UNIVERSIDADE D
COIMBRA

Miguel António Silva Neves

A REINFORCEMENT LEARNING APPLICATION TO AN ASSEMBLY DECISION-MAKING PROBLEM

Dissertação no âmbito do Mestrado Integrado em Engenharia Mecânica, no ramo de produção e projeto orientada pelo Professor Doutor Pedro Mariano Simões Neto e apresentada ao Departamento de Engenharia Mecânica da Universidade de Coimbra.

Setembro de 2020

1 2 9 0

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

A reinforcement learning application to an assembly decision-making problem

Submitted in Partial Fulfilment of the Requirements for the Degree of Master in Mechanical Engineering in the speciality of Production and Project

Aplicação de reinforcement learning num problema de tomada de decisão

Author

Miguel António Silva Neves

Advisor[s]

Pedro Mariano Simões Neto

Jury

President	**Professor Doutor Telmo Miguel Pires Pinto** **Professor Auxiliar da Universidade de Coimbra**
Vowels	**Doutor Miguel Jorge Vieira** **Investigador Doutorado da Universidade de Coimbra** **Doutor Miguel Ângelo Fernandes Castanheiro e Simão** **Investigador Doutorado da Stratio**
Advisor	**Professor Doutor Pedro Mariano Simões Neto** **Professor Auxiliar da Universidade de Coimbra**

Coimbra, September, 2020

ACKNOWLEDGEMENTS

It would not have been possible to write this dissertation without the guidance and support from all the people that accompanied me throughout this journey as a student of Mechanical Engineering at the University of Coimbra.

To Prof. Dr. Pedro Neto, advisor, for all the guidance, enthusiastic encouragement, useful critiques and for all the valuable resources provided throughout this dissertation.

To Dr. Miguel Vieira, for his valuable and constructive suggestions, advice and constant readiness that were crucial to the completion of this work.

To my friends and colleagues in the laboratory, for their friendship and all the support given.

To all my friends that have been by my side on this journey, with whom I have memorable memories and whose friendship I could always count on.

To my parents and my brother, for all the unconditional support given, for their constant patience and their ceaseless encouragement.

To my girlfriend, that has always been by my side during the development of this dissertation and whom I must thank for all the support given and for always believing in me.

Abstract

Reinforcement learning is a methodology with great potential of applicability in manufacturing decision-making problems due to the reduced need of previous training data, i.e., the system learns along time with actual operation.

This dissertation focuses on the implementation of a reinforcement learning algorithm in an assembly decision-making problem of an airplane, from the Yale-CMU-Berkeley Object and Benchmark Dataset, aiming to identify the effectiveness of the proposed approach in the assembly time optimization. There are numerous types of reinforcement learning algorithms, with Q-Learning being the algorithm chosen for this dissertation. This algorithm is based on the learning of a matrix of Q-values (Q-table) from the successive interactions with the environment to find an optimal state-action policy that maximizes the accumulated reward, formalized as a Markov Decision Process (MDP).

This implementation was achieved in three scenarios with increasing complexity. In the first scenario, the reinforcement learning agent could only distinguish between feasible and impossible assembly sequences. In a second scenario the actions' average time were included so that different assembly sequences corresponded to solutions with diverse accumulated rewards. This scenario allowed an initial optimization of the algorithm's parameters and rewards. Finally, in the last scenario, the tasks' average time were measured with the corresponding time variances, so that the assembly sequences would have a larger distribution on accumulated rewards. This last scenario allowed the further optimization of the algorithm's parameters and rewards.

The implemented algorithm, after optimization, achieved very promising results by learning the optimal assembly sequence 95.83% of the times.

Keywords Reinforcement Learning, Q-Learning, Assembly Sequence, Optimization.

Resumo

Reinforcement learning é uma metodologia com grande potencial de aplicabilidade em problemas de tomada de decisões na manufatura devido à reduzida necessidade prévia de dados, isto é, o sistema aprende durante a real operação.

Esta dissertação foca-se na implementação dum algoritmo de *reinforcement learning* num problema de tomada de decisões na montagem de um avião, pertencente ao *dataset* de objetos e *benchmark* de Yale-CMU-Berkeley, com o objetivo de identificar a eficácia da abordagem proposta na otimização dos tempos de montagem. Existem inúmeros algoritmos de *reinforcement learning*, tendo sido o algoritmo *Q-Learning* o escolhido para o trabalho desta dissertação. Este algoritmo baseia-se na aprendizagem duma matriz de *Q-values*, conhecida como *Q-table*, através de sucessivas interações com o ambiente de forma a determinar a *state-action policy* que maximiza as *rewards* acumuladas e formalizada como um *Markov Decision Process* (MDP).

Esta implementação foi conseguida em três cenários distintos, com um nível de complexidade crescente. No primeiro cenário, o *reinforcement learning agent* apenas poderia distinguir entre sequencias de montagem possíveis ou impossíveis. Num segundo cenário os tempos médios de duração das ações foram adicionados com a consequência de diferentes sequências de montagem corresponderem a diferentes soluções com valores de *rewards* acumuladas. Este cenário permitiu uma primeira otimização dos parâmetros e *rewards* do algoritmo. Por fim, no terceiro cenário os tempos médios das ações foram medidos com as respetivas variações, o que tornou a distribuição de *rewards* acumuladas mais dispersas. Este cenário permitiu uma nova otimização dos parâmetros e *rewards* do algoritmo.

O algoritmo implementado, após a sua otimização, apresentou resultados promissores ao aprender a sequência de montagem ótima 95.83% das vezes.

Palavras-chave: Reinforcement Learning, Q-Learning, Sequência de Montagem, Otimização.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF SIMBOLS AND ACRONYMS/ ABBREVIATIONS

List of Symbols

α – Learning rate

α^* – Significance level

ε – Probability of choosing a random action in the Epsilon-Greedy Algorithm

γ – Discount factor

π – Reinforcement learning's policy

π_θ – Reinforcement learning's policy π with parameters θ

a – Action

a_t – Action at time t

$\mathcal{A}$ – Action space

${}_x^y C$ – Combinations of y objects x at a time

E_{π_θ} – Expectation operator by following the policy π_θ

$\mathcal{E}$ – Emission probabilities

$\max_a Q(s_{t+1}, a)$ – Estimate of the optimal future Q-value

n – Sample size

o – Observation

o_t – Observation at time t

$\mathcal{O}$ – Observation space

$p(a|s)$ – Probability of selecting the action a in the state s

$p(s_{t+1}|s_t)$ – Probability of transition at time t from s_t to s_{t+1}

$p(s_{t+1}|s_t, a_t)$ – Probability of transition at time t from s_t to s_{t+1} under a_t

${}_x^y P$ – Permutations of y objects x at a time

Q – Current Q-value

Q^{new} – New Q-value

$Q^\pi(s_t, a_t)$ – Q-function corresponding to the policy π from taking a_t in s_t

$r(s, a)$ – Reward from taking the action a in the state s

r_c – Completion reward

r_m – Reward multiplier

r_p – Reward penalty

r_s – Reward shift

r_t – Reward at time t

$\boldsymbol{R}$ – Matrix of rewards for all states and actions

s^* – Standard deviation of the sample

s – State

s_t – State at time t

s_{t+1} – State after state s_t

$\mathcal{S}$ – State space

$\boldsymbol{T}$ – Matrix of time durations for all states and actions

$\mathcal{T}$ – Transition operator

$t_{\alpha^*/2}$ – Student t value for the significance level of α^*

$V^{\pi}(s_t, a_t)$ – Value function corresponding to the policy π from taking a_t in s_t

x_i – Element i of the sample

$\bar{x}$ – Mean of the sample

Acronyms/Abbreviations

A2C - Advantage Actor-Critic

A3C - Asynchronous Advantage Actor-Critic

D4PG - Distributed Distribution Deep Deterministic Policy Gradient

DDPG - Deep Deterministic Policy Gradient

DJSS - Dynamic Job Shop Scheduling

DNN - Deep Neural Network

DPG - Deterministic Policy Gradient

DQN - Deep Q-Network

GMR - Generative Motor Reflexes

GPS - Guided Policy Search

GUI - Graphic User Interface

HRL - Hierarchical Reinforcement Learning

IRL - Inverse Reinforcement Learning

MDP - Markov Decision Process

NAF - Normalized Advantage Function

POMDP - Partially Observed Markov Decision Process

PPO - Proximal Policy Optimization

RL - Reinforcement Learning

RL-DT - Reinforcement Learning with Decision Trees

RNN - Recurrent Neural Network

SAC - Soft Actor-Critic

SL - Supervised Learning

TRPO - Trust Region Policy Optimization

t.u. – Time units

UL - Unsupervised Learning

VICE - Variational Inverse Control with Events

VICE-RAQ - VICE Reinforcement Learning with Active Queries

1. INTRODUCTION

Along with the advent of product customization, industrial manufacturing tasks are increasingly more complex and required to be highly flexible and efficient. Reinforcement learning (RL) is a current approach to such optimisation problems and as such, this dissertation intends to identify its efficacy in the assembly optimization of a product comprised of various parts and fasteners.

Reinforcement learning (RL) is, alongside with supervised learning (SL) and unsupervised learning (UL), a paradigm of machine learning (Sutton and Barto 2018) originally inspired by the way biological systems learn (Schultz, Dayan, and Montague 1997; Schmajuk and Zanutto 1997; Touretzky and Saksida 1997), where an agent (e.g. a human, a robot, a vehicle) interacts with the environment by taking actions (Figure 1.1). As a consequence of the action taken, the environment defines states and rewards. A state is essentially a description of the environment's situation and the rewards are an abstract concept that describes the feedback, by which the success or failure of the agent is measured. This kind of learning is often used in problems that require decision making, by dynamically exploring a solution that maximizes the total rewards.

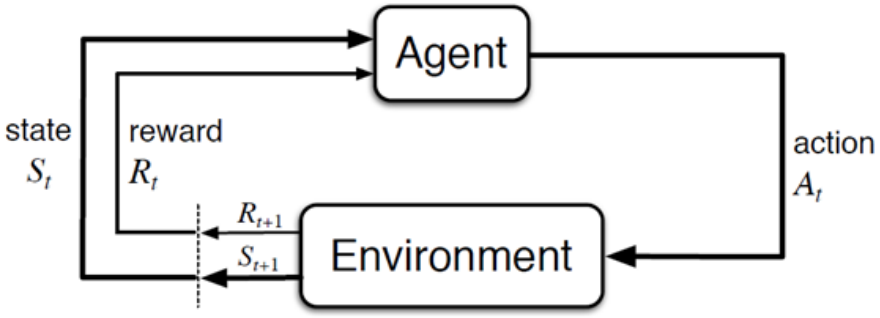

Figure 1.1. Agent and environment interaction.
Taken from (Sutton and Barto 2018).

In SL, the learning phase is based on labelled input-output examples, allowing the learned model to extrapolate to new situations. This learning method is often used in classification and regression problems. Contrarily, in UL, neither features nor outputs are

known and its typical uses are in the detection of patterns in data, also known as clustering problems. The differences in RL are the fewer data available in the learning phase, which are the current states and expected rewards, and a continuous learning process, fuelled by the interaction between the agent and the environment.

Given its characteristics, RL methods are used in complex problems where there appears to be no obvious or easily programmable solution, such as game playing, robotics, control problems or operational research and in problems where there is not enough labelled data such as anomaly detection problems.

The first uses of RL algorithms were in game playing problems since the possible states during a game can be very large, which deems infeasible the common approach of manually designing rules for the player to follow. By applying RL algorithms the agent can play against humans or other agents in order to continuously improve its playing capabilities.

In the field of robotics, since most of robot programming tasks are tedious, require years of experience and expertise, RL algorithms can be applied to replace it with an intuitive process comprehensible even by an unskilled user. Also, hard-coding controllers for robots may have limitations when the robot must adapt to new situations or when the robot/environment cannot be sufficiently modelled (Deisenroth 2011).

For example, in control problems, the perfect knowledge of the system is not possible, which is a common assumption in control-optimization. Therefore, in these situations, RL methods can be applied since they only learn on measured data and rewards. Also, the learning could be done in a simulation environment, which would be further improved by learning in the real world while the system remains online (Kober, Bagnell, and Peters 2013).

Lastly, RL could be used in operational research in the area of targeting marketing, for example, to learn cross channel integration (Abe et al. 2004) or to optimize a product delivery system with various transportation vehicles (Proper and Tadepalli 2006).

Reinforcement learning can be also used in anomaly detection problems to tackle the insufficient data availability problem. In recent years, the industry has transitioned from a corrective maintenance to a predictive maintenance standpoint (C. Huang et al. 2018). In other words, instead of only fixing and replacing components after they fail, methods are used to predict when such failures will occur, using statistics methods or anomaly detection models, so that actions can be taken before the failure happens. In this type of scenarios,

labelled anomalies are hard to obtain, enabling the potential of RL methods to improve continuously an anomaly detection model throughout the life of the equipment.

This dissertation is organized as follows. In the Section 2 reinforcement learning is discussed in detail. In the Section 3 the state of the art of the RL's algorithm is identified, as well as the application of RL in robotics and decision-making problems. In the Section 4 an assembly process is thoroughly analysed, and the Q-Learning algorithm is implemented in three different scenarios to optimize the assembly sequence. Finally, in the Section 5, the conclusions are presented as well as suggestions for future work.

2. REINFORCEMENT LEARNING

As previously stated, machine learning can be divided into three main categories, supervised learning, unsupervised learning and reinforcement learning.

Typically, in reinforcement learning problems the agent has a goal (or goals) related to the environment's state and has both the capabilities of taking actions that may affect the environment and extracting information regarding to the environment's current state. The RL agent is supposed to learn an optimal behavioural strategy where, based on the information available from the environment, it takes the optimal actions towards its desired goal (Sutton and Barto 2018). This mapping from the perceived states to actions is known as the policy π, which is the RL agent´s core. In other words, the policy is the set of stimulus-response rules or relations. This policy, given a state s and an action a, may be deterministic, $a = \pi(s)$, or stochastic which would imply the need to specify the probability for each action, $a \sim \pi(s, a) = p(a|s)$.

Contrarily to RL, in SL an external supervisor provides a training set of labelled examples, i.e. a set of situations together with the correct action the system should take in each situation, which the model uses to learn. The goal is to extrapolate the trained system response to situations not yet seen. Notwithstanding, it could not be adequate to learn from interaction since it's often impractical to obtain correct and representative examples of every desired behaviour (difficult to have labelled data). However, RL might be combined with SL in specific cases where it would be important to determine which capabilities are critical or not. UL on the other hand is about finding patterns hidden in sets of unlabelled data.

2.1. Modelling theory

To better understand the formalization of RL problems, the concepts of Markov chains, Markov decision processes (MDPs) and partially observed Markov decision processes (POMDPs) are introduced. The Markov chain, $\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$, is a simple graphical model proposed by Andrey Markov which can be defined by a state space $\mathcal{S}$ and a transition operator $\mathcal{T}$. The state space is a set of valid states s the system can occupy, $s \in \mathcal{S}$, and the

transition operator defines the conditional distribution of the next state over the previous state, i.e. $p(s_{t+1}|s_t)$. Figure 2.1 shows an example of a Markov chain.

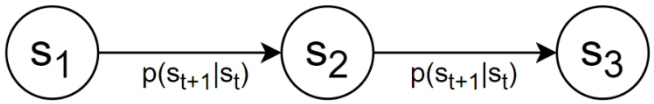

Figure 2.1. Markov chain.

To define a Markov decision process, $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$, an action space $\mathcal{A}$ and a reward function r are added to the Markov chain definition. The action space is, in other words, the set of possible actions a the agent can take such that $a \in \mathcal{A}$. In Markov decision processes, the transition probabilities are not only conditioned on the previous state but also on the previous action, $p(s_{t+1}|s_t, a_t)$. Since the policy is the mapping of the states to actions, $\pi_\theta(a_t|s_t)$, a graphical representation of a Markov decision process can be observed in Figure 2.2.

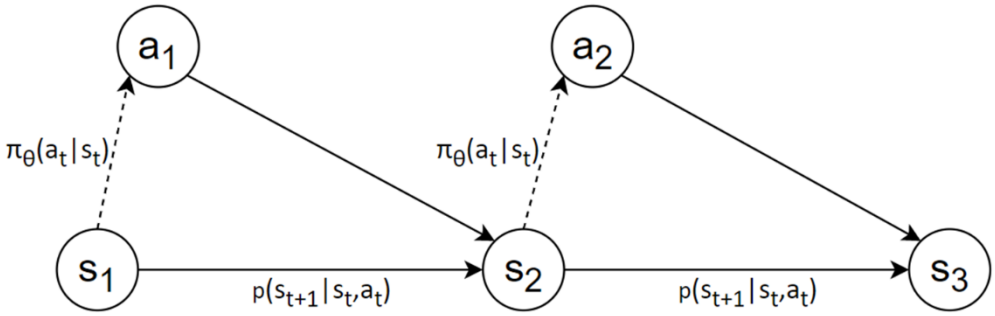

Figure 2.2. Markov decision process (MDP).

Classical RL problems are formalized as Markov decision processes. The reason for this formalism is the increased difficulty in computation, by considering all the states and actions taken from the initial state to the current state. Using MDPs the system only needs to keep track of the last state and action. However, it is important to understand that this Markov assumption leads to the loss of data, which in some situations might be relevant since rewards may be infrequent and delayed.

Lastly, POMDPs are a generalization of MDPs and are used in particular RL problems where the agent cannot sense every information regarding the environment's state,

and therefore, can only act according to the (partial) observations it makes of the environment. To define the POMDPs, an observation space $\mathcal{O}$ and the emission probabilities $\mathcal{E}$ are added to the MDPs' definition so that $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$. The observations space is the set of all possible observations the agent can make $o \in \mathcal{O}$ and the emission probability defines the probability of an agent making a certain observation o in regards to the environment state s, $p(o_t|s_t)$. Since the agent acts on the observations taken, the policy is now defined as $\pi_\theta(a_t|o_t)$. An example of such a process can be observed in the Figure 2.3.

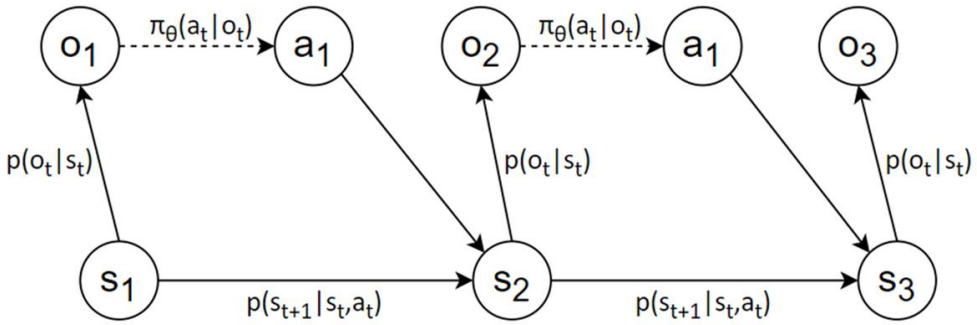

Figure 2.3. Partially observed Markov decision process (POMDP).

The learning agent must evaluate whether it is being successful in the task. An agent can discern good from bad events based on the reward signal, which is analogous to the way humans learn when experiencing pain or pleasure. This is the primary source of improvement of the policy, since if the action selected in a certain state returns a low reward, then the policy may be changed so that, when faced by the same exact state, the policy selects a different and more rewarding action. However, the agent's goal is to maximize the accumulated reward over time through the actions chosen. However, an action with a high immediate reward might not be the optimal choice, since it may lead to a lower accumulated reward over the future. To tackle this issue there are two important concepts, the value function and the quality function, usually known as Q-function.

Given a policy, the value function is defined as the total expected reward from a given state s_t:

$$V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t] \quad \textbf{(2.1)}$$

The Q-function is, on the other hand, the total expected reward from taking the pair action a_t in the state s_t:

$$Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t] \quad (2.2)$$

Actions must therefore be selected based on value judgements because the agent's goal is to maximize the accumulated reward over time. Unfortunately, while rewards are given directly by the environment, values must be estimated multiple times within the sequence of observations.

Another significant concept in RL is the model of the environment, which is an element used for planning a course of action considering the future states. This element describes the way the system and the environment will evolve over time as a stochastic function of the current state and actions (Abbeel 2008), which means that given a certain state and action the model might be able to predict the next probable state and reward.

2.2. Algorithm structure

RL algorithms can be structurally subdivided into three different parts (Figure 2.4). The first one corresponds to its ability to generate new sample experiments (i.e. run the initial policy) by interacting with the environment. After generating samples, the algorithm must fit a model and evaluate its performance. Lastly, the policy is improved.

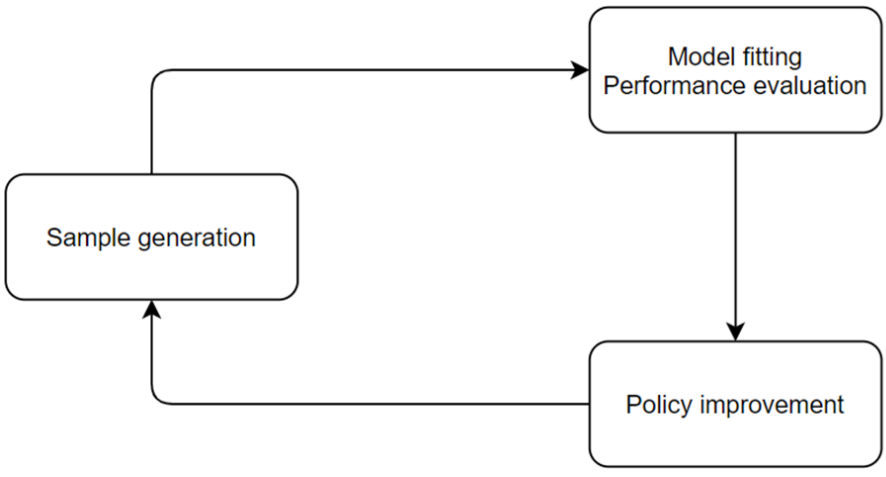

Figure 2.4. Structure of an RL algorithm.

Given the anatomy of RL algorithms, they can be classified into three categories of methods which are value-function methods, policy search methods and actor-critic methods. The value-function methods, also known as critic-only methods, are based on the idea of initially discovering the optimal value function by fitting a value-function or a Q-function and then deriving the optimal policy from this. On the other hand, the policy search methods, also known as actor-only methods, search directly in the policy space by summing the rewards of sample trajectories, which is only possible if the search space is restricted. In the particular case of policy search algorithms, known as policy gradient methods, one step of gradient ascent is applied on the expected reward objective. Lastly, the actor-critic methods are a combination of both where the critic monitors the agent's performance by fitting a value function or a Q-function to determine when the policy must be changed by the actor. Moreover, the models can be also divided into model-free algorithms, which do not use models of the environment and are explicitly trial-and-error learners and model-based algorithms, which use the model for planning or policy improvement.

The reason for the vast number of existing RL algorithms and approaches is the usage of reinforcement learning to tackle a large variety of different problems, with different specificities and setups. Different problems require different properties of RL algorithms which relate to sample efficiency, stability, ease of use and assumptions used.

Sample efficiency is related to the number of samples required to arrive at a good final policy. One of the most important factors in sample efficiency is regarded to the algorithm being on-policy or off-policy. An off-policy algorithm is able to improve without generating new samples from the new policy, while an on-policy algorithm requires the generation of new samples each time the policy is changed. Therefore, an on-policy algorithm is less sample efficient than an off-policy. Most policy gradient methods are on-policy and value-function methods are off-policy. The comparison of sample efficiency between the various classes of algorithms can be observed in the diagram of the Figure 2.5.

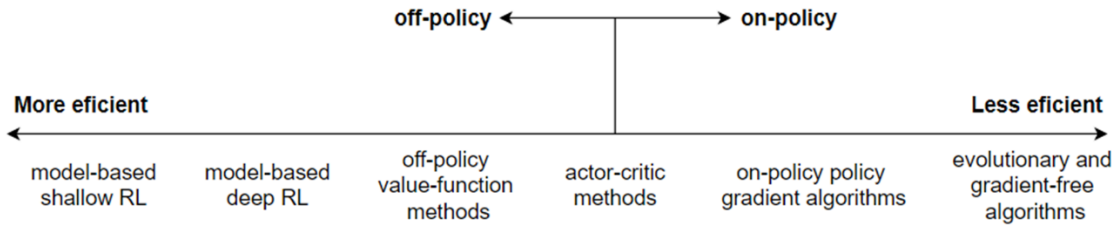

Figure 2.5. Sample efficiency of the different classes of algorithms. Adapted from (Levine 2018).

The sample efficiency, however, does not deem an algorithm better than the other, since they could require more computation time. Sample efficiency is particularly relevant when applying RL in problems which require samples from the real world, instead of a simulated environment. In terms of stability, only policy gradient methods are proven to converge since they use a method of gradient ascent. Moreover, algorithms often require certain assumptions, such as full observability, episodic learning and continuity or smoothness. Another important criterion is the difficulty of representing elements such as reward functions, policies and models being dependent on the type of problem in study.

2.3. Main challenges

Even though many advances were made in recent years, there are yet some challenges when applying RL algorithms. This is particularly evident when applying RL in robotic settings because the robot's states and actions are inherently continuous. Therefore, the resolution at which they are represented must be specified, often related to control problems (Kober, Bagnell, and Peters 2013).

The first challenge arises in the learning phase where the agent must do a trade-off between exploration and exploitation, which means that the agent must choose actions it knows from previous experience to be effective in producing reward (exploitation). However, in order to discover such actions, the agent must take actions not previously selected (exploration). This challenge is known as the exploration-exploitation dilemma (Sutton and Barto 2018).

Another common challenge in RL problems is the "curse of dimensionality" (Bellman 2003). In high-dimensional spaces, when the number of dimensions grows, the data and computation needed to cover the entire state-action space increases exponentially.

To ensure global optimality, data must be collected throughout the entire state-space, which may be infeasible in high-dimensional state-action spaces (Kober, Bagnell, and Peters 2013).

There is also the "curse of real-world samples" since robots inherently interact with the physical world. This means that there are restrictions on expensive hardware, component's wear and economical and logistical consequences of maintaining or repairing such systems. Consequently, when applying RL in robotic systems a safe exploration is critical, besides that most real robot learning tasks require human supervision (Kober, Bagnell, and Peters 2013).

In order to limit the need for real-world interactions, accurate models are often used as simulation systems. Ideally, such models would allow the system to learn on simulations which would be later transferred to the real scenario/robot. However, creating accurate models is very challenging and sometimes even impracticable. Small errors due to under-modelling can accumulate and make the simulation robot diverge from the real-world robot, which limits the direct transfer to the real system. This challenge is usually known as "curse of under-modelling and model uncertainty" (Kober, Bagnell, and Peters 2013).

Finally, in RL the desired behaviour is often specified by the reward function, which is frequently easier than defining the behaviour itself, however, in practice, in some problems it may be astonishingly difficult. This RL challenge is often known as "curse of goal specification". In order to specify reasonable reward functions, the reward function often needs to include intermediate rewards, in a process named reward shaping (Kober, Bagnell, and Peters 2013).

3. STATE OF THE ART

3.1. Background

As stated, RL algorithms can be generally divided into model-free algorithms and model-based algorithms. It is also relevant to further divide model-free algorithms into value-function methods, policy search methods and actor-critic algorithms.

In the case of value-function methods, (Sutton 1988) devised a temporal difference algorithm known as TD(λ) in the early stages of modern reinforcement learning. This algorithm led to the highly successful TD-Gammon, a game-learning program capable of beating professional gammon players (Tesauro 1995), which incentivised further work in the RL field.

One of the most well-known value function algorithms is Q-learning, which is also a temporal difference learning algorithm. Q-learning was introduced by (Watkins 1989) (Watkins and Dayan 1992) and is based on the idea of learning a table of Q-values, known as Q-table, from the successive interactions with the environment. After all the improvements in the Q-table, the actions are chosen based on the best Q-value available in the current state. This algorithm is highly affected by the curse of dimensionality since the increase in dimensions increases the Q-table's size. In order to tackle this problem, the deep Q-network algorithm was introduced (DQN).

The DQN algorithm combines Q-learning with deep neural networks (DNNs) by substituting the Q-table with a DNN, which takes the state and approximates the Q-values for each action. The use of DNN also allows the agent to learn directly from high-dimensional sensory inputs, such as images (Mnih et al. 2015). The use of DNNs in RL, also known as deep RL, proved to be a highly successful approach to tackle the "curse of dimensionality". (van Hasselt, Guez, and Silver 2015) proposed the Double Q-Learning algorithm, an improvement on the Q-Learning algorithm by tackling the problem of overestimation on the action values under certain conditions (Van Hasselt 2010). Similarly, since DQN was also known to suffer from overestimation of the values of the actions a similar improvement to DQN was made, surging the algorithm Double DQN.

Another well-known value-function method is the SARSA algorithm (Rummery and Niranjan 1994) which is identical to Q-Learning with the differences of being an on-policy algorithm, that follows a policy to find the next action instead of choosing an action in a greedy fashion.

In the category of policy search algorithms, policy gradient methods are the most widely used, however, alternative methods can be employed such as methods inspired by expectation maximization. One example of such algorithms is the Policy Learning by Weighting Exploration with Returns (PoWER) and was designed for robotic tasks. This method updates by a reward-weighted imitation of previously seen episodes and leverages the use of the motor primitives' concept. Motor primitives encode elemental motions which can be generalized, sequenced and combined into more complex tasks. In essence, this algorithm performs a local search around the policy learned from demonstration and previous knowledge (Kober and Peters 2009).

Regarding to policy gradient algorithms, the REINFORCE algorithm was proposed in (Williams 1992). This algorithm relies on estimate returns by Monte Carlo methods to update the policy weights.

Policy gradients, however, have two major flaws as they have low sample efficiency and poor convergence. In order to tackle the poor convergence problem, natural policy gradients where introduced. This approach, which had its origins in supervised learning (Amari 1998), was later introduced to the reinforcement learning field by (Kakade 2001). The way natural policy gradients tackle the poor convergence problem is by the usage of a fixed penalty coefficient in order to limit the policy change, so that the collapse of the training performance is prevented.

Similarly, the Trust Region Policy Optimization algorithm (TRPO) (Schulman et al. 2015), later devised, tackles the poor convergence problem in a similar fashion but instead of using a fixed penalty coefficient, it uses a method named as fixed KL divergence, which is basically a measure of difference between the old and the new policy. These last two algorithms are second-order optimization methods, and therefore, are computationally expensive.

Proximal Policy Optimization (PPO) (Schulman et al. 2017) was later introduced as an algorithm capable of attaining the data efficiency and the reliable performance of TRPO, while only using first-order optimization.

Regarding to actor-critic methods, the actor-critic architecture was introduced by (Barto, Sutton, and Anderson 1983) and was applied in a pole-balancing problem. The algorithms Advantage Actor-Critic (A2C) and Asynchronous Advanced Actor-Critic (A3C) are two classic actor-critic methods specialized on parallel training (Mnih et al. 2016). In these algorithms, the critics learn the value function while multiple actors are trained in parallel. In A3C the actors get synced from time to time and each actor talks independently with the global parameters. The A2C, on the other hand, has a coordinator who waits for the work completion of all the actors before updating the global parameters, and therefore, in the following iteration all the actors start from the same policy. A2C is essentially a synchronous deterministic version of the algorithm A3C.

Another interesting actor-critic algorithm is the Deterministic Policy Gradient (DPG), which considers deterministic policies instead of the usual stochastic policies. This algorithm was also able to outperform its stochastic counterparts in a RL problem with 20 continuous action dimensions and 50 state dimensions (Silver et al. 2014). Following DPG successes, a new algorithm named Deep Deterministic Policy Gradient (DDPG) was proposed by adapting the DQN's idea of combining deep neural networks in reinforcement learning algorithms (Lillicrap et al. 2015). In a similar fashion to DQN, the usage of a deep neural network allowed this new algorithm to process high-dimensional sensory inputs.

The algorithm Normalized Advantage Function (NAF), a continuous variant of the Q-Learning algorithm, was introduced in (Gu et al. 2016). This algorithm was devised to simplify the standard actor-critic style algorithms while conserving the benefits of nonlinear function approximation. This algorithm was tested against DDPG and was able to outperform it in the majority of the simulated tasks. This algorithm was also shown to improve its sample efficiency when incorporating learned models, however, the learned models were required to perfectly match the real model.

DDPG was further improved with the introduction of Distributed Distributional DDPG (D4PG) algorithm (Barth-Maron et al. 2018). This algorithm includes two major differences which are distributional updates and multiple distributed workers which write into the same replay table. When tested across a wide variety of simple control tasks, difficult manipulation tasks and hard obstacle-based locomotion tasks, the D4PG achieved state of the art performance.

Another relevant deep reinforcement learning algorithm is the Soft Actor-Critic (SAC) (Haarnoja et al. 2018). This algorithm is based on the maximum entropy RL framework, in which the actor aims to maximize the expected reward as well as maximizing the entropy, i.e. to be successful at the task while acting as randomly as possible. SAC is an off-policy maximum entropy actor-critic algorithm that can provide both sample efficient learning and stability. Effectively, this algorithm has been proved to extend to very complex high-dimensional tasks, such as the Humanoid benchmark (Duan et al. 2016) with 21 action dimensions, where off-policy methods such as DDPG tend to struggle to obtain good results.

Model-based algorithms, instead of learning directly from experience by performing actions in the environment, they use a reduced number of interactions with the environment to build a model and then use this model to simulate the further episodes. The model learning task is based on the experience acquired where for each action and state the environment provides a new state and a reward. This is essentially a supervised learning problem.

In summary, the main loop of model-based reinforcement learning starts with the collection of experience in the real environment. The experience is then used to generate a model, which is used to generate new samples. The value functions and policies are updated with the new samples and the new value functions and policies are used to select the next action to be performed in the environment (Figure 3.1).

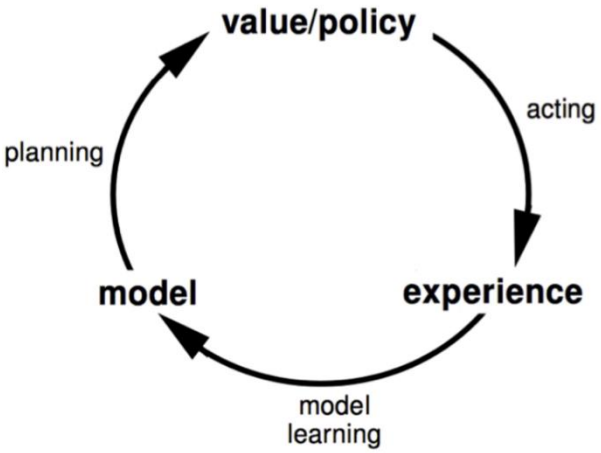

Figure 3.1. Model-based reinforcement learning's main loop.
Adapted from (Sutton and Barto 2018).

An alternative to this loop is known as the Dyna Architecture. In this architecture the real experience is not only used to build the model but also to update the value functions and policies (Figure 3.2).

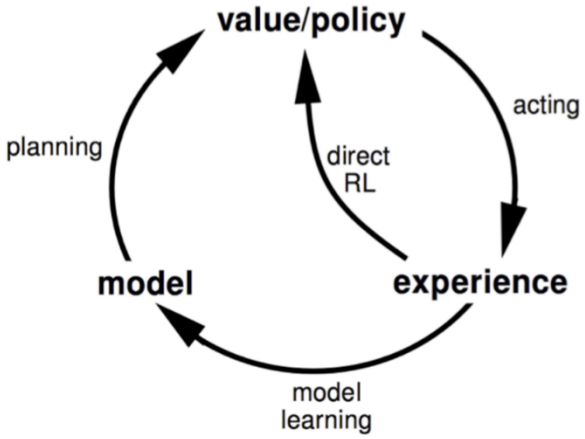

Figure 3.2. Model-based reinforcement learning based on the Dyna Architecture. Taken from (Sutton and Barto 2018).

An example of an algorithm that employs the Dyna architecture is the Dyna-Q algorithm. This algorithm proposed by (Sutton 1990) was built based on (Watkins 1989) Q-Learning algorithm.

When solving complex high-dimensional problems, standard policy gradient methods often require a large number of iterations and are prone to poor local optima. To solve these difficulties the algorithm Guided Policy Search (GPS) uses trajectory optimization to guide the policy away from poor local optima (Levine 2013). However, in a context with a small number of real-world samples, the resulting neural network is only robust in the neighbourhood of the trajectory distribution explored by real-world interactions. Generative Motor Reflexes (GMR) was introduced to tackle this exact problem, improving robustness by using motor reflexes and stabilizing actions (Ennen et al. 2019).

One of the key problems with model-based algorithms is the model bias, i.e. the assumption that the learned model resembles accurately the real environment. This can be a problem especially when only a few samples and no prior knowledge are available regarding to the task at hand. To tackle this issue an algorithm named PILCO (probabilistic inference for learning control) was introduced. This model-based policy search algorithm implements

a probabilistic dynamic model to express the model uncertainty and incorporates model uncertainty into planning and policy evaluation (Deisenroth and Rasmussen 2011).

Model-based algorithms often require exhaustive exploration to learn an accurate model. To tackle this the algorithm Reinforcement Learning with Decision Trees (RL-DT) was proposed. This method, as the name implies, uses decision trees to learn efficiently and rapidly a model of the domain, which is later used to compute a reasonable policy. This characteristic and the usage of an explicit exploration mode where the fewest visited states are explored turn RL-DT a sample efficient algorithm (Hester and Stone 2009).

Lastly, a new reinforcement learning model-based method, whose agent is known as Dreamer, was introduced recently. Dreamer learns long-horizon behaviours purely by latent imagination. This method was shown to outperform previous methods, such as A3C and D4PG, in data-efficiency, computation time and final performance on a variety of challenging continuous control tasks (Hafner et al. 2020).

All the algorithms previously mentioned have in common a reward function. However, designing a reward function is critical to obtain good results and if there is misspecification the reward function can be exploited by the agent and cause unintended behaviour. Further than that, in some complex RL problems, the design of the reward function might be very difficult or even unfeasible. To tackle such problems, regarding the "curse of goal specification", Inverse Reinforcement Learning (IRL) (Russell 1998) and Variational Inverse Control with Events (VICE) (Fu et al. 2018) were developed.

In IRL, instead of designing a reward function, the agent learns the correct behaviour by mimicking expert behaviour, i.e. modelling the preferences of another agent using its observed behaviour (Russell 1998).

However, IRL requires examples of expert behaviour to learn, which might be hard to obtain, and therefore, VICE was proposed to generalize IRL to alternative forms of expert supervision. An example of that would be the replacement of full demonstrations by examples of the desired outcome of the task (Fu et al. 2018). The comparison between IRL and VICE can be better understood by the Figure 3.3. In IRL a policy is learned by mimicking somewhat closely the correct expert behaviour (supervision). In contrast, in VICE the supervision is made solely from desired outcomes which can lead to a range of completely different policies despite the similar outcome.

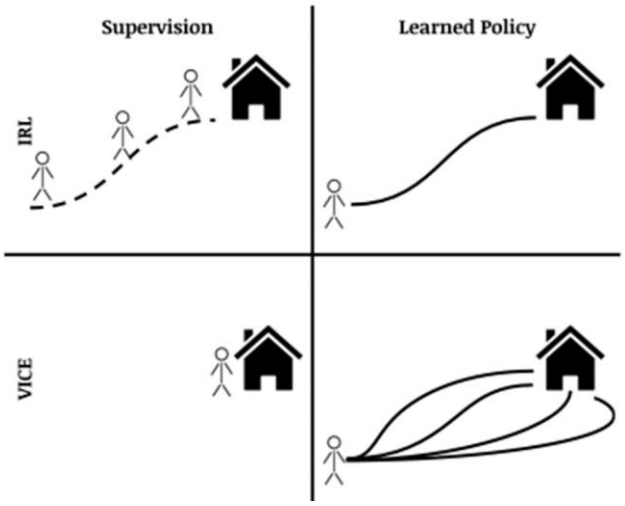

Figure 3.3. Comparison between IRL and VICE.
Taken from (Fu et al. 2018)

One implementation of VICE is the Off-policy VICE-RAQ with soft actor-critic algorithm, known simply as VICE-RAQ, where RAQ stands for RL with active queries (Singh et al. 2019). This algorithm is an integration of off-policy VICE, an extension of VICE to an off-policy setting by using the SAC algorithm, in an active query framework. The active query framework corresponds to the request of user labels of success for the current state.

Another important approach in the field of reinforcement learning is hierarchical reinforcement learning (HRL). This approach has the goal of solving more abstract and difficult problems that can be subdivided into simpler tasks. In other words, HRL methods were created with the objective of learning and planning while using high-level macro-actions instead of low-level primitive actions. The most well-known formulation for HRL is the Options framework (Sutton, Precup, and Singh 1999). An example of an option would be a navigation task, where if an agent does not have any obstacle it would move forward until one was found. In this particular case, the initiation set would be the non-existent obstacle in its' path, the policy would describe the forward movement and the termination condition would be the encounter of an obstacle. As shown in the Figure 10, once in a classical MDP in each time step an action is chosen, in HRL there is a decision of an option, in particular states represented by the white circles. This option initiates and is executed based on its policy until it terminates. This skill acquisition method builds a skill tree from a set of sample solution trajectories obtained. It uses a detection method to segment each

solution trajectory into chains of skills, in each one is allocated its abstractions and finally, the skill chains are then merged into a skill tree (Konidaris et al. 2010).

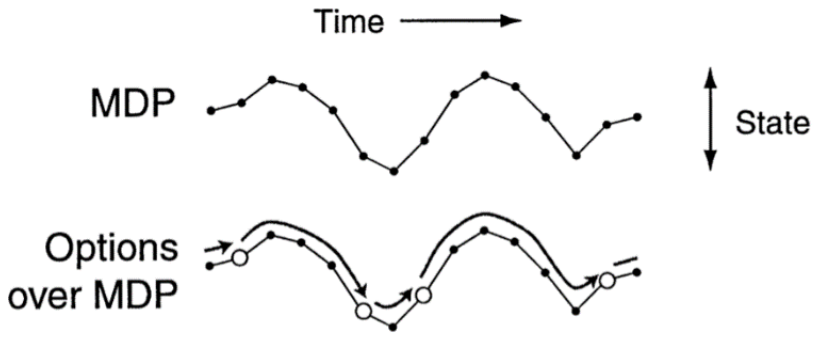

Figure 3.4. Comparison between a MDP and a MDP using the Options framework. Adapted from (Sutton, Precup, and Singh 1999).

3.2. Applications

Before introducing RL applications in the fields of robotics and anomaly detection examples, it is important to start by mentioning two examples of highly successes in the area of game playing.

The first one is the application of RL in a set of 49 Atari games by developing the DQN algorithm. This new algorithm was able to outperform the best RL methods at the time in 43 out of the set of 49 game. Furthermore, DQN performed at a level comparable to a professional player, achieving more than 75% of the human score in 29 out of the set of 49 games tested (Mnih et al. 2015).

The second one is the development of the Go playing program known as AlphaGo. This program achieved a 99.8% winning rate against other Go programs and defeated the European Go champion by 5 games to 0 (Silver et al. 2016). This program was trained by supervised learning from human expert moves and by reinforcement learning from self-play. This program was further trained and competed against the 9 dan player Lee Sedol, winner of 18 international titles. This new version of AlphaGo (AlphaGo Lee) won 4 out of the 5 games played. Later, a new algorithm, AlphaGo Zero, was later developed as an improvement on the previous successes of AlphaGo. This new algorithm is based solely on

reinforcement learning and does not use human data. AlphaGo Zero won against the previous AlphaGo 100 games to 0 (Silver et al. 2017).

In the robotics context, reinforcement learning was implemented both in simulated environments and real-world environments in a variety of highly complex tasks. The following advances in RL presented are divided in purely simulated applications and applications with real-world implementation.

In the work by (Maravall, de Lope, and Martín H. 2009), evolutionary algorithms and reinforcement learning were incorporated to combine the advantages of both methods. While RL is a method effective in real-time and on-line applications, it has some difficulties when applied in high-dimensional problems. On the contrary, evolutionary algorithms are powerful off-line optimizers, especially in extremely high dimensional spaces, however, they are not suited to real-time, on-line environments. Therefore, a hybrid approach was proposed in the application of autonomous navigation of a L-shaped two-link robot in a cluttered environment with unknown obstacles.

Another virtual application in robotics is proposed in (Maeda and Aburata 2013), where a virtual robot is initially trained by supervised learning to push a cubic object from an initial position to the target position. The robot is then trained further by an actor-critic RL method which improves its ability to solve this task from shifted initial positions.

Learning real-world tasks from scratch is an extremely complex problem since it requires usually big training times and a high sample-complexity. To tackle this issue, in (Gu et al. 2017), an asynchronous version of the NAF algorithm, parallel NAF, was used in a variety of robotic arm tasks both in simulation and in the real-world. This algorithm allows various robots to learn simultaneously in an asynchronous fashion, which reduces the training time proportionally to the number of robots in use. The robotic arms were trained with success in random target reaching tasks, in virtual and real environments, door opening tasks, in real and virtual environments, and pick and place tasks, in a virtual environment.

Similar tasks were also learned by 48x48 RGB image observations and active queries, both in virtual and real environments, in a 7-DoF robotic arm by the usage of the proposed VICE-RAQ algorithm (Singh et al. 2019). In this case, the tasks learned in a virtual environment were an object pushing task, where a mug is pushed onto a coaster, a door opening task and an object picking task. In the real environment, the tasks learned were an object pushing task, similar to the simulated one, a draping task, were the robot arm drapes

a cloth over an object, and a bookshelf task, where the robot arm places a book in an empty slot of the bookshelf. In (Rupam Mahmood et al. 2018) a UR5 robot was trained a real-world reaching task using the TRPO algorithm with the objective of observing the impact of task setups elements in the performance of the robot. A six axes Kinova Jaco 2 robot arm was trained in a similar reaching task and in a task where the robot was expected to place a block in a hole using GMR, an improvement on the GPS algorithm, both in a simulation and in the real-world. GMR proved itself as more robust than the GPS algorithm (Ennen et al. 2019). In (Kormushev, Calinon, and Caldwell 2010) a WAM robot was successfully trained in a similar reaching task and in a highly complex pancake flipping task with the help of the policy search algorithm PoWER.

In order to reduce the programming effort required by an expert, in (Akkaladevi et al. 2018) an approach based on Interactive Reinforcement Learning is proposed, where a complete collaborative assembly process is learned. The learning approach is done in two steps. The first one consists of modelling simple tasks that constitute the assembly process, using task-based formalism. These modelled simple tasks are then used by the robotic system by proposing to the user a set of possible actions at each step of the assembly process via a graphic user interface (GUI). After the user selects the action, the robot performs it, progressing the assembly process while learning the assembly order. The framework also allows different users to teach different assembly processes to the robot. This proposed approach is based on Q-Learning and IRL and was successfully applied in a UR10 robot in an assembly process comprising tasks such as, picking, holding, mounting and receiving objects.

An especially relevant task in robotics is the locomotion task. Since the Sony Aibo robot ERS-210A has by default a fairly slow gait, its gait is usually enhanced by hand-coding or by the usage of learning algorithms. Hand-coding a parametrized gait is very time-consuming and requires human expertise. Therefore, in (Kohl and Stone 2004) the application of policy gradient RL in the continuous 12-dimensional space problem was proposed. The final gait obtained surpassed previous hand-coded and learned solutions.

An interesting and challenging application of RL in robotics is the usage of the model-based RL-DT algorithm to learn how to do penalty kick goals on the Aldebaran Nao humanoid robot. This algorithm was proposed since it generalizes during model-learning, which limits the number of trials needed for learning. The robot was first trained in a

simulated environment with a standard ball location and then with a randomized ball location. Finally, this task was learned on the physical robot (Hester, Quinlan, and Stone 2010).

IRL algorithms proved to be successful in solving, very challenging, previously unsolved control tasks. They have enabled a quadruped robot, designed by Boston Dynamics, to traverse challenging, previously unseen terrain. IRL algorithms also extended the state-of-the-art in autonomous helicopter flight and achieved the ability to perform the most challenging aerobatic manoeuvres performed by any autonomous helicopter to date, including manoeuvres such as continuous in-place flips, rolls and tic-tocs. Effectively were achieved performances in aerobatic flight, in the XCell Tempest and the Synergy N9 helicopters, comparable to that of the best human pilots (Abbeel 2008).

Another highly complex and relevant RL development in robotics is the usage of HRL. In (Konidaris et al. 2011), with the goal of solving several tasks in a room, such as handle pulling, button pushing and switch pressing, in a specific order an uBot-5 robot was first trained how to interact with each object and in which order. Then, using the CST HRL algorithm these skills were able to be transferred so that the robot would learn how to solve a different room with similar tasks.

In the anomaly detection field, a time series anomaly detector was created with the help of a recurrent neural network (RNN) combined with Q-Learning. This detector was formulated with the required features making no assumption about the concept of the anomaly, being threshold-free and being able to improve dynamically. This detector was trained on Yahoo benchmark datasets. From the experiments made, the anomaly detector was capable of identifying shifts on means, point anomalies and anomalous patterns, and achieved high-quality results (C. Huang et al. 2018).

In recent years, the research on the applicability of RL is also increasing in the fields of decision-making and system control problems. In (Fernandes 2019) Q-Learning was applied in a stock optimization problem and was able to achieve better results, up to 25%, when compared with traditional stock management algorithms. (Wang and Usher 2007) studied the implementation of the Q-Learning algorithm for the usage of job agents when establishing routing decisions in a job shop environment. In this work the effects of the Q-Learning application were investigated and guidelines for future applications and recommendations for factor settings were devised. Also, in dynamic job shop scheduling

problem (DJSS) (Shahrabi, Adibi, and Mahootchi 2017) proposed the usage of RL with a Q-factor algorithm to improve the scheduling method's performance while considering random job arrivals and machine breakdowns. In a simulated environment this proposed method achieved high performances. (J. Huang, Chang, and Chakraborty 2019) proved, through a simulation study, the effectiveness of the usage of Q-Learning in a maintenance problem where random failures of machines are highly disruptive. The performance in manufacturing work cells that utilize gantries to load and unload the materials and parts needed is highly dependent on the gantry movements in real operation. (Ou et al. 2018) formulated the gantry scheduling problem as a RL problem through the usage of Q-Learning and demonstrated, from the simulation results, the capability of effectively reducing system production losses in real-time operations. In the manufacturing field (Watanabe and Inada 2020) proposed the usage of RL to improve assembly efficiency by a dual-arm robot and achieved higher performance when compared with other methods. By the usage of RL, (Low, Neo, and Kumar 2020) devised a process to automatically designing the fixtures used in a machining or measurement process to mitigate the dependence on the user's experience of conventional methods. A common task in industrial manufacturing is the sorting of small parts. This task is typically done by the usage of vibratory bowl feeders, which are designed manually in a expensive trial-and-error approach. To tackle this issue (Stocker, Schmid, and Reinhart 2019) proposed the usage of Q-Learning. In the welding process RL was implemented in order to achieve an intelligent weld control capable of attaining and maintaining the desired weld pool width, (Jin, Li, and Gao 2019). Lastly, the usage of robots to replace simple manual manufacturing tasks sometimes encompasses some control issues. An example is the usage of a robot in a grinding task where the grinding force signal can easily overshoot during the impact stage and instabilities can occur during the process stage. To tackle these issues, (Zhang et al. 2020) propose a force control algorithm based on a press-and-release model and model-based reinforcement learning. This approach attained a fast convergence of the normal force in the impact and processing stages as well as a reduced surface roughness in the workpiece.

4. CASE STUDY

In this dissertation the assembly process of an airplane, from the Yale-CMU-Berkeley Object and Benchmark Dataset (Figure 4.1), is studied and optimized through the implementation of a RL methodology. This process is representative of an assembly job of a complex product containing different parts and tools, decomposed in a number of tasks which can be assembled in different sequences by an assigned resource (for the purpose of this study we will disregard if it is a human or robotic resource). However, improving the time efficiency of such an assembly process is often impractical due to the complexity of measuring all tasks sequences' time. For that reason, the goal of this work is to identify the effectiveness of the RL framework in the resolution of such a problem.

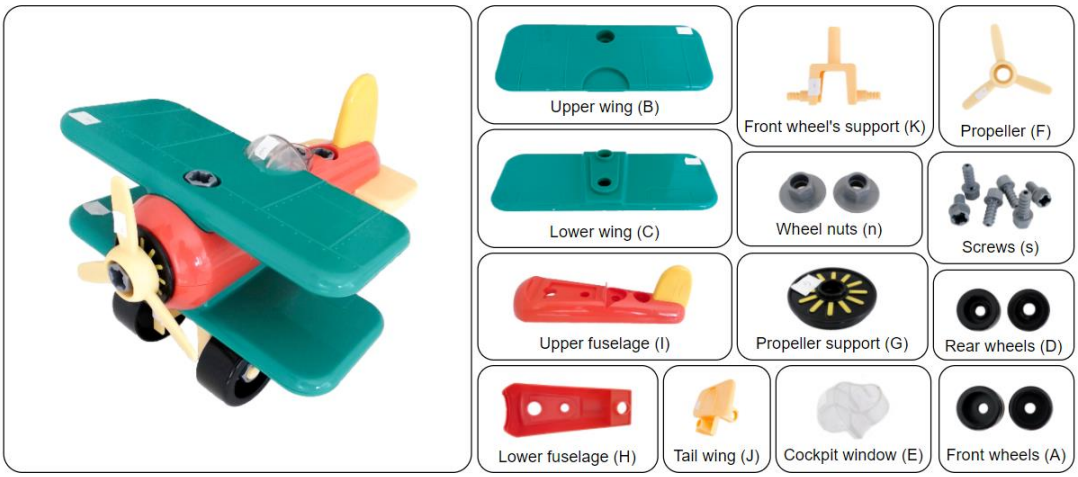

Figure 4.1. Airplane from the Yale-CMU-Berkeley Object and Benchmark Dataset.

In the first subchapter, the airplane's components and assembly structure are thoroughly analysed. Later, in the second subchapter, the assembly problem is formulated using Q-Learning and is implemented in three different scenarios. The first scenario corresponds to the situation where the agent must learn a feasible assembly sequence, i.e. an assembly sequence that respects all the task precedences. In the second scenario, estimated time durations of each task are incorporated in the decision-making algorithm to foment the learning of the most time-efficient assembly sequences. In this scenario, the algorithm's

parameters are individually analysed in order to improve the agent's performance. Lastly, in the third scenario, more accurate and disperse time durations are implemented and the parameters are once again optimized.

4.1. Problem description: airplane's components and assembly structure

The airplane is comprised of 9 structural parts and 2 types of fasteners, which are displayed in Table 4.1 and Table 4.2.

Table 4.1. Airplane's structural parts.

Part	Airplane's part description	Number of parts
A	Front wheels	2
B	Upper wing	1
C	Lower wing	1
D	Rear wheels	2
E	Cockpit window	1
F	Propeller	1
G	Propeller support (engine)	1
H	Lower fuselage (lower body of the airplane)	1
I	Upper fuselage (upper body of the airplane)	1
J	Tail wing (rear body of the airplane)	1
K	Front wheel's support	1

Table 4.2. Airplane's fasteners.

Fastener	Fastener's head type	Number of fasteners
n	Wheel nuts	2
s	Screws	5

After subdividing the airplane in parts and fasteners, the assembly process was subdivided in a total of 8 tasks. In the Table 4.3 each task is associated with the corresponding parts and fastener required. It is important to note that some parts can be used

in more than one task. For the assembly to be complete, every task must be executed without repetitions, therefore the number of different assembly sequences is $n! = 8! = 40320$.

Table 4.3. Parts and fasteners associated with each task and their respective quantities.

	Parts											Fasteners	
Task	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**s**	**n**
1							1	1	1				
2										1		1	
3						1						1	
4			1									1	
5		1									1	1	
6		1			1							1	
7	1										1		2
8				1						1		1	

However, the feasible number of assembly sequences is lower than the previously calculated one, due to the fact that certain tasks require other tasks to be previously completed. Such precedence sequence dependencies are displayed in the Table 4.4.

Table 4.4. Precedence task's dependencies.

Task	Precedence task
1	None
2	1
3	1
4	1
5	1 and 4
6	1
7	None
8	None

In order to determine the feasible number of assembly sequences, the actions can be placed in 8 slots, resembling the order in which the tasks are executed. Since the actions 7 and 8 have no dependencies they can be placed in any one of the 8 slots, therefore they can be calculated as a permutation of 8 slots taken 2 at a time $({}_{2}^{8}P)$. After assigning the tasks 7 and 8, the action 1 must be executed in the first available slot since all the other actions are dependent to this task. The actions 4 and 5 can be placed in any available slot as long as the action 4 is executed before the action 5, which can be described as the combination of 5 objects taken 2 at a time $({}_{2}^{5}C)$. Finally, the remaining actions can be executed in any order, which is represented as the permutation of 3 objects taken 3 at a time $({}_{3}^{3}P)$ (Figure 4.2). Therefore, there are ${}_{2}^{8}P \times 1 \times {}_{2}^{5}C \times {}_{3}^{3}P = 3360$ feasible assembly sequences.

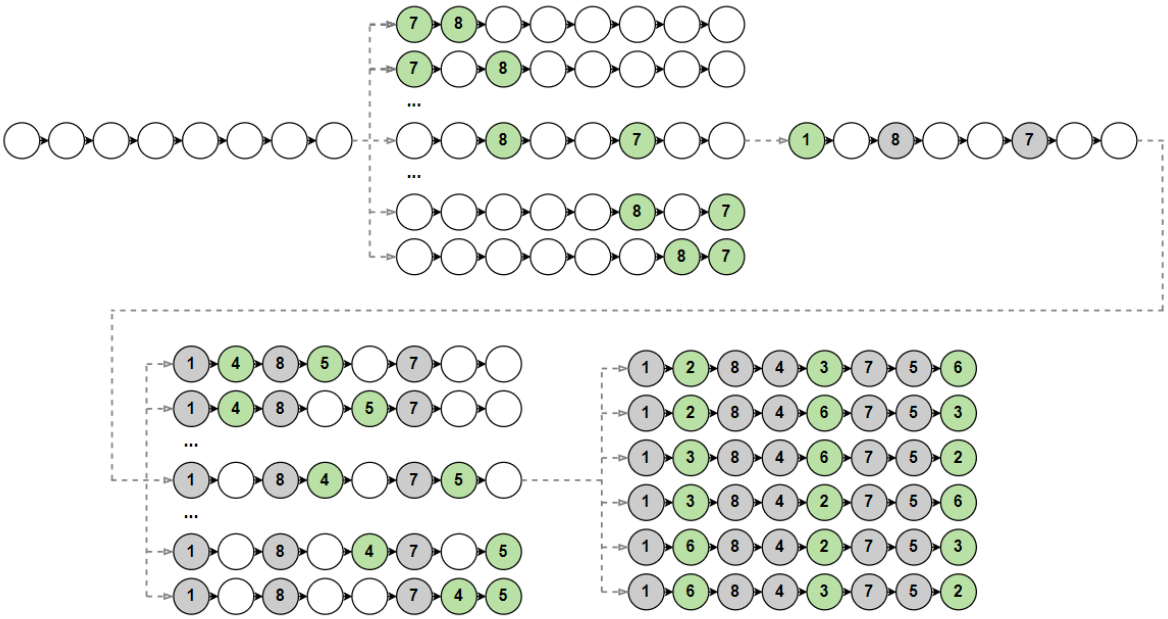

Figure 4.2. Feasible assembly sequences' scheme.

4.2. MDP formulation

As previously stated, the assembly process is subdivided in 8 different tasks, or actions, and the assembly can be considered as complete when all the 8 tasks have been executed. Therefore, the MDP's states can be defined by the updated assembly status at each task, where the initial state would correspond to the situation where none of the actions were

executed, and the final state when all the actions were completed and the airplane is assembled. Each action can be assigned to a binary variable which would have the value of 0 if the task is yet to be completed, and 1 if the task was already executed. To calculate the existing number of states, each state could be associated with an 8 digit binary number, in which the leftmost digit corresponds to the action 1, the following digit corresponds to the sequential action 2 and so on until the rightmost digit (Figure 4.3). The initial state would then be represented as 00000000 and the final state would correspond to 11111111, which in decimal notation corresponds to 0 and 255 respectively, thus, the number of states is 256.

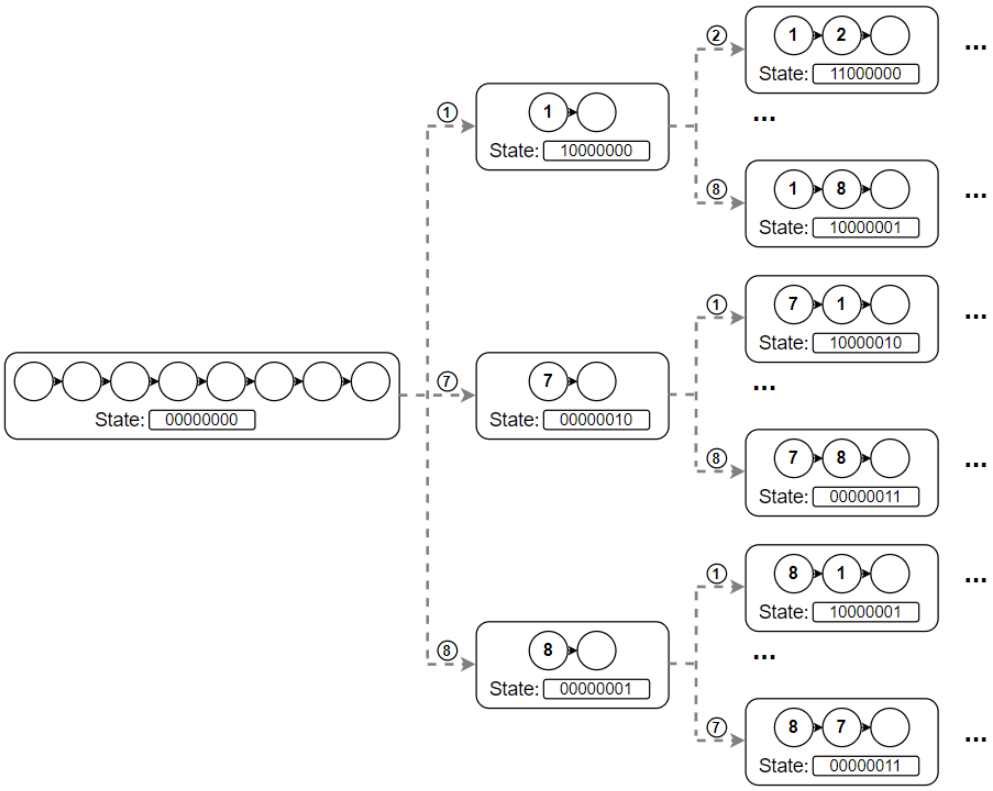

Figure 4.3. MDP's states and actions scheme.

However, some states are impossible to be achieved due to task precedences. There are two main groups of impossible states. The first one is the case where the first action was not executed and at least one of the actions 2, 3, 4, 5 and 6 has been executed, i.e. action 1 has the value of 0 and the actions 2, 3, 4, 5 and 6 can have the value of either 0 or 1, which can be calculated as $(2^5 - 1)$ states. Since the actions 7 and 8 can be either executed or not, the number of impossible states in the first group can be calculated as $(2^5 - 1) \times 2^2$.

The second group of impossible states corresponds to the case where the action 5 is executed previously to the action 4 while action 1 has already been done. In this case, the action 1 has been executed, the action 4 has not been executed, the action 5 has been executed. All the other actions were either executed or not executed, which mathematically corresponds to 2^5 states. Thus, the number of impossible states is $1 \times (2^5 - 1) \times 2^2 + 2^5 = 156$, which in turn means that the number of possible states is $256 - 156 = 100$.

4.3. Q-Learning implementation

After defining the MDP's states and actions, the Q-Learning algorithm was gradually implemented in three scenarios using MATLAB. In order to implement the Q-Learning algorithm, it is necessary to understand the Q-Learning parameters used, therefore, it is important to understand how the Q-table is updated. The value iteration update is done at each step through the Bellman Equation, which consists on the weighted average of the old Q-value and the new information obtained, where α corresponds to the learning rate, γ to the discount factor, r_t to the received reward when moving from state s_t to s_{t+1}, $Q^{new}(s_t, a_t)$ to the new Q-value of the state s_t and action a_t, $Q(s_t, a_t)$ to the old Q-value of the state s_t and action a_t and $\max_a Q(s_{t+1}, a)$ to the estimate of the optimal future Q-value:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times \left(r_t + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \quad \textbf{(4.1)}$$

The learning rate parameter has values between 0 and 1 and influences to what extent the new information changes the old information, which means that a lower learning rate leads to a longer learning time. However, it is important to note that a higher learning rate may lead to suboptimal results or even divergence. The discount factor determines the importance of future rewards, so the lower its value the less meaningful are the future rewards. If the discount factor has the value 0, only the current reward is considered.

The selection of the action is made using an epsilon greedy search, i.e. the agent selects a random action with probability ε and otherwise selects the action greedily, with probability $1 - \varepsilon$, by selecting the action with the highest Q-value. The value of epsilon (ε) decays based on a decay rate known as epsilon decay.

The learning phase takes place over various episodes. In this specific case, an episode starts with no tasks done and ends when all tasks have been successfully completed or when the maximum number of steps has been reached. In all scenarios, the algorithm considered that, whenever the Q-Learning agent selected an impossible action, the current state does not change, which means that the sequence is penalised to require more than 8 steps to complete an episode i.e. the agent could not successfully assemble the airplane with only 8 actions. The number of episodes required to complete the learning phase is dictated by the maximum number of episodes per experiment. At the end of the experiment, the agent selects the actions based solely on the Q-Values, which means that after learning, the agent selects always the same assembly sequence (learned assembly sequence), which could be for example the assembly sequence $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$.

The values of the default parameters used in the Q-Learning algorithm implementation can be observed in the Table 4.5.

Table 4.5. Values for the Q-Learning parameters.

Parameter	Value
Learning rate (α)	1
Discount factor (γ)	1
Epsilon (ε)	0.9
Epsilon decay	0.01
Max steps per episode	8
Max episodes per experiment	1500

4.3.1. Scenario 1: Learning a feasible assembly sequence

In the first scenario of the Q-Learning algorithm implementation, the main goal was for the agent to be able to learn one feasible assembly sequence, as previously detailed, which corresponds to $100 \times \frac{3360}{40320} = 8.333\%$ of all the assembly sequences.

In order to do so, it is essential to previously define the rewards to account in the equation (4.1). In this simpler case, three types of rewards were defined: partial rewards, completion rewards and reward penalties. The partial rewards are attributed to each executed

task, the completion reward is granted when all tasks have been executed, and reward penalties are assigned when an impossible task is selected by the agent.

This first implementation was executed in three different sets of rewards, as shown in Table 4.6, with the parameters in the Table 4.5 and was repeated in 120 experiments. As previously stated, in each one of the 120 experiments one assembly sequence is learned.

Table 4.6. Scenario 1: rewards.

Rewards	Set 1	Set 2	Set 3
Partial reward	0	0	10
Completion reward	0	100	100
Reward penalty	-10000	-10000	-10000

In this scenario, all the 120 experiments were able to successfully learn one of the feasible assembly sequences, but not necessarily the same, which in turn did not allow to understand the impact of the partial and completion rewards in the determination of the most efficient assembly sequences. Without the ability to conclude the efficacy of each reward type in the Scenario 1, the complexity of the problem was increased in the Scenario 2.

4.3.2. Scenario 2: Learning an assembly sequence based on estimated task average times and variances

4.3.2.1. Part 1: Introduction of estimated assembly times

With the objective of understanding how well the agent would be able to compare the efficiency of the feasible assembly sequences based on the rewards, a new reward system was designed to ponder the time spent to perform each task. In this scenario, the tasks' average times were estimated (Table 4.7) as well as the increase/decrease variances on the average times with respect to the tasks previously done (Table 4.8).

Table 4.7. Tasks' average time.

Task	1	2	3	4	5	6	7	8
Average time (time units, t.u.)	10	7	8	6	12	8	11	9

Table 4.8. Tasks' variance in respect to the average time (t.u.).

Task	1	2	3	4	5	6	7	8
Task 1 done							0	0
Task 2 done			-1	-1.5	0	-1	0	1
Task 3 done		0		0	0	0	0	0
Task 4 done		-0.5	0			0	0	0
Task 5 done		-1	-0.5			-2	1	0
Task 6 done		0	0	0	0		0	0
Task 7 done	0	0	0	0	0	0		0
Task 8 done	0	0	0	0	0	0	0	

The rewards were then defined through the following equation, where $\boldsymbol{R}(s,a)$ is the reward for taking the action a in the state s, r_m is the reward multiplier, r_s the reward shift, r_p the reward penalty and $\boldsymbol{T}(s,a)$ the predefined time it takes to complete the action a in the state s, that is calculated by summing the respective variances to the task's average time:

$$\begin{cases} R(s,a) = r_m \times (-T(s,a) + r_s) & if\ possible\ action \\ R(s,a) = r_p & if\ impossible\ action \end{cases} \quad (4.2)$$

Since the RL algorithm's goal is to maximize the accumulated reward, in order to learn the most time-efficient assembly sequence (minimize the assembly sequence time), the matrix $\boldsymbol{T}(s,a)$ must be subtracted in the equation (4.2). The r_s, as the name implies, shifts each reward by its value and, as a result, shifts the accumulated reward by eight times its value. The r_m, on the other hand, multiplies the shifted reward. Consequently, the accumulated reward is also multiplied by r_m. If r_m and r_s have the values of 1 and 0

respectively the accumulated reward is equal, in absolute values, to the duration of the assembly sequence.

The accumulated rewards for all feasible assembly sequences for these exact values of r_m and r_s are displayed in the Figure 4.4.

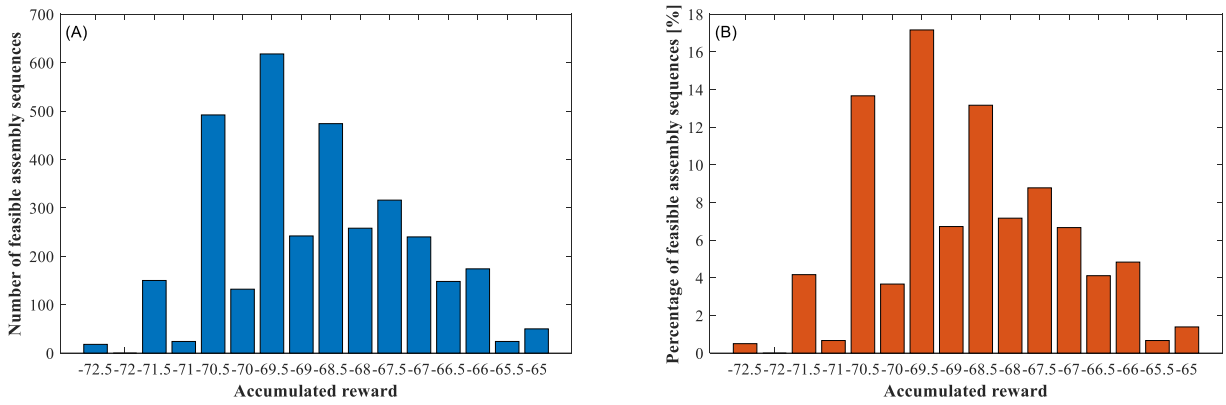

Figure 4.4. Distribution by number (A) and percentage (B) of feasible assembly sequences' accumulated rewards.

As it can be observed in the Figure 4.4 (B), the most common accumulated reward, corresponding to 18.39% of all feasible assembly sequences, is -69.5, which correlates to the corresponding total assembly time. An example of such an assembly sequence is $8 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 2 \rightarrow 6 \rightarrow 5$ where the accumulated reward is $\sum R(s,a) = -9 - 10 - 8 - 6 - 11 - 6.5 - 7 - 12 = -69.5$.

It is also possible to observe in the Figure 4.4 (A) that there are 50 feasible assembly sequences with the maximum accumulated reward of -65 (optimal accumulated reward).

As in the scenario 1, each set of parameters and rewards was replicated in 120 experiments for each set, so that a statistically relevant change could be observed for a 95% confidence interval. As an example, the mean, sample standard deviation and 95% confidence interval for the first set in Scenario 2 are calculated in the equations (4.3), (4.4) and (4.5) respectively based on the problem results presented in the Table 4.9 for the 120 experiments. In the equation (4.4), for a 95% confidence interval the significance level (α^*) has the value of 5% and since we are dealing with a sample size of 120 the number of degrees of freedom is 119. Therefore, the Student t value is equal to $1.6578 \approx 1.658$.

Table 4.9. Replication results of 120 experiments considering set 1.

Accumulated reward	Count	Accumulated reward	Count
-65	3	**-69**	6
-65.5	0	**-69.5**	21
-66	10	**-70**	4
-66.5	5	**-70.5**	15
-67	12	**-71**	1
-67.5	8	**-71.5**	6
-68	12	**-72**	0
-68.5	16	**-72.5**	1

Table 4.10. Parameters of set 1.

Rewards	Value
Reward shift	0
Reward multiplier	1
Reward penalty	-10000

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = -68.558 \quad (4.3)$$

$$s^* = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} = 1.652 \quad (4.4)$$

$$\bar{x} \pm t_{\alpha^*/2} \times \frac{s^*}{\sqrt{n}} = -68.558 \pm 1.658 \times \frac{1.652}{\sqrt{120}} == -68.558 \pm 0.250 \quad (4.5)$$

In an initial sensitivity analysis, the parameters learning rate, discount factor, and the rewards (r_s and r_p) were tested individually (Figure 4.5) while using the parameters in the Table 4.5. The comparison of the performances for the various sets of parameters and rewards considers three indicators: the mean accumulated reward, normalized for a $r_m$of 1 and a r_s of 0; the percentage of times the agent learned one of the 50 optimal assembly sequences; and the percentage of times the agent failed to learn one feasible assembly

sequence in the 120 experiments. When the agent failed to learn a feasible assembly sequence the number of experiments was increased so that the mean would reflect 120 correctly learned assembly sequences. The reason for this rule was the high penalty on the mean of an incorrectly learned assembly, which would turn unfeasible a correct comparison between sets.

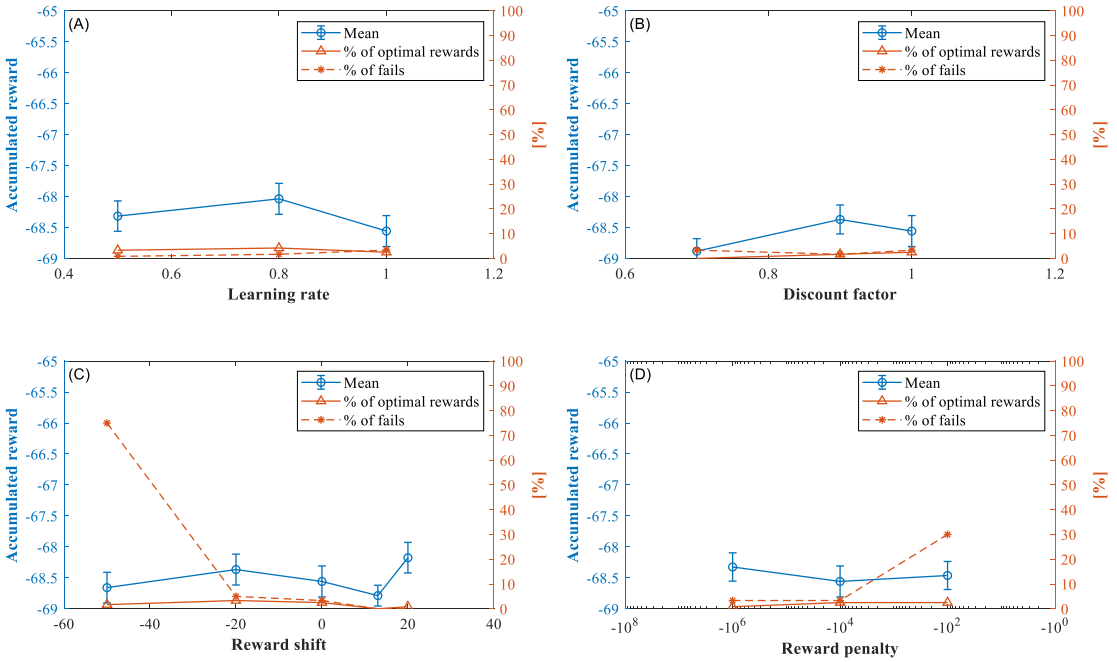

Figure 4.5. Impact of the learning rate (A), discount factor (B), reward shift (C) and reward penalty (D) on the agent's performance.

When examining the Figure 4.5 (A) and (B) we can observe that changing the learning rate and the discount factor seems to not affect significantly the percentage of fails and the percentage of optimal assembly sequences. However, it may suggest that a value lower than 1 in the learning rate and a higher discount factor could lead to better results. In regard to the r_s in the Figure 4.5 (C), a negative value increases the likelihood of incorrectly learning an impossible assembly sequence because the agent is not able to differentiate between the penalties and the accumulated rewards shifted to values increasingly more negative. Also, a positive r_s may lead to better results as seen in the mean increase for the value of 20. The reward penalty, as seen in the Figure 4.5 (D), understandably, influences the percentage of fails since an action with a higher r_p is more likely identified as an incorrect action, i.e. the bigger the penalty the lower the percentage of fails.

With the objective of understanding the impact of the r_m and the maximum number of episodes they were individually changed maintaining the parameters in the Table 4.5 and Table 4.10, except for the r_s and r_p with the new values of 20 and -1000000 respectively. Also, in the experiments where the maximum number of episodes was altered, the selected r_m used was 5. Even though the results may suggest better outcomes with a learning rate lower than 1, the parameter was kept unchanged for the following simulations since the higher the value the faster the learning (Figure 4.6).

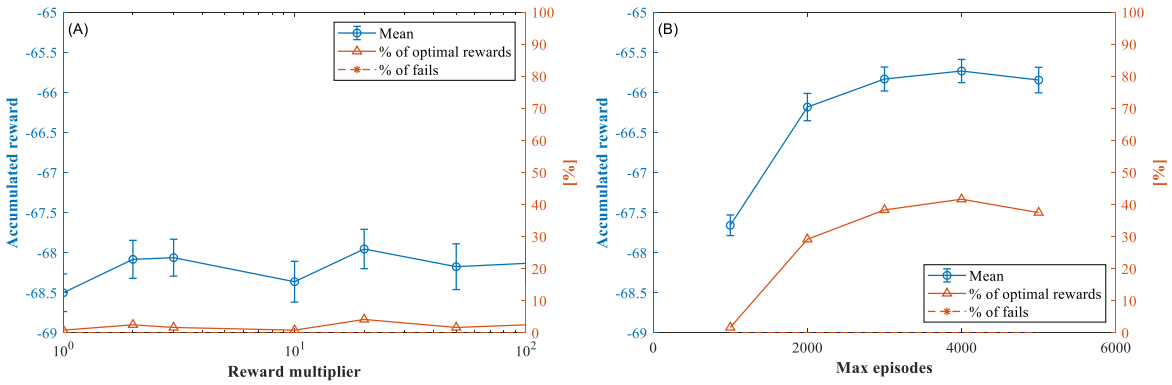

Figure 4.6. Impact of the reward multiplier (A) and maximum number of episodes per experiment (B) on the agent's performance.

In terms of the r_m sensitivity analysis displayed in the Figure 4.6 (A), there is no statistically significant improvements in the agent's performance. However, the value chosen for r_m in future sets of parameters and rewards is 20 as it accomplished the best results. In respect to the maximum number of episodes, presented in the Figure 4.6 (B), it is possible to conclude that increasing the maximum number of episodes per experiment leads to an increase in performance (both visible in the mean and in the percentage of optimal accumulated rewards) until a certain value in which it seems to plateau. Though, is important to remember that a higher value for the maximum number of episodes is related to the amount of times the experiment has to be repeated for the agent to learn, which means that it is essential to maintain the number as low as possible because in a real scenario the maximum number of episodes in an experiment corresponds to the amount of assemblies required to learn the most efficient assembly sequence. Thus, for these parameters, and especially, for the epsilon decay of 0.0001, the optimal maximum number of episodes is 3000. The optimal maximum number of episodes is dependent on the epsilon decay's value

since a lower epsilon decay leads to a slower increase in the greedy selection by the agent and, as such, requires more episodes in the learning phase. In order to identify the relation between these two parameters, a graph of the evolution of the episodic accumulated reward in one of the experiments with 5000 maximum number of episodes is analysed (Figure 4.7).

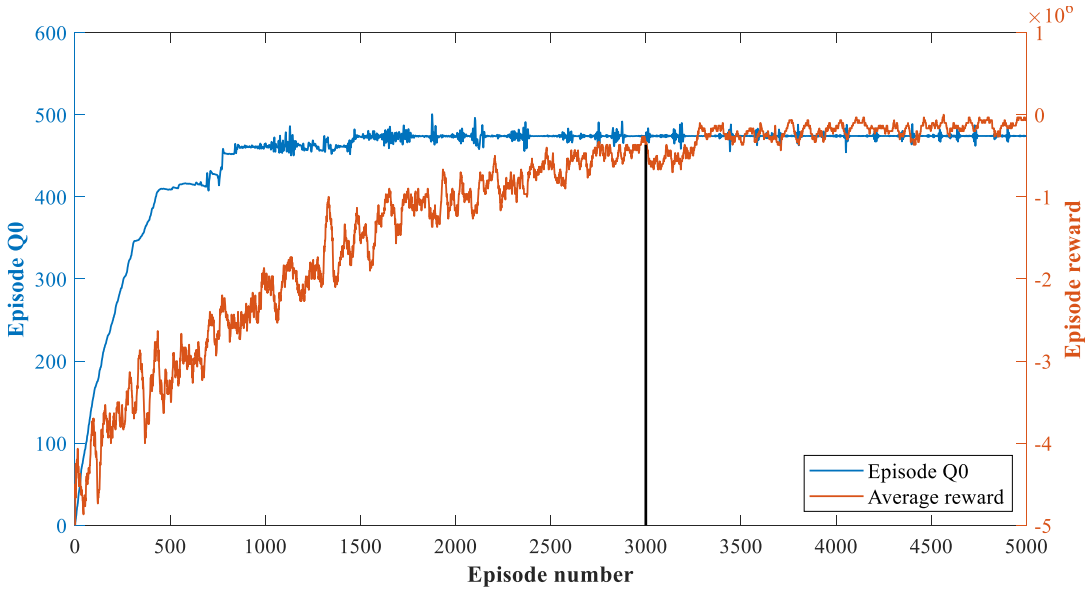

Figure 4.7. Evolution of the episode reward over the episodes for a value of epsilon decay of 0.0001.

When analysing the Figure 4.7, it is possible to observe that after the previously identified optimal maximum number of episodes the episodic accumulated reward does not increase greatly, which could explain why increasing the maximum number of episodes further does not lead to a significant change in the results in the Figure 4.6. Similar graphs were analysed (Figure 4.8) for different values of epsilon decay in order to identify in which episode the episodic accumulated plateaus during an experiment to select this value as the optimal maximum number of episodes per experiment for the given epsilon decay. The optimal pairs of maximum number of episodes per experiment and epsilon decay are displayed in the Table 4.11.

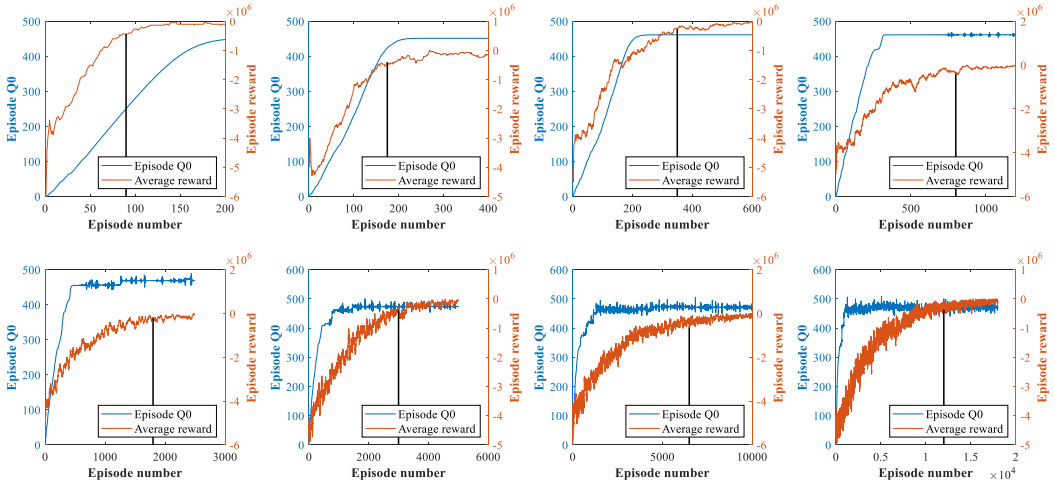

Figure 4.8. Evolution of the episode reward over the episodes for various values of epsilon decay.

Table 4.11. Maximum number of episodes selected for each epsilon decay value.

Epsilon decay	0.005	0.002	0.001	0.0005	0.0002	0.0001	0.00005	0.00003
Max episodes	90	175	350	800	1800	3000	6500	12000

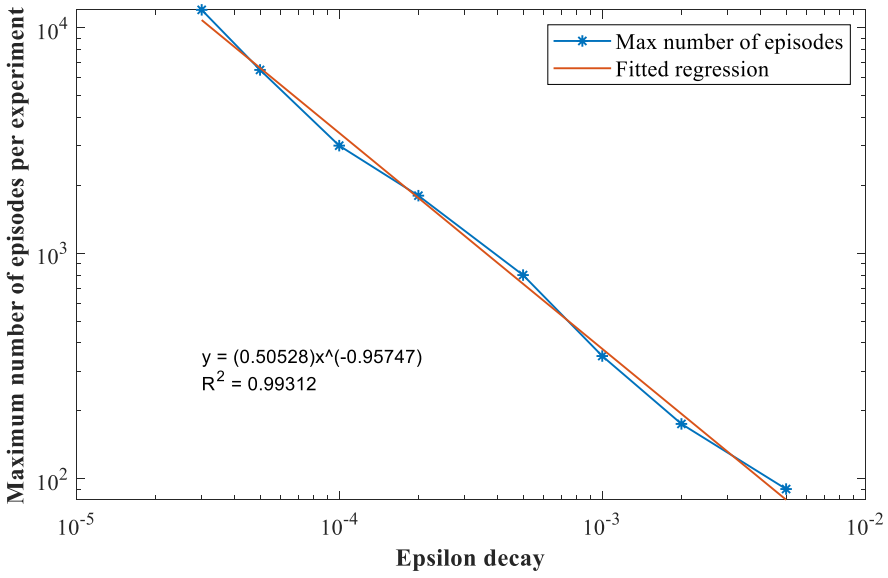

Figure 4.9. Graph of maximum number of episodes over epsilon decay.

From the values available in the Table 4.11, a linear regression was devised using a logarithmic scale in both axes. As shown in the Figure 4.9, the data (Table 4.11)

approximated fits the function $y = 0.50528 \times x^{-0.95747}$ where y is the maximum number of episodes and x the epsilon decay.

Additional new experiments were run with the parameters shown in the Table 4.12, apart from the epsilon decay and maximum number of episodes which were based on the Table 4.11. Their results are displayed in the Figure 4.10.

Table 4.12. Parameters for the pair epsilon decay and maximum number of episodes experiment.

Parameter	Value
Learning rate	1
Discount factor	1
Epsilon	0.9
Max steps per ep.	8
Reward shift	20
Reward multiplier	20
Reward penalty	-1000000

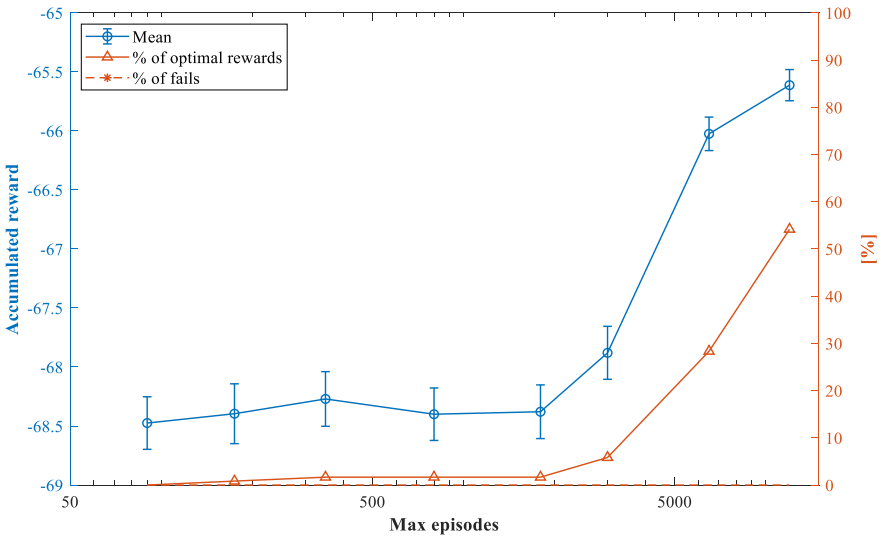

Figure 4.10. Impact of the pairs of epsilon decay and maximum number of episodes on the agent's performance.

As one can observe in the Figure 4.10 the increase in the maximum number of episodes, accompanied by the respective decrease in the epsilon decay, leads to better results

with an increase in the mean and in the percentage of optimal rewards. It is also possible to notice that such an increase only starts around 1800 and 3000 maximum number of episodes.

4.3.2.2. Part 2: Reintroduction of completion rewards

In the second part of this scenario, an earlier concept was reintroduced to explore if it was able to improve the results, the completion reward r_c. For its' reintroduction, the reward function (4.2) needs to be updated to the following equation.

$$\begin{cases} R(s,a) = r_c + r_m \times (-T(s,a) + r_s) & if\ last\ possible\ action \\ R(s,a) = r_m \times (-T(s,a) + r_s) & if\ possible\ action \\ R(s,a) = r_p & if\ impossible\ action \end{cases} \quad \textbf{(4.6)}$$

Three different values for $r_c$were used (0, 100 and 500) with the parameters displayed in the Table 4.12 and with an epsilon decay of 0.000025 and a maximum number of episodes of 13000. The results of these three attempts are visible in the Figure 4.11.

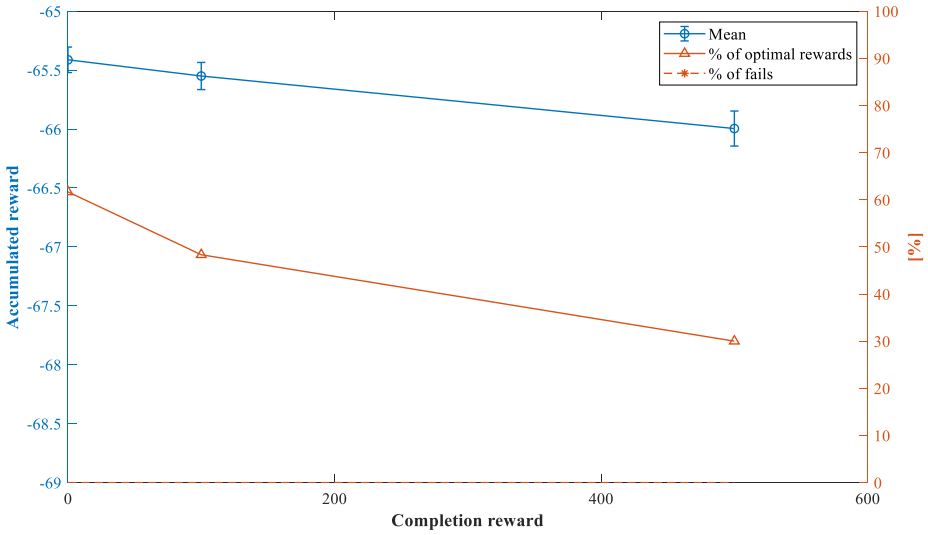

Figure 4.11. Impact of the completion reward on the agent's performance.

As expected, the completion reward worsens the accumulated reward results. This is explained by the addition of the completion reward value to all feasible assembly sequences, which leads to a less noticeable accumulated reward difference between them. The completion reward could be effective to minimize the percentage of fails, however, this percentage is already 0%. For a completion reward of 100, the mean difference is not

statistically relevant, however, both the percentage of optimal rewards and the mean seem to decrease with the increase of the completion reward.

4.3.3. Scenario 3: Learning an assembly sequence based on measured task average times and estimated variances

In the third scenario of this project, each exact task time was measured and repeated 10 times (Table 4.13) so that the average processing times (Table 4.14) would be accurate. Before the measurements, each task was tested several times so that the variability in the times would be as small as possible.

Table 4.13. Task's time measurements.

	Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Measured times [t.u.]	**1**	6.36	9.10	8.59	6.75	10.62	9.87	12.36	10.06
	2	6.28	9.15	10.30	8.22	10.52	11.31	12.39	8.50
	3	4.71	8.00	8.70	7.90	9.82	11.56	12.88	8.03
	4	5.38	8.15	10.49	8.29	9.25	11.00	10.49	9.09
	5	5.80	8.30	8.75	7.25	9.44	8.95	11.22	8.90
	6	6.71	8.52	8.97	7.40	10.04	12.16	11.04	9.37
	7	5.73	8.17	9.00	7.95	9.07	9.24	11.94	7.70
	8	7.31	8.05	8.29	6.30	10.76	10.01	10.34	8.70
	9	5.16	7.62	8.33	7.75	9.95	9.56	10.14	8.15
	10	5.89	6.55	8.50	7.00	9.30	10.09	11.98	9.71
Mean [t.u.]		5.93	8.16	8.99	7.48	9.88	10.38	11.48	8.82

Table 4.14. Task's average time.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Average time [t.u.]	6	8	9	7.5	10	10.5	11.5	9

As in scenario 2, the tasks' average times have variations in respect to the corresponding precedence tasks. These variations are difficult to measure and very time-consuming and for that reason, as in the scenario 2, were estimated which means they are not an accurate depiction of the real scenario. (Table 4.15). It is important to notice, nevertheless, that the tasks variability complexity has increased from the scenario 2 so that there would be a larger variety of accumulated rewards and a lower number of assembly sequences with the largest accumulated reward, i.e. optimal assembly sequences. The variability was also increased by introducing the tool changeover time. Since there are two different types of fasteners, the fastening device's tool must be switched during the assembly process. Regarding the fastening device, there are two main assumptions made, that the assembly process starts without any tool placed and the tool changeover lasts three time units to be performed. In order to introduce the tool changeover, the number of states must be increased since there is a new state information. The new states, apart from the 8 binary variables that define the completion of each task, have a new variable that can have three possible values (0, 1 and 2). The value 0 indicates that the fastening device does not have any tool placed (start of the assembly), the value 1 indicates that the fastening device has the screwdriver applied, and the value 2 indicates the nut driver is applied. The new total number of states is $1 + 255 \times 2 = 511$ since in the first state the fastening device has no tool and in any other state it can have either the first or the second tool. Apart from the impossible states similar to the ones in the scenario 2, which in this case are twice as much (312), there are additional impossible states where the tool is incorrect. Such impossible states are of two types: states where only the task 7 was executed and the first tool is applied, and states where the task 7 was not yet executed and the second tool is applied. The first kind of impossible states only occurs once, when only the task 7 has occurred. The second type of impossible tasks can be subdivided in three groups. In the first group, the task 1 has not been taken, and therefore, tasks 2 through 6 have also not been taken, task 8 has already been executed and the second tool is applied, which accounts for 1 impossible assembly sequence. In the second group, the task 1 has already been executed, while tasks 4 and 5 have not yet been executed, tasks 2, 3, 6 and 8 were either executed or not and the second tool is applied. This group accounts for $2^4 = 16$ impossible states. Lastly, in the third group, the task 1 has already been taken as well as the task 4 and the tasks 2, 3, 5, 6 and 8 were either executed or not, which

accounts for $2^5 = 32$. Therefore, the number of impossible states is $312 + 1 + 1 + 16 + 32 = 362$ and the number of possible states $511 - 362 = 149$.

Table 4.15. Task's variance in respect to the average time in time units.

State	1	2	3	4	5	6	7	8
Task 1 done							0	0
Task 2 done			-2	-3	-0.5	-2	0	1.5
Task 3 done		0		0	0	0	0	0
Task 4 done		-1	0			-1.5	0	0
Task 5 done		-2	-1			-3	2	0
Task 6 done		-1	-0.5	1	-0.5		0	0
Task 7 done	0	0	0	0	0	0		0
Task 8 done	0	0	0	0	0	0	0	

With the respective changes to the average times and time variances the new distribution of accumulated rewards from the feasible assembly sequences can be observed in the Figure 4.12.

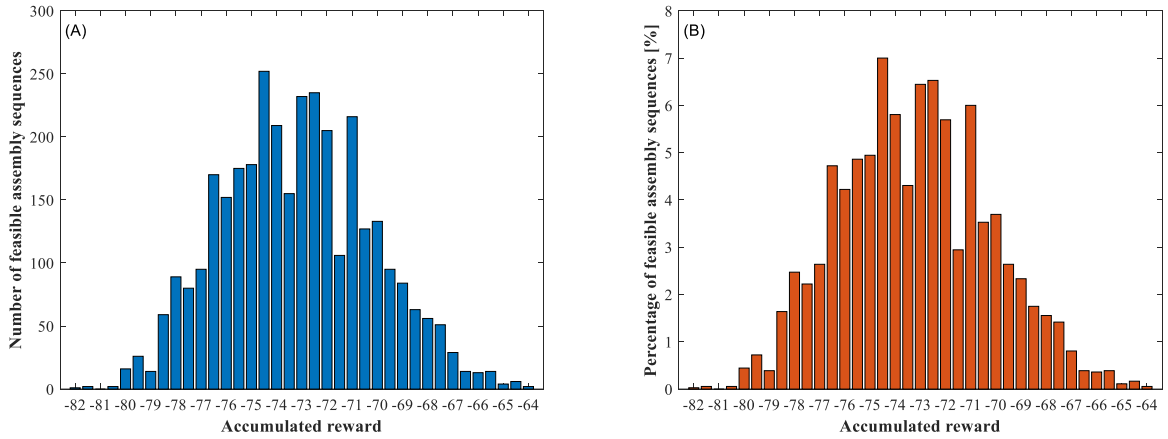

Figure 4.12. Distribution by number (A) and percentage (B) of feasible assembly sequences' accumulated rewards.

The new distribution has, as previously stated, a larger variety of accumulated rewards and the highest accumulated reward or optimal accumulated reward is shared only

by 2 assembly sequences (Figure 4.12 (B)), which are 7 → 1 → 8 → 2 → 4 → 5 → 6 → 3 and 7 → 8 → 1 → 2 → 4 → 5 → 6 → 3, and has the value of -64. In this new scenario, it may be easier to understand the impact of the set's parameters on the agent's performance. With that idea in mind, the reward shift was modified for the values (0, 3, 6, 7, 8, 9, 10 12, 15) while using the values of the Table 4.12, apart from the epsilon decay and maximum number of episodes, which had the values of 0.00005 and 6500 respectively. The results of the multiple sets of 120 experiments are displayed in the Figure 4.13.

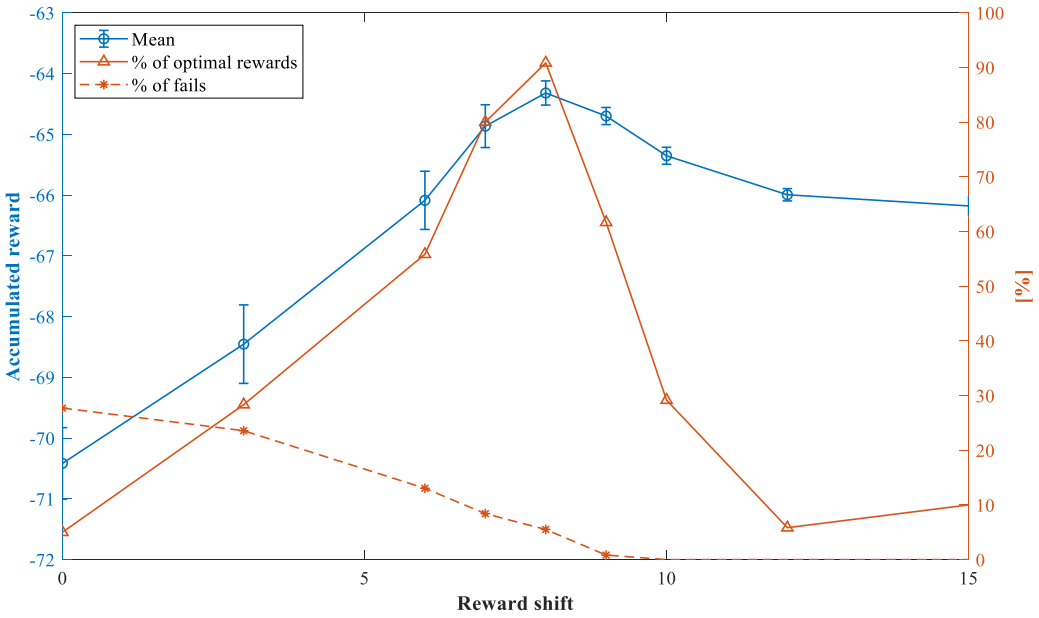

Figure 4.13. Reward shift's impact on the agent's performance.

The mean of all the 37 possible values of accumulated reward is -73 (for a r_s of 0 and r_m of 1), and therefore, if subdivided evenly, each task would have a reward of -9.125, which we will define as mean task reward. A value of r_s equal to the mean task reward shifts the accumulated rewards to a position where they are evenly separated into positive and negative. When analysing the Figure 4.13 it is possible to identify that the best accumulated reward occurs for a value of the reward shift of 8. Also, it is important to notice that the percentage of fails decreases with the increase of the reward shift and is approximately 0 for values higher or equal to 9. Thus, the optimal reward shift may be related to the mean task reward, but it may be relevant to confirm this relation with a different scenario. A value of the reward shift slightly lower than the mean task reward may improve the mean

accumulated reward since a larger number of the accumulated rewards are negative. However, it also increases the percentage of fails, which is highly prejudicial for a real scenario. For that reason, the optimal value for the reward shift is 9. With this new r_s value, the learning rate and the discount factor were individually analysed (Figure 4.14).

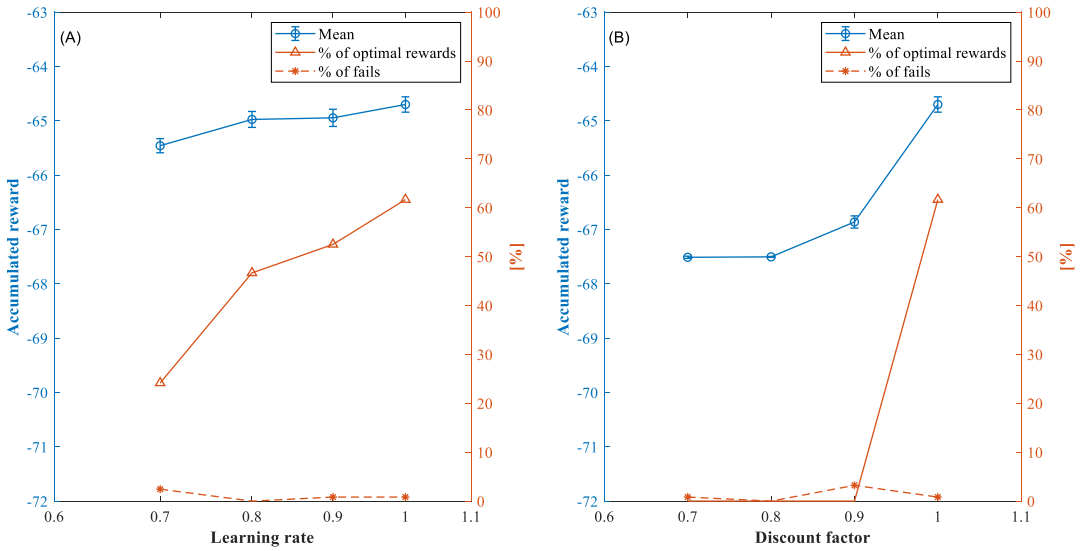

Figure 4.14. Learning rate's and discount factor's impact on the agent's performance.

Since in both cases the confidence interval is very small, even small differences in the mean are significant. Both in the learning rate and in the discount factor, it can be concluded that the optimal value is 1, however, in the discount factor case the difference in the mean and in the percentage of optimal results is more accentuated.

Then, with the same set's parameters as before, the maximum steps per episode was individually analysed (Figure 4.15).

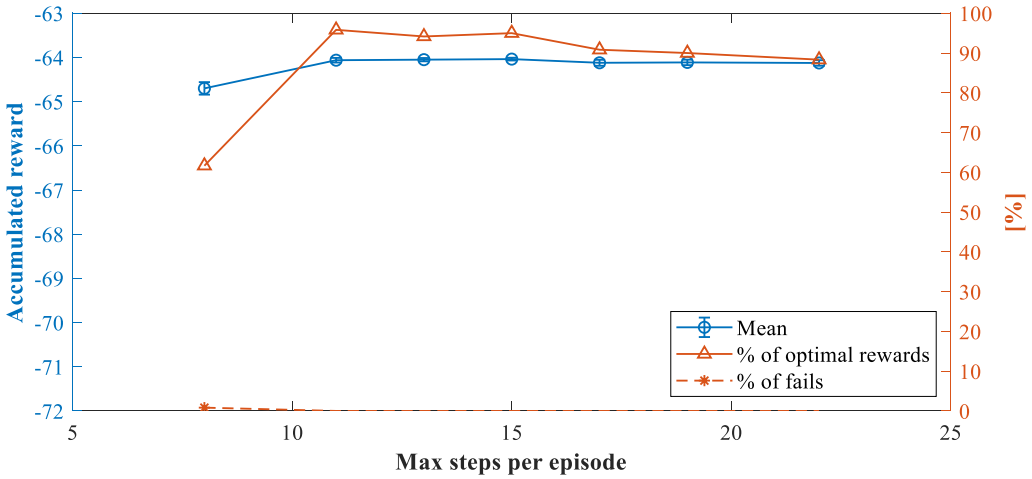

Figure 4.15. Maximum steps per episode's impact on the agent's performance.

It is possible to conclude that an early increase in the maximum number of steps per episode leads to a significant increase both in the mean and in the percentage of optimal rewards. The further increase in this value does not significantly alter the results. The value 15 was selected for this parameter.

Lastly, with all the other parameters decided, the reward multiplier's impact was studied with various values (1, 5, 9, 11, 13, 15, 19, 25, 30) and the experiments results are visible in the Figure 4.16.

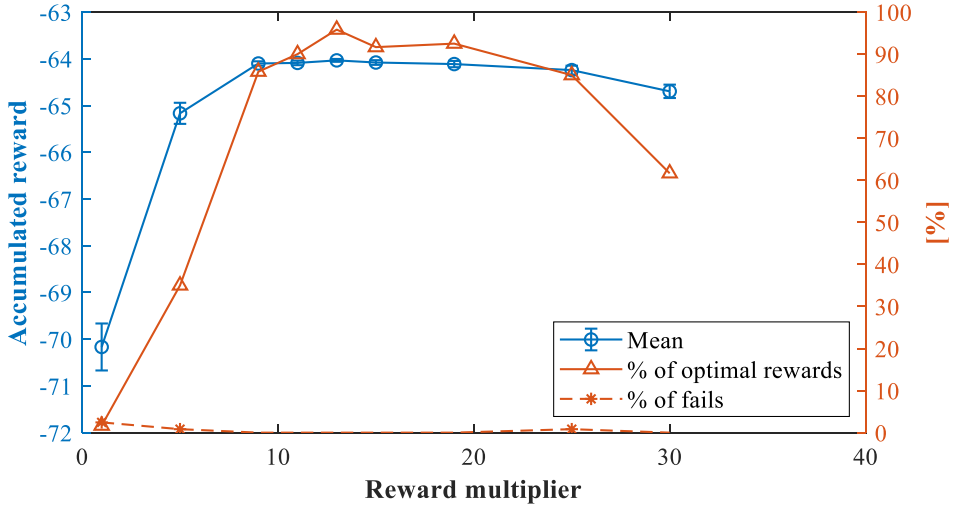

Figure 4.16. Reward multiplier's impact on the agent's performance.

It can be observed that there is an optimal value for r_m since the increase in the value of the reward multiplier leads to an early increase both in the mean and in the percentage of optimal rewards followed by a peak. For the smaller values of the reward multiplier the percentage of fails is nonzero. Based on the graph it can be defined that the optimal reward multiplier value is 13.

After all the experiments and comparisons, it is possible to conclude that the best set's parameters are the ones expressed in the Table 4.16, with which the agent, in 120 experiments, was able to learn one of the 2 optimal assembly sequences 115 times ($\cong$ 95.83%), one of the assembly sequences with the second best accumulated reward 4 times ($\cong$ 3.33%) and one with the fifth best accumulated reward once ($\cong$ 0.83%), while never failing to learn a feasible assembly sequence.

Table 4.16. Optimal set's parameters.

Parameter	Value
Learning rate	1
Discount factor	1
Epsilon	0.9
Epsilon decay	0.00005
Max steps per ep.	15
Max episodes	6500
Reward shift	9
Reward multiplier	13
Reward penalty	-1000000

5. CONCLUSIONS

In this dissertation, the challenges in the application of a reinforcement learning algorithm in a decision-making problem are explored, considering the implementation of a Q-Learning model-free algorithm. By formulating the problem as a MDP, it is shown that Q-Learning finds an optimal state-action policy that maximizes the accumulated reward over a succession of given steps. This allows to verify the application of a scalable method to address the optimization of an assembly sequence problem as a sequential decision process, where the action in one state influences the transition to the subsequent state. Despite its reduced application in the literature, RL methods show a straightforward versatility in complex problems where uncertainty plays a significant role, such as on-line industrial environments, where until now mostly traditional optimization and heuristic approaches are considered.

The improvement of the time efficiency on the assembly of complex products is often impractical due to the complexity of measuring all tasks sequences' times. Such complex problems, however, are proven to be tackled successfully through the usage of reinforcement learning. In fact, this approach has the advantages of achieving good optimization results, where the optimal assembly sequence was learned 95.83% of the times, while never learning an assembly sequence outside the top 1.16% of the assembly sequences (corresponding to the assembly sequences in the top 5 accumulated rewards), and being capable of learning the best assembly sequence by assembling in real-time. However, in the current situation, the algorithm requires 6500 assemblies to correctly learn the best assembly sequence, which is higher than the number of feasible assembly sequences. This is due to the fact that impossible actions are not restricted, but only penalised, which increases the assembly sequence's search space from 3600 (feasible assembly sequences) to 40320 (all assembly sequences), i.e. the algorithm searches the best assembly sequence from all assembly sequences (including impossible ones). When considering this increase in the search space we can conclude that the number of assemblies required by the reinforcement learning algorithm is only 16.12% of all assembly sequences.

5.1. Future work

RL provides an easily understandable and accessible platform to address diverse problems. The work done in this dissertation can be further improved in three main aspects. First of all, a similar analysis could be done with a different assembly process to understand the relation between the optimal Q-Learning algorithm's parameters and rewards, and therefore, comprehend the adaptability of this approach to different scenarios. Then, the reward definition could be improved by measuring the time durations during assembly, without requiring previous knowledge on the task's average time and variabilities. Finally, the efficiency of the approach could be enhanced by implementing the restrictions on impossible actions, which would require a lower number of assemblies to learn the optimal assembly sequence. Withal, the exploration of optimization problems could ultimately expand this approach to combine RL algorithms and mathematical programming approaches to improve results and computing time with the speed and flexibility of RL.

BIBLIOGRAPHY

Abbeel, Pieter. 2008. “Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control.” Stanford, California: Stanford University.

Abe, Naoki, Naval Verma, Chid Apte, and Robert Schroko. 2004. “Cross Channel Optimized Marketing by Reinforcement Learning.” In *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 767–72. New York, NY, United States: Association for Computing Machinery. https://doi.org/10.1145/1014052.1016912.

Akkaladevi, Sharath Chandra, Matthias Plasch, Sriniwas Maddukuri, Christian Eitzinger, Andreas Pichler, and Bernhard Rinner. 2018. “Toward an Interactive Reinforcement Based Learning Framework for Human Robot Collaborative Assembly Processes.” *Frontiers in Robotics and AI* 5 (NOV): 126. https://doi.org/10.3389/frobt.2018.00126.

Amari, Shun Ichi. 1998. “Natural Gradient Works Efficiently in Learning.” *Neural Computation* 10 (2): 251–76. https://doi.org/10.1162/089976698300017746.

Barth-Maron, Gabriel, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. “Distributed Distributional Deterministic Policy Gradients.” *ArXiv*, April, 1–16. http://arxiv.org/abs/1804.08617.

Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson. 1983. “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems.” *IEEE Transactions on Systems, Man and Cybernetics* SMC-13 (5): 834–46. https://doi.org/10.1109/TSMC.1983.6313077.

Bellman, Richard. 2003. *Dynamic Programming*. Mineola, New York: Dover Publications.

Deisenroth, Marc Peter. 2011. *A Survey on Policy Search for Robotics*. *Foundations and Trends in Robotics*. Vol. 2. Now Foundations and Trends. https://doi.org/10.1561/2300000021.

Deisenroth, Marc Peter, and Carl Edward Rasmussen. 2011. “PILCO: A Model-Based and Data-Efficient Approach to Policy Search.” In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 28:465–72. Bellevue, Washington, USA.

Duan, Yan, Xi Chen, C Xi@eecs Berkeley Edu, John Schulman, Pieter Abbeel, and Pabbeel@cs Berkeley Edu. 2016. “Benchmarking Deep Reinforcement Learning for Continuous Control.” *ArXiv*, May. https://github.com/.

Ennen, Philipp, Pia Bresenitz, Rene Vossen, and Frank Hees. 2019. “Learning Robust Manipulation Skills with Guided Policy Search via Generative Motor Reflexes.” In *Proceedings - IEEE International Conference on Robotics and Automation*,

2019-May:7851–57. https://doi.org/10.1109/ICRA.2019.8793775.

Fernandes, Sérgio. 2019. "Faculdade de Engenharia Da Universidade Do Porto Reinforcement Learning Para Problemas de Otimização." FEUP. https://repositorio-aberto.up.pt/handle/10216/122190.

Fu, Justin, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. 2018. "Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition." *Advances in Neural Information Processing Systems* 2018-Decem (May): 8538–47. http://arxiv.org/abs/1805.11686.

Gu, Shixiang, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2017. "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates." In *Proceedings - IEEE International Conference on Robotics and Automation*, 3389–96. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICRA.2017.7989385.

Gu, Shixiang, Timothy Lillicrap, Ilya Sutskever, Sergey Levine, and Slevine@google Com. 2016. "Continuous Deep Q-Learning with Model-Based Acceleration." In *33rd International Conference on Machine Learning, ICML 2016*, 2829–38. New York, New York, USA: PMLR.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." In *35th International Conference on Machine Learning, ICML 2018*, 5:2976–89. International Machine Learning Society (IMLS).

Hafner, Danijar, Timothy Lillicrap Deepmind, Jimmy Ba, Mohammad Norouzi, and Google Brain. 2020. "Dream to Control: Learning Behaviours by Latent Imagination." *ArXiv*.

Hasselt, Hado Van. 2010. "Double Q-Learning." In *Advances in Neural Information Processing Systems 23*, 2613–21.

Hasselt, Hado van, Arthur Guez, and David Silver. 2015. "Deep Reinforcement Learning with Double Q-Learning." In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 2094–2100. AAAI press. http://arxiv.org/abs/1509.06461.

Hester, Todd, Michael Quinlan, and Peter Stone. 2010. "Generalized Model Learning for Reinforcement Learning on a Humanoid Robot." In *Proceedings - IEEE International Conference on Robotics and Automation*, 2369–74. Anchorage, AK, USA: IEEE. https://doi.org/10.1109/ROBOT.2010.5509181.

Hester, Todd, and Peter Stone. 2009. "Generalized Model Learning for Reinforcement Learning in Factored Domains." In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2:717–24. Budapest, Hungary.

Huang, Chengqiang, Yulei Wu, Yuan Zuo, Ke Pei, and Geyong Min. 2018. "Towards Experienced Anomaly Detector Through Reinforcement Learning." In *32nd AAAI Conference on Artificial Intelligence, AAAI-18*, 8087–88. AAAI Press.

Huang, Jing, Qing Chang, and Nilanjan Chakraborty. 2019. "Machine Preventive Replacement Policy for Serial Production Lines Based on Reinforcement

Learning." In *IEEE International Conference on Automation Science and Engineering*, 2019-Augus:523–28. IEEE Computer Society. https://doi.org/10.1109/COASE.2019.8843338.

Jin, Zeshi, Haichao Li, and Hongming Gao. 2019. "An Intelligent Weld Control Strategy Based on Reinforcement Learning Approach." *International Journal of Advanced Manufacturing Technology* 100 (9–12): 2163–75. https://doi.org/10.1007/s00170-018-2864-2.

Kakade, Sham. 2001. "A Natural Policy Gradient." *Advances in Neural Information Processing Systems* 14 (January). https://repository.upenn.edu/statistics_papers/471.

Kober, Jens, J. Andrew Bagnell, and Jan Peters. 2013. "Reinforcement Learning in Robotics: A Survey." *The International Journal of Robotics Research* 32 (11): 1238–74. https://doi.org/10.1177/0278364913495721.

Kober, Jens, and Jan Peters. 2009. "Learning Motor Primitives for Robotics." In *2009 IEEE International Conference on Robotics and Automation*, 2112–18. Kobe, Japan: Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/robot.2009.5152577.

Kohl, Nate, and Peter Stone. 2004. "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion." In *Proceedings - IEEE International Conference on Robotics and Automation*, 2004:2619–24. New Orleans, LA, USA: IEEE. https://doi.org/10.1109/robot.2004.1307456.

Konidaris, George, Scott Kuindersma, Roderic A. Grupen, and Andrew G. Barto. 2011. "Autonomous Skill Acquisition on a Mobile Manipulator." In *25th AAAI Conference on Artificial Intelligence*, 1468–73. AAAI press.

Konidaris, George, Scott Kuindersmay, Andrew Barto, and Roderic Grupen. 2010. "Constructing Skill Trees for Reinforcement Learning Agents from Demonstration Trajectories." In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, 1162–70.

Kormushev, Petar, Sylvain Calinon, and Darwin G. Caldwell. 2010. "Robot Motor Skill Coordination with EM-Based Reinforcement Learning." In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, 3232–37. Taipei, Taiwan: IEEE. https://doi.org/10.1109/IROS.2010.5649089.

Levine, Sergey. 2013. "Guided Policy Search." In *30th International Conference on Machine Learning, ICML2013*, 28:1–9. Atlanta, Georgia, USA: PMLR. https://doi.org/10.1109/ICRA.2015.7138994.

———. 2018. "Introduction to Reinforcement Learning." CS 285. 2018. http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-4.pdf.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. "Continuous Control with Deep Reinforcement Learning." *CoRR*. https://goo.gl/J4PIAz.

Low, Darren Wei Wen, Dennis Wee Keong Neo, and A. Senthil Kumar. 2020. "A

Study on Automatic Fixture Design Using Reinforcement Learning." *International Journal of Advanced Manufacturing Technology* 107 (5–6): 2303–11. https://doi.org/10.1007/s00170-020-05156-6.

Maeda, Yusuke, and Ryohei Aburata. 2013. "Teaching and Reinforcement Learning of Robotic View-Based Manipulation." In *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, 87–92. Gyeongju, South Korea: IEEE. https://doi.org/10.1109/ROMAN.2013.6628454.

Maravall, Darío, Javier de Lope, and José Antonio Martín H. 2009. "Hybridizing Evolutionary Computation and Reinforcement Learning for the Design of Almost Universal Controllers for Autonomous Robots." *Neurocomputing* 72 (4–6): 887–94. https://doi.org/10.1016/j.neucom.2008.04.058.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. "Asynchronous Methods for Deep Reinforcement Learning." In *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–69. PMLR. http://arxiv.org/abs/1602.01783.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. 2015. "Human-Level Control through Deep Reinforcement Learning." *Nature* 518 (7540): 529–33. https://doi.org/10.1038/nature14236.

Ou, Xinyan, Qing Chang, Jorge Arinez, and Jing Zou. 2018. "Gantry Work Cell Scheduling through Reinforcement Learning with Knowledge-Guided Reward Setting." *IEEE Access* 6 (February): 14699–709. https://doi.org/10.1109/ACCESS.2018.2800641.

Proper, Scott, and Prasad Tadepalli. 2006. "Scaling Model-Based Average-Reward Reinforcement Learning for Product Delivery." In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning*, 735–42. Berlin, Germany: Springer. https://doi.org/10.1007/11871842_74.

Rummery, G. A., and M. Niranjan. 1994. "On-Line Q-Learning Using Connectionist Systems." Cambridge, England. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.2539.

Rupam Mahmood, A., Dmytro Korenkevych, Brent J. Komer, and James Bergstra. 2018. "Setting up a Reinforcement Learning Task with a Real-World Robot." In *IEEE International Conference on Intelligent Robots and Systems*, 4635–40. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/IROS.2018.8593894.

Russell, Stuart. 1998. "Learning Agents for Uncertain Environments (Extended Abstract)." In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory - COLT' 98*, 101–3. New York, New York, USA: ACM Press. https://doi.org/10.1145/279943.279964.

Schmajuk, Nestor A., and B. Silvano Zanutto. 1997. "Escape, Avoidance, and Imitation: A Neural Network Approach." *Adaptive Behavior* 6 (1): 63–129. https://doi.org/10.1177/105971239700600103.

Schulman, John, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. “Trust Region Policy Optimization.” *32nd International Conference on Machine Learning, ICML 2015* 3 (February): 1889–97. http://arxiv.org/abs/1502.05477.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov Openai. 2017. “Proximal Policy Optimization Algorithms.” *ArXiv*, August.

Schultz, W., P. Dayan, and P. R. Montague. 1997. “A Neural Substrate of Prediction and Reward.” *Science* 275 (5306): 1593–99. https://doi.org/10.1126/science.275.5306.1593.

Shahrabi, Jamal, Mohammad Amin Adibi, and Masoud Mahootchi. 2017. “A Reinforcement Learning Approach to Parameter Estimation in Dynamic Job Shop Scheduling.” *Computers and Industrial Engineering* 110 (August): 75–82. https://doi.org/10.1016/j.cie.2017.05.026.

Silver, David, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. “Deterministic Policy Gradient Algorithms.” In *Proceedings of the 31st International Conference on Machine Learning*, 387–95. Beijing, China: ICML.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, et al. 2016. “Mastering the Game of Go with Deep Neural Networks and Tree Search.” *Nature* 529 (7587): 484–89. https://doi.org/10.1038/nature16961.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, et al. 2017. “Mastering the Game of Go without Human Knowledge.” *Nature* 550 (7676): 354–59. https://doi.org/10.1038/nature24270.

Singh, Avi, Larry Yang, Chelsea Finn, and Sergey Levine. 2019. “End-To-End Robotic Reinforcement Learning without Reward Engineering.” In *Robotics: Science and Systems 2019*. Robotics: Science and Systems Foundation. https://doi.org/10.15607/rss.2019.xv.073.

Stocker, Cosima, Marc Schmid, and Gunther Reinhart. 2019. “Reinforcement Learning–Based Design of Orienting Devices for Vibratory Bowl Feeders.” *International Journal of Advanced Manufacturing Technology* 105 (9): 3631–42. https://doi.org/10.1007/s00170-019-03798-9.

Sutton, Richard S. 1988. “Learning to Predict by the Methods of Temporal Differences.” *Machine Learning* 3 (1): 9–44. https://doi.org/10.1007/bf00115009.

———. 1990. “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming.” In *Machine Learning Proceedings 1990*, 216–24. Elsevier. https://doi.org/10.1016/b978-1-55860-141-3.50030-4.

Sutton, Richard S., Doina Precup, and Satinder Singh. 1999. “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning.” *Artificial Intelligence* 112 (1): 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1.

Sutton, Richard S, and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction*. *The Lancet*. Vol. 258. The MIT Press.

https://doi.org/10.1016/S0140-6736(51)92942-X.

Tesauro, Gerald. 1995. “Temporal Difference Learning and TD-Gammon.” *Communications of the ACM* 38 (3): 58–68. https://doi.org/10.1145/203330.203343.

Touretzky, David S., and Lisa M. Saksida. 1997. “Operant Conditioning in Skinnerbots.” *Adaptive Behavior* 5 (3–4): 219–47. https://doi.org/10.1177/105971239700500302.

Wang, Yi Chi, and John M. Usher. 2007. “A Reinforcement Learning Approach for Developing Routing Policies in Multi-Agent Production Scheduling.” *International Journal of Advanced Manufacturing Technology* 33 (3–4): 323–33. https://doi.org/10.1007/s00170-006-0465-y.

Watanabe, Keijiro, and Shuhei Inada. 2020. “Search Algorithm of the Assembly Sequence of Products by Using Past Learning Results.” *International Journal of Production Economics* 226 (August): 107615. https://doi.org/10.1016/j.ijpe.2020.107615.

Watkins, Christopher J.C.H. 1989. “Learning from Delayed Rewards.” King’s College.

Watkins, Christopher J.C.H., and Peter Dayan. 1992. “Technical Note: Q-Learning.” *Machine Learning* 8 (3): 279–92. https://doi.org/10.1023/A:1022676722315.

Williams, Ronald J. 1992. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.” *Machine Learning* 8 (3–4): 229–56. https://doi.org/10.1007/bf00992696.

Zhang, Tie, Meng Xiao, Yanbiao Zou, and Jiadong Xiao. 2020. “Robotic Constant-Force Grinding Control with a Press-and-Release Model and Model-Based Reinforcement Learning.” *International Journal of Advanced Manufacturing Technology* 106 (1–2): 589–602. https://doi.org/10.1007/s00170-019-04614-0.