Instance_4 . . . . .	47
5.5	The Policy/Reward Cliff . . . . .	57
5.6	Clipped surrogate function . . . . .	57

# List of Tables

5.1	The four academic instances . . . . .	41
5.2	Results of RLPS compared to heuristics and other approaches . . . . .	42
5.3	EMS correction step impact demonstrated on Instance_1 . . . . .	43
5.4	Comparison of the RLPS approach with and without LSTM cell use . . . . .	43
5.5	Comparison of RLPS approach agent to a random agent on BR1 instances . . .	44
5.6	Comparison of RLPS approach agent to a random agent on BR10 instances . . .	44

# List of Acronyms

<b>AI</b>	Artificial Intelligence
<b>BPP</b>	Bin Packing Problem
<b>RL</b>	Reinforcement Learning
<b>SA</b>	Simulated Annealing
<b>MDP</b>	Markov Decision Process
<b>POMDP</b>	Partially Observable Markov Decision Process
<b>EMS</b>	Empty Maximal Space
<b>PPO</b>	Proximal Policy Optimization
<b>DFTRC</b>	Distance to the Front-Top-Right Corner
<b>DBL</b>	Deepest Bottom Left
<b>BBL</b>	Back Bottom Left
<b>BTL</b>	Back Top Left
<b>HA</b>	Height first–Area second
<b>RLHO</b>	Reinforcement Learning driven Heuristic Optimization

<b>RLPS</b>	Reinforcement Learning with Placement Strategies
<b>LSTM</b>	Long Short-Term Memory
<b>BR</b>	Bischoff and Ratcliff
<b>SPPO</b>	Sequential Proximal Policy Optimization

# General Introduction

THE three-dimensional packing problem is a generalization of the classical one- and two-dimensional problems. Such a problem has many industrial and commercial applications, but the most common one is the transportation of goods that may be packed directly into containers or trucks. Arranging boxes into a container is one of the most complex packing problems with respect to real-world constraints. An underestimate of the number of containers required to transport goods may be very costly in terms of delay and an under-utilization of containers may result in wasted resources and huge financial losses for companies.

Publications on this problem area are relatively few in comparison to its 1D and 2D counterparts. However, recent years have seen rapid growth in research and publications on this problem. Tremendous progress has been seen in Artificial Intelligence in the last few years, mainly, thanks to the ever-growing computational power that had become available and also, thanks to the major advances in algorithmic ideas. Such ideas have proved to be capable of solving complex problems and have been applied in many fields. The bin packing problem is no exception, as researchers have been developing and testing AI techniques to solve such an intricate problem. Most notably, reinforcement learning has emerged as a promising field in the AI world. Although its popularity is commonly associated with complex games solving such as Chess, Go and video games, over the last few years, attempts to use reinforcement learning to solve real-world problems have dramatically increased.

In this direction, in the present report, we investigate reinforcement learning's potential to solve the 3D bin packing problem. We give a detailed description of the approaches that seemed to perform the best, and compare our results with contemporary algorithms applied to the container loading problem.

This report goes for the following plan. In the first chapter, we will introduce the host company, the general frame of the project, and the expected results. Throughout the second chapter, we present a preliminary study in which we introduce algorithmic ideas required to fully understand the approaches we developed. In the third chapter, we will describe some of

the best performing existing solutions, give a brief critique of these solutions and introduce the proposed solution. In the fourth chapter, we will go through the implementation details. The fifth chapter is devoted to present the obtained results and benchmark our results with other existent algorithms. Finally, we will finish our report with a general conclusion and future prospects.

# Chapter 1

## General Context

### Introduction

In this first chapter we introduce the general context of the conducted work. We start by presenting the host company. Afterwards, we set the project context and we specify the goals to be achieved. Finally, we introduce the project management methodology adopted during this internship.

### 1.1 | Context of the Internship

This end-of-studies project is a graduation requirement to obtain the engineering diploma at the National School of Computer Science (ENSI). It involves doing an internship at a company for at least four months prior to graduation. My internship extends over a period of 6 months conducted at "InstaDeep" starting from February 10<sup>th</sup>, 2020.

### 1.2 | Presentation of the Host Company

In the following, we present the host company, and its solutions.



### 1.2.1 About InstaDeep

InstaDeep was founded in 2014. It has since grown to an established AI firm with more than 90 employees spread across its headquarter in London, and offices in Paris, Tunis, Lagos, Cape Town and Dubai.

InstaDeep puts forward different AI products. The solutions it offers include accelerated pattern-recognition through the use of GPUs and a focus on decision-making systems. These products are used in a wide range of industries, we cite energy, logistics, mobility and manufacturing.

Carefully selected from prestigious engineering schools grads in Tunisia, Europe, and around the world, InstaDeep teams are working on various domains of expertise such as computer vision, natural language processing, reinforcement learning and data science.

### 1.2.2 Main services

InstaDeep's services are focused mainly on three axes:

- **Decision-making systems:** Decision making is of vital importance for the the success of companies. InstaDeep's reinforcement learning systems help companies to improve their decision making and help them to maintain the margins and pricing to stay competitive. This technology can be beneficial for many sectors ranging from robotics and mobility to logistics and and the medical field.
- **AI-enabled Manufacturing:** AI use is advantageous for manufacturers and help reduce manufacturing costs. InstaDeep knows this very well and applies AI to IoT data to help predict with high accuracy machine failures and boost system reliability.
- **AI-powered Energy to fuel the world:** Classic methods for oil and gas detection are outdated and less efficient to find oil in very deep and underground resources. InstaDeep's advanced prediction and recommendation systems help energy companies reduce their exploration costs by improving their systems efficiency through the use of AI.

### 1.2.3 Values

In 2020, CB Insights ranked InstaDeep among the top 100 most promising private artificial intelligence companies in the world. InstaDeep is a leading AI power in Tunisia and Africa. It

knows well what African talent is truly capable of and encourages the African community by providing mentoring programs, training and by its participation and sponsorship of many AI events in Africa.

## 1.3 | Context of the Project

The goal of this research project is to implement a reinforcement learning algorithm to solve the bin packing problem. Mainly, the work should be based on the recently published scientific paper "Reinforcement Learning Driven Heuristic Optimization", on jun 2019 by a Google Brain team. The paper is, in short, about combining reinforcement learning and meta-heuristics to solve a 1D bin packing problem. The achieved work is not supposed to be fully based on the paper and any path that seems to yield good performance is allowed, as the goal of the internship is to implement a reinforcement learning algorithm for the bin packing problem.

### 1.3.1 Expected Results

In the end of the internship, the designed solution must satisfy the following requirements:

- The designed algorithm must prove success in the learning phase
- The approach is considered acceptable if it yields competitive results when compared to state-of-the-art heuristics and algorithms.
- The algorithm run-time has to be reasonable and as efficient as possible.
- A proper documentation must be provided.
- A benchmarking on academic instances must be performed.
- The implemented work should be well documented and written clearly and concisely.

## 1.4 | Project Management Methodology

For a smooth and successful project execution, it is important to adopt a project management methodology. Based on the nature of our project, we decided to adopt the *Test and*

*Learn* methodology. The *Test and Learn* methodology is well suited for projects characterized by uncertainties and for projects that require a lot of experimentation. The process of this methodology is illustrated in figure 1.1

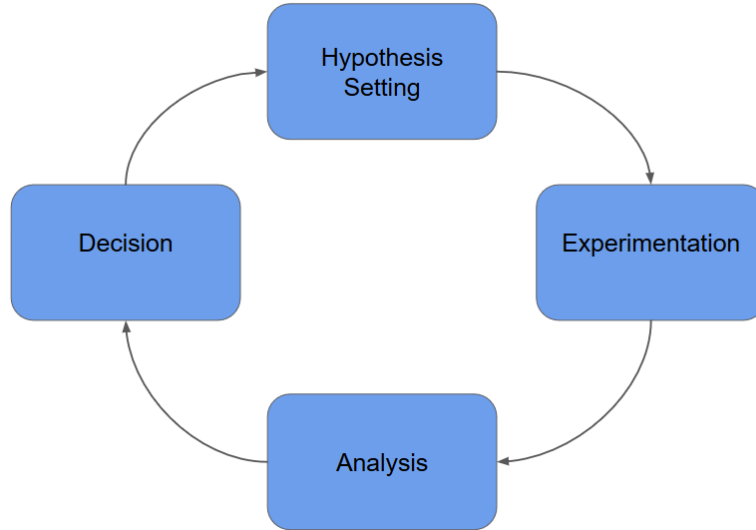


Figure 1.1: Test and Learn Methodology Process

The *test and learn* methodology consists first of setting hypotheses for the possible solutions of a given problem. The second phase is the testing phase in which we conduct experiments to test the validity of our hypotheses. Then, based on the results of the experiments, we decide whether the made hypotheses are valid and we improve the initial solution. The *learn and test* methodology is an iterative process and usually a number of round trips is needed so that the final solution meets the project requirements and reaches the expected results.

In some cases, following the tests results, the objectives of the project might be reviewed and sometimes, the whole project might be abandoned due to hypotheses leading to unsatisfactory results. [1]In some cases, following the tests results, the objectives of the project might be reviewed and sometimes, the whole project might be abandoned due to hypotheses leading to unsatisfactory results. ??

## Conclusion

In this chapter, we introduced the general context of the internship and the project, we presented the host company and we defined a project management methodology. The next chapter will be a theoretical study, in which we will present a few key theoretical concepts.

# Chapter 2

## Preliminary Study

### Introduction

The bin packing problem is a classical and well-studied problem. There is no shortage of algorithmic ideas that were developed throughout the years to solve this problem. In this chapter, we will go through some of these ideas, that we asses crucial to introduce, which would allow us later to get a better understanding of our solution.

### 2.1 | Problem Definition

The three-dimensional bin packing problem (3D-BPP) consists in packing, with no overlapping, a set of three-dimensional rectangular shaped boxes (items) into the minimum number of three-dimensional rectangular shaped bins (containers). The 3D-BPP is considered a strongly NP-hard problem. This implies that an exhaustive search methods to find the optimal solution are generally computationally unmanageable, and also that there is hence no known computationally feasible optimal solution method for the problem. So other means to get a solution have be established. Three-dimensional packing problems have numerous relevant industrial applications such as loading cargo into vehicles, containers or pallets, or in packaging design. The 3D-BPP has many variants, but during this project, we will mainly focus on 2 types of problems:

- **Weakly heterogeneous problem:** consists of a few types of items with high demand for each one.

- **Strongly heterogeneous problem:** consists of many types of elements with small demand for each one. The worst case is when all the items have different sizes and there is only one item of each shape.

### 2.1.1 Problem Formulation

To formulate the problem we will consider each item  $i$  in the finite set  $S$  to have three dimensions  $w_i$ ,  $h_i$  and  $d_i$ . Each identical bin  $b$  has dimensions  $W$ ,  $H$  and  $D$ . The items and bins are rectangular boxes and the three dimensions correspond to the width, height and depth values. To find the solution for a bin  $b$  we assume that  $\sum_{i \in b} w_i \leq W$ ,  $\sum_{i \in b} h_i \leq H$ ,  $\sum_{i \in b} d_i \leq D$ . As such it is correct to conclude that  $\sum_{i \in b} w_i \times h_i \times d_i \leq W \times H \times D$ . For each bin  $b$  we intend to minimize the wasted volume given by  $W \times H \times D - \sum_{i \in b} w_i \times h_i \times d_i$ .

### 2.1.2 Constraints

Practical constraints should be considered in the real-world container loading problem. In the following, we introduce the constraints we considered in this project:

- **Overlapping constraint:** Boxes should not overlap.
- **Placement constraint:** Boxes should lie entirely in the container.
- **Rotation of objects:** orthogonal rotations are allowed for the items. Rotating an item simply means swapping its width ( $w_i$ ), height ( $h_i$ ) and depth ( $d_i$ ) values. In a three-dimensional space, each item can have 6 different rotations.
- **Stability of objects:** The measure of stability of an object which is allocated onto other objects is defined as follows:

$$S = \frac{\text{contact space between upper object and lower objects}}{\text{underside space of upper object}}$$

We call  $S$  the ratio of contact between the upper object and lower objects.  $S \neq 1$  means the underside of the upper object is not completely supported by the lower objects. In this project, upper object must completely be contacted onto the lower objects, which means we must have a contact ratio  $S = 1$ .

## 2.2 | Meta-heuristics

Meta-heuristics are search methods used to solve complex optimization problems. They are used in many fields: business, engineering, industry and many other areas. Meta-heuristics often use concepts derived from mathematical, biological, natural and physical sciences to improve the performance of heuristics. The main difference between heuristics and meta-heuristics is that heuristics often depend on the problem to solve, mainly for choosing the nearest neighbour, while meta-heuristics are more general and can be applied to a wide range of problems. The goal of a meta-heuristic is to find a global optimum. This is achieved by both browsing the search area and exploring promising regions, but without being trapped by a local optimum. Meta-heuristics can be mainly classified into 2 types:

- **Constructive:** build a solution, step by step, according to a set of rules defined beforehand.
- **Improvement:** Start from a feasible solution and improve it by applying successive small changes.

Meta-heuristics are often inspired by natural processes. Our main focus will be on a meta-heuristic called Simulated Annealing.

### 2.2.1 Simulated Annealing

Simulated Annealing is a meta-heuristic for approximating the global optimum of a given function in a large search space. It is often used in combinatorial problems where the search space is discrete and has a finite set of solutions.

The simulated annealing algorithm is inspired by a naturally occurring phenomena: physical annealing. Physical annealing is the process of heating up a material until it reaches an annealing temperature and then cooling it down slowly so as to reduce the hardness of the material, relieve the tension and change its structure.

In high temperatures, the molecular structure of the material becomes weaker and more susceptible to change. During the cooling procedure, the molecular structure is harder and less susceptible to change. Simulated Annealing mimics the physical annealing process but is used for optimizing a function.

The details of SA are shown in Figure 2.1, where  $S$  is an initial solution,  $\psi(S)$  is the neighborhood of  $S$ ,  $S'$  is a solution generated by a change in  $S$  and  $S' \in \psi(S)$ ,  $f$  represents the objective function used for computing the cost of a solution  $S'$ ,  $S^*$  is the best solution found,  $\alpha$  is a cooling ratio,  $SA_{max}$  is a maximum number of iterations,  $T_0$  and  $T_f$  are an initial and final temperature respectively.

```

1   $S \leftarrow$  an initial solution;
2   $S^* \leftarrow S$ ;
3   $iter \leftarrow 0$ ;
4   $T \leftarrow T_0$ ;
5  while ( $T > T_f$ ) do
6      while ( $iter < SA_{max}$ ) do
7           $iter \leftarrow iter + 1$ ;
8          PICK (a random neighbor  $S' \in \psi(S)$ );
9           $\Delta \leftarrow f(S') - f(S)$ ;
10         if  $\Delta < 0$  then
11              $S \leftarrow S'$ ;
12             if  $f(S') < f(S^*)$  then
13                  $S^* \leftarrow S'$ ;
14             end
15         else
16             TAKE ( $x \in [0, 1]$ );
17             if  $x < e^{-\Delta/T}$  then
18                  $S \leftarrow S'$ ;
19             end
20         end
21     end
22      $T \leftarrow \alpha * T$ ;
23      $iter \leftarrow 0$ ;
24 end
25 Return  $S^*$ ;

```

Figure 2.1: Simulated Annealing pseudo-code

We can interpret the slow cooling procedure present in the simulated annealing algorithm as a slow decrease in the probability of accepting worse solutions while the solution space is explored. Accepting worse solutions is a mechanism that is often used in meta-heuristics that enables this family of algorithms to escape local optimums and go for a more extensive search for the global optimum.

Generally, simulated annealing algorithms follow these steps:

We start by initializing the temperature to an initial positive value. During the search, the temperature progressively and slowly decreases to zero. At each time-step, the algorithm randomly selects a solution close to the current one, measures its quality, if the new solution is superior, we accept it, else, if it is worse, we calculate an acceptance probability that depends on the current temperature and use it to make the decision of accepting or rejecting the worse solution.

## 2.3 Reinforcement Learning

Reinforcement learning (RL) is a sub-field of machine learning that enables an agent to learn in an interactive environment by trial and error using a feedback from its own actions and experiences. The RL paradigm is inspired by the Trial and Error Learning theory in Behavioral Psychology. When an organism faces a new difficult problem, the organism learns through trial and error, and with repeated trials, errors reduce.

### 2.3.1 Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical framework often used for modeling decision making in situations where the decision maker has partial control over the outcomes of his decisions. Most of the RL problems can be formalized as MDPs.

#### 2.3.1.1 The Agent–Environment Interface

First, let's define a few key terms that describe the elements of a RL problem:

- **Agent:** the learner and decision maker
- **Environment:** comprising everything outside the agent, is called the environment. It is the entity the agent interacts with.
- **Action:** is a manifestation of the interaction between the agent and the environment. It is a decision made by the agent.
- **Reward:** special numerical values that the agent seeks to maximize over time through its choice of actions.
- **State:** a representation of the environment. It includes all the information and variables that would describe perfectly an environment at a given time.
- **Policy:** Method to map agent's state to actions



### 2.3.1.2 MDP Formal Definition

To properly define what an MDP is, let's first introduce the Markov property. The Markov property states that *"The future is independent of the past given the present"* which can be formalized mathematically by:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1...S_t) \quad (2.1)$$

In other words, knowing the current state is equivalent to knowing all of the past states. Hence, a state is Markov if and only if it covers all relevant information from history.

An MDP is a tuple

$$(S, A, P, R, \gamma)$$

where:

- **S**: is the state space
- **A**: is the action space. It is a set of all possible actions that an agent can take. The behaviour of an agent can be perfectly described via its policy, which is a distribution over actions given states.

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \quad (2.2)$$

- **P**: is the state transition probability function. P completely characterize the environment's dynamics

$$P(s'|s, a) = \mathbb{P}[S_t = s'|S_{t-1} = s, A_{t-1} = a] \quad (2.3)$$

- **R**: the expected rewards for state-action pairs. It is a scalar feedback signal that indicates how well the agent is doing at time step  $t$ .

$$R(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] \quad (2.4)$$

- $\gamma$ : the discount factor,  $\gamma \in [0, 1]$ . It determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future.

We can introduce the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (2.5)$$

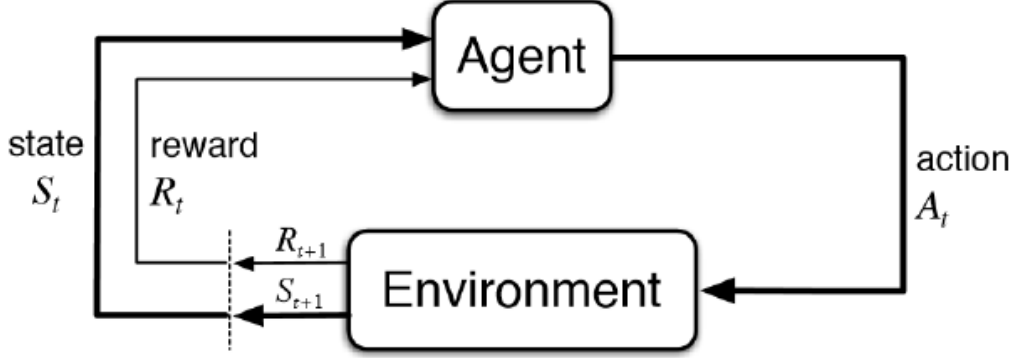


Figure 2.2: The agent–environment interaction in a Markov decision process

Figure 2.2 illustrates the agent-environment interaction. At each time step  $t$ , the agent receives some representation of the environment’s state,  $S_t \in S$ , and on that basis selects an action,  $A_t \in A(s)$ . One time step later, in part as a consequence of its action, the agent receives a numerical reward,  $R_{t+1} \in R \subset \mathbb{R}$ , and finds itself in a new state,  $S_{t+1}$ .

## 2.4 Placement Strategy

In the bin packing problem, when trying to place a box in the bins, we are faced with an infinite number of possible placements. In this section, we will see how we can discretize the solution space and will describe the main components of the placement strategy.

### 2.4.1 Empty Maximal Spaces

In order to manage the space in the bins, we use a concept called Empty Maximal Spaces (EMSs). EMSs are the largest empty parallelepiped spaces available for filling with boxes. Figure 2.3 illustrates the concept.

The maximal spaces are represented by their vertexes with minimum and maximum coordinates,  $(x_i, y_i, z_i)$  and  $(X_i, Y_i, Z_i)$  respectively. In the beginning, there is only one EMS with

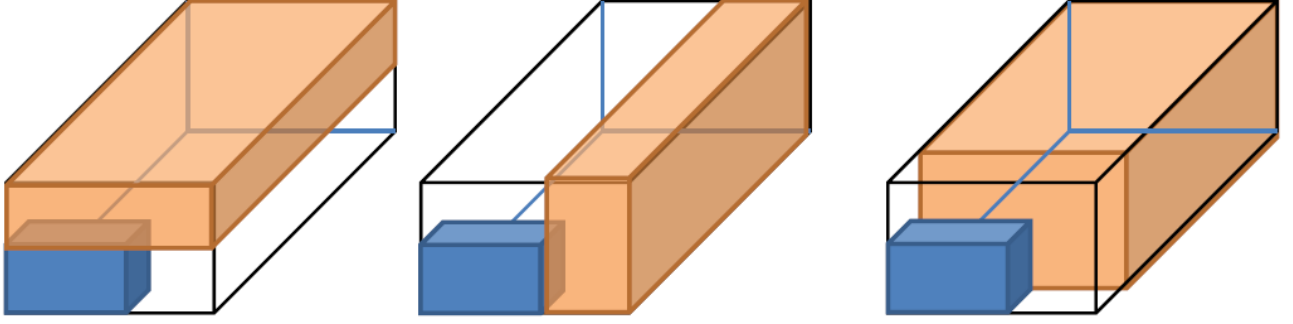


Figure 2.3: The generated Empty Maximal Spaces after placing one element in the bin

a size equal to the size of the container. When one blue box is placed at the origin of the bin (origin of the initial EMS), it generates three empty maximal spaces which are represented by the yellow cuboids. These three yellow cuboids result from the intersection of the box placed with the initial EMS (the empty container). In the next step, when packing the next box, the yellow cuboids represent the three possible placements to place the next box.

To generate and keep track of the EMSs, we make use of the difference process (DP), developed by Lai and Chan (1997) [2]. The DP process consists of the following three main steps:

- Place a box in a EMS
- Generate new EMSs resulting from the intersection of the box being placed with the existing EMSs and remove the intersected EMSs
- Eliminate the EMSs which have infinite thinness or those that are totally inscribed by other EMSs.

The elimination of EMSs that are totally inscribed by other EMSs is the most time consuming task in the DP process. In order to reduce the computation time for this task we added the following rules to the difference process introduced by Gonçalves and Resende (2013) [3]. Using these simple rules we could speed up the generation of EMSs by up to 55 %:

- If the volume of a newly created EMS is smaller than the volume of each of the boxes remaining to be packed, do not add it to the list of EMSs.
- If the smallest dimension of a newly created EMS is smaller than the smallest dimension of each of the boxes remaining to be packed do not add it to the list of EMSs.

## 2.4.2 Placement Heuristics

When searching for a position to pack a box, only the origin  $(x_i, y_i, z_i)$  of the feasible EMSs are considered as candidate positions. An EMS is considered feasible for a box if the box fits in it using at least one of its possible orientations. In this section, we introduce placement heuristics that will select the EMS where a box will be placed and the orientation of the box.

### 2.4.2.1 Distance to the Front-Top-Right Corner-1 (DFTRC-1)

Distance to the Front-Top-Right-Corner [3] is a heuristic used to select the EMS in which we will put a box.

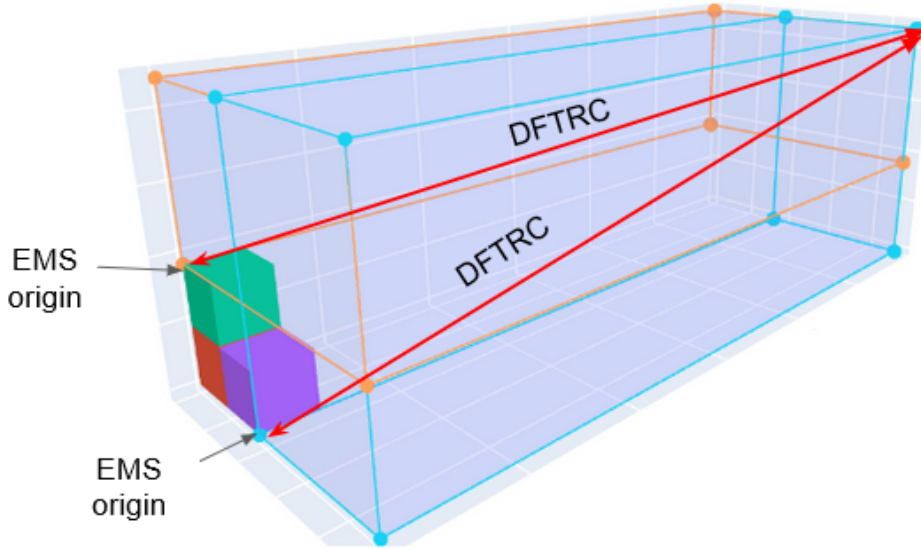


Figure 2.4: DFTRC-1 heuristic

Figure 2.4 illustrates the heuristic. There are 3 boxes that are already packed represented by the 3 fully colored cuboids. For the sake of the example, we are only considering two of the EMSs. The two EMSs are the yellow and blue cuboids. To compare the EMSs, we calculate the distance from the origin of the EMS to the front-top-right corner of the container and we use this distance as a measure of compactness. The greater the DFTRC the more compact we will consider the packing of the box. In this example, the yellow EMS has a higher DFTRC, so the next box will be placed in the yellow EMS on the top of the green box as shown in Figure 2.5. Once the box is placed, we update the list of EMSs and sort it according to the DFTRC-1 heuristic.

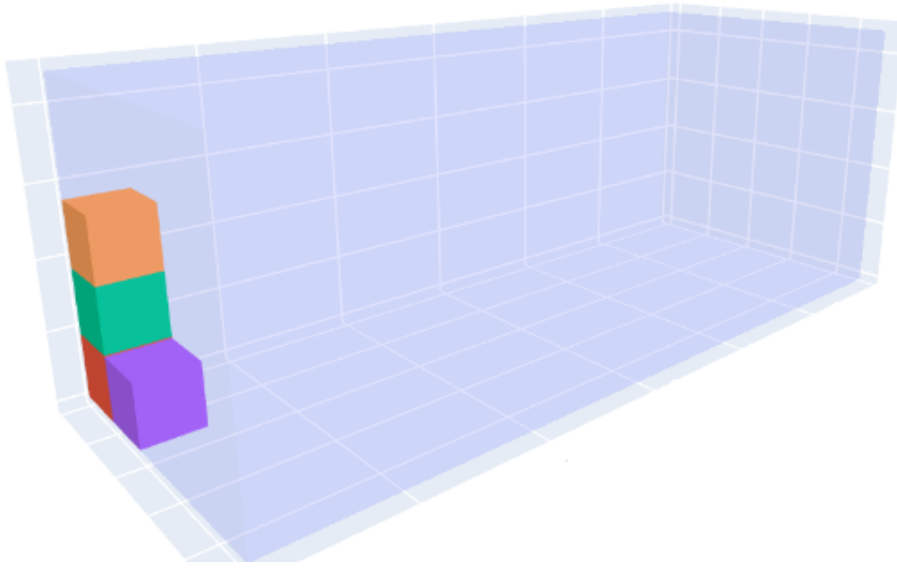


Figure 2.5: Placed box

The DFTRC-1 heuristic is only responsible for selecting the spatial location of the box. In a 3D environment, there are 6 possible orientations for a box. To select an orientation, we use another heuristic introduced in [5]. Figure 2.6 illustrates the idea in a 2-dimensional space which can be easily extended to a 3D space.

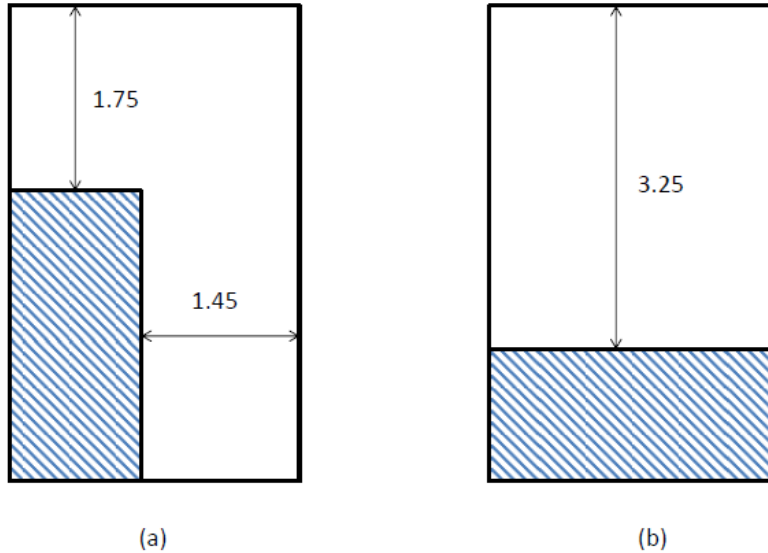


Figure 2.6: Orientation heuristic in a 2 dimensional space

When one box has several feasible orientations in one empty space, the one having the smallest margin is selected. For example, in Figure 2.6, the box can be placed in the EMS in two different

orientations, the first placement (a) has a margin (1.45; 1.75) while the second placement (b) has a margin (0; 3.25). We compare the minimums of each tuple, the placement with the least minimum margin is selected. In this case placement (b) will be selected because  $0 < 1.45$  and hence it generates less margin in one direction. If there are no feasible orientations in the EMS, we take the next EMS in the list of EMSs. The idea behind the orientation heuristic is that, when choosing the orientation as described, the EMSs that will be generated after placing the box are more likely to host the rest of the boxes.

#### 2.4.2.2 Deepest-Bottom-Left Heuristic (DBL)

The Deepest-Bottom-Left heuristic [9] has been extensively used to tackle the bin packing problem. DBL is a sorting algorithm that operates over the list of EMSs. The idea is that an object should be placed in the deepest available position, and then as far as possible to the bottom and then as far as possible to the left.

---

**Algorithm 1:** Deepest-Bottom-Left Sorting (S)

---

**Result:** Sorted EMSs list S

initialization;

Set  $sw = \text{false}$  ;

Let  $n$  be the maximal index of list  $S$  ;

**for**  $i \leftarrow 2$  **to**  $n$  **step** 1 **do**

**for**  $j \leftarrow i$  **to** 2 **step** -1 **do**

**if**  $x_j < x_{j-1}$  **then**

$\text{swap}(S_j, S_{j-1})$  ;

// First check the length order

$sw = \text{true}$  ;

**else if**  $x_j = x_{j-1}$  **then**

**if**  $z_j < z_{j-1}$  **then**

$\text{swap}(S_j, S_{j-1})$  ;

// Second check the height order

$sw = \text{true}$ ;

**else if**  $z_j = z_{j-1}$  **then**

**if**  $y_j < y_{j-1}$  **then**

$\text{swap}(S_j, S_{j-1})$  ;

// Third check the width order

$sw = \text{true}$  ;

**end**

**end**

---

The DBL heuristic allows us to select the EMS in which we will pack a box. As for the orientation of the box, we use the same orientation heuristic used in the DFTRC-1 heuristic.

### 2.4.2.3 Distance to the Front-Top-Right Corner-2 (DFTRC-2)

In DFTRC-1, the orientation of the box is selected using an orientation heuristic. In DFTRC-2, we use the distance to the front top right corner to select the EMS in which we will pack a box, as well as the orientation of the box. DFTRC-2 calculates, for all feasible EMSs and all feasible box orientations in each EMS, the DFTRC of the front-top-right corner of the box being packed and chooses the EMS which maximizes DFTRC.

Figure 2.7 illustrates the idea of this heuristic. We are considering a two-dimensional partial packing of boxes 1, 3, and 4 and we want to choose a place to pack box 2. If we use heuristic DFTRC-1, box 2 will be put in point A and if we use heuristic DFFTRC-2, box 2 will be put in point C. In DFTRC-2, we are calculating the distance from the front-top-right corner of the box to the front-top-right corner of the container, and we choose the placement that maximizes this distance. Note that in this example, the EMS having as minimum coordinates point B is not feasible for box 2 and that the EMSs having as minimum coordinates points A and D only allow one feasible orientation of box 2.

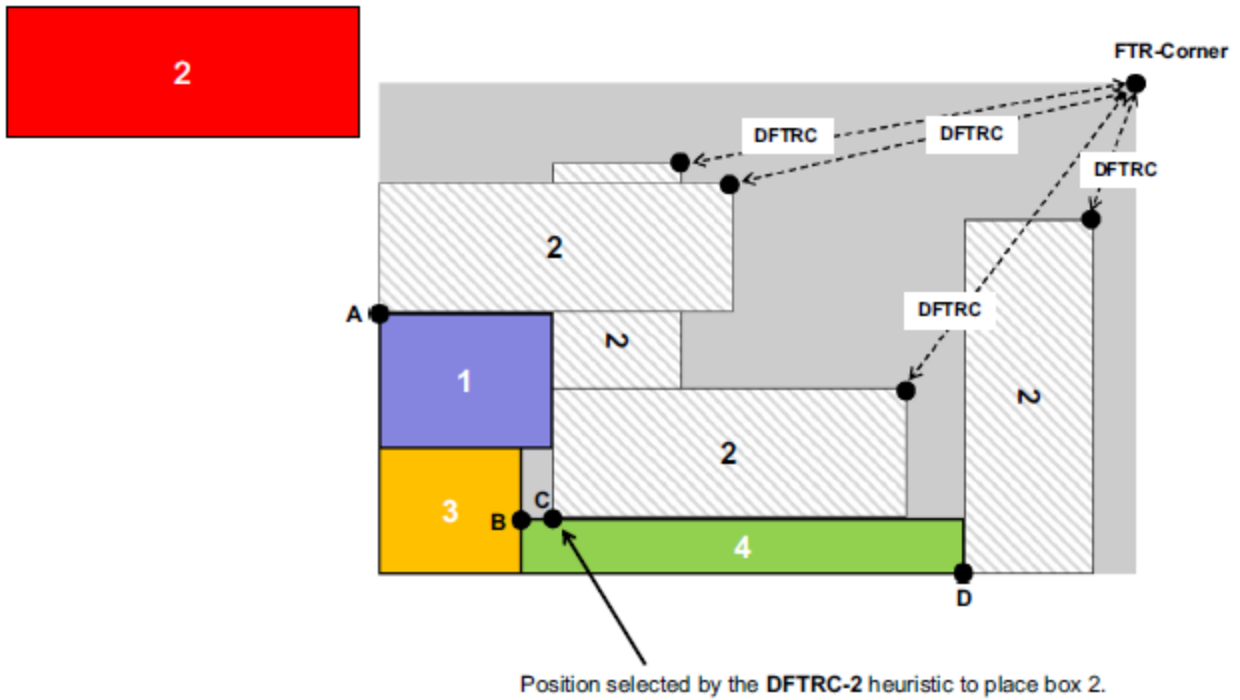


Figure 2.7: Example of applying DFTRC-2 heuristic

## Conclusion

In this chapter, we formulated the bin packing problem, then we introduced the main concepts that will be used to solve it, which are meta-heuristics and reinforcement learning. We have also introduced the placement heuristics which allow us to simulate the packing operation and manage the space in the bins. In the next chapters, we will build upon these ideas to introduce the Reinforcement Learning Driven Heuristic Optimization framework.



# Chapter 3

## State of The Art

### Introduction

We begin this second chapter by introducing the existing solutions, then we provide a brief critique of these approaches, followed by an introduction to the proposed solution.

## 3.1 | Existing Solutions

### 3.1.1 Heuristics

In any heuristic solution approach, there should be a mechanism to decide how to put the boxes in the container. When the mix of boxes is weakly heterogeneous, the most common used heuristics are Wall Building and Layer Building. This is mainly because the weakly heterogeneous case has many similar items and the aforementioned heuristics make use of this information. In the other hand, in the strongly heterogeneous case, boxes will be placed one at a time. In the following, we will describe the main ideas of Wall Building heuristic. Layer Building is a less adopted approach but is similar in principle to the wall building heuristic.

#### 3.1.1.1 Wall Building

Wall building is a well-known constructive heuristic that deals with the 3D bin packing problem. The heuristic is based on the concept of filling the container layer by layer. The heuristic uses

the assumption that it is often desirable that all boxes of the same type should be placed together, hence the boxes of one type will tend to be stacked in proximity. The Wall Building heuristic is basically a set of rules that dictates how to pack the boxes in the container. The main concept is how to build layers. Each layer is a section of the length of the container over its complete height and width. A new layer is not commenced until all the previous layer is packed. The depth of each layer is determined by the size of the first box packed in that layer. When filling a space in a layer, a box type is selected and used to fill as many complete columns width-wise as possible. The selection of the box type is done using a set of rules. For example, one of the criterion is that the boxes are ranked according to the largest smallest dimension, and that's because a box with a large smallest dimension is hard to fit late in the packing. The algorithm is complex and it's impossible to mention all the details in this brief overview. Figure 3.1 illustrates the concept of Wall Building.

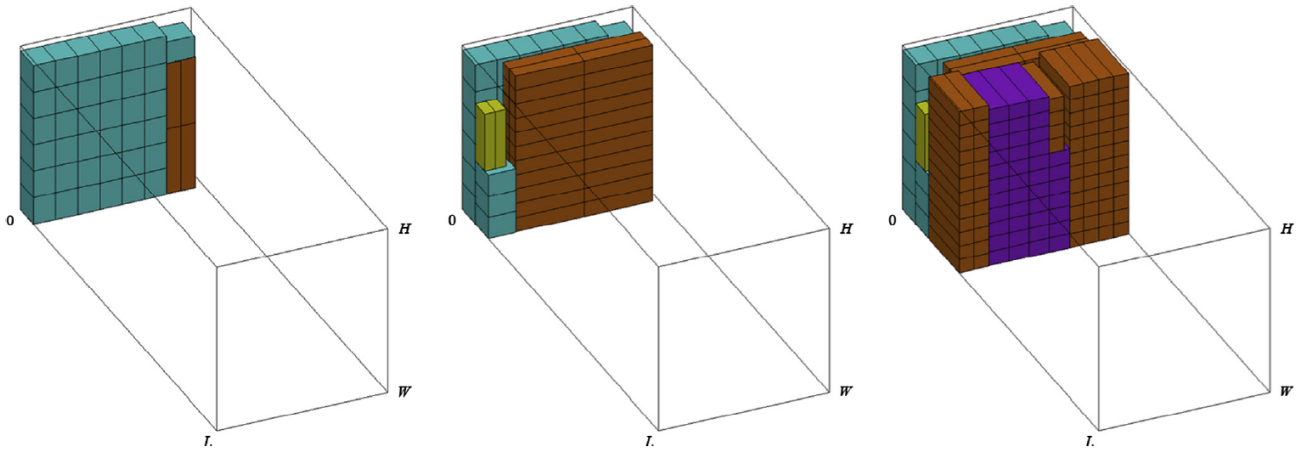


Figure 3.1: Wall Building

### 3.1.2 Placement Heuristics

Placement heuristics are algorithms that uses some criteria to decide where to pack a box in the container. The most common techniques in this family of heuristics are empty maximal spaces and extreme points. Many efficient algorithms adopt these techniques to solve the bin packing problem.

### 3.1.2.1 Empty Maximal Spaces based Approaches

Maximal spaces are the set of largest cuboid spaces that entirely cover the unpacked volume of the container. These approaches are based on the use of EMSs. The main idea is to identify maximal spaces and then fill them with the best configuration of boxes according to few criteria. For example, once a box is packed and the new empty maximal spaces are generated, the next box could be packed in the closest space to the bottom left corner of the container that is large enough to accommodate the box.

### 3.1.2.2 Extreme Points based approaches

Extreme points are an alternative strategy of identifying placement points. These points arise from contact between the face of two boxes or a box and the container and are located at the point where three edges coincide to create a fully concave corner. These become the candidate placement positions for placing the unpacked boxes. A set of criteria are used to select the best extreme point to pack the next box in. Figure 3.2 is an example of extreme points in three dimensions.

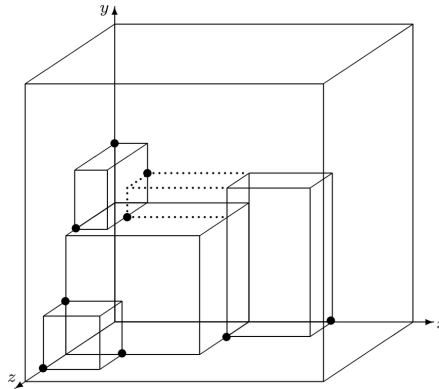


Figure 3.2: Extreme Points

### 3.1.3 Meta Heuristics

Meta heuristics usually build on the placement heuristics described in the previous section. While placement heuristics will provide a fast and reasonable quality solution, meta heuristics are usually used to improve the solution of the placement heuristics. Meta heuristics are often used to determine the ordering of the boxes referred to as sequencing. Sequencing can be critical

to the effectiveness of the packing heuristic and modifications to the placement heuristics often involve altering the box sequence.

### 3.1.3.1 Genetic Algorithms

Genetic algorithms have been extensively used in this context and its popularity may be due to the many genetic operators designed to deal with permutation representations. The main idea is that chromosomes are used to encode the ordering and the rotation of each box, and each chromosome is transformed to a solution by a placement heuristic. The population is ranked according to a fitness value, crossover and mutation are applied and the best solution is copied to the next generation.

### 3.1.3.2 Tabu Search

Tabu Search is also one of the most popular meta heuristics in the bin packing problem. Most commonly, a complete initial solution is first generated either randomly or through a basic heuristic. There are mainly two approaches when using Tabu Search: either to perform a search that alters the box packing order using a swap neighborhood and use a placement heuristic to determine the layout within the container, or, to decide the ordering according to some criterion, and alter the free space assigned to a box. Simulated Annealing has also been used in a similar way to Tabu Search to solve the bin packing problem. Simulated Annealing was later combined with Tabu Search to form a hybrid search system. Figure 3.3 illustrates the flowchart of a standard Tabu Search algorithm.

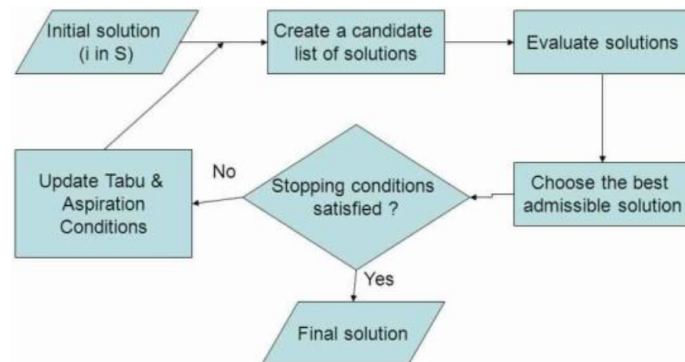


Figure 3.3: Flowchart of a Standard Tabu Search algorithm

### 3.1.4 Reinforcement Learning Methods

In recent years, Reinforcement Learning has been increasingly applied to solve combinatorial problems. The bin packing problem is not an exception. Reinforcement learning has been used in different ways to solve the bin packing problem.

In 2016, Bello et al. [B1] developed a neural combinatorial optimization framework with RL to solve the travelling salesman problem, and then demonstrated the flexibility of their work by applying their approach on the the KnapSack problem which can be considered as a one-dimensional bin packing problem with constraints. Mainly, RL was used to train a neural network that points to items to include in the knapsack and stops when the total weight of the items collected exceeds the weight capacity.

Deep-Pack [B2] is a deep reinforcement learning framework proposed to solve the online two-dimensional bin packing problem that takes an image showing the current state of the bin as input and gives out the pixel location where the incoming object needs to be placed as the output.

Hu et al. [B3] combined a heuristic approach with reinforcement learning to solve a different 3D version of the problem. In fact, in many real business environments, bins do not have a fixed size, and the cost of a bin is proportional to its surface area. The goal is to find a way to place the items while minimizing the surface area of the bin. The role of the RL agent is to order the input sequence and the main role of the heuristic is to transform the output sequence generated by the RL agent into a feasible solution, then, a reward signal can be computed and used by the agent to improve. This technique proved to outperform previous well-designed heuristics.

## 3.2 | Critique of the Existing Solutions

First of all, it is crucial to mention that the bin packing problem is a very popular research direction in optimization area. Although there is an extensive scientific literature about the BPP, it is still an active research domain. Mainly, researchers aim to design new approaches that not only outperform the existing solutions in terms of quality, but also approaches that can be generalized to a wide range of packing problems and that can be applied in a reasonable amount of time. Despite the fact that the existing approaches can find good solutions for the BPP, there is always a need for improvement. Besides, generally, these methods suffer from the

lack of generality. For instance, the Wall Building heuristic does a great job in the homogeneous bin packing problem which is the problem that consists of few boxes shapes, while placement heuristics tend to do a better job at the heterogeneous version of the problem. Finally, some of the proposed methods are hard to adapt for real world problems and for commercial use because of their inability to respect some constraints that are omnipresent in the industrial world.

## 3.3 | Proposed Solution

Our solution is based on a research paper entitled "Reinforcement Learning Driven Heuristic Optimization" [B4] published in 2019 by a Google team. The approach is a hybrid method that combines reinforcement learning with meta-heuristics. Meta-heuristics are efficient algorithms in optimization problems, but their result depends heavily on the initial solution. The main idea of the paper is to train a RL agent to learn to provide good initial solutions for meta-heuristics. Originally, the idea was designed and tested on the one-dimensional Bin Packing Problem. The items are assigned a weight and the bins have a maximum capacity. Initially, the items are placed randomly in the bins and the role of the RL agent and the meta-heuristic is to change the assignment of the items in the bins in such a way that it minimizes the number of used bins.

In this paper, the authors demonstrate a novel approach to combinatorial optimization where reinforcement learning and heuristic algorithms are combined to yield superior results to reinforcement learning alone on a combinatorial optimization problem. They also demonstrate that it's possible to train reinforcement learning to enable heuristic algorithms to achieve superior performance than when they are decoupled on combinatorial optimization problems. Further details about the paper will be discussed in later chapters.

In this project, our work consists mainly of testing if the framework presented in the paper can be scaled and adapted to solve a 3D bin packing problem.

## Conclusion

This chapter sought to establish a state of the art. We started by presenting the existing modern solutions to solve the problem of container loading, then we provided a brief critique of the existing solutions and we finished by introducing our solution. In the next chapter, we

will look into the implementation details.

# Chapter 4

## Implementation Details

### Introduction

This chapter presents the implemented work. First, we tried to extend the RLHO paper to be used in the 3D bin packing problem. But then we noticed that the developed approach was suffering from some challenges that we mention. Second, we developed an RL based approach to tackle the problems the RLHO approach was facing. In the upcoming sections, we will be delivering its detailed implementation.

### 4.1 | Work Environment

In the following section, we introduce the hardware and the software environments used in this work.

#### 4.1.1 Hardware Tools

We used a desk computer with the following specs

- **Processor:** Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
- **GPU:** NVIDIA GeForce GTX 1060 10 GB of memory
- **Memory:** 16 GB of RAM



- **Operating system:** Ubuntu
- **Hard Disk:** 1 TO

### 4.1.2 Software Environment

- **Visual Studio Code:** is a code editor redefined and optimized for building and debugging modern applications
- **Jupyter Notebook:** is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
- **Python:** Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
- **OpenAI Gym:** Gym is a toolkit for developing reinforcement learning environments.
- **Ray:** Ray is a fast and simple framework for building and running distributed applications. It enables users to parallelize single machine code, with almost no changes at all.
- **RLlib:** is an open-source library for reinforcement learning that offers both high scalability and a unified API for a variety of applications.
- **TensorFlow:** TensorFlow is an open source software library for numerical computation using data-flow graphs.
- **Keras:** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow
- **Plotly Python:** Plotly is a python library used to create interactive graphs.

## 4.2 | Reinforcement Learning Driven Heuristic Optimization (RLHO) Overview

Reinforcement Learning Driven Heuristic Optimization (RLHO) is a framework that was originally designed and tested on the 1D Bin Packing Problem. The main motive behind this

framework is that meta-heuristics are commonly known to find good approximate solutions to NP-hard problems, but the run-time and the quality of the solution are heavily dependent on the initial solution. The idea is to use reinforcement learning to teach an agent to provide for a meta-heuristic algorithm a good initial solution, resulting in an overall better final solution, outperforming pure reinforcement learning algorithms. The RLHO framework can be

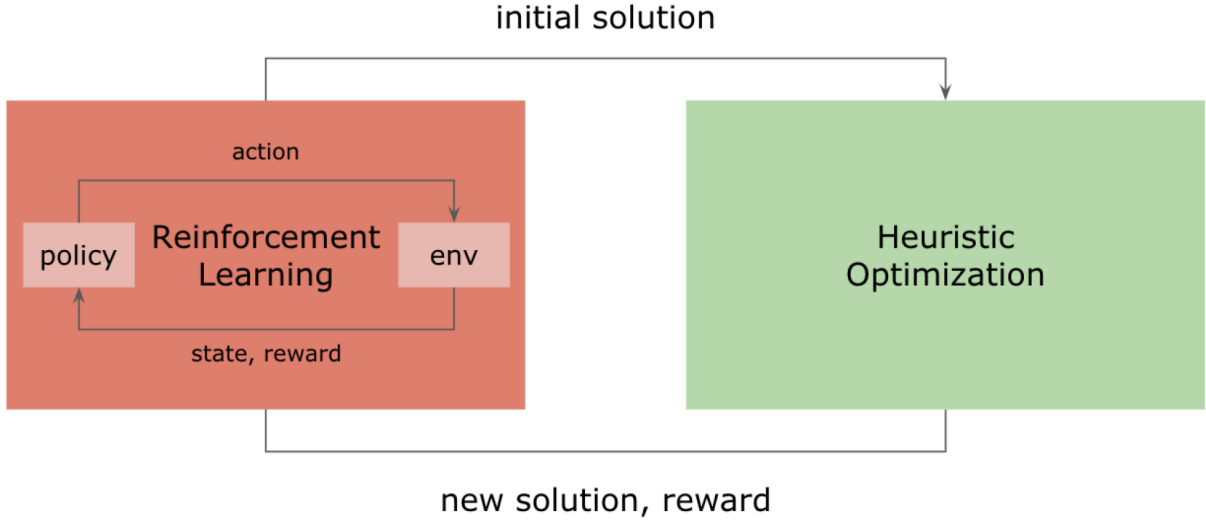


Figure 4.1: The RLHO framework

decomposed into two major components: the Reinforcement learning component (RL) and the Heuristic Optimization component (HO).

In the following we will describe the dynamics of the RLHO framework: we have an RL agent interacting with an environment, the agent makes an action, the environment returns a reward and an observation about the state of the environment. We run the RL agent for an  $x$  number of steps. The generated solution after  $x$  number of steps will serve as an initial solution for the Heuristic Optimization component. In HO, we use a meta-heuristic that will optimize further the solution, we run for  $y$  number of steps. After  $y$  steps, the final solution is evaluated, and a second reward signal is generated and propagated to the RL agent. In this approach, the action space consists of perturbations and thus the MDP does not have a terminal state. The agent is free to continue perturbing the state forever, however, we train the agent with a limited number of steps. The additional optimization provided by HO gives us an additional training signal to RL as to how RL actions contribute to the future return provided by HO. The second reward signal gives us an estimation of the accumulated reward if the RL agent were to be run infinitely.

In the original paper, in the RL component, the authors used Proximal Policy Optimization (PPO) which is a policy gradient based reinforcement learning algorithm (the details of the algorithm are presented in the Appendix), and Simulated Annealing as a meta-heuristic in the Heuristic Optimization component. It should be noted that they have used this framework to solve the 1D Bin Packing Problem, in which items and containers are characterized by a single value (size). Using RLHO for the 3D version of the problem is a challenge to overcome, especially that the 3D version is a much more complex problem.

## 4.3 | The Proposed Solutions

### 4.3.1 RLHO for the 3D Bin Packing Problem

The container loading problem is way more complex than the 1 dimensional version. It can be decomposed into 3 sub-problems:

- Decide the order of boxes to be packed (commonly referred to as the sequence)
- Decide the spacial location of the box
- Decide the orientation of the box

It is hard for an RL agent to do all of the 3 mentioned decisions, especially that the space in the container is of continuous nature, and hence, the number of possible solutions is infinite. Moreover, the RLHO framework requires the use of a meta-heuristic such as Simulated Annealing. Thus, we have to find a formulation to the problem that can be adapted to both the RL agent and the Simulated Annealing algorithm at the same time. We propose that the RL agent and SA will only be responsible to define the sequence (the order of boxes). As for the spatial location of each box and its orientation, these will be decided by a placement algorithm that is based on heuristics introduced in 2.4. The main role of the placement algorithm is to transform a sequence into a complete packing solution. The quality of the solution outputted by the placement algorithm heavily depends on the input sequence, and thus, we formulated the bin packing problem as an optimization problem over sequences. In figure 4.2 we introduce the role of each component in the proposed solution as well as the relationship between them. The RL agent decides the order of the boxes, the allocation algorithm transforms the generated sequence into a packing solution, the solution is evaluated and a reward is returned to the

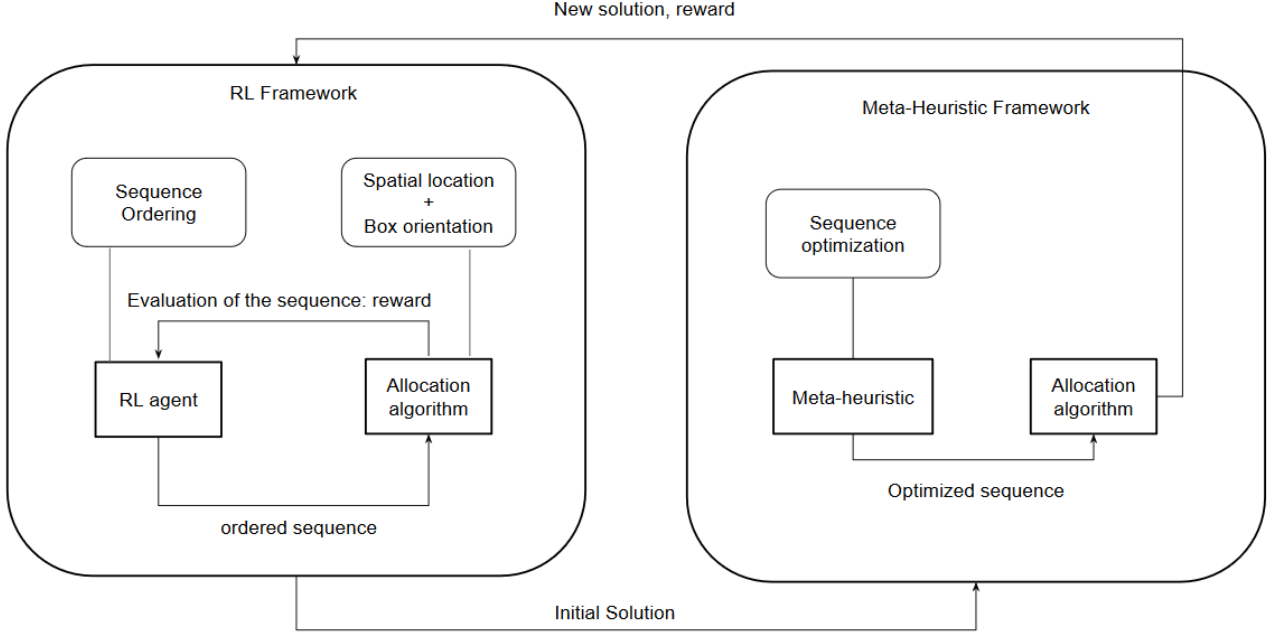


Figure 4.2: Proposed Solution Components

agent. The generated sequence by the RL agent will serve as an initial solution for the SA algorithm. The role of the meta-heuristic is to optimize further the sequence. SA is run for a certain number of steps, evaluating the quality of the solution after each step. At the end of the execution of SA, the final sequence is evaluated and a second reward signal is propagated to the RL agent.

#### 4.3.1.1 Environment Settings

During this project, we had to implement the bin packing environment from scratch. We use the OpenAI Gym library as a toolkit to develop the environment. It allows us to build customized environments and it is compatible with any numerical computation library.

When implementing the dynamics of the environment, we tried to keep the main ideas presented in the RLHO paper. In the original paper, we start from a random association of the items into the bins. Then, at every timestamp, the environment presents the agent with an item  $i$ . The agent then needs to decide which other item  $j$  to swap locations with item  $i$  based on the current state. In our implementation, we start from a random sequence of boxes, the environment presents the agent with a randomly selected box from the sequence, and the agent decides which other item to swap locations with in the sequence.

Figure 4.3 explains the dynamics of the environment. Here, we consider an instance composed

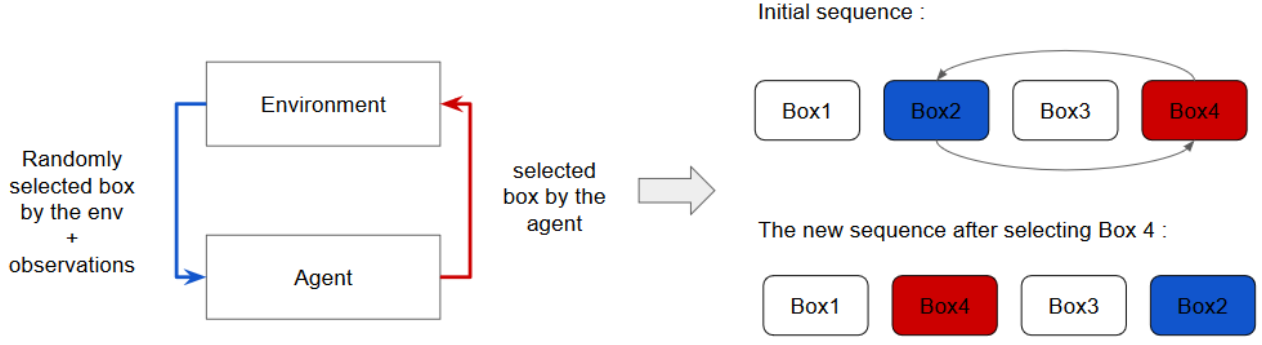


Figure 4.3: The environment dynamics

of 4 boxes, the initial sequence order is randomly generated:  $(Box1, Box2, Box3, Box4)$ . The environment selects randomly  $Box2$ . Based on the current state, the agent selects  $Box4$ , this results in a new sequence in which  $Box2$  and  $Box4$  are swapped  $(Box1, Box4, Box3, Box2)$ .

In the following, we will define the observations, the actions and the reward used in this environment.

- **Observations:** They are composed of the dimensions of the boxes and their positions in the sequence, the dimensions and the position of a randomly chosen box by the environment
- **Action:** An action can be defined as selecting a box from the input sequence and swap its position with the position of the randomly chosen box by the environment
- **Reward:** We define a score to evaluate a solution:  $XX.YYY$ , where the integer part  $XX$  is the number of containers used including the last one, and the decimal part  $YYY$  is the percentage of the used volume in the last bin. We propose the negative of this score as a reward. Intuitively, the lower the score is the better the solution quality is.

#### 4.3.1.2 RLHO Environment Changes

In the original paper, for the design of the reward function, they defined the intermediate reward as the difference between the cost of the previous solution and the cost of the current solution, meaning that the solution is evaluated after every timestep, and the agent receives a reward for every action it takes. In a 3D bin packing environment, it's almost impossible to evaluate the solution in every timestep because of the massive calculations it requires. The

generation and the update of the list of the Empty Maximal Spaces that happens after every time a box is put in the container is very time consuming. Using dense rewards is not an option in this case and we opted to use sparse rewards, where the agent receives a single reward at the end of an episode. We should note that an episode is a sequence of states, actions and rewards  $(S_1, A_1, R_1 \dots S_n, A_n, R_n)$ , which ends with terminal state. A terminal state marks the end of the agent's work.

As a first step, we wanted to validate that the RL agent was able to learn strategies in the implemented environment. We run an experiment in which we tried to optimize an instance using uniquely the RL framework and we noticed that the agent was finding difficulties optimizing the container. The learning process was not stable and the mean reward had high variance mainly because of the fact that the agent was presented with a randomly selected box by the environment, and the agent had control only over the item with which positions will be swapped. We changed the environment in a way that now, at every timestep, the agent selects the 2 items and swaps their positions. The environment no longer selects randomly one of the boxes. Figure 4.4 is a comparison of the performance of an agent (mean reward) before (left) and after (right) the environment changes.

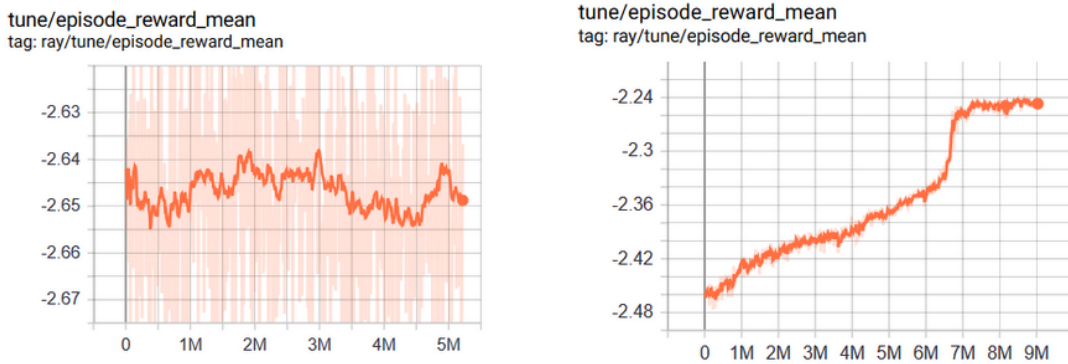


Figure 4.4: Comparison between the performance of an agent before (left) and after (right) the environment changes

#### 4.3.1.3 Combining RL and SA

Once we made sure that our reinforcement learning agent is working properly, we proceeded to combine RL and SA. The agent interacts with the environment, taking actions and receiving a non-zero reward at the end of an episode. The generated solution is then used as an initial solution for SA. SA is run for a certain number of steps, applying random perturbations to the sequence. The final solution is then evaluated and a second reward signal is propagated to the

RL agent. This marks the end of one iteration in the RLHO framework. At this point, the agent can update its parameters using the sum of the two reward signals.

#### 4.3.1.4 Challenges Facing RLHO

When testing the implemented RLHO framework, we have noticed that there are some challenges that are restraining the framework from reaching good solutions. In the following, we mention what we assess to be the most important obstacles:

- The RLHO framework is very slow to train. Simulated Annealing required the evaluation of the solution at every timestep, hence slowing the training process because of the time consuming evaluation step.
- RLHO is sensitive to hyperparameters. It is hard to find the optimal hyperparameters (number of RL steps and SA steps in one RLHO iteration, temperature for SA). Moreover, large instances required more RL and SA steps. We noticed that sometimes RLHO failed to converge when setting a high number of steps.
- The rotation heuristic was certainly limiting the performance of the framework. RLHO was unable to build all of the possible solutions, resulting in a partial search space.
- Working on large instances resulted in a high number of possible actions for the RL agent, as at every timestep, the agent selects 2 boxes from the set of all the boxes in the instance, and thus, the number of possible actions is equal to the number of boxes in the instance. This presented a major difficulty for the RL agent to learn good strategies.
- The role of the agent is to optimize the sequence of boxes, the coordinates of the boxes and their rotations are only known at the end of the episode when we transform the sequence into a packing solution. The observations the RL agent is receiving are really sparse and do not describe the state of the environment.

#### 4.3.2 Combining Reinforcement Learning and Placement Strategies

We developed this approach mainly to tackle the challenges faced in the RLHO approach. The idea is that we use placement strategies like Deepest-Bottom-Left introduced in 2.4.2.2 to select the Empty Maximal Space, and it's up for the agent to decide which type of box and its orientation to fill the selected EMS with. At every timestep, the agent selects a box type

and its rotation, the process is repeated until all of the boxes are packed. We will refer to this approach as **Reinforcement Learning with Placement Strategies (RLPS)**. We should note that this is a constructive approach: At every step a box selected by the agent is put in the EMS selected by the placement strategy, the process is repeated until a complete solution to the problem is reached. The **RLPS** approach succeeds to overcome all of the challenges we mentioned previously:

- The algorithm is much faster and only one solution evaluation is required at the end of an episode
- There are fewer hyperparameters to tune
- The agent has the freedom to select the orientation of the boxes. It has been theoretically proven in previous works, that the Deepest-Bottom-Left heuristic is capable of reconstructing optimal solutions and hence, the search space is now complete.
- The number of possible actions is significantly reduced, especially for the homogeneous instances (Number of actions is equal to the number of types of boxes).
- Since this is a constructive approach, the observations can be enriched. At every step, we possess information about the coordinates of the boxes, their orientations, the state of the container etc.

### 4.3.3 RLPS Environment Settings

We will define actions, observations and reward in this environment, then we will provide the pseudo-code of this constructive approach.

- **Observations:**
  - Information about the bin: length, height, width, volume, remaining empty space
  - Information about the selected EMS: length, height, width, origin coordinates, volume
  - Information about each box: length, height, width, origin coordinates, volume
- **Action:** Select one type of boxes and its orientation. The selected box is placed in the selected EMS. An action mask is provided to the agent so that it can only select valid



actions. An action is considered valid if there is still at least one box that needs to be packed of the selected type and the box in the selected orientation must fit in the EMS.

- **Reward:** We use the same score and reward we defined previously.

---

**Algorithm 2:** RLPS Trajectory Sampling Pseudo-Code

---

```

init
    initialize list_of_boxes;
    initialize ems_list;
    initialize selected_ems;
    initialize observations;
while list_of_boxes is not empty do
    selected_box ← agent.select_box(observations);
    update selected_box_info();
    list_of_boxes.remove(selected_box);
    update container_info() ;
    update ems_list();
    selected_ems ← Deepest_Bottom_Left_heuristic(ems_list);
    update_observations() ;
    if list_of_boxes is empty then
        | reward ← evaluate_solution() ;
    else
        | reward ← 0 ;
    end
end

```

---

### 4.3.4 Environment Improvements

In the following, we introduce a few tricks that were used to enhance the agent's performance.

#### 4.3.4.1 Empty Maximal Spaces Correction

The Bin Packing Problem we are considering is characterized by a few constraints, one of them is the vertical stability. In our case, a box can be packed in the container if and only if the bottom surface of the box is either supported by the ground of the container or is **fully** supported by boxes underneath it. This constraint makes the problem harder to solve. It should

be recalled that we are using Empty Maximal Spaces (EMSs) to handle the spatial location in the bin. One particularity of EMSs, is that, when putting a box inside an EMS, the box's origin will have the same EMS's origin coordinates. Thus, so that an EMS is able to host a box, its origin must be supported by a surface. We have noticed that many of the generated EMSs, have an origin that is not supported by any surface, which makes the EMS totally useless and incapable of hosting boxes at that timestep. We introduced an EMS correction algorithm, that loops over the generated EMSs after each step, and corrects them in a way, that their origin is now supported by a surface. This procedure is deemed to have a huge impact on the agent's performance. In figure 4.5, on the left is an image of an EMS (blue) before the correction step. Despite the fact that there is a box underneath this EMS and there is plenty of space to put a box in this EMS, the EMS's origin is not supported by any surface which makes it unusable. On the right, is an image of the same EMS after the correction step. Modifications were made to the EMS dimensions and origin, so that now it is possible to put a box in the EMS.

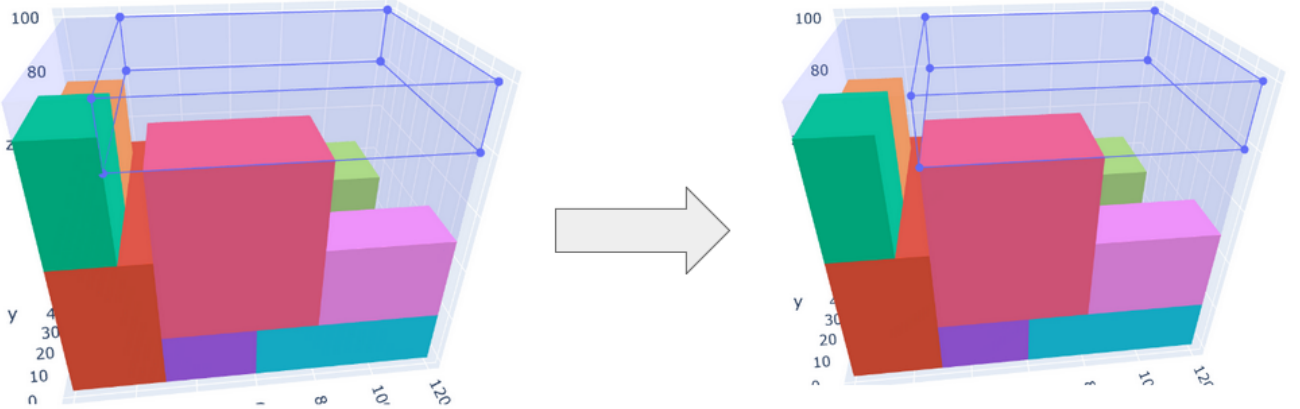


Figure 4.5: EMS before (left) and after (right) the EMS correction step

#### 4.3.4.2 Long Short-Term Memory For Partially Observable MDPs

In reinforcement learning, we need to be concerned with the Markov Property. The state must provides enough information, so that the agent can accurately predict expected next rewards and next states given an action, without the need for any additional information. This does not need to be perfect, it may make the agent less optimal overall, but the theory will still work. Our state representation is not perfect. The bin packing task requires a memory of the past states and actions and it will appear non-Markovian because the future states and rewards depend on more than just the agent current input. Instead of a Markov Decision Process

(MDP), the environment becomes a Partially-Observable Markov Decision Process (POMDP). Providing the agent with observations that accurately describe the environment in our bin packing problem will result in an extremely huge number of states, which will prevent the agent from learning.

In order to construct a better state from available data, we tried to use techniques designed for partially observable MDPs that effectively try to build missing parts of state data statistically. We used an LSTM (Long Short-Term Memory) cell in the neural network architecture as a memory mechanism, allowing the agent to remember past actions and states. The use of an LSTM cell resulted in an overall better performance. Figure 4.6 represents the neural network architecture we used. An input vector is fed to the neural network, then it is processed with a fully connected layer that uses Rectifier Linear Unit as an activation function. The resulted features representation is then fed to an LSTM cell. The output of the LSTM cell is post-processed with another fully connected layer that uses a Softmax distribution to yield the action distribution.

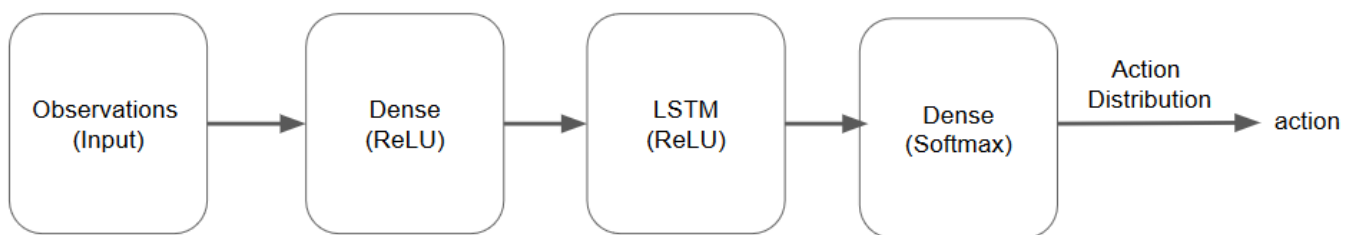


Figure 4.6: Neural Network Architecture

## Conclusion

In this chapter, we first introduced the work environment, then, we presented an overview of the RLHO framework. We then presented a detailed description of the environment and the challenges facing this framework. We then introduced an RL based alternative solution that we called RLPS and we described the dynamics of the implemented environment. We finished by presenting techniques that helped improving the agent's performance. In the next chapter, we will discover how efficient the proposed solutions are compared to a few powerful heuristics and previous InstaDeep's solutions.

# Chapter 5

## Results and Discussion

### Introduction

In this chapter, we present the achieved work. We start by introducing the data we used for training and testing, then, we present the results achieved by the two developed approaches. We also compare the results to state-of-the-art heuristics, in the end, we give a graphic representation of the solutions and wrap it all up with a brief discussion.

### 5.1 | Data set Overview

First of all, it is noteworthy to mention that here we are not referring to the same type of data used in supervised learning. We are actually referring to a set of problems that we will call instances, and these instances are generated by the environment. In the 3D bin packing literature, it is common to use the Bischoff and Ratcliff data sets [4]. These data sets consist of 15 sets of 100 instances each (commonly referred to as BR 1-15), ranging from the most homogeneous data set (BR1, low number of box shapes) to the most heterogeneous data set (BR 15, high number of box shapes). The goal of this project is not to build a model capable of achieving good results on all of the instances, as generalization is out of the scope of this project, but rather, to design an algorithm capable of reaching good results on very specific instances for a customer's needs. Mainly, we used 4 academic instances that differ in terms of the number of items and the number of shapes. These instances represent both of the homogeneous case and the heterogeneous case. In the following table we give a name and present the number of items and shapes in each instance:

Instance	Number of items	Number of shapes
Instance_1	214	3
Instance_2	124	50
Instance_3	1156	3
Instance_4	1280	100

Table 5.1: The four academic instances

Instance\_1 is instance 100 from BR1. It represents the homogeneous case. Instance\_2 is instance 100 from BR10. It represents the heterogeneous case. Instance\_3 and Instance\_4 are 2 academic instances that were provided by the client. Our goal is to get the best possible score on these 4 instances.

## 5.2 Experimental Results

In this section, we present the results of the RLHO approach and the RLPS approach.

### 5.2.1 RLHO Results

First of all, it is worth noting that, when reporting all of the following results, we are using the score we introduced in 4.3.1.1. The integer part is the number of containers used including the last one, and the decimal part is the percentage of the used volume in the last bin. Intuitively, the lower the score is, the better it is considered.

For the RLHO approach, we conducted experiments only on **Instance\_1**. We set the number of permutations for the RL agent to 100, and the number of simulated annealing steps to 500. Mainly, we conducted two experiments, the first one using only pure reinforcement learning, and the second one using the whole RLHO framework where we combine reinforcement learning with simulated annealing. Surprisingly, in the two experiments we got the same score of **2.13**. The score is considered good compared to the classic methods, but a bit far from the best performing heuristics. We speculate that using simulated annealing did not help the RL agent to find better solutions, because of the limiting capabilities of the orientation heuristic, as well as to the small number of simulated annealing steps. Setting a high number of SA steps is impossible, as the training takes extremely long time with high number of SA steps. At this

point, we decided to abandon the RLHO approach, as it seems to be not practical and limiting.

### 5.2.2 RLPS Results

In this section, we report the scores obtained by the RLPS approach. We compare the results to state-of-the-art heuristics, as well as to two approaches developed in InstaDeep, the first one is a simple **PPO** algorithm and the second one we will call it **Complex Sequential PPO with BTL heuristic**. These two approaches are two Reinforcement Learning based approaches: the agent selects the EMS, the Box and its orientation. The main difference between the two approaches is that Complex SPPO with BTL heuristic sometimes uses a heuristic to take actions instead of the agent, and the agent is capable of learning from the actions taken by the heuristic, while PPO is a pure RL solution. The following table represents a comparison of the different approaches. We are using the same score we mentioned above.

Score	Instance_1	Instance_2	Instance_3	Instance_4
<b>LEGO</b>	2.346	2.235	12.515	14.056
<b>DFTRC</b>	2.27	3.016	12.257	18.31
<b>BBL</b>	2.201	2.4	12.01	16.022
<b>BTL</b>	2.177	2.462	12.03	15.512
<b>HA</b>	2.20	2.23	12.57	14.20
<b>PPO</b>	2.13	2.164	11.430	18.25
<b>Complex SPPO with BTL heuristic</b>	2.101	2.115	11.35	13.39
<b>RLPS</b>	<b>2.075</b>	<b>2.125</b>	<b>11.47</b>	<b>12.61</b>

Table 5.2: Results of RLPS compared to heuristics and other approaches

### 5.2.3 Environment Improvements Impact

In this section, we will demonstrate the effect of the improvements we conducted on the environment and presented in 4.3.4, mainly, the EMS correction step and the use of an LSTM cell in the neural network architecture.

### 5.2.3.1 EMS Correction Step Impact

Score	Without EMS correction	With EMS correction
<b>Random Agent</b>	3.185	2.314
<b>Trained RL agent</b>	2.240	2.115

Table 5.3: EMS correction step impact demonstrated on Instance\_1

The EMS correction step was not only yielding better results in terms of score, but it had also accelerated the training phase. This is probably due to the fact that, when training an agent without the EMS correction step, a lot of time is wasted on trying to find the best EMS (according to the heuristic used) that respect the vertical stability constraint. On the other hand, when using the EMS correction step, most of the created EMSs respect the vertical stability constraint, hence, finding the best EMS becomes faster.

### 5.2.3.2 LSTM Cell Use Impact

The use of an LSTM cell in the neural network architecture to "memorize" past actions and observations seen by the agent has a considerable impact on the results. We trained 2 reinforcement learning agents using the RLPS approach, one **without** the use of an LSTM cell and the other **with** the use of an LSTM cell. In both experiments, we are applying the EMS correction step. In the following table, we report the results we found.

Score	Instance_1	Instance_2	Instance_3
<b>RLPS</b> without LSTM cell	2.115	2.160	11.525
<b>RLPS</b> with LSTM cell	2.075	2.125	11.470

Table 5.4: Comparison of the RLPS approach with and without LSTM cell use

### 5.2.4 RLPS on Unseen Data

Although generalization is out of the context of this project, we wanted to make sure that the strategies learnt by the agents can be transferred and used for other instances. For each of the 4 instances we mentioned previously, we trained a separate agent. The unique goal of each agent was to solve the particular instance it's optimizing, so, during the training phase, each agent was seeing only one instance and was over-fitting to solve the instance. To conclude whether the strategies the agents are learning can be useful to solve new instances, we considered the two agents that were trained on **Instance\_1** and **Instance\_2** and compared their performance to a random agent on randomly selected instances respectively from the BR1 dataset and BR10 dataset. The following two tables display the results. We run the agents 200 times on each instance and we report the obtained score averaged over these 200 tries.

Instance	RLPS trained on Instance_1	Random Agent
BR1_instance1	2.2504	2.3465
BR1_instance10	2.2484	2.3866
BR1_instance53	2.1329	2.3932
BR1_instance92	2.3120	2.4046
BR1_instance45	2.1528	2.3889
BR1_instance38	2.2507	2.3843

Table 5.5: Comparison of RLPS approach agent to a random agent on BR1 instances

Instance	RLPS trained on Instance_2	Random Agent
BR10_instance1	2.2596	2.4296
BR10_instance23	2.3075	2.4290
BR10_instance47	2.2922	2.4123
BR10_instance60	2.2943	2.4082
BR10_instance73	2.2733	2.4290
BR10_instance87	2.3060	2.4298

Table 5.6: Comparison of RLPS approach agent to a random agent on BR10 instances

The reported experiments prove that, although the agents were trained using only one in-



stance, they can perform considerably better than a random agent on unseen instances, and the learnt strategies can be transferred to solve new instances. With a proper input normalization during the training phase and an adequate reward function, generalization shouldn't be a problem for the RLPS approach.

### 5.2.5 Container Visualization

For debugging purposes, we developed a visualization tool that enable us to see how the boxes are placed inside the containers. We present 4 figures displaying the state of the first container respectively of Instance\_1, Instance\_2, Instance\_3 and Instance\_4.

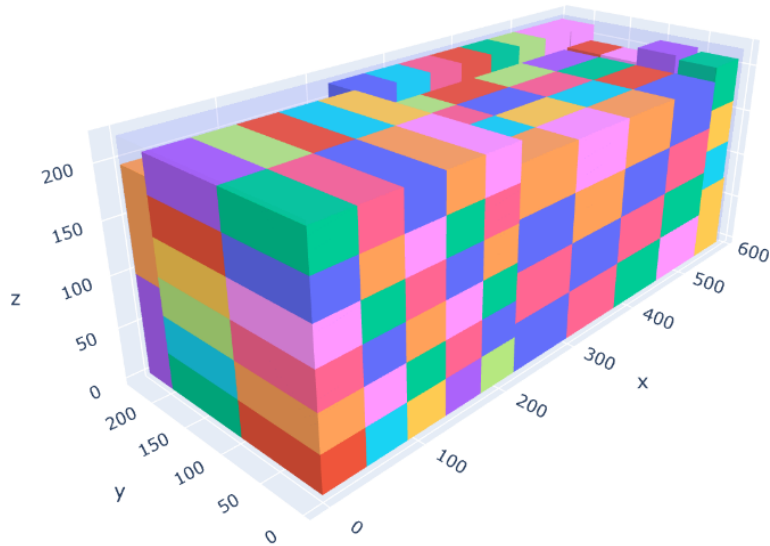


Figure 5.1: The first container of Instance\_1

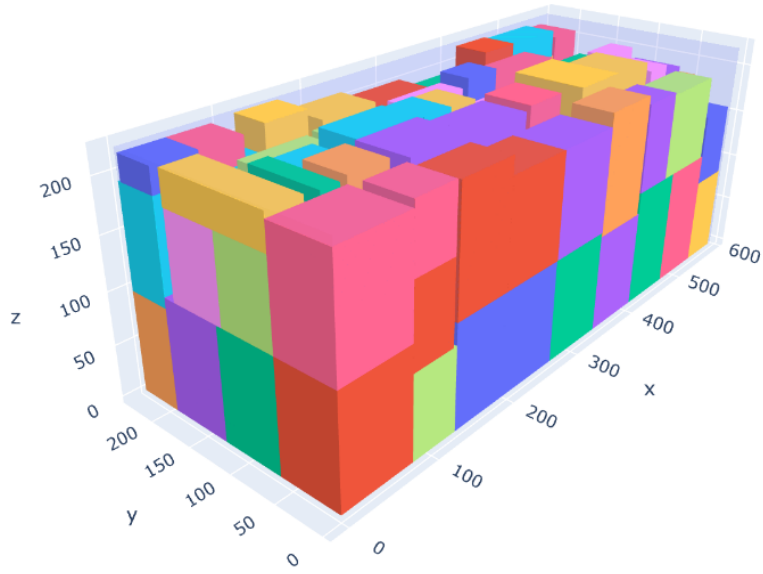


Figure 5.2: The first container of Instance\_2

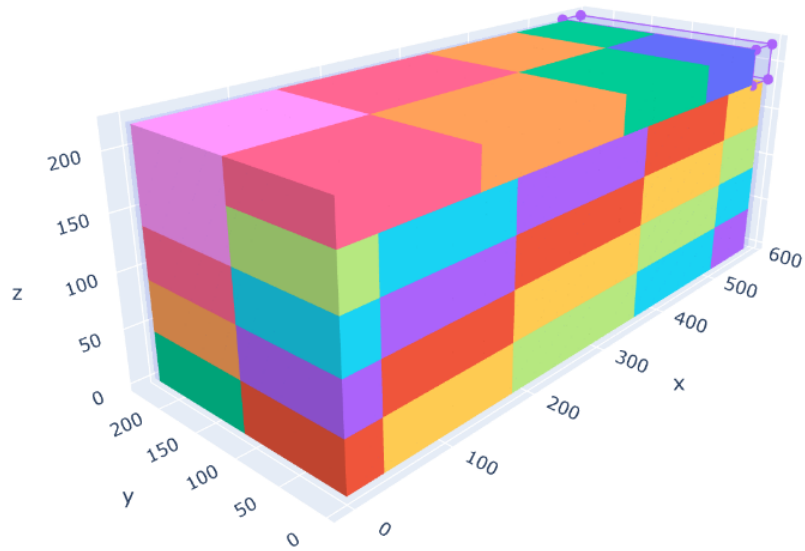


Figure 5.3: The first container of Instance\_3

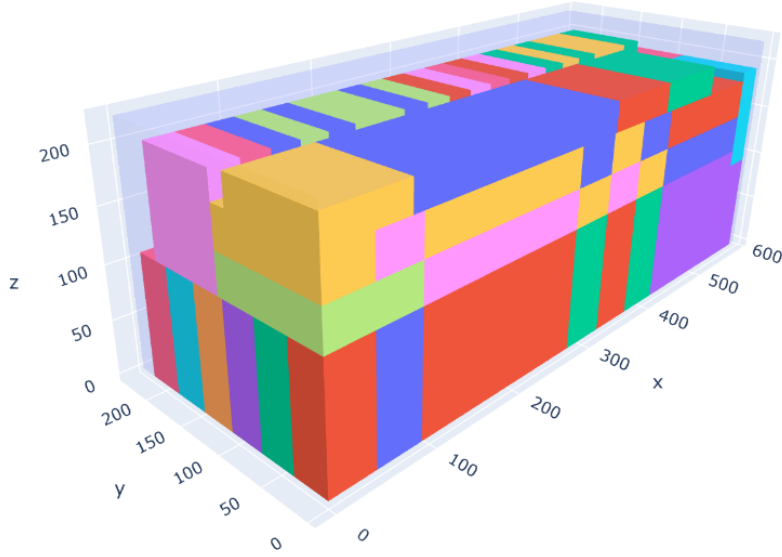


Figure 5.4: The first container of Instance\_4

### 5.3 Discussion

Despite the fact that the **RLHO** approach offers a novel and promising framework to solve combinatorial problems, it seems hard to apply it on the 3D bin packing problem, and this is mainly due to the high amount of calculations involved and its inability to construct all of the possible solutions. However, it would be interesting to see this framework applied on the 1D bin packing problem or in other combinatorial problems such as the travelling salesman problem.

In the other hand, it is quite amazing to see the **RLPS** approach doing well. It has outperformed all of the heuristics we tested on the 4 instances (LEGO, DFTRC, BBL, BTL and HA). Thanks to the improvements we carried out on the environment, it has achieved exciting results, outperforming simple **PPO**, and attaining better scores on 2 instances when compared to **Complex SPPO with BTL heuristic**, and very competitive scores on the other 2 instances. With a better hyperparameters setting and a few improvements, we expect the RLPS approach to perform even better and attain superior scores on all of the instances.

When it comes to the training time and sample efficiency, it is hard to compare these approaches, as the hardware setup (mainly the number of CPU cores and GPUs) is different. Using the RLPS approach, it takes the algorithm around 8 hours on average to solve one instance. Nevertheless, we should keep in mind that, although reinforcement learning systems require a lot of training time, when it comes to inference, a trained agent usually solve new

instances in a matter of seconds.

## Conclusion

In this chapter, we introduced the dataset we worked on and benchmarked our results against state-of-the-art heuristics and other methods. We had also presented the impact of the improvements we implemented and provided a visualization of the results we obtained. Fortunately, our approach has shown a great success in solving the 3D bin packing problem and outperformed most of the aforementioned methods. In the next chapter, we will conclude our work and open perspectives.

# General Conclusion

Throughout this report, we described the steps taken to design and implement a successful reinforcement learning approach to solve the bin packing problem and demonstrated that reinforcement learning approaches are competitive alternatives to heuristics. Without a doubt, this internship was insightful and instructive. Having the opportunity to work with such a skilled team was definitely fruitful and we learned a lot from this great experience at InstaDeep.

We have begun our project by understanding the bin packing problem and its different variants, and then spent a considerable amount of time studying the required theoretical knowledge in order to guarantee a solid understanding of the problem. In the next phase, we described some state-of-the-art methods that had been used to solve our problem. This phase allowed us to get a sound knowledge of what methods might work, and an in-depth understanding of the pros and cons of each method. Next, we presented an overview of the RLHO approach, and its implementation details. Then, we decided to drop it because of the numerous challenges this method failed to solve, and opted for a reinforcement learning based method combined with placement strategies. During this phase, we described the implementation details and the characteristics of each method. This part was particularly challenging technically and theoretically, but in the end we managed to handle it successfully. Finally, we introduced the dataset we worked on, presented the experimental results, and benchmarked our results against other algorithms.

Our implementation succeeded in tackling the 3D bin packing problem and it yielded competitive results, surpassing the old approach implemented in the host company on some instances. It had proved to be efficient on both, the homogeneous and the heterogeneous versions of the problem. The code was documented in a way that, new members who would work on improving our approach, would find it easy to read, to understand, to scale and to maintain it.

Our implementation is considered to be a first version. Various improvements can be added. Mainly, we could try other reward functions, or use another concept to manage the space inside the containers, such as Extreme Points. Moreover, to successfully build a system capable of

solving real world bin packing instances, a generalization process must be conducted, training over different distributions must be carried out, and other neural network architectures and hyperparameters must be tested.

Finally, we believe that reinforcement learning has a huge potential to solve real world problems when used properly. We hope to see this field thriving in the future and successfully solving other complex real world problems

# Bibliography

- [B1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio, "Neural Combinatorial Optimization with Reinforcement Learning" arXiv:1611.09940
- [B2] Olyvia Kundu ; Samrat Dutta ; Swagat Kumar, "Deep-Pack: A Vision-Based 2D Online Bin Packing Algorithm with Deep Reinforcement Learning", 2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)
- [B3] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, Yinghui Xu, "Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method" arXiv:1708.05930
- [B4] Qingpeng Cai, Will Hang, Azalia Mirhoseini, George Tucker, Jingtao Wang, Wei Wei "Reinforcement Learning Driven Heuristic Optimization" arXiv:1906.06639
- [B5] Richard S. Sutton and Andrew G. Barto "Reinforcement Learning: An Introduction" <http://incompleteideas.net/book/the-book-2nd.html>

# Netography

- [1] <https://www.manager-go.com/marketing/test-and-learn.htm> (Last visit 25/06/2020 )
- [2] <https://www.sciencedirect.com/science/article/abs/pii/S0360835296002057> (Last visit 25/06/2020 )
- [3] <http://dx.doi.org/10.1016/j.ijpe.2013.04.019> (Last visit 25/06/2020 )
- [4] <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/thpackinfo.html> (Last visit 1/9/2020 )
- [5] [https://www.researchgate.net/publication/273121476\\_A\\_genetic\\_algorithm\\_for\\_the\\_three-dimensional\\_bin\\_packing\\_problem\\_with\\_heterogeneous\\_bins](https://www.researchgate.net/publication/273121476_A_genetic_algorithm_for_the_three-dimensional_bin_packing_problem_with_heterogeneous_bins) (Last visit 26/06/2020 )
- [6] <https://arxiv.org/abs/1611.09940> (Last visit 13/07/2020 )
- [7] <https://www.groundai.com/project/a-multi-task-selected-learning-approach-for-solving-new-type-3d-bin-packing-problem/1> (Last visit 1/09/2020 )
- [8] [https://www.researchgate.net/publication/222401651\\_Heuristic\\_algorithms\\_for\\_the\\_three-dimensional\\_bin\\_packing\\_problem](https://www.researchgate.net/publication/222401651_Heuristic_algorithms_for_the_three-dimensional_bin_packing_problem) (Last visit 1/09/2020 )
- [9] <http://openscholar.dut.ac.za/handle/10321/3180> (Last visit 1/09/2020 )
- [10] <https://www.udacity.com/course/deep-reinforcement-learning-nanodegree-nd893> ) (Last visit 1/09/2020 )



# Appendix

In the appendix, we introduce the Proximal Policy Optimization (PPO) algorithm and specify the key differences between PPO and REINFORCE, which is a classic policy gradient method.

## Beyond REINFORCE

PPO is a policy gradient method that improved the REINFORCE algorithm, which is one of the first policy gradient methods in reinforcement learning. So in order to properly introduce the PPO algorithm, we first need to present briefly the REINFORCE algorithm and its logic.

REINFORCE works as follows:

First, we initialize a random policy  $\pi_\theta(a_t | s_t)$  and using the policy we collect a trajectory which basically a list of (state, actions, rewards) at each time step:

$$s_1, a_1, r_1, s_2, a_2, r_2, \dots \quad (5.1)$$

Second, we compute the total reward of the trajectory  $R = r_1 + r_1 + r_1 + \dots$  and compute an estimate the gradient of the expected reward,  $g$ :

$$g = R \sum_t \nabla_\theta \log \pi_\theta(a_t | s_t) \quad (5.2)$$

Third, we update our policy using gradient ascent with learning rate  $\alpha$ :

$$\theta \leftarrow \theta + \alpha g \quad (5.3)$$

The REINFORCE algorithm has a few issues:

1. The update process is very inefficient. We run the policy once, update once, and then throw away the trajectory.

2. The gradient estimate  $g$  is very noisy. By chance the collected trajectory may not be representative of the policy.
3. There is no clear credit assignment. A trajectory may contain many good/bad actions and whether these actions are reinforced depends only on the final total output.

The PPO algorithm improve the REINFORCE algorithm by resoloving these 3 issues. In the following, we present how PPO overcomes these problems.

### • Noise Reduction

The easiest option to reduce the noise in the gradient is to simply sample more trajectories. Using distributed computing, we can collect multiple trajectories in parallel, so that it won't take too much time. Then we can estimate the policy gradient by averaging across all the different trajectories.

$$\left. \begin{array}{l} s_t^{(1)}, a_t^{(1)}, r_t^{(1)} \\ s_t^{(2)}, a_t^{(2)}, r_t^{(2)} \\ s_t^{(3)}, a_t^{(3)}, r_t^{(3)} \\ \vdots \end{array} \right\} g = \frac{1}{N} \sum_{i=1}^N R_i \sum_t \nabla_{\theta} \log \pi_{\theta} \left( a_t^{(i)} \mid s_t^{(i)} \right) \quad (5.4)$$

Another advantage for running multiple trajectories is that we can collect all the total rewards and get a sense of how they are distributed. Learning can be improved if we normalize the rewards, where  $\mu$  is the mean, and  $\sigma$  the standard deviation.

$$R_i \leftarrow \frac{R_i - \mu}{\sigma} \quad \mu = \frac{1}{N} \sum_i R_i \quad \sigma = \sqrt{\frac{1}{N} \sum_i (R_i - \mu)^2} \quad (5.5)$$

### • Credit Assignment

The total reward  $R$ , is just a sum of reward at each step:

$$R = r_1 + r_1 + r_1 + \dots \quad (5.6)$$

At time step  $t$ , even before an action is decided, the agent has already received all the rewards up until step  $t - 1$ . So we can think of that part of the total reward as the reward from the past. The rest is denoted as the future reward.

$$\overbrace{(\dots + r_{t-1} + r_t + \dots)}^{R_t^{\text{past}}} \quad \overbrace{(\dots + r_t + \dots)}^{R_t^{\text{future}}} \quad (5.7)$$

Because we have a Markov process, the action at time-step  $t$  can only affect the future reward, so the past reward shouldn't be contributing to the policy gradient. So to properly assign credit to the action  $a_t$ , we should ignore the past reward. So a better policy gradient would simply have the future reward as the coefficient .

$$g = \sum_t R_t^{\text{future}} \nabla_{\theta} \log \pi_{\theta} (a_t \mid s_t) \quad (5.8)$$

## • Importance Sampling

After the update, the trajectories we've just generated are simply thrown away. If we want to update our policy again, we would need to generate new trajectories once more, using the updated policy. This sounds a little wasteful. The trajectories we generated using the policy  $\pi_{\theta}$  had a probability  $P(\tau; \theta)$  to be sampled. Just by chance, the same trajectory can be sampled under the new policy, with a different probability  $P(\tau; \theta')$ . If we want to compute the average of some quantity, say  $f(\tau)$ . We could simply generate trajectories from the new policy, compute  $f(\tau)$  and average them.

$$\sum_{\tau} P(\tau; \theta') f(\tau) \quad (5.9)$$

Now we could modify this equation, by multiplying and dividing by the same number,  $P(\tau; \theta)$  and rearrange the terms.

$$\sum_{\tau} \overbrace{P(\tau; \theta)}^{\text{sampling-under- old-policy}} \quad \overbrace{\frac{P(\tau; \theta')}{P(\tau; \theta)}}^{\text{re-weighting factor}} f(\tau) \quad (5.10)$$

Intuitively, this tells us we can use old trajectories for computing averages for new policy, as long as we add this extra re-weighting factor, that takes into account how under or over-represented each trajectory is under the new policy compared to the old one.

The re-weighting factor can be expressed a chain of products of each policy at different time-step:

$$\frac{P(\tau; \theta')}{P(\tau; \theta)} = \frac{\pi_{\theta'}(a_1 \mid s_1) \pi_{\theta'}(a_2 \mid s_2) \pi_{\theta'}(a_3 \mid s_3) \dots}{\pi_{\theta}(a_1 \mid s_1) \pi_{\theta}(a_2 \mid s_2) \pi_{\theta}(a_2 \mid s_2) \dots} \quad (5.11)$$

In practice, we want to make sure the re-weighting factor is not too far from 1 when we utilize importance sampling.

## • The Surrogate Function

Let's suppose we are trying to update our current policy,  $\pi_{\theta'}$ . To do that, we need to estimate a gradient,  $g$ . But we only have trajectories generated by an older policy  $\pi_{\theta}$ . Mathematically, we could utilize importance sampling to compute the gradient. A normal policy gradient can be expressed as:

$$g = \frac{P(\tau; \theta')}{P(\tau; \theta)} \sum_t \frac{\nabla_{\theta'} \pi_{\theta'}(a_t | s_t)}{\pi_{\theta'}(a_t | s_t)} R_t^{\text{future}} \quad (5.12)$$

We can rearrange these equations, and the re-weighting factor is just the product of all the policy across each step. We can cancel some terms, but we're still left with a product of the policies at different times, denoted by "...".

$$g = \sum_t \frac{\cdots \pi_{\theta'}(a_t | s_t) \cdots \nabla_{\theta'} \pi_{\theta'}(a_t | s_t)}{\cdots \pi_{\theta}(a_t | s_t) \cdots \pi_{\theta'}(a_t | s_t)} R_t^{\text{future}} \quad (5.13)$$

This is where proximal policy comes in. If the old and current policy is close enough to each other, all the factors inside the "..." would be pretty close to 1, and then we can ignore them. Then the equation simplifies

$$g = \sum_t \frac{\nabla_{\theta'} \pi_{\theta'}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} R_t^{\text{future}} \quad (5.14)$$

Now that we have the approximate form of the gradient, we can think of it as the gradient of a new object, called the surrogate function.

$$\begin{aligned} g &= \nabla_{\theta'} L_{\text{sur}}(\theta', \theta) \\ L_{\text{sur}}(\theta', \theta) &= \sum_t \frac{\pi_{\theta'}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} R_t^{\text{future}} \end{aligned} \quad (5.15)$$

## • Clipping Policy Updates

If we keep reusing old trajectories and updating our policy, at some point the new policy might become different enough from the old one, so that all the approximations we made could become invalid. Updating our policy and ignoring the fact that the approximations are not valid anymore can lead to a really bad policy that is very hard to recover from.

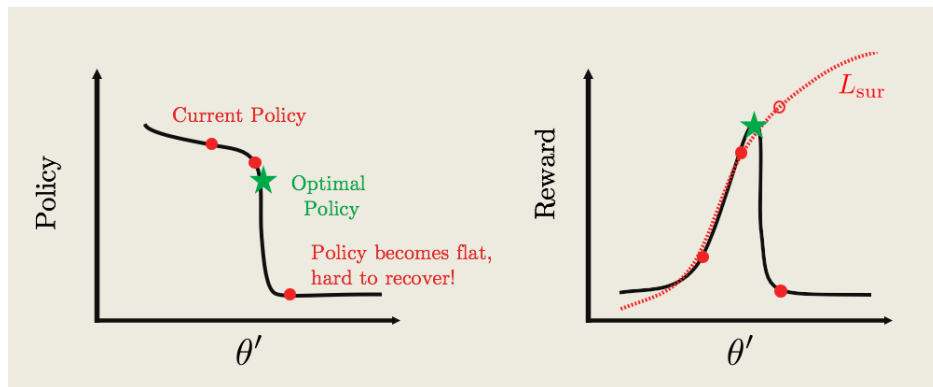


Figure 5.5: The Policy/Reward Cliff

Say we have some policy parameterized by  $\pi_{\theta'}$  (shown on the left plot in black) in figure 5.5, and with an average reward function (shown on the right plot in black).

The current policy is labelled by the red text, and the goal is to update the current policy to the optimal one (green star). To update the policy we can compute a surrogate function  $L_{sur}$  (dotted-red curve on right plot). So  $L_{sur}$  approximates the reward pretty well around the current policy. But far away from the current policy, it diverges from the actual reward.

If we continually update the policy by performing gradient ascent, we might get something like the red-dots. The big problem is that at some point we hit a cliff, where the policy changes by a large amount. From the perspective of the surrogate function, the average reward is really great. But the actually average reward is really bad.

What's worse, the policy is now stuck in a deep and flat bottom, so that future updates won't be able to bring the policy back up. we are now stuck with a really bad policy.

To fix this, we can flatten the surrogate function (blue curve) in figure 5.6.

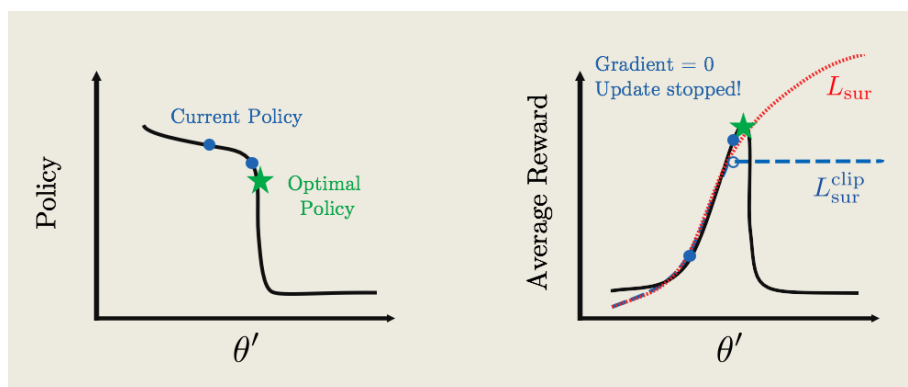


Figure 5.6: Clipped surrogate function

So starting with the current policy (blue dot), we apply gradient ascent. The updates remain the same, until we hit the flat plateau. Now because the reward function is flat, the gradient is zero, and the policy update will stop. The formula that will automatically flatten our surrogate function to avoid all the cliffs:

$$L_{\text{sur}}^{\text{clip}}(\theta', \theta) = \sum_t \min \left\{ \frac{\pi_{\theta'}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} R_t^{\text{future}}, \text{clip}_{\epsilon} \left( \frac{\pi_{\theta'}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} \right) R_t^{\text{future}} \right\} \quad (5.16)$$

We can finally summarize the PPO algorithm:

1. collect some trajectories based on some policy  $\pi_{\theta}$ , and initialize theta prime  $\theta' = \theta$
2. compute the gradient of the clipped surrogate function using the trajectories Update  $\theta'$  using gradient ascent  $\theta' \leftarrow \theta' + \alpha \nabla_{\theta'} L_{\text{sur}}^{\text{clip}}(\theta', \theta)$
3. Repeat step 2-3 without generating new trajectories. Typically, step 2-3 are only repeated a few times
4. Set  $\theta = \theta'$ , go back to step 1, repeat.