# Reinforcement Learning for Multi-Function Radar Resource Management

Marc Vincent

Sorbonne Université
LIP6

Doctoral School **École Doctorale Informatique, Télécommunications et Électronique**

University Department **LIP6**

Thesis defended by **Marc Vincent**

Defended on **September 8, 2023**

In order to become Doctor from Sorbonne Université

Academic Field **Computer Science**

# Reinforcement Learning for Multi-Function Radar Resource Management

| | | |
|---|---|---|
| **Thesis supervised by** | Amal El Fallah Seghrouchni | Supervisor |
| | Vincent Corruble | Co-Supervisor |

**Committee members**

| | | | |
|---|---|---|---|
| *Referees* | Ann Nowé | Professor at Vrije Universiteit Brussel | |
| | Tristan Cazenave | Professor at Université Paris-Dauphine | |
| *Examiners* | Olivier Sigaud | Professor at Sorbonne Université | Committee President |
| | Gauthier Picard | Senior Researcher at Office National d'Etudes et de Recherches Aérospatiales | |
| *Guest* | Frédéric Barbaresco | Segment Leader at Thales LAS France | |
| *Supervisors* | Amal El Fallah Seghrouchni | Professor at Sorbonne Université | |
| | Vincent Corruble | Associate Professor at Sorbonne Université | |

# Colophon

Doctoral dissertation entitled "Reinforcement Learning for Multi-Function Radar Resource Management", written by Marc Vincent, completed on October 9, 2023, typeset with the document preparation system LaTeX and the yathesis class dedicated to theses prepared in France.

**Keywords:** reinforcement learning, radar, scheduling, multi-objective
**Mots clés :** apprentissage par renforcement, radar, ordonnancement, multi-objectif

This thesis has been prepared at

**LIP6**
Sorbonne Université
Campus Pierre et Marie Curie
4 place Jussieu
75005 Paris
France

Web Site `https://www.lip6.fr/`

## Reinforcement Learning for Multi-Function Radar Resource Management

**Abstract**

In the wake of recent advances in the field of machine learning, much progress has been accomplished in one of its sub-fields, reinforcement learning, whose aim is to solve sequential decision problems under uncertainty. Radar resource management seems to represent an ideal application case for this type of technique. Indeed, a radar emits signals, called dwells, whose echoes are used to measure the state of surrounding objects; these dwells vary according to numerous parameters (duration, beam width...) and must be executed sequentially. The surveillance strategy of a multi-function radar thus consists in continuously selecting the dwells to perform, with the aim of searching the surrounding space while tracking already detected targets. The methods currently used to address this problem are largely heuristic, and are likely to run into difficulties in a range of complex situations involving hyper-velocity or hyper-maneuvering targets.

First, we propose applications of reinforcement learning techniques adapted to the current architecture of multi-function radars. These contributions focus on two aspects : dwell scheduling on the antenna using *model-based* methods, and active tracking dwell optimization using *model-free* methods. Secondly, we highlight the limitations of current resource management architectures, which leads us to consider an alternative architecture for which we propose new reinforcement learning algorithms designed to address the problems it raises. These contributions focus both on the multi-objective aspect, which is useful in multi-function radars to reflect the trade-offs to be made between different functions, and on the combinatorial aspect, which is due to the large number of tasks that the radar must carry out in parallel.

**Keywords:** reinforcement learning, radar, scheduling, multi-objective

**Résumé**

Dans le sillage des avancées récentes dans le champ de l'apprentissage automatique, de nombreux progrès ont été réalisés dans l'un de ses sous-domaines, l'apprentissage par renforcement, dont le but est de résoudre des problèmes de décision séquentielle dans l'incertain. La gestion de ressources radar semble représenter un cadre d'application propice pour ce type de techniques. En effet, un radar émet des signaux, appelés pointages, dont l'écho permet de mesurer l'état des objets alentour ; ces pointages varient selon de nombreux paramètres (durée, largeur de faisceau...) et doivent être exécutés séquentiellement. La stratégie de surveillance d'un radar multi-fonctions revient ainsi à sélectionner en continu les pointages à effectuer dans le but de surveiller l'espace environnant tout en pistant les cibles déjà détectées. Les méthodes utilisées actuellement pour répondre à cette problématique sont en grande partie heuristiques et risquent d'être mises en difficulté dans une gamme de situations complexes impliquant des cibles hyper-véloces ou hyper-manœuvrantes.

Dans un premier temps, nous proposons des applications de techniques d'apprentissage par renforcement adaptées à l'architecture courante des radars multi-fonction. Ces contributions portent sur deux aspects : l'ordonnancement des pointages sur l'antenne par méthodes *model-based* et l'optimisation des pointages de poursuite active par méthodes *model-free*. Dans un second temps, nous mettons en avant les limites des architectures de gestion de ressources actuelles, ce qui nous amène à envisager une architecture alternative pour laquelle nous proposons de nouveaux algorithmes d'apprentissage par renforcement destinés à répondre aux problèmes qu'elle soulève. Ces contributions portent à la fois sur un aspect multi-objectif, utile dans les radars multi-fonctions pour refléter les compromis à réaliser entre les différentes fonctions, et sur l'aspect combinatoire qui est dû au grand nombre de tâches que le radar doit mener à bien en parallèle.

**Mots clés :** apprentissage par renforcement, radar, ordonnancement, multi-objectif

**LIP6**
Sorbonne Université – Campus Pierre et Marie Curie – 4 place Jussieu – 75005 Paris – France

# Contents

# Introduction

Radar, short for "radio detection and ranging," is a technology that uses radio waves to detect and track objects. From their inception as an air defense system in the early 20th century, radars have evolved and diversified, finding applications in weather forecasting, air traffic control, navigation, and various other fields. They have become indispensable tools due to their ability to provide accurate measurements of distant objects and phenomena and gather valuable information about the surrounding environment. The past few decades have witnessed remarkable advancements in radar technology, leading to the development of highly sophisticated and intricate systems with enhanced capabilities.

These significant strides in radar technology can be attributed to the rapid progress in digitalization and electronic systems. The integration of digital components has paved the way for a multitude of improvements, enabling radars to perform a diverse range of missions more efficiently than ever before. One notable advancement is the utilization of phased array antennas, which use multiple individual radiating elements and controlled phase shifting to steer and focus the transmitted or received electromagnetic waves with precision and flexibility. This has brought about the concept of digital beam forming. Beam forming is the process of shaping and directing the radar's waves into a specific pattern or direction to achieve desired coverage and angular resolution. Radars can now dynamically adjust their beam pattern electronically, freeing it from traditional mechanical constraints. Digital beam forming enhances the radar's agility and responsiveness by enabling rapid beam scanning and reconfiguration. This capability is particularly crucial in scenarios where a radar needs to monitor multiple targets simultaneously or rapidly switch between different tasks.

In this context, multi-function radars, which are able to manage searching, tracking, and missile guidance in parallel, have come to represent a key component of air defense systems. However, the efficient utilization of resources, including power, computation

time, and antenna time, poses a significant challenge for these systems. The current resource management strategies, originally designed for earlier generations of radar systems, often fail to fully exploit the potential of these adaptive radars. Consequently, several capabilities inherent in these advanced systems remain underutilized. Furthermore, the optimization of these radars' behavior has become increasingly complex due to the increased number of degrees of freedom of the systems. With a myriad of parameters and variables to consider, achieving optimal radar behavior has become a much more challenging task. In the defense context, the emergence of new threats adds another layer of urgency to the development of advanced radar systems. Of special concern are hypermaneuvering targets, which can change direction exceptionally quickly and frequently, and hypervelocity targets, which travel at extremely high speeds, typically exceeding several kilometers per second. These characteristics are a challenge as they allow the targets to surpass the tracking capabilities of traditional radar systems, underscoring the critical need for advanced radar systems capable of detecting and tracking such elusive targets.

In this context, much effort has been devoted to development of "cognitive radars". The idea is to obtain a system that can learn through interactions with its surrounding environment and use this feedback to intelligently adapt its illumination of the environment, taking target characteristics into account. Analogously to biological systems, this requires the radar to maintain a "mental model" of its surroundings that it can update continuously. The adaptivity that the radar should display is necessitated by the non-stationarity of its environment, whose causes include weather variation and the appearance of unknown targets. This framework naturally presents a favorable application case for artificial intelligence techniques.

Artificial intelligence has evolved significantly since its inception, witnessing remarkable progress in various fields. AI originated in the 1950s with the goal of developing machines that could simulate human intelligence. Early AI focused on symbolic reasoning and rule-based systems, attempting to replicate human decision-making processes through logical algorithms. However, progress was limited by the complexity of real-world problems and the lack of computational power. Gradually, a shift occurred with the rise of machine learning approaches. Machine learning leverages statistical techniques to enable systems to learn from data and improve performance over time. This marked a turning point in AI development. Initially, machine learning focused on tra-

ditional algorithms like decision trees and linear regression. However, the emergence of neural networks and deep learning unlocked new possibilities for AI. Neural networks, inspired by the structure of the human brain, allowed for the creation of more complex and layered models. Deep learning algorithms gained popularity due to their ability to automatically extract hierarchical features from data, leading to breakthroughs in image recognition, speech processing, and natural language processing. The availability of vast amounts of data and the increasing computational power fueled the growth of AI. Today, AI has permeated numerous sectors, including healthcare, finance, transportation, and more.

Machine learning is traditionally divided into three fields: supervised learning, unsupervised learning, and reinforcement learning. While supervised and unsupervised learning operate on labeled and unlabeled data, respectively, reinforcement learning takes a different approach by learning from interaction and feedback received from an environment. At its core, it involves an agent that takes actions in an environment to maximize its long-term expected rewards. The agent learns from the consequences of its actions, receiving feedback in the form of rewards or penalties. Reinforcement learning algorithms learn through an iterative process of exploration and exploitation, where the agent explores the environment to gather data and gradually improves its decision-making policy based on the observed rewards. These principles provide a foundation for developing intelligent agents capable of autonomously learning and making optimal decisions in complex and uncertain environments. Initially, reinforcement learning algorithms were limited by the curse of dimensionality and computational complexity, making practical applications challenging. However, in recent years, they have made significant strides, fueled by the breakthroughs in deep learning. The combination of deep neural networks with reinforcement learning algorithms, known as deep reinforcement learning, has revolutionized the field and led to remarkable success in complex domains, including playing games like Go and chess at a superhuman level and mastering complex control tasks.

Despite these successes, the application of machine learning techniques to real-world scenarios remains an ongoing issue, particularly in the case of reinforcement learning. Ethical considerations and responsible AI development are also gaining prominence to ensure AI is used ethically and with accountability. One specific challenge is ensuring safe reinforcement learning, where algorithms are designed to make reliable decisions without causing harm or engaging in undesirable behavior. A multiplicity of approaches

have been proposed for safe learning: one of them is to cast the problem as multi-objective reinforcement learning, where agents need to balance and optimize multiple conflicting objectives. This allows for the consideration of trade-offs between different criteria such as performance, fairness, and safety.

In the context of the digitization of radars, machine learning techniques represent an alternative worth investigating. However, the scarcity of available data for emerging forms of threat to radar systems constitute an impediment to the application of supervised learning. The interactive nature of radar resource management rather points to reinforcement learning. Given reinforcement learning's ability to handle complex decision-making tasks and adapt to changing conditions, the management of radar resources seems to represent a favorable application framework for this type of technique. Indeed, the surveillance strategy of a multi-function radar must answer a sequential decision problem in the face of uncertainty. The uncertainty arises from the radar's partial information acquisition mechanism, which can only benefit from localized and noisy observations, the obtaining of which requires active listening phases (called dwells) that consumes energy, computing time and, crucially, antenna time. The radar must therefore continuously choose the dwells it performs in succession, which vary according to numerous parameters (duration, beam width, etc.), in order to track the detected targets while maintaining search over its surveillance space. Reinforcement learning could therefore provide a framework to address the challenges posed by the increased degrees of freedom in modern radar systems. It would allow for the development of intelligent agents that can learn from experience and effectively navigate the complex optimization landscape, optimizing the radar's dynamic resource allocation based on environmental conditions, operational requirements, and mission objectives.

However, the multi-function radar use case is also characterized by the presence of multiple tasks to be performed in parallel. The balance between these tasks and the consideration of their interaction represent a combinatorial problem. Combinatorial aspects have been studied in both model-based and model-free reinforcement learning: they represent a key issue, as their exploitation allows extending reinforcement learning methods to large-scale systems. In addition, the performance of each task can be evaluated by a separate criterion, which lends itself to a multi-objective approach. Multi-objective reinforcement learning addresses these problems by learning from multiple reward functions: this allows the agent to take into account trade-offs between different objectives.

This combination of combinatorial and multi-objective aspects represents a convincing perspective for the development of learning agents capable of handling more complex systems.

In this work, we present the following contributions:

- we propose applications of reinforcement learning techniques adapted to the current architecture of multi-function radars. These contributions focus on two aspects: antenna-level scheduling by *model-based* methods and active tracking optimization by *model-free* methods. We also highlight the limitations of current resource management architectures, which leads us to consider an alternative architecture and to conclude that current reinforcement learning methods do not immediately solve the problems raised by these alternatives.
- we investigate new reinforcement learning algorithms to address the problems raised by this alternative architecture. These contributions concern both the multi-objective aspect, which is useful in multi-function radars to reflect the trade-offs to be made between the different functions, and the combinatorial aspect, which is due to the large number of tasks that the radar must carry out in parallel. The first contribution is specifically aimed at the conjunction of multi-objective learning and factorized MDPs, which is a formalism that allows to exploit the internal structure of complex environments. The second contribution focuses only on the multi-objective side: it consists in proposing an algorithm allowing to specify the agent's objective by a non-linear utility function, which would allow to better translate the preferences of the radar operators.

The outline of the dissertation is as follows:

- In chapter 1, we give an overview of the aspects of a multi-function radar system relevant to our work, especially the resource management components.
- In chapter 2, we present the main concepts and algorithms of reinforcement learning.
- In chapter 3, we present our applications of reinforcement learning techniques to current multi-function radars for scheduling and active tracking.
- In chapter 4, we propose multi-objective reinforcement learning algorithms, focusing first on combinatorial environments, and then on stochastic environments

where the objective is defined by non-linear scalarization.

# Chapter 1

# An overview of multi-function radars

A radar (*Radio Detection And Ranging*) is a type of sensor that uses electromagnetic wave echoes to detect and locate objects in its environment. While radars have many applications ranging from flight control to weather forecasting, the use case considered in this thesis is that of a land or maritime radar dedicated to air surveillance in a military context. In this context, radars are mainly used to search for objects entering its surveillance space, to identify them, and to track their movement.

Radars are complex systems involving multiple aspects: high-performance hardware is required to deliver electromagnetic signals of sufficient power; signal processing is necessary to extract meaningful detections from raw sensor data; data processing uses these detections to estimate the presence and characteristics of targets in the radar's environment. Multi-function radars (MFR) use electronically steered phased array antennas, which gives them a certain amount of beam agility thanks to which they are able to carry out several functions in parallel, mainly searching and tracking. This adds a further aspect, resource management, as the radar has to balance its resources between each of its different functions. In this chapter we present an overview of MFRs and discuss the components relevant to this thesis.

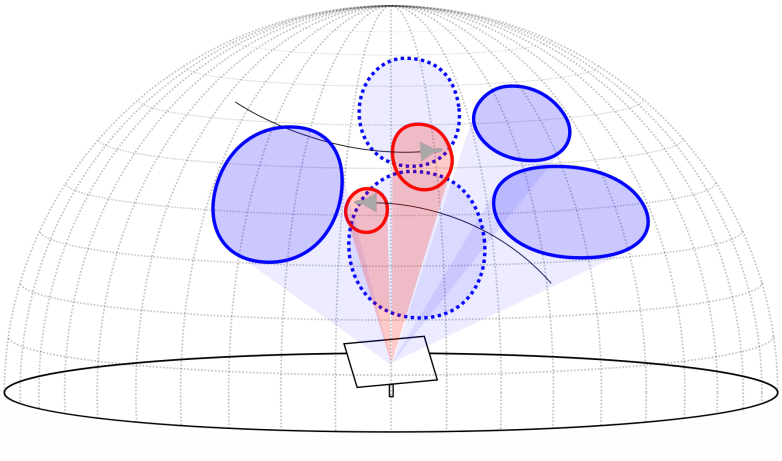

Figure 1.1: Multi-function radars carry out both search and tracking tasks.

## 1.1 Radar fundamentals

In our use case, the radar's main role is to produce position and velocity measurement of distant objects. These measurements are obtained by catching radio signals emitted by these objects via a receiving antenna. Passive radars only use a receiving antenna, while active radars additionally employ an emitting antenna which transmits waves of electromagnetic energy. The environment scatters the energy, and some of the scattered energy is subsequently detected by the receiving antenna. In this thesis, we assume an active monostatic radar, i.e. one that uses the same antenna to transmit and receive signals.

In this section, we explain how the radar creates its emissions, how these interact with the environment to determine the signal received by the radar (as modeled by the radar equation), and what measurements can be obtained from these signals.

### 1.1.1 Dwell model

The radar's emissions are organized by dwells. A dwell is the combination of an illumination law, which dictates the direction and shape of the electromagnetic wave (known as a beam), and a waveform, which corresponds to the structure of the emitted signal. The idea is to create waveforms with characteristic patterns, so that when the radar receives a signal, it tries to find an echo of this pattern to infer the presence of the target that reflected the signal back. A waveform is composed of one or more bursts; a burst is in turn a series of pulses. Each burst is parameterized by the number $N_p$, frequency

$1/T_p$, and duration $\tau$ of its pulses and by the frequency $f$ of the carrier wave used (or alternately, its wavelength $\lambda = \frac{c}{f}$). $T_p$ is known as the pulse repetition interval (PRI) and $\tau$ as the pulse width; together they define the burst's duty cycle $D = \frac{\tau}{T_p}$ which is the ratio of time spent emitting. These parameters define the observation time $T_{obs}$ which is the effective duration of the emitted signal:

$$T_{obs} = T_p N_p D \tag{1.1}$$

Varying these parameters from one burst to another also allows limiting the losses and the ambiguities in the received signal. The accumulation of these pulses (by *integration*) makes it possible to obtain a signal strong enough to allow detection. Since this thesis is focused on the downstream resource management aspect, we do not detail the signal processing component and only provide an energetic model.

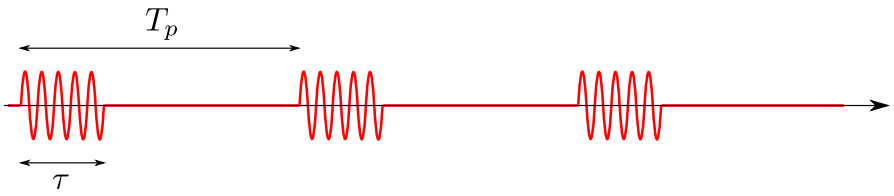


Figure 1.2: Structure of a burst with $N_p = 3$.

### 1.1.2 Beamforming

An electronically scanned phased-array antenna, like the ones used in multi-function radars, comprises an array of radiating elements whose phase and amplitude can be controlled. Array control in modern antennas is fast enough to allow for highly-maneuverable and adaptable beamforming. Thanks to these beamforming capabilities, the antenna can focus its emitting power $P$ in a given direction. This concentration of power is modeled by its transmit gain $g_t$:

$$g_t = 4\pi \frac{A_e}{\lambda^2} \tag{1.2}$$

where $A_e = \eta A_T$ is the antenna's effective aperture area, with $\eta$ the aperture efficiency and $A_T$ the actual aperture area. The sum of the emissions of each element creates a radiation pattern with a main lobe and several side lobes. The main lobe is characterized by its half-power beamwidth:

$$\theta_{3dB} = \frac{0.886\lambda}{l} \tag{1.3}$$

where $l = \hat{l}\eta$ is the length of aperture and $\hat{l}$ is the length efficiency. As we can see, the wavelength $\lambda$ and the aperture efficiency $\eta$ both have opposite effects on the gain and the beamwidth: therefore we cannot increase the beamwidth without decreasing the gain. Phased-array antennas also allow steering the pointing angle of the main beam by applying linear phase increments to each array element. However, when the angle between the center of the beam and the center of the antenna $\theta_{b|a}$ increases, this also increases the deflection loss:

$$L_s = \cos(\theta_{b|a})^{-1} \tag{1.4}$$

which affects both the effective transmit gain and the half-power beamwidth. Intuitively this makes the beam larger and therefore less "focused", which reduces the gain at the center of the beam:

$$\tilde{g}_t = \frac{g_t}{L_s} \tag{1.5}$$

$$\tilde{\theta}_{3dB} = L_s \theta_{3dB} \tag{1.6}$$

When the target is offset from the center of the beam by an angle $\theta_{t|b}$, the directed power received by the target $\tilde{P}$ is further decreased relative to the antenna's peak power $P$ (Blackman and Popoli, 1999):

$$\tilde{P} = P \exp\left(\frac{-F{\theta_{t|b}}^2}{\tilde{\theta}^2_{3dB}}\right) \tag{1.7}$$

where $F$ is a constant. If the target is offset from the beam by an angle corresponding to the half-power beamwidth, i.e. $\theta_{t|b} = \frac{\tilde{\theta}_{3dB}}{2}$, then by definition $\tilde{P} = \frac{P}{2}$, from which we can deduce the value of the constant $F = 4\ln 2$ (Blackman and Popoli, 1999).

### 1.1.3 The radar equation

Assuming the target is situated at range $R$, we can model the power $\tilde{P}$ emitted by the antenna as distributed over a sphere of area $4\pi R^2$. By factoring in the effective transmit

gain $\tilde{g}_t$, the antenna power that reaches the target is therefore:

$$\frac{\tilde{P}\tilde{g}_t}{4\pi R^2} \tag{1.8}$$

The "reflectiveness" of a target is characterized by its radar cross section $\sigma$. Then, similarly, the fraction of power reflected back to the antenna by the target is given by:

$$\frac{\sigma}{4\pi R^2} \tag{1.9}$$

Finally, this signal is intercepted by the antenna's effective aperture $A_e = 4\pi\frac{\tilde{g}_t}{\lambda^2}$. Since the antenna is monostatic, the receive gain $g_r$ is equal to the transmit gain $g_t$. Therefore the deflection loss $L_s$ applies twice. The energy received by the radar combines these three terms and the observation time $T_{obs}$ and is expressed by the radar equation (Skolnik, 2008):

$$E_r = \frac{\tilde{P}\tilde{g}_t}{4\pi R^2}\frac{\sigma}{4\pi R^2}\frac{\lambda^2\tilde{g}_r}{4\pi}T_{obs} \tag{1.10}$$

The radar equation is a fundamental formula that can be used for radar design and performance estimation of a given radar system. In particular, it allows reasoning about the ratio between received signal energy and noise, which determines the radar's detection capabilities and is called the signal-to-noise ratio (SNR):

$$SNR = \frac{2E_r}{N} \tag{1.11}$$

where $N = N_0 B$ is the noise power. Here $B = \frac{1}{\tau}$ represents the radar's bandwidth and $N_0 = kT_s$ is the receiver thermal noise. $N_0$ depends on the Boltzmann constant $k$ and the effective temperature $T_s = F_n T_0$ where $F_n$ is the antenna's noise factor and $T_0 = 290\text{K}$ is the ideal temperature.

The higher the SNR, the simpler the signal will be to detect and provide accurate information about the target. The radar equation informs us on how we can increase the SNR, namely by decreasing the beam width $\theta_{3dB}$ (which increases the directed power $\tilde{P}$) or by increasing the observation time, which both increase the amount of energy emitted at the center of the beam. On the other hand, the larger the offsets $\theta_{b|a}$ between the antenna and the beam on one hand and $\theta_{t|b}$ between the beam and the target on the other hand, the lower the returned signal, and thus the SNR, will be (Barton, 2004).

The radar equation can also be rearranged to express the detection range of a dwell given a required SNR:

$$R = \sqrt[4]{\frac{\tilde{P}\tilde{g}_t \cdot \sigma \cdot \lambda^2 \tilde{g}_r \cdot T_{obs}}{(4\pi)^3 \cdot N \cdot SNR}} \tag{1.12}$$

### 1.1.4 Position and velocity measurements

When the transmitted signal encounters a target, it is reflected and its echo reaches the antenna. The distance $R$ between the target and the radar can be inferred from the time $\Delta t$ between transmission and reception, during which the signal makes a round trip at the speed of light $c$:

$$\Delta t = \frac{2R}{c} \tag{1.13}$$

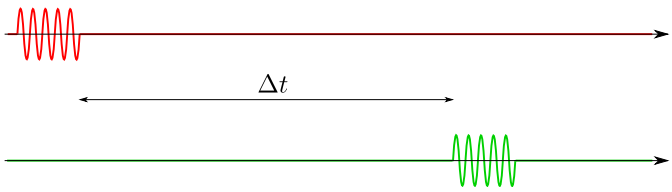


Figure 1.3: Emitted signal at the top, delayed received echo at the bottom.

Targets can only be distinguished from each other if the radial distance separating them is greater than the radar's range resolution $R_{res}$. The range resolution corresponds to the distance traveled during the duration of the pulse width $\tau$:

$$R_{res} = \frac{c\tau}{2} \tag{1.14}$$

Sending several pulses may create range ambiguities if a pulse is received and its associated original transmit pulse cannot be determined. This can happen for targets beyond the unambiguous range $R_{una}$, which depends on the pulse repetition interval:

$$R_{una} = \frac{T_p c}{2} \tag{1.15}$$

The radial velocity $v_r$ of the target causes a shift between the frequencies of the emitted and received signals. This Doppler shift $f_d$ can be used to infer $v_r$:

$$f_d = \frac{2v_r}{\lambda} \tag{1.16}$$

The Doppler resolution is inversely proportional to the duration of the coherent pulse train. Similarly to range ambiguities, Doppler ambiguities happen at high pulse repetition intervals when it is unclear how many multiples of the pulse repetition frequency are contained in the measured Doppler shift.

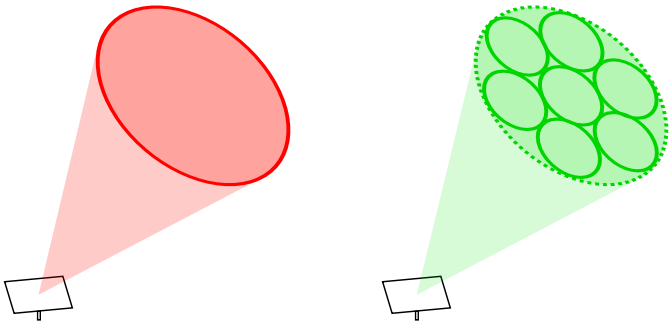

Figure 1.4: Beamforming at emission (left) and at reception (right).

The angular position of the target is estimated by comparing the amplitude of the signal received by several simultaneous receiving beams of a single pulse at slightly different angles. Given two Gaussian receiving beams $b_1(\theta)$ and $b_2(\theta)$, the difference in amplitude between them $\Delta(\theta) = b_1(\theta) - b_2(\theta)$ and the sum of these amplitudes $\Sigma(\theta) = b_1(\theta) + b_2(\theta)$, the error signal response is:

$$k_s(\theta) = \frac{\Delta(\theta)}{\Sigma(\theta)} \tag{1.17}$$

The sensitivity of the measurement is determined by the gradient of the discrimination slope $k_s'(\theta)$ and is expressed as $k_m = k_s'(0)$, which corresponds to the point where the measurement slope and the measurement axis intersect.

The beamwidth and signal-to-noise ratio affect the receiver's thermal noise, which is reflected in the angular error standard deviation:

$$\sigma_\theta = \frac{\tilde{\theta}_{3dB}}{k_m\sqrt{2SNR}} \tag{1.18}$$

Furthermore, if the target is off the center of the beam by an angle $\theta_{t|b}$, then the effective angular accuracy $\tilde{\sigma}_\theta$ further decreases as a factor of the beamwidth:

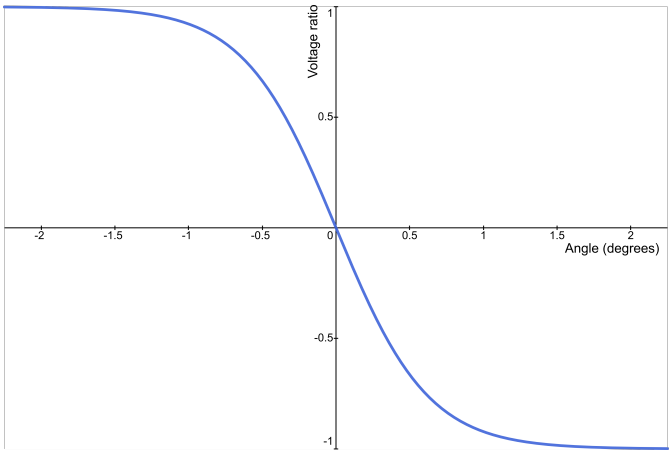


Figure 1.5: Example of error signal response $k_s(\theta)$.

$$\tilde{\sigma}_\theta = \sigma_\theta \sqrt{1 + \left( k_m \frac{\theta_{t|b}}{\tilde{\theta}_{3dB}} \right)^2} \tag{1.19}$$

## 1.2 Data processing

The signal processing components of the radar produce raw measurement data. This data must however be further processed to distinguish clutter (i.e. false detections) from real targets and to infer the trajectories of these targets from sequential detections.

### 1.2.1 Detection

Data obtained in the signal processing phase first goes through a detection phase. The idea is to test the hypothesis $H_0$ of absence of a target and the hypothesis $H_1$ of presence of a target in the received echo. This test is not carried out on each pulse of the dwell, since the pulses are individually too weak to allow detection. Instead, the pulses can be combined to improve detection in a process known as integration. The two main types of integration are coherent integration, when the amplitude and phase of the signal are taken into account, and incoherent integration, when only the amplitude is used.

This process entails a trade-off between the probability of detection $P_d$, which is when a real target is correctly detected, and the probability of false alarm $P_{fa}$, which is when noise in the receiver is incorrectly interpreted as a target. For a desired probability of false alarm, the optimal decision is defined as a threshold $T$ on the likelihood ratio $\Lambda(x) = \frac{p(x \mid H_1)}{p(x \mid H_0)}$ of an observation $x$, such that $H_1$ is chosen if $\Lambda(x) > T$, $H_0$ otherwise; the

probability of false alarm determines the threshold, as $P_{fa} = p(x > T \mid H_0)$. Assuming a square law envelope detector, the probability of false alarm can be written as $P_{fa} = e^{-T}$.

For a given detection threshold, the probability of detection depends on the fluctuation model of the target. For example, in Swerling models 1 and 2, complex targets are modeled as multiple small scatters, and the associated probability of detection is approximated as a function of SNR and probability of false alarm:

$$P_d = P_{fa}^{\frac{1}{1+SNR}} \tag{1.20}$$

As a consequence, to increase the probability of detection, one must increase the detection threshold, yet this will also result in a higher probability of false alarm, so that a trade-off between missed targets and false alarms must be made.

### 1.2.2 Tracking

When a detection occurs, the information is transmitted to a tracking module. A tracking module is composed of a filtering algorithm, a data associator, and a track manager. The filter's role is to estimate and predict the trajectory of the detected targets. Data association establishes correspondences between received detections and existing tracks. Finally, track management determines the life cycle of tracks, in particular when to create a track and when to delete it.

#### Filtering

The most common filtering algorithm for radars is the Kalman filter. Given linear models with Gaussian perturbation describing the target kinematics and its measurement function, the Kalman filter provides a closed-form solution to estimate the target state (position, velocity, possibly acceleration) and the variance of the estimation error (i.e. the tracking accuracy) from the observations collected by the antenna, and to give a prediction on the evolution of this state. In this framework, the evolution of the target's state in Cartesian coordinates $\mathbf{x}_t = [x, \dot{x}, y, \dot{y}]^\top$ at time $t$ is described by the linear equation:

$$\mathbf{x}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{w}_{t-1} \tag{1.21}$$

where $\mathbf{A}_t$ is the transition matrix and $\mathbf{w}_{t-1}$ is the process noise with covariance $\mathbf{Q}_t$, which together form the transition model. Similarly, measurements $\mathbf{z}_t$ are represented as derived from the underlying state by a linear transformation:

$$\mathbf{z}_t = \mathbf{H}_t\mathbf{x}_t + \mathbf{v}_t \tag{1.22}$$

where $\mathbf{H}_t$ is the measurement matrix and $\mathbf{v}_t$ is the measurement noise with covariance $\mathbf{R}_t$, which together form the measurement model. The value of $\mathbf{R}_t$ can be readily derived from the range and angle accuracy estimations in section 1.1.4.

The filter maintains mean estimates of $\mathbf{x}_t$: $\hat{\mathbf{x}}_t^-$ is the a priori state estimate given $\mathbf{z}_{t-1}$ and $\hat{\mathbf{x}}_t$ is the a posteriori state estimate given $\mathbf{z}_t$. From the estimate errors $\mathbf{e}_t^- = \mathbf{x}_t - \hat{\mathbf{x}}_t^-$ and $\mathbf{e}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$, we can express the estimate error covariance matrices $\mathbf{P}_t^- = \exp[\mathbf{e}_t^-\mathbf{e}_t^{-\top}]$ and $\mathbf{P}_t = \exp[\mathbf{e}_t\mathbf{e}_t^\top]$. The mean and covariance are estimated iteratively in three phases, starting with a *prediction* phase:

$$\hat{\mathbf{x}}_t^- = \mathbf{A}_t\hat{\mathbf{x}}_{t-1} \tag{1.23a}$$

$$\mathbf{P}_t^- = \mathbf{A}_t\mathbf{P}_{t-1}\mathbf{A}_t^\top \tag{1.23b}$$

This is followed by a *measurement* phase which uses the measurement $\mathbf{z}_t$ to determine the innovation covariance matrix $\mathbf{S}_t$, the measurement residual $\delta_t$ and the gain $\mathbf{K}_t$:

$$\delta_t = \mathbf{z}_t - \mathbf{H}_t\hat{\mathbf{x}}_t^- \tag{1.24a}$$

$$\mathbf{S}_t = \mathbf{H}_t\mathbf{P}_t^-\mathbf{H}_t^\top + \mathbf{R}_t \tag{1.24b}$$

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}_t^\top\mathbf{S}_t^{-1} \tag{1.24c}$$

Finally, the *update* phase corrects the estimates:

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t\delta_t \tag{1.25a}$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_t^- \tag{1.25b}$$

Variations of this algorithm such as the extended Kalman filter and the unscented

Kalman filter allow to take into account non-linear transition models.

Another issue is that most often the target dynamics will evolve over time, which requires switching between different transition models when the target maneuvers. The most effective approach is the Interacting Multiple Model (IMM). On a given track, the IMM applies several filters in parallel, each using a different transition model. The IMM uses the measurements to evaluate the likelihood of each model and assign them a probability, taking into account the a priori transition probability matrix between models. The combined state estimate is a mean over the estimates of the different models, weighted by their respective probabilities.

More formally, an IMM is parameterized by $N$ models (e.g. Kalman filters), where each model $i$ has an initial probability $\mu_0^i$ and the probability of transition from model $i$ to $j$ is $p^{ij}$. The IMM maintains estimated states, covariances, and probabilities $(\hat{\mathbf{x}}_t^i, \mathbf{P}_t^i, \mu_t^i)$ for each model $i$. Similarly to the Kalman filter, the IMM goes through three phases, applied in parallel to each model $j$. The *prediction* phase takes the transition probabilities and the current estimated model probabilities to form a prediction for each model $j$:

$$\tilde{\mu}_t^j = \sum_{i=1}^{N} \hat{\mu}_{t-1}^i p^{ij} \tag{1.26a}$$

$$\tilde{\mu}_t^{ij} = \frac{\hat{\mu}_{t-1}^i p^{ij}}{\tilde{\mu}_t^j} \tag{1.26b}$$

$$\hat{\mathbf{x}}_{t-1}^{(j)}, \hat{\mathbf{P}}_{t-1}^{(j)} = combination((\hat{\mathbf{x}}_{t-1}^i, \hat{\mathbf{P}}_{t-1}^i, \tilde{\mu}_t^{ij})_{1 \leqslant i \leqslant N}) \tag{1.26c}$$

$$\tilde{\mathbf{x}}_t^j, \tilde{\mathbf{P}}_t^j = prediction(\hat{\mathbf{x}}_{t-1}^{(j)}, \hat{\mathbf{P}}_{t-1}^{(j)}) \tag{1.26d}$$

where the per-model *prediction* is that of a Kalman filter as in eq. (1.23) and the *combination* forms new estimates of the state and covariance based on the updated model probabilities:

$$\mathbf{x} = \sum_{i=1}^{N} \mu^i \mathbf{x}^i \tag{1.27a}$$

$$\mathbf{P} = \sum_{i=1}^{N} \mu^i [\mathbf{P}^i + (\mathbf{x}^i - \mathbf{x})(\mathbf{x}^i - \mathbf{x})^\top] \tag{1.27b}$$

The predicted estimates of each model then go through the *measurement* phase spe-

cific to each Kalman filter (cf. eq. (1.24)):

$$\delta_t^j, \mathbf{S}_t^j, \mathbf{K}_t^j = measurement(\tilde{\mathbf{x}}_t^j, \tilde{\mathbf{P}}_t^j, \mathbf{z}_t) \tag{1.28}$$

The measurements are used to *update* the state and covariance according to each model (cf. eq. (1.25)) and the model probabilities according to their likelihoods $\Lambda_t^j$:

$$\hat{\mathbf{x}}_t^j, \hat{\mathbf{P}}_t^j = update(\tilde{\mathbf{x}}_t^j, \tilde{\mathbf{P}}_t^j, \delta_t^j, \mathbf{S}_t^j, \mathbf{K}_t^j) \tag{1.29a}$$

$$\Lambda_t^j = \frac{1}{\sqrt{|2\pi \mathbf{S}_t^j|}} \exp(-\frac{1}{2}(\delta_t^j)^\top (\mathbf{S}_t^j)^{-1}(\delta_t^j)) \tag{1.29b}$$

$$\hat{\mu}_t^j = \frac{\Lambda_t^j \tilde{\mu}_t^j}{\sum_{i=1}^N \Lambda_t^i \tilde{\mu}_t^i} \tag{1.29c}$$

Finally, the aggregated estimates of the state and covariance are obtained through a combination from each model weighted by their probabilities:

$$\hat{\mathbf{x}}_t, \hat{\mathbf{P}}_t = combination((\hat{\mathbf{x}}_t^j, \hat{\mathbf{P}}_t^j, \hat{\mu}_t^j)_{1 \leqslant j \leqslant N}) \tag{1.30}$$

**Transition models**

The definition of adequate transition models is a major problem for radar systems: different models are needed to cover all the maneuvers that can be performed by a given type of target. Among the most commonly used models for linear dynamics are constant $n$th-derivative models, which describe the evolution of an arbitrary position coordinate $x$ by assuming that its $n$th-derivative $x^{(n)}$ remains approximately constant over time while its $(n+1)$th-derivative is represented by a white noise process of strength $q$. This can be described in terms of stochastic differential equations as:

$$dx^{(n-1)} = x^{(n)} \cdot dt \tag{1.31a}$$

$$dx^{(n)} = q \cdot dW_t \text{ with } W_t \sim \mathcal{N}(0, q^2) \tag{1.31b}$$

For $n = 1$, this yields a constant velocity model with white noise acceleration applied to a state $\mathbf{x}_t = [x, \dot{x}]^\top$ that has the following matrix form:

$$\mathbf{A}_t = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \tag{1.32}$$

$$\mathbf{Q}_t = \begin{bmatrix} \frac{dt^3}{3} & \frac{dt^2}{2} \\ \frac{dt^2}{2} & dt \end{bmatrix} q \tag{1.33}$$

Similarly, for $n = 2$, we obtain a constant acceleration model with white noise jerk applied to a state $\mathbf{x}_t = [x, \dot{x}, \ddot{x}]^\top$:

$$\mathbf{A}_t = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \tag{1.34}$$

$$\mathbf{Q}_t = \begin{bmatrix} \frac{dt^5}{20} & \frac{dt^4}{8} & \frac{dt^3}{6} \\ \frac{dt^4}{8} & \frac{dt^3}{3} & \frac{dt^2}{2} \\ \frac{dt^3}{6} & \frac{dt^2}{2} & dt \end{bmatrix} q \tag{1.35}$$

To describe maneuvers, constant turn models can be used. In these, the target is assumed to move at constant velocity with a constant turn rate $\omega$ in an $(x, y)$ plane. For state $\mathbf{x}_t = [x, \dot{x}, y, \dot{y}]^\top$ and acceleration noise coefficients $q_x$ and $q_y$, the matrix form of a constant turn model is:

$$\mathbf{A}_t = \begin{bmatrix} 1 & \frac{\sin \omega dt}{\omega} & 0 & -\frac{1-\cos \omega dt}{\omega} \\ 0 & \cos \omega dt & 0 & -\sin \omega dt \\ 0 & \frac{1-\cos \omega dt}{\omega} & 1 & \frac{\sin \omega dt}{\omega} \\ 0 & \sin \omega dt & 0 & \cos \omega dt \end{bmatrix} \tag{1.36}$$

$$\mathbf{Q}_t = \begin{bmatrix} q_x^2 \frac{dt^3}{3} & q_x^2 \frac{dt^2}{2} & 0 & 0 \\ q_x^2 \frac{dt^2}{2} & q_x^2 dt & 0 & 0 \\ 0 & 0 & q_y^2 \frac{dt^3}{3} & q_y^2 \frac{dt^2}{2} \\ 0 & 0 & q_y^2 \frac{dt^2}{2} & q_y^2 dt \end{bmatrix} \tag{1.37}$$

Alternative linear models include $n$th-derivative decay models, where the $n$th-derivative exponentially decays to zero over time. Similarly to before, the $(n+1)$th-derivative is described by a white noise process of strength $q$:

$$dx^{(n-1)} = x^{(n)} \cdot dt \tag{1.38a}$$

$$dx^{(n)} = -Kx^{(n)} \cdot dt + q \cdot dW_t \text{ with } W_t \sim \mathcal{N}(0, q^2) \tag{1.38b}$$

These are called Ornstein-Uhlenbeck models and Singer models for $n = 1$ and $n = 2$ respectively.

### Data association

Data association is required to determine if a detection corresponds to one of the existing tracks before its tracking filter can be updated. This is necessary to avoid updating tracks with false alarms caused by thermal noise or clutter, or when multiple targets are present in the environment. The problem is compounded when multiple detections occur (near-)simultaneously. This involves considering all association hypotheses between each detection and each track.

The first step is to filter out the most unlikely hypotheses. To do so for a given hypothesis, the gap between the track's predicted measurement and the detection measurement is evaluated with a chosen measure, and the hypothesis is rejected if the distance between the two measurements exceeds a predefined threshold. A common measure is the Mahalanobis distance

$$\sqrt{\delta_t \mathbf{S}_t^{-1} \delta_t} \tag{1.39}$$

where $\mathbf{S}_t$ is the innovation covariance matrix and $\delta_t$ the measurement residual as defined in 1.25. The advantage of this distance is that it takes into account the uncertainty of the Kalman filter over the actual position of the target.

Once unlikely hypotheses have been excluded, track–detection association can be performed. The simplest method is global nearest neighbor (GNN), which assigns each detection to a track in a way that minimizes the sum of distances between the detections and the track predictions. A more computationally-intensive alternative is probabilistic data association (PDA), where each hypothesis is assigned a probability and tracks are updated with the weighted average of their associated hypotheses.

#### Track management

The management of a track is generally organized in the form of a life cycle, with logic rules defining the transitions between the different stages of the cycle: when a new target is detected, a certain number of confirmation hits are usually required before initiating the track; the track must then be updated regularly to maintain a desired level of accuracy; if no detection is made at the expected location of the target, a reacquisition procedure can be initiated; finally, after a certain number of missed detections or if a given track metric such as angular covariance exceeds a given threshold, the target is considered lost and the track is abandoned. More advanced methods make use of Markov chains or estimate the likelihood of the track to determine its status.

Furthermore, due to constraints on the radar's angular and radial resolution, targets that are too close together cannot be distinguished: it may therefore be necessary to merge or split tracks when targets move closer or further apart.

## 1.3 Resource management

### 1.3.1 Multi-function radars

Multi-function radars (MFR) are able to concurrently perform a wide range of functions by varying parameters related to waveform selection and beamforming, such as the observation time and the half-power beamwidth. Radar functions center on searching for new targets and tracking existing ones. Additional functions may include missile guidance, target recognition and calibration. Each function is accomplished by tasks; for example a tracking task refers to the tracking of a single target. A task is performed by a number of dwells.

Figure 1.6: Thales' GF300, an example of multi-function radar.

Having to perform multiple functions in parallel poses a resource management problem. Radar resource management (RRM) consists in allocating resources such as time, power, and processing between functions according to the radar's mission, which sets the level of priority of each function. Among these resources, time budget is the most critical: it includes the transmission time, the waiting time (before the signal returns) and the reception time associated with each dwell played. The challenge is therefore to generate dwells that fulfill the various functions assigned to the radar with a limited antenna time. Using a rotating antenna (with a fixed rotation frequency) creates an additional constraint on the possible execution dates of the dwells. Since each task is performed by executing a number of dwells (or tasks), it is necessary to continuously schedule these dwells by assigning them order and execution times. All dwells have optimal execution times, and minimizing delays from these is one of the scheduling objectives. More importantly, it may happen that temporal constraints prevent from executing all dwells, which leads to an overload situation and forces the scheduler to drop some of them according to their level of priority.

Resource management also affects dwell optimization. The greater number of degrees of freedom for beamforming in MFRs means that the space of possible dwell configurations is significantly larger. This makes optimizing waveforms more difficult, especially for functions like tracking whose tasks must be dynamically adapted (Moo and Ding,

2015). Moreover, the choice of dwell parameters should aim not only to maximize its function's performance, but also to lower resource requirements for the scheduler in order to avoid overload.

Resource management operates on the representation of the tactical situation around the radar in terms of tracks resulting from data processing. Radar resource management is generally divided into three phases, which we detail below: task management, prioritization and scheduling.

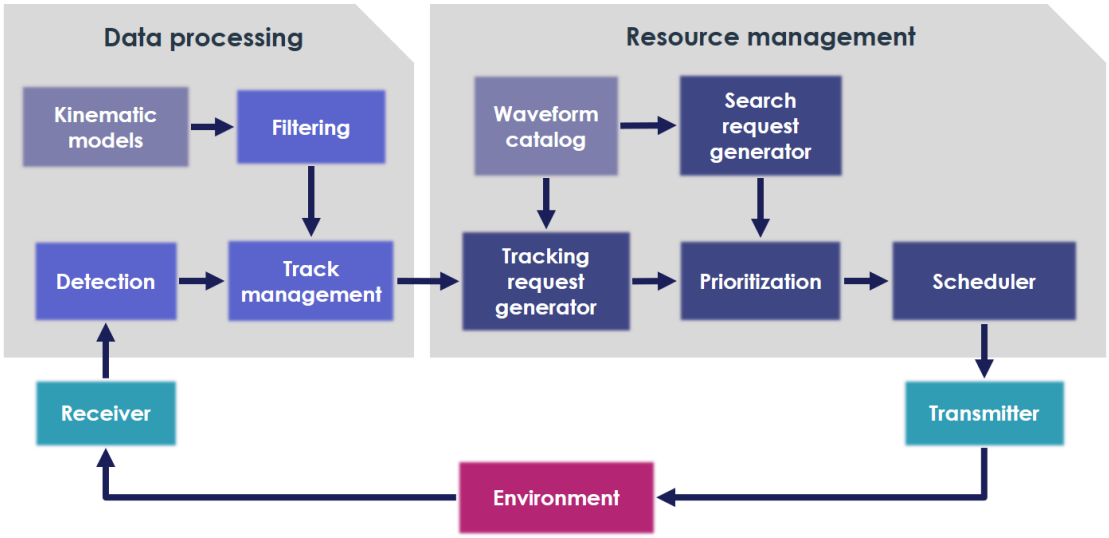


Figure 1.7: High-level view of a MFR.

### 1.3.2 Dwell optimization

Task management consists of generating the dwell parameters for the different functions of the radar.

For the search function, the relevant operational parameters include the detection threshold, the pattern of beams covering the surveillance space, and the revisit interval and observation time for each search dwell. Alert-confirm techniques are enabled by beam agility, where a lowered detection threshold acts as a first stage to produce alerts, followed by a confirmation dwell. The beam pattern over the surveillance space must take into account the spacing between the beams and the beam width, since larger beamwidths create larger losses when the target is offset from the centre of the beam, which can be compensated by a smaller spacing. Each of these correspond to a transmit beam, while the electronically scanned antenna allows to use multiple simultaneous receiving

beams (cf. fig. 1.4), enabling higher angular resolution and faster search. Finally, the observation time and revisit interval of each dwell are inversely related and need to be balanced. Heuristic solutions using e.g. triangular search pattern are usually the nom, yet the need for resource management techniques that can adapt to dynamic and uncertain scenarios has prompted research into mathematical optimization applications for search patterns (Briheche, 2017).

Updating a track can be done in two ways: either by a dedicated dwell, which constitutes active tracking, or by exploiting the detections resulting from search dwells (*Track-While-Scan*, TWS). TWS is generally reserved for distant or low-threat targets. Active tracking dwells are centered on the predicted target position, thus more accurate than a search dwell. Besides waveform parameters, which can improve range resolution when dynamically adapted, the critical parameter of active tracking is the track update rate, which must maintain the necessary track accuracy without using too much resource. The classic approach to update rate selection is Blackman and Van Keuk's method, which selects the next update as the the earliest time after which the filter's angular prediction error surpasses a given fraction of the beamwidth. Tracking dwells length are usually chosen to maintain a desired SNR. Aside these heuristic methods, optimization approaches exist that include Q-RAM (Quality of Service Resource Allocation Method) and information-theoretic optimization.

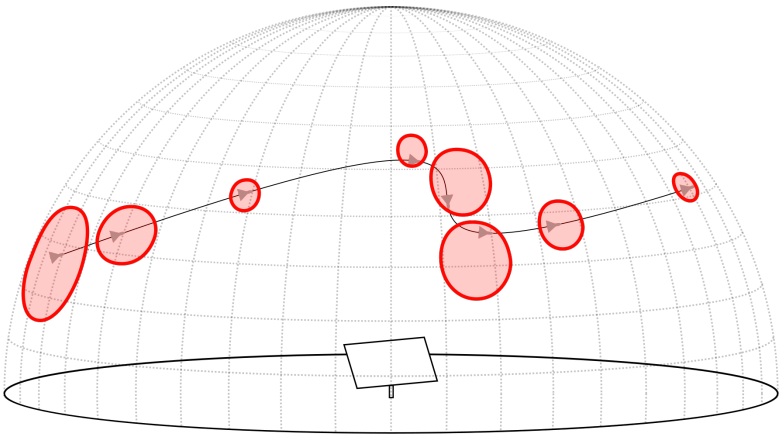

Figure 1.8: Maneuvering targets require higher update rates and may require to adapt the beam width.

### 1.3.3 Task prioritization and scheduling

The request managers corresponding to different radar functions continuously update a list of requested dwells. These dwells must then be assigned a priority level, which reflects how critical the dwell in question is to fulfill the radar's mission. This priority level may depend on mission parameters (e.g. prioritized search versus prioritized tracking), the nature of the associated task, or the threat level of a given target. It comes into play when the radar is overloaded, i.e. not all requested dwells can fit onto the antenna timeline and the system must select which dwells to postpone or abandon. In practice, priority assignment is determined by rule-based systems or by ranking dwells according to heuristically-computed scores.

Finally, the dwell requests and their priorities are transmitted to a scheduler whose goal is to program the order and the execution dates of the various dwells on the antenna, and to reject certain dwells if all the requests cannot be satisfied in the allotted time. The two main approaches are frame-based and best-first schedulers (Blackman and Popoli, 1999). Frame-based methods attempt to fill in a fixed-duration time frame with dwell requests while a previous frame is being performed by the antenna, whereas best-first methods maintain a queue of dwell requests ordered by heuristic criteria and repeatedly retrieve the top dwell to execute it. Frame-based schedulers allow to minimize delays between the dwells' requested and effective execution time, while best-first schedulers are more reactive and ensure greater antenna occupancy.

# Chapter 2

# Reinforcement learning background

This chapter is intended to provide an overview of the foundations and main approaches in the field of reinforcement learning (RL). RL is one of the three main areas of machine learning, along supervised learning and unsupervised learning. The specificity of RL, in its basic setting, is that a learning agent interacts with an environment in order to discover by trial-and-error a behavior that maximizes a certain reward. RL is a very broad framework for artificial intelligence in general; it is a challenging research field with numerous promising potential applications.

RL was formalized in its modern form in the 1980s, even though it is built on a much longer history, notably tracing its origins back to the study of animal learning in 19th-century psychology and to Richard Bellman's work on optimal control and dynamic programming in the 1950s (Sutton and Barto, 2018). The deep learning revolution of the early 2010s allowed RL to tackle problems that had seemed out of range before, and led to several high-profile successes in game playing that renewed interest in the field (Mnih et al., 2013; Silver et al., 2016; Vinyals et al., 2019). As a result, over time, a great variety of approaches have been developed, each with their own strengths and weaknesses.

Given the sequential decision-making problem at the heart of radar resource management, RL provides an appealing alternative to existing techniques. The versatility of RL and the wide range of methods developed for it mean it can be applied in different ways and to different aspects of the radar problems at hand. Our intent here is to present a large gamut of RL approaches, so that we can later discuss their applicability to radar use cases in the context of our work.

First we describe the RL framework, with its formalization, objective and learning

process (section 2.1). We then go over the three main families of algorithms designed to solve RL problems in their basic setting: value-based, policy-based, and model-based methods (sections 2.2 to 2.4). We finally discuss a number of specialized techniques, alternative models and extensions to RL; as RL is a growing field with many research directions, we only give a short overview of these, but focus especially on multi-objective RL, as part of our work is dedicated to methods falling in this domain.

## 2.1 The RL framework

This section describes the basics of RL. RL deals with stochastic sequential decision-making problems. The learner and decision-maker, called an agent, interacts with an environment, starting from an initial state; at each time step, when the agent selects an action to take, it transitions to a new state and receives a reward. The general objective is to maximize the cumulative sum of rewards (Sutton and Barto, 2018; Francois-Lavet et al., 2018).

### 2.1.1 Markov Decision Processes

The basic RL setting is usually formalized as a Markov decision process (MDP). An MDP is defined as:

- a finite state space $\mathcal{S}$;
- a finite action space $\mathcal{A}$;
- a transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, which gives the conditional probability $T(s' \mid s, a)$ of transition to the next state $s'$ given the previous state $s$ and selected action $a$;
- a reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ that gives the expected reward associated with a transition. We may overload this notation with $R(s, a) = \sum_s T(s' \mid s, a)\, R(s, a, s')$. Note that a reward can be negative and "punish" the agent.

When the agent interacts with the environment, it thus creates a sequence of states, actions, and rewards: starting from an initial state $s_0 \in \mathcal{S}$ sampled from the distribution of initial states $\mu_0$, it takes an action $a_0 \in \mathcal{A}$, then transitions to a new state $s_1$ according to $T$ and receives a reward $r_1$ according to $R$; this process repeats in every state, yielding a sequence $(s_0, a_0, r_1, s_1, a_1, r_2, s_3, ...)$.

A distinction is made between continuing and episodic settings. In the episodic setting, there is a terminal state $s_T$: when this terminal state is reached, the agent is brought back to the initial state and starts a new episode.

It is also useful to model problems in which the agent does not have a full view of its environment and may as a consequence be unable to distinguish between similar states. This property is called partial observability. Partially observable MDPs (POMDPs) include an additional function $\mathcal{O} : \mathcal{S} \times \Omega \to [0, 1]$ which is a conditional probability over the observation $\omega \in \Omega$ made by the agent in the current state. Most real-life applications of RL correspond to POMDPs, which makes partial observability a transversal issue in the field.

The goal of an RL agent is to find a policy that maximizes the sum of its rewards. A policy determines the action to take according to the current state. Policies can be stochastic: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and give a conditional probability over the next action, or they can be deterministic: $\pi : \mathcal{S} \to \mathcal{A}$.

In order to keep the sum of rewards finite, a discount factor $\gamma \in [0, 1)$ is usually introduced that makes short-term rewards more valuable than long-term ones. We can then define the state-value function $V^\pi$, which gives the value of a state $s$ under policy $\pi$, that is, the expected return starting in $s$ if the agent follows $\pi$:

$$V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid S_t = s \right] \tag{2.1}$$

with $R_t = R(S_{t-1}, A_{t-1}, S_t)$ a random variable of the reward at step $t$. State-value functions verify a recurrent relationship known as the Bellman equation:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s' \mid s, a)[R(s, a, s') + \gamma V^\pi(s')] \tag{2.2}$$

In control tasks, the goal is to find a policy that yields the optimal state-value function:

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s) \tag{2.3}$$

where $\Pi$ is the policy space. The policy associated with this optimal value function is the optimal policy $\pi^*$. This optimal state-value function verifies a specific recurrent relationship known as the Bellman optimality equation:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T(s' \mid s, a)[R(s, a, s') + \gamma V^*(s')] \tag{2.4}$$

Other value functions are used that are more practical for action selection, namely the action-value function $Q$ and the advantage function $A$, which follow similar recurrent relationships:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid S_t = s, A_t = a \right] \tag{2.5}$$

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{2.6}$$

### 2.1.2 Classes of algorithms

An RL agent includes one or more of the following components (Francois-Lavet et al., 2018):

- a representation of a value function that provides a prediction of how good each state or each state/action pair is (this estimate is denoted by $V$ to distinguish it from the true value function $V^\pi$);
- a direct representation of the policy;
- a model of the environment (the estimated transition function and the estimated reward function).

This allows to distinguish different approaches to RL problems.

Value-based methods (section 2.2) learn a value-function and derive a policy from it, whereas policy-based methods (section 2.3) directly learn their policy. Actor-critic methods are an intermediate approach where one component, the actor, directly improves the current policy while using value-function estimates provided by the second component, the critic.

Model-free methods are based solely on this kind of learning. Model-based methods (section 2.4), on the other hand, only learn a model of the environment, then use a planning algorithm to find an optimal policy. Again, intermediate algorithms may do both learning and planning. Note that representations of a value function or of the policy are also fundamentally models of the agent's knowledge about its environment; however in the context of RL, "model" usually designates a model of the environment.

During the learning process, once the agent has started gathering knowledge about its environment through experience, it faces a choice in every state: it can either select the action that maximizes the expected return according to current knowledge (the greedy action), or it can select another, a priori suboptimal action which could potentially lead to greater rewards. A trade-off has to be made between exploiting the best strategy found so far, and further exploring the environment to determine if a better strategy is to be found.

This dilemma between exploration and exploitation is especially important when the agent faces sparse, delayed rewards (such rewards give rise to the *credit assignment problem*). This problem is primarily studied in the context of $k$-armed bandits, which are MDPs with only one state: the objective is to determine as fast and reliably as possible which action yields the highest rewards. The simplest approach to balancing exploration and exploitation is to choose actions in an $\epsilon$-greedy way: at each step, a random action is chosen with probability $\epsilon < 1$, otherwise the agent takes the greedy action.

Additionally, RL methods are often classified as on-policy or off-policy. According to the definition given in Sutton and Barto (2018), on-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data. On-policy methods are unbiased and tend to be more stable. Off-policy methods, by leveraging data from different sources, facilitate exploration, are more data-efficient, and more general: they can learn from demonstrations, only they can be used in the offline setting. However, off-policy methods usually necessitate special means to correct the bias that comes from trying to learn a policy while exploring with another.

### 2.1.3 Approximation and generalization

The first RL algorithms addressed discrete problems and followed the tabular approach, storing the value of every state or action-state pair. This proved impractical for most applications for several reasons: the number of states increases exponentially in the number of state variables (which is known as the *curse of dimensionality*), the problem can be set with a continuous state space or action space, or it can be partially observable.

These shortcomings can be overcome with *function approximation*. In this approach, a parameterized function is used to approximate the components of the learning algo-

rithm (cf. section 2.1.2). Initially, the most common choice for approximation was to use linear functions. However, in recent years, major breakthroughs were achieved in high-dimensional problems using non-linear function approximation with deep neural networks (Goodfellow et al., 2016), giving birth to deep reinforcement learning. Common neural networks architectures like convolutional neural networks (CNN; LeCun and Bengio, 1995), recurrent neural networks (RNN; Hochreiter and Schmidhuber, 1997), and Transformers (Vaswani et al., 2017) have been applied successfully to RL problems (Mnih et al., 2013; Hausknecht and Stone, 2015; Zambaldi et al., 2018).

Despite empirical success, function approximation creates additional challenges. For instance, methods employing it in conjunction with bootstrapping and off-policy learning display increased instability and divergence. Sutton and Barto (2018) call this combination the *deadly triad*. Stabilizing off-policy learning remains a key issue in the field.

Function approximation represents a major step towards agents that can generalize the information obtained about a given state to similar states. One of the main objectives in RL research remains to design agents that generalize better. Generalization is defined as the ability of an agent to perform well in an environment about which it knows little, either because it has gathered limited information in this environment, or because this environment differs slightly (with regard to transition or reward) from the one used for learning (Francois-Lavet et al., 2018).

The ability to learn from as little data as possible is also termed *sample efficiency*. Sigaud and Stulp (2019) distinguish three aspects of sample efficiency:

- *data efficiency* consists in extracting as much information as possible from the available data,
- *sample choice* consists in collecting the most informative data from the environment, and
- *sample reuse* consists in improving the agent's knowledge several times with the same sample.

To date, state-of-the-art RL algorithms remain significantly sample-inefficient compared to human learners: deep RL especially requires vast amounts of computational power to solve complex tasks. Numerous research directions attempt to alleviate this core difficulty, ranging from multi-task RL (Espeholt et al., 2018; Hessel et al., 2018; Hausman et al., 2018) to meta-RL (Duan et al., 2016b; Oh et al., 2020) to data-augmentation (Kostrikov et al., 2021; Wang et al., 2022a) and pre-training techniques (Parisi et al.,

2022; Driess et al., 2022).

## 2.2 Value-based methods

We begin our overview of RL algorithms with the original approach to the field, value-based methods. Such methods try to estimate a value function from which they can derive a policy. Under the scheme of generalized policy iteration, the value function estimated from data generated by the current policy is used to improve this policy until both value function and policy converge towards optimality. Value-based methods can be tabular or approximate (Sutton and Barto, 2018). The methods presented in this section and the next are model-free methods set in the online setting.

### 2.2.1 Tabular methods

#### Monte Carlo methods

Monte Carlo methods are the most straightforward way to estimate value functions in the episodic setting. They are based on averaging complete returns in an incremental manner. For example, in the prediction task, a full episode is generated by following the policy $\pi$ to evaluate; then, the value of every visited state $s_t$ is updated:

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha[G_t - V(s_t)], \quad \text{with } \alpha = \frac{1}{\mathcal{T}(s_t)}, \\ G_t &= \sum_{k=0}^{T-t-1} \gamma^k r_{t+1+k} = r_{t+1} + \gamma G_{t+1}, \quad G_T = 0 \end{aligned} \tag{2.7}$$

where $\mathcal{T}(s_t)$ is the number of times $s_t$ was visited across all episodes and $G_t$ is the discounted return from time step $t$ to episode end $T$. State values are initialized arbitrarily (usually 0). The process is repeated until convergence.

Here $G_t$ is the *target* towards which we correct the estimated value function, and the parameter $\alpha$ is the *step size* which defines how much the estimate can change at each time step. The step size can also be a constant. This update is an instance of the general update rule used for estimates in RL:

$$\text{NewEstimator} \leftarrow \text{OldEstimator} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}] \times \text{Direction} \tag{2.8}$$

The difference [Target − OldEstimate] is called the error.

In the control task, a similar update is performed for the action-value function estimate of each visited state-action pair $Q_w(s,a)$ and a simple policy improvement updates the policy: $\pi(s) \leftarrow argmax_a\ Q_w(s,a)$ (the policy is also initialized arbitrarily). To ensure exploration, *soft policies* must be used, i.e. policies that maintain a non-zero probability for all actions. The most common choice is $\epsilon$-greedy policies.

The above applies to the on-policy setting. Monte Carlo methods can also be off-policy, generating information with a behavior policy $b$ to learn a target policy $\pi$. The most common technique for estimating expected values under one policy given samples from another is *importance sampling*, whereby returns are weighted by an importance-sampling ratio:

$$G_t^\rho = \rho_{t:T-1} \sum_{k=0}^{T-t-1} \gamma^k r_{t+1+k}, \text{ with } \rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{b(a_k|s_k)} \tag{2.9}$$

Such techniques, though, usually increase the variance of the target. There are variants that limit this increase in variance, such as *per-reward* importance sampling, which weights every reward with the adequate ratio, instead of weighting the whole sum of rewards:

$$G_t^\rho = \sum_{k=0}^{T-t-1} \rho_{t:t+k}\gamma^k r_{t+1+k} = \rho_t(r_{t+1} + \gamma G_{t+1}), \quad \rho_t = \rho_{t:t} \tag{2.10}$$

Unfortunately, pure Monte Carlo methods are restricted to the episodic setting and are computationally inefficient. Temporal-difference methods aim to correct these defects.

### Pure bootstrapping

Temporal-difference (TD) methods are a blend of Monte Carlo methods and dynamic programming: they learn from samples of experiences yet bootstrap on previous (action-)state values. Updates of the estimate are performed at each time step, in an on-line fashion, which makes TD methods usable in the continuing setting and more sample-efficient than Monte Carlo methods.

For example, the **TD(0)** algorithm solves the prediction problem by updating the state-value function in the following way at each step $(s_t, a_t, r_{t+1}, s_{t+1})$ generated by the

policy being evaluated:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.11}$$

The update error $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, named the *TD error* in this context, is often noted $\delta_t$.

The update for **Sarsa**, an on-policy control algorithm, is similar, using the sequence $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ generated at each step by the policy $\pi$ being learned; this must be a soft policy, greedy with regard to the action-state function $Q$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.12}$$

Another algorithm, **Q-learning**, adapts the approach to off-policy control. The behavior policy is still a soft policy derived from $Q$, but the target policy is deterministically greedy with regard to $Q$. This is reflected in the target of the update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \text{max}_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.13}$$

Finally, **Expected Sarsa** can be either on-policy or off-policy. The target of the update is expressed as an expectation over the next action according to target policy $\pi$. The soft behavior policy can be identical to $\pi$, or it can be derived from $Q$ in another way. Expected Sarsa includes Q-learning as a special case. The update is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.14}$$

**Multi-step bootstrapping**

A significant improvement to TD methods is brought by *n-step bootstrapping*. In this scheme, updates are delayed by a number $n$ of steps, and the target of the update includes the (discounted) rewards accumulated during this delay. Multi-step bootstrapping is an intermediate between one-step bootstrapping and Monte Carlo methods, and usually performs better than both by propagating rewards faster. As an example, the update for on-policy $n$-step Sarsa for time step $t$ becomes:

$$
\begin{aligned}
Q_{t+n}(s_t, a_t) &\leftarrow Q_{t+n-1}(s_t, a_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(s_t, a_t)], \\
G_{t:t+n} &= \sum_{k=t}^{t+n-1} \gamma^{k-t} r_{k+1} + \gamma^n Q_{t+n-1}(s_{t+n}, a_{t+n})
\end{aligned}
\tag{2.15}
$$

where $Q_{t+n}$ refers to the estimate of $Q$ at time step $t+n$.

Off-policy $n$-step bootstrapping can be implemented in two ways. The first one is to use importance sampling. The second one, put forth in the **Tree-Backup** algorithm, is to generalize Expected Sarsa's target to every step of the delayed update; the resulting target is defined recursively as:

$$
\begin{aligned}
G^{TB}_{t:t+n} &= r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q_t(s_{t+1}, a) + \gamma\, c_{t+1} \left(G_{t+1:t+n} - Q_t(s_{t+1}, a_{t+1})\right) \\
&= Q_{t-1}(s_t, a_t) + \sum_{k=t}^{t+n-1} \left( \prod_{i=t+1}^{k} \gamma c_i \right) \left( r_{k+1} + \gamma \sum_a \pi(a|s_{k+1}) Q_k(s_{k+1}, a) - Q_{k-1}(s_k, a_k) \right)
\end{aligned}
\tag{2.16}
$$

where $G_{t+1:t+1} = Q_t(s_{t+1}, a_{t+1})$, and $c_i = \pi(a_i|s_i)$ is the coefficient that corrects the off-policy discrepancy.

All these algorithms were designed for the tabular setting. This facilitates theoretical analysis, especially regarding proofs of convergence. However, approximate methods, despite being less theoretically grounded, allow application in more complex environments with a large or continuous state space.

### 2.2.2 Approximate methods

Usually, the goal set for RL algorithms with function approximation is to minimize a loss function, the mean squared value error:

$$
\mathcal{L}^V(w) = \mathbb{E}_{s \sim \mu_b}[(y - V_w(s))^2] \tag{2.17}
$$

where $\mu_b(s)$ is a probability distribution over states, proportional to the time spent in each state while learning with behavior policy $b$ (known as the *on-policy distribution*), $y$ is the target of the update, and $V_w(s)$ (or $V(s\,;\,w)$) is the value of state $s$ approximated with weight vector (or parameter vector) $w$. This loss function is in turn optimized by gradient descent, which is another instance of the general update rule (eq. (2.8)),

performed at every step $t$:

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \frac{1}{2}\alpha \nabla_{w_t} \mathcal{L}^V(w_t) \\ \text{i.e. } \quad w_{t+1} &\leftarrow w_t + \alpha \mathbb{E}_{s\sim\mu_b}[(y_t - V_{w_t}(s_t))\nabla_{w_t} V_{w_t}(s_t)] \end{aligned} \tag{2.18}$$

For practical purposes, stochastic gradient descent (SGD) is used, whereby the expectation is evaluated as a sum over a minibatch of transitions. A similar loss function $\mathcal{L}^Q(w)$ can be defined for the action-value function.

However, if the target bootstraps (which is desirable for computational efficiency), a bias is introduced, since the target then depends on the current value of $w$. Such *semi-gradient* methods have fewer convergence guarantees and are vulnerable to the deadly triad. Alternatives have been studied, such as *gradient-TD* methods, that use true gradients with other loss functions (Sutton and Barto, 2018); however, they do not extend to non-linear control settings.

### Deep Q-Networks

Mnih et al. (2013) created a breakthrough by achieving RL state-of-the-art performance on Atari games, a reference RL benchmark, with an agent learning only from raw pixels, without hand-crafted features or game-specific hyperparameter tuning. The proposed algorithm, **DQN**, is a variant of Q-learning with a number of improvements:

- to deal with partial observability, states are represented as a sequence of 4 game frames;
- the SGD minibatch is sampled uniformly from a memory $D$ of the last $N$ transitions, in a process known as *experience replay* (Lin, 1992). This technique avoids the correlations that would come from consecutive states (whereas supervised learning assumes independent samples) and therefore reduces variance; it improves sample efficiency, as each transition is used several times, which is especially important for rare ones; and it avoids risks of divergence or local minima by smoothing learning. Use of experience replay is made possible by the off-policy nature of Q-learning;
- in a subsequent article (Mnih et al., 2015), the concept of *target network* is introduced. Two weight vectors $w, w^-$ are maintained: at each step, $w^-$ is used for target evaluation while $w$ is updated and used for action selection; a parameter

update $w^- \leftarrow w$ takes place periodically. This further reduces divergence and oscillations of the $Q$-values.

- finally, rewards are clipped to $[-1, +1]$ so that their range is the same over all games.

The loss function for DQN summarizes these changes:

$$\mathcal{L}^Q(w) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \text{max}_{a'} Q(s', a'; w^-) - Q(s, a; w))^2] \tag{2.19}$$

A CNN serves as function approximator: its input is the current sequence $s$, its output is the vector $Q(s, \cdot)$ of action-state values for all actions. This architecture is computationally efficient as it allows to determine the greedy action in one forward pass of the network.

Despite the empirical stability of the algorithm, there are no theoretical guarantees regarding its convergence. Its success nevertheless prompted a number of proposed improvements.

### Improvements to DQN

A known limitation of Q-learning is that it suffers from positive maximization bias: using the maximum over estimates of the $Q$-values as an estimate of the maximum $Q$-value causes a positive ("optimistic") bias that can lead the $Q$-value to diverge, creating suboptimal policies. This phenomenon is experimentally demonstrated by van Hasselt et al. (2016) who propose **Double DQN** to counter it. This technique takes advantage of the presence of the two weight vectors $w, w^-$ to stabilize the target value: $w$ is always used to determine the maximizing action and $w^-$ to give the value of the associated state-action pair. This reduces overestimation and facilitates convergence:

$$\mathcal{L}^Q(w) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; w); w^-) - Q(s, a; w))^2] \tag{2.20}$$

Another issue with DQN is that it replays transitions at the same frequency that they are originally experienced although some transitions may provide more interesting information than others. Schaul et al. (2016) propose **prioritized experience replay** where transitions with high expected learning progress are replayed more frequently. A

level of priority $p_t$ is associated with each transition; it is proportional to the TD error $\delta_t$, which is a good proxy for how "surprising" a transition is. In order to avoid always replaying the same transitions and thus losing diversity, prioritization is made stochastic. The probability of replaying the transition from step $t$ is then defined as $\mathcal{P}(t) = \frac{p_t^\eta}{\sum_k p_k^\eta}$ where $\eta$ determines how much prioritization is used. As the replay distribution doesn't match the uniform distribution we are trying to evaluate, the algorithm risks getting stuck in local optima at the end of the optimization, so there is need for importance sampling: $\rho_t = \left(\frac{1}{N\mathcal{P}(t)}\right)^{\beta(t)}$ where $N$ is the replay memory size. $\rho_t\delta_t$ is then used instead of $\delta_t$ in the DQN loss function.

Wang et al. (2016) present **dueling Q-networks**, a network architecture usable with any deep RL value-based algorithm. The specificity of the dueling architecture is to estimate both the state value $V^\pi(s)$ and the vector of advantage value $A^\pi(s,a)$ for all actions, which are then combined to output the estimates of action-state values $Q^\pi(s,a)$. The weights used to compute $V$ and $A$ are shared until the output layer. Separating the representations of $V$ and $A$ has several benefits. By learning $V$, the network directly learns which states are valuable, and learning is accelerated since $V$ (and thus $Q$) is updated at each transition; by contrast, previous architectures learn each $Q(s,a)$ separately. Furthermore, $A$ is sufficient for action selection, and may be irrelevant in situations where the action does not affect the outcome, while $V$ is always important; the values of $A$ also tend to be significantly smaller than the associated $V$, so separating them stabilizes optimization. If the value of $Q$ was simply computed as $Q = V + A$, it might result in inaccuracies due to $A$ with regard to the greedy action $a^*$. In order to always have $Q(s,a^*) = V(s)$, the preferred upgrade is:

$$Q(s_t,a_t) \leftarrow V(s_t) + A(s_t,a_t) - \max_a A(s_t,a) \tag{2.21}$$

However, replacing the max operator with averaging makes the value of $Q(s,a)$ more stable and gives better experimental results.

Another trend of improvements over DQN consists in **distributional methods**, which essentially aim to learn the distribution of the random return (the value distribution) instead of the expectation of the return (the value). Learning the value distribution helps in a number of ways: it improves stability, especially in the presence of stochasticity, by using smoother target and loss function, and provides auxiliary predictions that ease

learning through inductive bias. Bellemare et al. (2017) proposed the first distributional method, where the value distribution is represented as a discrete distribution over a set of atoms whose probabilities are parameterized by Dirac distributions. Other methods in this direction seek to learn more expressive representations of this distribution, e.g. via quantile functions (Dabney et al., 2018; Yang et al., 2020; Nguyen-Tang et al., 2020).

The original DQN deals with partial observability by stacking the $N$ most recent observations. In some environments it may be necessary to give the agent more memory to understand more fragmented observations. Hausknecht and Stone (2015) propose replacing the frame-stacking technique of (Mnih et al., 2013) with a recurrent layer after the convolutional part of the neural network as a more general way to deal with partial observability. The recurrent layer is an LSTM (Hochreiter and Schmidhuber, 1997). Training the LSTM is done by storing sequences of transitions in the replay memory and initializing the recurrent state to zero before each sequence is used for learning. The resulting algorithm, **Deep Recurrent Q-Networks**, increases DQN's resilience to partial observability.

Standard DQN is limited in applicability with regard to action space: since it outputs $Q(s, a)$ for every action $a$, it cannot be used with continuous action spaces or too large discrete ones. In the continuous case, discretization could be an option, but when applied to $N$-dimensional actions, this creates a combinatorial explosion in the number of actions. Tavakoli et al. (2018) propose **Branching dueling Q-networks**, an extension of dueling Q-networks where the advantage value function is predicted independently for each action dimension. However this solution does not allow coordination between the different sub-actions. Metz et al. (2017) overcome this limitation by sequentially generating actions, one dimension at a time. This is done by modifying the original MDP: the original transitions are broken up into a trajectory of intermediate states that consist of the current actual state and the sub-actions chosen so far; one 1-dimensional sub-action is chosen at each step, and the intermediate states do not have rewards or discounting. The Q-value $Q_L$ of this modified MDP can be learned with standard DQN. However, the increase in the number of states makes training harder and more unstable. In order to mitigate this, the original Q-value $Q_U$ is also learned with TD(0), and a soft equality is enforced between $Q_U$ and $Q_L$ in the states of the modified MDP where no action has yet been chosen. The algorithm is robust for the choice of the number of bins for discretization and ordering of sub-actions. Generating sub-actions one dimension at

a time also allows better exploration of the action space, especially when faced with multimodal distributions. Sequential DQN thus manages to outperform DDPG (cf. section 2.3.3) on several continuous benchmarks.

## 2.3 Policy-based methods

As an alternative to value-based methods, policy-based methods directly learn a policy $\pi(a \mid s\,; \theta)$ (also noted $\pi_\theta(a \mid s)$) parameterized by a weight vector $\theta$. Therefore, they always use function approximation. Policy-based methods present several advantages. They can represent stochastic policies, as opposed to deterministic policies in value-based methods: for example, this means that policy-based methods can deal with continuous action spaces by expressing $\pi(a \mid s\,; \theta)$ as a probability density function. Additionally, deterministic policies may not be best suited for certain settings, such as POMDPs, adversarial settings, and multi-objective problems. Finally, they are usually more stable than value-based methods, but less sample-efficient.

### 2.3.1 Policy-gradient methods

Policy-gradient methods attempt to optimize the policy by gradient ascent:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_{\theta_t} J(\theta_t) \tag{2.22}$$

where $J(\theta_t)$ can be defined in two equivalent manners, with a specific definition of the on-policy distribution $\mu_\theta$ in each case (Sutton and Barto, 2018; Silver et al., 2014):

| Option $a$ | Option $b$ (continuing setting only) |
|---|---|
| $J^a(\theta) = \mathbb{E}_{s_0\sim\mu_0}[V^{\pi_\theta}(s_0)]$ | $J^b(\theta) = \mathbb{E}_{s\sim\mu_\theta^b,\, a\sim\pi_\theta}[R(s,a)]$ |
| $= \mathbb{E}_{s\sim\mu_\theta^a,\, a\sim\pi_\theta}[R(s,a)]$ | $= (1-\gamma)\mathbb{E}_{s\sim\mu_\theta^b}[V^{\pi_\theta}(s)]$ |
| $= \sum_{t=0}^{\infty} \mathbb{E}_{\mu_0,\pi_\theta}\left[\gamma^t R_{t+1}\right]$ | $= \lim_{h\to\infty} \frac{1}{h}\sum_{t=0}^{h} \mathbb{E}_{\mu_0,\pi_\theta}[R_{t+1}]$ |
| expected discounted return | average reward |
| $\mu_\theta^a(s) = \sum_{t=0}^{\infty}\gamma^t \Pr(S_t = s \mid \pi, \mu_0)$ | $\mu_\theta^b(s) = \lim_{t\to\infty}\Pr(S_t = s \mid \pi, \mu_0)$ |
| discounted state distribution | stationary state distribution |

(2.23)

The issue is that the gradient of the performance depends both on the policy and the state distribution induced by the policy, and this state distribution is itself dependant on the usually unknown transition function. Fortunately, the *policy gradient theorem* (Sutton and Barto, 2018) conveniently links the gradient of the performance to the gradient of the policy only:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_s \mu_\theta(s) \sum_a Q_B(s,a) \nabla_\theta \pi(a \mid s;\theta) \\ &= \mathbb{E}_{s\sim\mu_\theta,\, a\sim\pi_\theta}\left[Q_B(s,a)\frac{\nabla_\theta \pi(a \mid s;\theta)}{\pi(a \mid s;\theta)}\right]\end{aligned} \tag{2.24}$$

with $Q_B(s,a) = Q(s,a) - B(s)$ where $B(s)$ is an arbitrary baseline, used to reduce variance. $B(s)$ may be a constant (like 0) but a better choice is to use the estimate of the state-value function $V(s)$, in which case we obtain the advantage function.

Degris et al. (2012) extend this result to the off-policy setting, by proving that improvement to a local optimum of $J(\theta)$ is guaranteed by the following approximation, which differs only by the presence of an importance sampling ratio that compensates the use of a behavior policy $b$:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s_t\sim\mu_b,\, a_t\sim b}\left[\rho_t\, Q_B(s_t,a_t)\frac{\nabla_\theta \pi(a_t \mid s_t;\theta)}{\pi(a_t \mid s_t;\theta)}\right] \tag{2.25}$$

The ratio $\frac{\nabla_\theta \pi(a \mid s;\theta)}{\pi(a \mid s;\theta)}$ is called the *likelihood ratio* and is usually computed as $\nabla_\theta \ln \pi(a \mid s;\theta)$. The right-hand side of eq. (2.24) can then be interpreted as the gradient

of the following objective function:

$$\mathcal{L}^{PG}_{Q_B}(\theta) = \mathbb{E}_{s\sim\mu_\theta,\, a\sim\pi_\theta}[Q_B(s,a)\ln\pi(a\,|\,s;\theta)] \tag{2.26}$$

An entropy regularizer $H_\pi(s) = -\sum_a \pi(a|s)\ln\pi(a|s)$ is often added with a coefficient $c$ to force the policy to stay stochastic and maintain exploration (O'Donoghue et al., 2017; Francois-Lavet et al., 2018):

$$\mathcal{L}^{PG}_{Q_B}(\theta) = \mathbb{E}_{s\sim\mu_\theta,\, a\sim\pi_\theta}[Q_B(s,a)\ln\pi(a\,|\,s;\theta)] + c\,\mathbb{E}_{s\sim\mu_\theta} H_{\pi_\theta}(s) \tag{2.27}$$

From this theorem, a simple policy gradient algorithm, **REINFORCE** (Williams, 1992) can be derived that updates $\theta$ using episodic returns, in a Monte Carlo fashion; that is, at the end of every episode, the update for each step $t$ of the episode is:

$$\theta \leftarrow \theta + \alpha\gamma^t(G_t - B(s_t))\nabla_\theta \ln\pi(a_t\,|\,s_t;\theta) \tag{2.28}$$

REINFORCE is thus an on-policy approximate method with strong convergence properties. Estimates of the expected return obtained by Monte Carlo sampling are unbiased, but also display high variance and sample-inefficiency.

### 2.3.2 Actor-critic methods

Pure policy-gradient methods need to form an estimate of $Q_B$ at each time step, which is achieved in the algorithms presented so far by Monte Carlo sampling. A more sample-efficient approach is to learn a value function parameterized by a vector $w$, called a critic, in parallel. The critic is used to estimate $Q^B$, at the cost of some bias, while the actor outputs the policy distribution. This is the concept of *actor-critic* algorithms (Konda and Tsitsiklis, 2000).

The first prominent actor-critic algorithm in deep RL was **Advantage Actor-Critic** (A2C) (Mnih et al., 2016; Wu et al., 2017). This algorithm also innovated in a number of other ways: its neural network architecture contained layers that were shared between the policy and value function; it used multiple actor-learners that ran in parallel on a multi-threaded CPU, generating their own experience and averaging their gradients to update common synchronized network parameters; and it used fixed-length segments of experience to compute estimators of the multi-step returns and advantage function.

A2C did not use a replay buffer, which came at a cost in sample-efficiency. Later actor-critic methods improved on this aspect by incorporating experience replay and importance sampling to enable off-policy learning, like ACER (Wang et al., 2017), or by using lower-variance estimators of the policy gradient and learning distributional Q-values, like Reactor (Gruslys et al., 2017).

### 2.3.3 Deterministic-policy gradient

Silver et al. (2014) adapt policy-gradient methods to the use of a deterministic target policy $\pi(s;\theta)$ in the presence of a continuous action space. This approach is motivated by the fact that the gradient of a stochastic policy integrates over both the state space and the action space, while the gradient of a deterministic policy integrates only over the state space, which means fewer samples to estimate it and more computational efficiency, especially with large action spaces. Furthermore, stochastic-policy gradients display high variance when the stochastic policy converges towards a deterministic policy; using a deterministic policy from the start prevents this and makes learning more stable.

An analog of the policy gradient theorem is provided for the **deterministic-policy gradient** (DPG):

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\mu_b}[\nabla_\theta \pi(s;\theta) \nabla_a Q^\pi(s,a)|_{a=\pi(s;\theta)}] \tag{2.29}$$

where $\mu_b$ is the state distribution under a policy $b$. This relation applies to on-policy and off-policy settings. In the on-policy case ($b = \pi$), the relation is an equality. The off-policy case is of greater practical interest since $b$ can then be a stochastic policy which encourages exploration; $b$ is often defined by adding noise to the deterministic target policy so that $b(s) \sim \mathcal{N}(\pi(s;\theta), \sigma^2)$. The fact that the target policy is deterministic means no importance sampling is needed in eq. (2.29). The authors further prove that the deterministic policy gradient is the limiting case of stochastic policy gradient, when the variance of the stochastic policy tends to zero.

The above result can be applied to learn the actor of an actor-critic architecture where the critic provides an estimate of $Q^\pi(s,a)$. In particular, Lillicrap et al. (2016) propose to use DQN's approach (cf. section 2.2.2) to learn the critic, as a way to apply DQN to continuous action spaces, yielding **deep deterministic-policy gradient** (DDPG). Fujimoto et al. (2018) adapt double learning for actor-critic algorithms and apply it to

DDPG to obtain the **Twin Delayed Deep Deterministic** policy-gradient algorithm (TD3).

### 2.3.4 Trust-region methods

One issue with policy-gradient methods is that they follow the steepest-ascent direction $\nabla_\theta J(\theta)$ in the parameter space. This entails the risk of destructive policy updates, since small changes in parameter space may result in radically different policies in practice. One possible improvement to this problem is to try to follow the steepest ascent in the policy space. To do so, natural policy gradient (Kakade, 2001; Amari, 1998) follows the corrected direction $F_\theta^{-1}\nabla_\theta J(\theta)$ where $F_\theta$ is the Fisher information matrix. Following the steepest ascent in policy space improves the optimization's stability while keeping vanilla policy gradient's convergence guarantees; the update in fact moves the policy in the direction of a greedy policy improvement. However, computing, inverting and storing the Fisher matrix is very expensive, which makes natural policy gradient unsuitable for large parameter vectors, such as those of deep neural networks.

Trust-Region Policy Optimization (Schulman et al., 2015) makes this line of thought more practical by subjecting the gradient of the policy to a constraint on the KL-divergence between the old policy and the updated policy. While this method exhibits strong convergence guarantees and is applicable to neural networks, it requires the use of conjugate gradient descent and line search to enforce the KL constraint. **Proximal Policy Optimization** (Schulman et al., 2017b) improves on TRPO by dropping the KL divergence constraint and clipping the original objective function:

$$\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_{s\sim\mu_{\bar{\theta}},\, a\sim\pi_{\bar{\theta}}}\left[\min\left(\hat{A}^{GAE}_{\bar{\theta}}(s,a)\frac{\pi(a|s;\theta)}{\pi(a|s;\bar{\theta})},\ \hat{A}^{GAE}_{\bar{\theta}}(s,a)\left[\frac{\pi(a|s;\theta)}{\pi(a|s;\bar{\theta})}\right]_{1-\epsilon}^{1+\epsilon}\right)\right] \tag{2.30}$$

where $[x]_l^u$ is the value of $x$ clipped in the interval $[l,u]$ and $\epsilon$ is a hyperparameter. The clipping of the policy ratio ensures that the next policy is not too different from the current one, which allows to drop the KL constraint. The min operator ensures that the monotonic improvement guarantees of TRPO are maintained. This new formulation is simpler and more generally applicable (e.g. it allows parameter sharing, unlike TRPO). It is also more sample-efficient thanks to the use of Generalized Advantage Estimation (Schulman et al., 2016) which estimates $A$ with low variance. It adapts a common

estimator of $A$, based on $n$-step returns:

$$\hat{A}^{(n)}(s_t, a_t) = \sum_{l=0}^{n-1} \gamma^l r_{t+l} + \gamma^n V(s_{t+n}) - V(s_t) = \sum_{l=0}^{n-1} \gamma^l \delta_{t+l} \tag{2.31}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the TD error. This estimator is unbiased if $V$ is the true state-value function. When using an approximation of $V$, the bias decreases as $n$ tends to infinity. This motivates the definition of the generalized advantage estimator as an exponentially weighted sum of estimators $\hat{A}_t^{(n)}$:

$$\hat{A}^{GAE}(s_t, a_t) = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{A}_t^{(n)}(s_t, a_t) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{2.32}$$

The value of $\lambda \in [0, 1]$ determines a trade-off between variance (higher as $\lambda$ increases) and bias (higher as $\lambda$ decreases).

### 2.3.5 Maximum-entropy methods

Policy-gradient algorithms often include an entropy regularizer that helps to keep the policy stochastic so as to maintain exploration. In this case, the objective can be defined as:

$$\pi^* = \text{argmax}_\pi \sum_t \mathbb{E}_{s_0 \sim \mu_0, a_{0:t} \sim \pi}[R(s_t, a_t)] + c\, \mathbb{E}_{s \sim \mu_\pi} H_\pi(s) \tag{2.33}$$

Another option is to maximize the entropy of the entire trajectory distribution under policy $\pi$. This is the objective of maximum entropy RL:

$$\pi_H^* = \text{argmax}_\pi \sum_t \mathbb{E}_{s_0 \sim \mu_0, a_{0:t} \sim \pi}[R(s_t, a_t) + c\, H_\pi(s_t)] \tag{2.34}$$

This can be interpreted as trying to learn not a deterministic optimal behavior, but a stochastic policy that encompasses all quasi-optimal behaviors. This approach can be justified by framing reinforcement learning as a probabilistic inference problem (Levine, 2018). A discount parameter $\gamma$ is usually added for variance reduction.

Maximum entropy methods have a number of benefits. They help exploration, especially in the case of multi-modal objectives (section 2.5.1). They are also robust to model and estimation errors and in adversarial settings. Finally, they are suited for multi-task and transfer learning; for example, they can be used to pre-train general-purpose

stochastic policies that can then be fine-tuned for specific tasks (Haarnoja et al., 2017).

Besides earlier methods (Haarnoja et al., 2017; Nachum et al., 2017, 2018b), the main maximum-entropy algorithm is **Soft actor-critic** (Haarnoja et al., 2018). Similarly to policy iteration, SAC alternates between policy evaluation, which is here given by:

$$\begin{aligned} Q_H(s_t, a_t) &= r_{t+1} + \gamma \mathbb{E}_{s_{t+1} \sim T(s_t, a_t)}[V_H(s_{t+1})] \\ V_H(s_t) &= \mathbb{E}_{a_t \sim \pi}[Q_H(s_t, a_t) - \ln \pi(a_t | s_t)] \end{aligned} \tag{2.35}$$

and policy improvement, given for all states $s$ by:

$$\pi_{\text{new}} = \text{argmin}_{\pi'} D_{KL} \left( \pi'(\cdot|s) \,||\, \frac{\exp\left(Q_H^{\pi_{\text{old}}}(s, \cdot)\right)}{Z^{\pi_{\text{old}}}(s)} \right) \tag{2.36}$$

where $Z^{\pi_{\text{old}}}$ is a partition function that can be ignored in optimization. This process is approximated with SGD applied to three neural networks $V_\psi$ (not indispensable but improves stability), $Q_w$, and $f_\theta(\epsilon_t;\, s_t) \to a_t$ (that induces the actual policy $\pi_\theta$).

Interestingly, Schulman et al. (2017a) demonstrate that the gradient for a "soft" version of Q-learning is actually the same as the policy gradient for the maximum entropy objective; that is, their expectation is the same even though their variance can be different. In any case, the learning curves of these two algorithms are empirically similar given the right hyperparameters.

## 2.4 Planning and model-based methods

Being able to predict the dynamics of the environment opens a host of opportunities for the learning agent. The ability to predict the next state and reward are a given when a perfect model of the environment can be provided, for example in board games like chess; this case has been extensively studied in the contexts of planning and optimal control. However, in most RL applications, a model of the environment is not available beforehand and instead has to be learned by supervised learning before planning can take place.

The main promise of model-based approaches is to reduce sample complexity, by favoring model sampling over environment sampling, to the expense of increased computational requirements. Another advantage would be improved generalization, as planning methods can reason about previously unseen situations, unlike reactive model-free poli-

cies. Planning could also potentially help to locally smooth out function approximation errors associated with value- and policy-based methods, which would result in better performance, provided the model's predictions are accurate enough.

### 2.4.1 Model learning

Model-based methods make use of a model $M$ of the transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ (also called the dynamics of the models). In most cases this model is not initially available and has to be learned. Learning accurate models for complex environments can prove a significant challenge. It is vital, however, as planning over poor models will degrade performance compared to model-free methods, a problem known as model bias (Deisenroth and Rasmussen, 2011). To limit inaccuracy in predictions, model-based methods either try to stay close to data, as in guided policy search (Levine et al., 2016), or directly estimate epistemic uncertainty (i.e. the uncertainty on the model's accuracy, as opposed to aleatoric uncertainty, which is synonymous with stochasticity in the environment). Uncertainty can be evaluated with bootstrapped ensembles or Bayesian neural networks (Chua et al., 2018; Depeweg et al., 2017), which both amount to learning separate models of the same environment: the degree of disagreement between models serves as a measure of model uncertainty.

Learning the dynamics of the environment fundamentally is a supervised learning problem. For continuous state spaces, the standard practice in model-based RL is to train the model to predict the change in state $\Delta s = s_{t+1} - s_t$ rather than the next state $s_{t+1}$, similarly to the concept of residual nets (He et al., 2015): this allows more accurate predictions in the case of small changes (Deisenroth and Rasmussen, 2011). In order to scale reasonably, neural networks are the most common approach, even though iteratively refitted linear models have also given excellent results (Levine et al., 2016).

The most common and obvious issue in model learning is having to deal with a stochastic environment. A first approach is descriptive models, that capture the entire probability distribution of all transitions, for example with Gaussian processes (Deisenroth and Rasmussen, 2011) or probabilistic neural networks (Chua et al., 2018; Hafner et al., 2019). A second approach is generative models, that simply produce samples of the next state (and reward), for example using Bayesian neural networks (Depeweg et al., 2017; Gal et al., 2016). In this case, a common trick is to reparametrize the model as a

function of some stochastic noise $\xi$: $s_{t+1} = f(s_t, a_t; \xi)$ (Heess et al., 2015).

In the case of a non-stationary environment, the standard approach is partial models: the agent learns a different model for every environment mode and switches to the relevant model when the mode changes (Nagabandi et al., 2019). This is actually a matter of multi-task learning and meta-learning.

When multi-step prediction is required, the most natural option is to chain one-step model predictions. However this tends to compound one-step errors. Alternatively, one can learn $n$-step predictions for all steps $1 \leqslant n < N$ to increase accuracy (Hafner et al., 2019). Mishra et al. (2017) take the idea further and propose to learn temporal-segment models, which predict the distribution over future state trajectories.

As for partial observability, the same solutions as for model-free methods apply; that is, the most common are RNNs (Ha and Schmidhuber, 2018) and frame stacking (Depeweg et al., 2017), although memory-based approaches have also been proposed (Gemici et al., 2017).

Finally, the difficulty of planning increases in high-dimensional state spaces; many papers attempt to perform dimensionality reduction in order to plan at the level of the environment's latent dynamics, especially by extending VAEs to the temporal setting (Watter et al., 2015; Krishnan et al., 2015; Karl et al., 2017).

### 2.4.2 Planning with a learned model

The problem of planning actions in an environment whose model is known has been studied extensively, for example with dynamic programming. Planning has been given many definitions. In its strictest sense, it refers to algorithms that solve decision-making problems without making use of learning. A simple approach to complex decision-making problems, then, is to learn a model of the environment and apply planning methods on it. Note that the model may be learned offline (before planning) or online (at the same time as planning). Discrete actions are suited for look-ahead search methods like MCTS (Browne et al., 2012). Continuous actions are tackled with trajectory optimization methods that range from direct optimal control, where trajectories are optimized with metaheuristics like random search or CEM (as in Nagabandi et al., 2018; Chua et al., 2018), to differential planning methods like iLQR, which approximates non-linear systems with linear dynamics and quadratic rewards (Todorov and Li, 2005).

These methods can be used for open-loop planning, where a plan is determined at the beginning of an episode and then executed entirely. However, it is usually preferable to use closed-loop planning, where feedback is allowed after the plan has started being executed, especially when the planning is applied to a learned model of a complex, stochastic environment. Most approaches use model predictive control, meaning that planning is repeated at each step, and only the first step of each plan is executed.

### 2.4.3 Model-based RL

In a broader sense, planning refers to any process that takes a model as input and outputs (or improves) a policy (Sutton and Barto, 2018). Under this definition, planning may be integrated with model-free RL in what is known as model-based RL.

Given a policy, the most direct way of using a model is to generate fictional transitions or trajectories that can then be exploited as additional training data for model-free optimization. This is sometimes termed *imagination*: the model-free algorithm makes use of both real and imagined transitions for training. This approach was pioneered with **Dyna-Q** (Sutton and Barto, 2018), which applied it to the classic Q-learning, with a tabular model. Recent approaches often use model ensembles to maintain model uncertainty (Kurutach et al., 2018; Clavera et al., 2018). Models can also be used to estimate $n$-step returns (Feinberg et al., 2018; Buckman et al., 2019).

Model-free methods may also be used in combination with the pure planning algorithms. This can go two ways: we can plan with learned priors, for example by using value functions and policies learned with model-free RL as the inputs of a planning algorithm, or we can learn with planned targets, by using planning algorithms to determine targets that drive the updates of the policy or value function of a model-free algorithm. Both approaches can be combined to form a procedure that alternates between planning and learning, where both provide information to improve the other (Silver et al., 2017a, 2018; Levine and Koltun, 2013; Levine and Abbeel, 2014; Levine et al., 2016).

### 2.4.4 Implicit planning

All methods presented so far in this section rest on two fundamental ideas:

- learning a model of the environment's dynamics, i.e. a module that predicts the next *environment state* (or next *observation* in the case of a POMDP), via *super-*

*vised learning*;

- exploiting this model to obtain or improve a policy by applying a *predefined* planning algorithm: pure planning, imagination, value gradients, etc.

A number of recently proposed algorithms diverge from this process, by training their model differently or by learning how to plan on their model. These approaches are sometimes grouped as implicit planning.

Differentiable planning algorithms use a classic supervised model and focus on learning how to plan. Learning how to plan allows to adapt the planning algorithm to the problem at hand. This adaptable planning algorithm is implemented as a neural network, so that the planning module is differentiable and can be trained by backpropagation. This approach can be considered a form of meta-learning. One approach in learning how to plan is to learn how to exploit imagination rollouts (Pascanu et al., 2017; Racanière et al., 2017). However, imagination-based agents do not make use of the very efficient search schemes of classic planning methods like MCTS, and may be unlikely to rediscover them. This is why another line of approach is to use these planning methods, and attempt to learn subcomponents of it. For instance, Guez et al. (2018) approximate MCTS with learnable operations for evaluation, selection, backup, and final decision.

Over the years, deep learning techniques have been developed that allow almost perfect pixel-level predictions for dynamics models (Oh et al., 2015). However, these prediction models often learn a lot of useless information about the environment state/observation (like background noise): this information is useless in the sense that it doesn't help to predict the value of the next states. A possible solution, then, is to train a value function with an embedded transition model to predict the value of future states. The value function is trained with a model-free algorithm, so that the embedded model is trained indirectly via backpropagation, not with supervised learning. This makes it an abstract model, since the states it predicts do not have to correspond to real environment states: only their values must match true state values. This is the origin of the term *value-equivalent model*. These ideas were pioneered in the **Predictron** (Silver et al., 2017b), which was used for value prediction in Markov reward processes. The exact output of the Predictron is a compound of imagined $n$-step returns, which is trained against real returns. The idea has since been applied to control problems (Oh et al., 2017; Farquhar et al., 2018; Tamar et al., 2016) and proved its usefulness when applied to AlphaZero to obtain **MuZero** (Schrittwieser et al., 2020), that was able to match Al-

phaZero's performance on board games without having access to a known environment model and to beat the state-of-the-art performance on Atari. Despite interesting results, it remains unclear whether value-equivalent models are able to learn the most complex environment dynamics accurately without supervised learning.

The most extreme form of implicit planning, proposed by Guez et al. (2019), does not rely on any form of model of the environment, and uses black-box recurrent networks of stacked ConvLSTMs (Shi et al., 2015) instead. Surprisingly, when this architecture is trained with the IMPALA actor-critic method, the resulting algorithm still exhibits typical planning-like characteristics (generalization in combinatorial spaces, sample-efficiency, improved performance form additional thinking time) and surpasses other implicit planning algorithms on a benchmark of planning tasks (Sokoban, Pacman...). Based on these results, the authors argue in favor of a behaviorist definition of planning.

## 2.5 Extensions

### 2.5.1 Exploration

As mentioned in section 2.1, RL agents must explore their environment during learning to try and discover greater sources of reward. However most classic algorithms rely on random exploration, either via an $\epsilon$-greedy strategy like DQN, or by applying a stochastic policy with Gaussian noise and entropy regularization like most policy-based methods. Other random exploration methods include Boltzmann exploration (Cesa-Bianchi et al., 2017), which uses the exponential of the standard Q-function as the probability of an action (and is closely related to maximum entropy methods), and noisy networks (Fortunato et al., 2017), which add parametric noise to the network weights.

These techniques hit their limits when applied to environments with sparse and delayed rewards, which instead require a form of *directed* exploration. This can take the form of *temporally-extended* exploration, for example via Thompson sampling (Osband et al., 2016) or self-imitation (Ecoffet et al., 2019, 2020; Oh et al., 2018). A lot of research also focuses on intrinsic motivation (Oudeyer and Kaplan, 2007, 2009), that is, on mimicking curiosity and "exploration for the sake of exploration". This notably covers novelty search, which drives the agent towards underexplored states (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017), and prediction-based exploration,

where the agent focuses on exploring transitions whose dynamics it struggles to predict (Achiam and Sastry, 2016; Pathak et al., 2017; Burda et al., 2018).

An alternative to advanced exploration methods is to guide the agent with human input. This can take the form of reward shaping (Ng et al., 1999), that is, crafting a reward function that doesn't only reflect the end behavior we want the agent to adapt, but also provides intermediate rewards that steer it in this direction. When the reward function is difficult to specify, one can resort to imitation learning, the problem of learning to perform a task from expert demonstrations. One approach is to recover an expert's cost function with inverse reinforcement learning (Ng and Russell, 2000; Abbeel and Ng, 2004). A more direct way to use demonstrations is to perform behavioral cloning (Hester et al., 2017; Ho and Ermon, 2016) which learns a policy as a supervised learning problem over state-action pairs from expert trajectories. Behavioral cloning can especially used to initialize policies (Silver et al., 2016).

### 2.5.2 Distributed and evolutionary methods

One major impediment to current RL methods is that their training time can be very long: for example, DQN and its variants are commonly trained for more than a week on the standard Atari benchmark. Distributed methods allow to scale up computational resources and speed up learning. Considerable work has also been done to make use of distributed computation in RL (Babaeizadeh et al., 2017; Clemente et al., 2017; Stooke and Abbeel, 2018; Horgan et al., 2018; Barth-Maron et al., 2018; Espeholt et al., 2018; Kapturowski et al., 2018): research focuses on maximizing GPU occupation time for gradient computation, maximizing multi-threaded CPU usage for experience generation, and synchronizing actor and learner processes.

This is especially the case for evolutionary methods, an interesting alternative to solve RL problems. The idea is to apply these techniques to optimize the policy parameters (e.g. neural network weights), using episodic return as the fitness function; this is known as direct policy search (Salimans et al., 2017). Several classic evolutionary methods have been applied to RL, such as the **Cross-entropy method** (CEM; de Boer et al., 2005; Szita and Lörincz, 2006), **CMA-ES** (Hansen, 2016; Duan et al., 2016a), and genetic algorithms (Petroski Such et al., 2018). Alternatives include neuro-evolution (Stanley et al., 2009; Kelly and Heywood, 2018) and combinations of gradient-based

and evolutionary methods (Khadka and Tumer, 2018; Pourchot and Sigaud, 2019). Of note is the ability of evolutionary methods to perform exploration naturally through population-based methods (Lehman and Stanley, 2011; Conti et al., 2018).

### 2.5.3 Learning multiple tasks

In its basic setting, the role of an RL agent is to learn to perform a single task. This is unlike natural agents, who possess a repertoire of skills which they can put to use to accomplish a variety of tasks. These temporally-extended behaviors have also been described in terms of options (Sutton et al., 1999) and goals (Sutton et al., 2011; Schaul et al., 2015). Mastering several abilities has many advantages.

First, it allows hierarchization, i.e. the composition of low-level skills to perform a high-level task. Hierarchical RL is long studied approach (Dayan and Hinton, 1992; Parr and Russell, 1998; Dietterich, 2000). The difficulty of building a hierarchical agent revolves around the coordination of its high-level and low-level controllers, and around autonomously learning the subgoals/skills required to solve the top-level task (Vezhnevets et al., 2017; Bacon et al., 2017; Nachum et al., 2018a).

Second, learning different skills "for the sake of it" can enable generalization to similar skills, allowing the agent to rapidly adapt to the most relevant aspects of the actual task. This is the framework of unsupervised (or self-supervised) RL (Jaderberg et al., 2017; Shelhamer et al., 2017; Andrychowicz et al., 2017), which places emphasis on autonomous option discovery (Fox et al., 2017; Achiam et al., 2018; Eysenbach et al., 2018).

Finally, when skills are learned in a sequence, with their complexity and difficulty increasing at each step, this forms a curriculum which may ease the agent's learning. The curriculum can consist in giving increasingly difficult goals to the agent, or exposing it to environments of increasing complexity. Of interest is the development of methods that perform automatic curriculum generation, for example via intrinsic motivation (Colas et al., 2019; Sukhbaatar et al., 2018; Florensa et al., 2018; Jabri et al., 2019).

Unsupervised learning and curriculum learning both rely on a form of transfer learning. Transfer learning can also occur when the same agent is tasked with learning to solve several unrelated environments, for example when trying to play different Atari games with one agent (Espeholt et al., 2018). This multi-task RL can be carried out by

training simultaneously on all environments (Kelly and Heywood, 2018; Teh et al., 2017; Hessel et al., 2018), or learning to solve each environment sequentially (Hausman et al., 2018; Kansky et al., 2017). Variants include online and lifelong learning. Both deal with a non-stationary task distribution over time: online learning focuses on fast adaptation, lifelong learning focuses on avoiding catastrophic forgetting (Nagabandi et al., 2019).

Meta-RL pushes the idea of transfer learning even further by attempting to train a model to be able to quickly learn a new task from a small amount of new data on a large number of different tasks. This meta-learning ability can involve automating the discovery of model's update rules or to train the model's initial parameters in order to maximize its few-shot performance on new tasks (Duan et al., 2016b; Finn et al., 2017; Oh et al., 2020).

Recently, there have been proposals to build upon pre-trained large language models to train agents that can display combinatorial generalization across different tasks (Li et al., 2022; Wang et al., 2023).

### 2.5.4 Multi-objective RL

Most research in RL implicitly follows Sutton's reward hypothesis, which states that "all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received *scalar* reward" (Sutton). This can be understood as a statement on the best way for a user to communicate their intent to the agent. This in turn implies two things: firstly, that the most practical method for a human to express the desired behavior to the agent is to describe it in terms of a single scalar reward function; secondly, that a scalar reward always constitutes a sufficient signal based on which the agent can efficiently optimize its behavior.

Both of these implications can be questioned. The first one, in particular, will probably run counter to the experience of most RL practitioners, who discover early on that it is difficult to specify a reward function that matches the desired behavior: RL algorithms have a tendency to exploit their reward, that is, to maximize it in unintended ways. In many cases, it is more natural to describe a desirable behavior in terms of multiple objectives which may conflict with each other: in most RL applications, the reward ends up begin defined as a scalarization of multiple subrewards corresponding to different criteria. This scalarization, if interpreted as a utility function, must reflect the

user's preferences over the criteria; yet in some cases these preferences may be unknown at training time, or they may vary depending on the situation. The second implication is contradicted by the fact that even when a task can be easily described by a scalar reward function, this reward might not prove simple to optimize either because it is sparse or delayed. This is why practitioners often resort to reward shaping, auxiliary skills learning, etc. On the other hand, if many criteria/subrewards are involved, subsuming them into a scalar signal may constitute an information bottleneck: the relationship between an action and the rewards it leads to is less obvious than if the agent received a signal that kept all subrewards intact. In other words, it can make credit assignment more difficult.

These limitations motivate the study of multi-objective RL (MORL) (Roijers et al., 2013). In MORL, we consider environments with vector-valued reward functions $\mathbf{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$. Each component of this vector of rewards is associated to one of the agent's objectives. As a consequence, we can define the value function in vector form:

$$\mathbf{V}^\pi(s) = \mathop{\mathbb{E}}_{\pi,T}[\sum_{k=0}^{\infty} \gamma^k \mathbf{R}(S_{t+k}, A_{t+k}) \,|\, S_t = s] \tag{2.37}$$

Since $\mathbf{V}^\pi$ is a vector, in contrast to the single-objective case, there doesn't necessarily exist a unique optimal value, i.e. a value $\mathbf{V}$ that dominates all other values $\mathbf{V}'$ such that $V_i(s) > V_i'(s)\ \forall s, i$. This can be replaced with the notion of Pareto-optimal values, that is, values that are not dominated by any other. These form the Pareto front, which represents the set of potentially optimal solutions depending on the user's preferences.

The most common way to represent these preferences is in the form of a scalarization function, which gives a complete ordering over values. Given vector-valued rewards, there are actually three ways we can adapt the standard RL objective to the use of a scalarization. First we can scalarize the rewards directly:

$$\max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\pi,T}[\sum_{t=0}^{\infty} \gamma^t u(\mathbf{R}(S_t, A_t))] \tag{2.38}$$

This reduces to a scalar reward and is not considered part of MORL. MORL algorithms instead try to solve for either the scalarization of expected returns (SER):

$$\max_{\pi \in \Pi} u(\mathop{\mathbb{E}}_{\pi,T}[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(S_t, A_t)]) \tag{2.39}$$

or the expected scalarization of returns (ESR):

$$\max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\pi,T}[u(\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(S_t, A_t))] \tag{2.40}$$

SER scalarizes the expectation over multiple returns, and is therefore more adequate when the associated policy is executed many times; conversely, in ESR, returns do not accumulate across episodes, so that this formulation is a better fit when the policy is run once per user, for example in medical applications where the treatment of one patient corresponds to a single episode. In practice, the most commonly adopted criterion is SER.

Due to its simplicity, the most usual scalarization is the weighted average $u(\mathbf{G}) = \sum_i w_i G_i$, where each objective is attributed a reward weight $w_i$ with $\sum_i w_i = 1$. However, linear scalarization has limited expressive power as it cannot express preferences for Pareto-optimal solutions outside of the convex hull. As a consequence, non-linear scalarizations, such as the thresholded lexicographic order, have also been studied, but they present more challenges as they make the rewards non-additive (Dornheim, 2022; Vamplew et al., 2022). Note that when using linear scalarization, the three formulations outlined above are all equivalent. However, in the multi-objective setting, the network can approximate the vector-valued action-state function

$$\mathbf{Q}^{\pi}(s, a, \mathbf{w}) = \mathop{\mathbb{E}}_{\pi_{\mathbf{w}},T}[\sum_{k=0}^{\infty} \gamma^k \mathbf{R}(S_{t+k}, A_{t+k}) \mid S_t = s, A_t = a] \tag{2.41}$$

Scalarizing this Q-value vector instead of directly outputting a scalar value typically yields more accurate estimations of the global Q-value (Tajmajer, 2018). Also, treating the problem as multi-objective is useful when the value of the reward weights at execution is not known in advance. The classic approach in this case is to learn a convex coverage set, which is a subset of undominated policies containing at least one optimal policy for any reward weight vector: the idea of such algorithms is to separately learn the optimal policies of a number of corner weights (Mossalam et al., 2016; Nguyen, 2018). This approach, however, is impractical in the dynamic weights setting, where the reward weights can change between episodes. In order to generalize over different reward weights, the agent's network can be conditioned on the current weights so it can learn and execute policies corresponding to different overall objectives. This results in an adapted DQN

loss function to minimize:

$$\mathop{\mathbb{E}}_{(s,a,\mathbf{r},s')\sim U(D)}[(\mathbf{r} + \gamma \max_{a'} \mathbf{Q}_{\theta^-}(s', a', \mathbf{w}) - \mathbf{Q}_\theta(s, a, \mathbf{w}))^2] \tag{2.42}$$

Several extensions of DQN have been proposed using this type of conditioning, dealing in particular with how the reward weights in the above loss function should be chosen. Abels et al. (2019) modify the loss function and experience replay to avoid forgetting the policies for weight vectors that haven't been experienced recently. Yang et al. (2019) propose a new loss function that lets the agent use information gathered with other policies to update the policy for the current reward weight vector. Wang et al. (2022b) propose near on-policy experience replay, which reduces the extrapolation error caused by data distribution mismatch.

## 2.6 Conclusion

In this chapter, we gave a tour of reinforcement learning. After presenting the field's terminology, we went through a typology of existing algorithms, from value-based and policy-based methods to model-based methods, which each present their own trade-offs. We concluded on extensions to the domain's core framework, notably multi-task and multi-objective reinforcement learning, which hold the promise of extending the applicability of reinforcement learning by tackling more complex settings that are relevant to real-world applications. Given this state of the art, in the next chapter we set out to apply these techniques to radar resource management.

# Chapter 3

# Applications of reinforcement learning to multi-function radars

In this chapter, we propose applications of reinforcement learning techniques to current multi-function radar resource management architectures. We focus on two components: task scheduling and active tracking dwell optimization. We tackle scheduling with model-based methods and active tracking with model-free ones. Improvements over existing resource management methods are highlighted.

## 3.1 Multi-function radar scheduling

Multi-function radars require efficient resource management strategies to fulfill their missions. In particular, task scheduling is crucial to mitigate difficult situations such as when all tasks cannot be accomplished. However, current approaches may prove insufficient in the face of emerging threats. Reinforcement learning could provide a more performant alternative to devise future radar schedulers. In the present chapter, we follow this line of thinking by making several proposals. After an overview of existing methods, we start by giving the scheduling problem a formulation that allows building schedules in a flexible way, which facilitates the discovery of high-value solutions. This opens the way for different resolution methods that we then present: first, a new heuristic that outclasses existing ones while keeping computational cost low; then, an adaptation of Monte Carlo Tree Search that is applicable to this new formulation. We show that our algorithms provide noticeable performance improvement over similar methods proposed

previously.

### 3.1.1 Literature review

Given that a radar runs under constraints of time, power, and processing, the radar's resources must be allocated between its different functions according to the radar's mission, which determines the priority level of each function (Moo and Ding, 2015; Charlish and Katsilieris, 2017). Among these resources, time budget is the most critical, as the time spent performing a given task closely reflects its importance. Each function consists of one or more tasks, that can be fixed, like search tasks, or assigned dynamically, for example tracking a specific target; each task is carried out by performing a number of dwells. Based on the current situation, the radar continuously generates dwell requests corresponding to its different functions. Dwell requests are characterized by requirements in desired execution time, duration, and power, and are assigned priority levels. The resulting list of requests is transmitted to a scheduler, whose role is to decide which dwells should be executed and at what time. Efficient scheduling is crucial in overload situations, which happen when temporal constraints prevent from executing all dwells, forcing the scheduler to drop some of them according to their level of priority. Prioritization therefore plays a key role in radar schedulers.

As mentioned in chapter 1, radar schedulers fall into two categories. Best-first schedulers execute requests from a heuristically-ordered queue. Common heuristics include earliest start time first (EST), earliest deadline first (EDF), and highest priority first (HPF). It is also usual to combine these simple heuristics into more complex rules (Butler et al., 1997). The alternative is frame-based schedulers, which pack tasks into fixed-duration frames. The advantage of this approach is that by considering a frozen list of tasks to schedule, it allows the problem to be well-defined mathematically, which opens the possibility of optimization methods. However, exact resolution is computationally too expensive for the required time frames, so that in practice heuristics are preferred (Orman et al., 1996; Winter and Baptiste, 2007; Jeauneau et al., 2013). Yet another advantage of frame-based schedulers over best-first ones is to permit greater control over the task parameters: for example, the task duration can be modeled as a variable, which allows for more flexibility and enhanced utilization of the radar timeline (Mir and Guitouni, 2014). It also enables task interleaving, where during a task's idle time be-

tween emission and reception another task may be scheduled, also increasing antenna utilization (Orman et al., 1996). Frame-based scheduling has also been approached with tabu-search methods (Abdelaziz and Mir, 2016) and genetic algorithms (Zhang et al., 2019).

Despite the appeal of heuristics in terms of low computational load, their performance is usually far below the optimal solution. This has prompted an interest in artificial intelligence techniques that could improve scheduling performance while conforming to real-time computational constraints. Among the first attempts of this kind was Izquierdo-Fuente and Casar-Corredera (1994)'s use of Hopfield networks to learn an interleaving scheduler. More recently, Shaghaghi and Adve (2018) studied the application of deep learning to task scheduling on multi-channel radars, i.e. radars that use different frequencies to execute multiple tasks simultaneously. This kind of radar handles multiple timelines, which improves antenna utilization but makes scheduling harder. The authors ran a branch-and-bound algorithm on instances of the problem to create a dataset which they used to train a neural network that estimates the value of the nodes of the search tree. These evaluations are then used to prune the search tree by eliminating the least promising nodes, which speeds up the search process. The proportion of pruned nodes is determined by a scaling factor, which controls how robust the algorithm is to estimation errors. The resulting combination of deep neural network and branch-and-bound method produces near-optimal solutions with significantly reduced computation, even if it remains too slow for real-time radar scheduling and requires a fixed number of tasks.

To reduce computation, Shaghaghi et al. (2018) address the problem with Monte Carlo Tree Search (MCTS). Along with branch-and-bound bound and dominance rules, they use a neural network to guide MCTS's exploration policy. The network is trained on a dataset similarly obtained from branch-and-bound solutions. This further cuts down time complexity compared to branch-and-bound methods while maintaining high-quality solutions. Their algorithm is applicable to an arbitrary number of active input tasks. To remove the need for a training dataset generated offline, Shaghaghi et al. (2019) use a similar solution but train the policy network with data produced by the radar's interactions with its environment. The tree search is modified to accommodate several constraints, such as non-homogeneous channels, blocked channels, and periodic tasks.

Similarly, Gaafar et al. (2019) propose a modified MCTS for single-channel radar

task scheduling to lower the computational complexity. The modified MCTS notably does not revisit states where all solutions branching from it have already been visited. The exploration of the search tree is guided by an earliest start time first heuristic. The authors also propose an RL variant of their algorithm using a scheme similar to AlphaZero (Silver et al., 2017a), whereby a value network and a policy network are trained based on MCTS rollouts and used to guide the exploration.

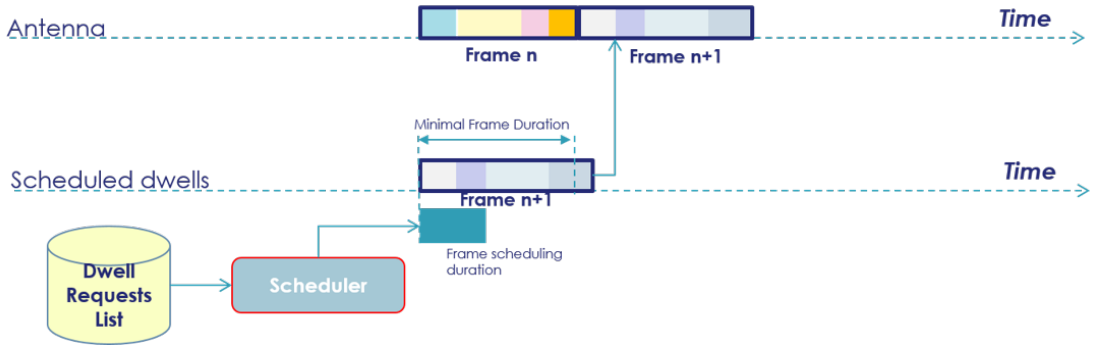


Figure 3.1: Schema of a frame-based scheduler.

### 3.1.2 Framework

We formalize the problem of radar task scheduling similarly to Gaafar et al. (2019). As we mentioned above, our model of an MFR scheduler processes one fixed set of tasks[1] at a time. An updated set of tasks is received at regular intervals. In the meantime, no tasks may be added to the set that is being processed. Each task in a set $I = \{1, ..., n\}$ is characterized by its temporal constraints and its priority level. The temporal constraints of a task $i$ are defined by its length $L^i$, its start time $T_s^i$ and drop time $T_{dr}^i$ (respectively the earliest and latest date at which it can start executing), and its due time $T_{du}^i$, which is the desired execution time. In this work, for any given instance, these constraints are all held constant over the course of the scheduling process: we do not consider preemptive tasks (e.g. for dwell interleaving) or variable duration time. Our goal is to determine for each task $i$ in $I$ whether to schedule it ($x^i = 1$, else 0), and if so at what execution time $t^i$, or to drop it ($y^i = 1$, else 0). Scheduled tasks must be entirely contained in a temporal frame $[0, T_{max}]$. In order to arbitrate between tasks in case of conflicts, each task is ascribed a drop cost $C_{dr}^i$ and a delay cost $C_{de}^i$ which reflect its priority level.

[1] In the rest of this chapter, we will use the standard terminology for scheduling, where "task" refers to the basic elements of a schedule—in our case, radar dwells. It should not be confused with the radar tasks mentioned earlier.

The drop cost is incurred only when the corresponding task is dropped while the delay cost determines how much the difference between the actual and ideal execution times is penalized; this difference is an absolute value, unlike in (Gaafar et al., 2019) where due times were not distinct from start times. How to assign relevant values to these costs in an operational context is left to future work. We write instances of the problem as $P = \{(L^i, T_s^i, T_{dr}^i, T_{du}^i, C_{dr}^i, C_{de}^i)\ \forall i \in I\ ; T_{max}\}$, and its (partial) solutions, or schedules, as $s = \{(x^i, y^i, t^i)\ \forall i \in I\}$. The objective is to minimize the sum of costs (or total cost) $C_P(s)$. We summarize the problem below, where $\oplus$ represents an exclusive or:

$$\min C_P(s) = \sum_{i \in I} x^i \left| t^i - T_{du}^i \right| C_{de}^i + y^i C_{dr}^i$$
$$\text{s.t.} \begin{cases} T_s^i \leqslant t^i \leqslant T_{dr}^i\ \forall i \in I \\ t^i + L^i \leqslant T_{max}\ \forall i \in I \\ t^i + L^i \leqslant t^j \oplus t^j + L^j \leqslant t^i \text{ if } x^i = x^j = 1,\ \forall (i < j) \in I^2 \\ t^i \in \mathbb{R}^+\ \forall i \in I \\ x^i, y^i \in \{0,1\} \text{ with } x^i = 1 - y^i,\ \forall i \in I \end{cases} \tag{3.1}$$

This problem can be solved to optimality with mixed-integer programming (MIP). Unfortunately, since problem (3.1) is in NP[1], the computation time for MIP grows exponentially with the number of tasks, making it prohibitive for radar applications. The exclusive-or constraint has an interesting implementation in MIP:

$$\begin{cases} \left.\begin{array}{l} t^i + L^i \leqslant t^j + M(n^{ij} + o^{ij}) \\ t^j + L^j \leqslant t^i + M(n^{ij} + 1 - o^{ij}) \end{array}\right\} \forall (i < j) \in I^2 \\ n^{ij} \leqslant 2 - x^i - x^j\ \forall (i < j) \in I^2 \\ n^{ij}, o^{ij} \in \{0,1\}\ \forall (i < j) \in I^2 \end{cases} \tag{3.2}$$

In order to give further insight into the structure of the problem, we offer its detailed formulation for MIP with pre-processing:

[1] This can be proven via a polynomial reduction with the knapsack problem by setting $T_s^i = C_{de}^i = 0$ and $T_{dr}^i = T_{max}$ for all $i$ in $I$.

$$\min C_P(s) = \sum_{i \in I} l^i_{de} C^i_{de} + (1 - x^i) C^i_{dr}$$

$$\text{s.t.} \begin{cases} \left.\begin{array}{l} T^i_s \leqslant t^i \leqslant T^i_{dr} \\ t^i + L^i \leqslant T_{max} \\ l^i_{de} \geqslant t^i - T^i_{du} \\ l^i_{de} \geqslant T^i_{du} - t^i \end{array}\right\} \forall i \in I \\ \left.\begin{array}{l} t^i + L^i \leqslant t^j + M(n^{ij} + o^{ij}) \\ t^j + L^j \leqslant t^i + M(n^{ij} + 1 - o^{ij}) \end{array}\right\} \forall (i < j) \in V \\ t^i + L^i \leqslant t^j + M n^{ij} \; \forall (i,j) \in U \\ x^i + x^j \leqslant 1 \; \forall (i < j) \in X \\ n^{ij} = 2 - x^i - x^j \; \forall (i < j) \in U \cup V \\ t^i, l^i_{de} \in \mathbb{R}^+ \; \forall i \in I \\ n^{ij} \in \mathbb{R}^+ \; \forall (i < j) \in U \cup V \\ x^i \in \{0,1\} \; \forall i \in I \\ o^{ij} \in \{0,1\} \; \forall (i < j) \in V \end{cases} \tag{3.3}$$

We use the following notations:

- $n^{ij}$: boolean, true if tasks $i$ and $j$ are not both scheduled;
- $o^{ij}$: boolean, true if task $j$ is executed before task $i$, false if it is the opposite;
- $M$: an arbitrarily large number such that $M \gg T_{max}$;
- $i \to j \equiv T^i_s + L^i \leqslant T^j_{dr} \; \wedge \; T^i_s + L^i + L^j \leqslant T_{max}$ ; $i$ can precede $j$ iff $i$ and $j$ can both be scheduled with $i$ executing before $j$;
- $(i,j) \in X \equiv i \nrightarrow j \; \wedge \; j \nrightarrow i$ ; $i$ and $j$ are incompatible (have incompatible interference) iff it is impossible to schedule both;
- $(i,j) \in V \equiv i \to j \; \wedge \; j \to i$ ; $i$ and $j$ are invertible (have bilateral compatible interference) iff they can be scheduled in both orders;
- $(i,j) \in U \equiv i \to j \; \wedge \; j \nrightarrow i \; \wedge \; T^j_s < T^i_{dr} + L^i$ ; $i$ and $j$ have unilateral compatible interference iff they can be scheduled in only one order and could overlap.

Of interest in formulation (3.3) is the presence of binary variables $o^{ij}$ in addition to $x^i$ (on the other hand, $n^{ij}$ is an intermediary variable entirely determined by $x^i$ and $x^j$). The values of $o^{ij}$ determine the order in which the scheduled tasks are placed. This highlights the fact that problem (3.1) can be subdivided in three successive sub-problems:

1. *inclusion*: determine which tasks to schedule;

2. *ordering*: determine the order in which these tasks should be scheduled;
3. *time setting*: determine the execution times of these ordered scheduled tasks.

Note that problem (3.1) can also be solved with heuristics like earliest start time first (EST) or earliest deadline first (EDF). These heuristics (and variations thereof) are common in radar resource management because they are fast and easy to implement. However they will often produce poor solutions, for two reasons: first, because they operate on the premise that the schedule must be built by adding tasks in chronological order; second, because they do not account for task priority.

By contrast, we aim to develop approximate algorithms for radar task scheduling that can approach optimal solutions while keeping reasonable computation times. Our first step is to model problem (3.1) as a Markov decision process (MDP), characterized by its state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T$ and reward function $R$. Once we have defined a suitable MDP, we will need an algorithm that outputs a strategy for this MDP, which we call the decision algorithm. Following Gaafar et al. (2019), we define a state as a partial schedule, where a number of tasks have been scheduled at certain execution times and a number of others dropped while respecting the constraints of (3.1). The only initial state $s_0$ for a given instance of the problem is the associated empty schedule, where no tasks are scheduled or dropped. We also define an action as the choice of one task to schedule in the set of available tasks $A = \{i \in I \mid x^i = y^i = 0\}$. Terminal states are states where no actions are available, that is, complete schedules, where all tasks are scheduled or dropped.

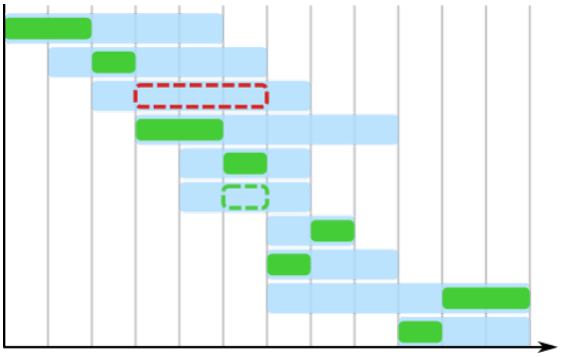

Figure 3.2: Example of a partial schedule.

We can define the reward as the cost difference between the two states involved in a transition, i.e. $R(s, a, s') = C_P(s) - C_P(s')$. With $\gamma = 1$, the value function becomes $V^\pi(s_0) = \mathbb{E}_\pi [C_P(s_0) - C_P(s_T)]$, which corresponds to our initial objective of finding a

schedule $s$ that minimizes $C_P(s)$. In practice, in our algorithms, we reason directly on costs.

The choice of the transition function is the most crucial aspect of this formalization. Transitions must deterministically decide what execution times to set and which tasks to drop when a new task is added to a partial schedule.

The simplest option, used in (Gaafar et al., 2019), is to add tasks chronologically: any newly added task $i$ is assigned an execution time greater than that of all previously scheduled tasks, and tasks that cannot be placed after $i$ are dropped. However, this restricts the range of viable strategies, since the decision algorithm has to solve both the inclusion and the ordering sub-problems at the same time, while the transition function only deals with the time setting sub-problem.

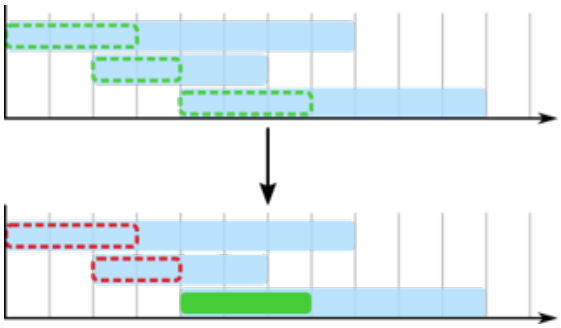

Figure 3.3: Example of a chronological transition.

Instead, we propose a transition model that allows building schedules in a more flexible way. Our idea is to model the transition function such that it solves both the ordering and the time setting sub-problems at each step. That is, given a set $S$ of scheduled tasks, the transition determines the order of tasks and their execution times by minimizing the total delay cost:

$$\min \sum_{i \in S} l^i_{de} C^i_{de}$$

$$\text{s.t.} \begin{cases} \left.\begin{array}{l} T^i_s \leqslant t^i \leqslant T^i_{dr} \\ t^i + L^i \leqslant T_{max} \\ l^i_{de} \geqslant t^i - T^i_{du} \\ l^i_{de} \geqslant T^i_{du} - t^i \end{array}\right\} \forall i \in S \\ \left.\begin{array}{l} t^i + L^i \leqslant t^j + M o^{ij} \\ t^j + L^j \leqslant t^i + M(1 - o^{ij}) \end{array}\right\} \forall (i < j) \in V \cap S^2 \\ t^i + L^i \leqslant t^j \; \forall (i,j) \in U \cap \\ t^i, l^i_{de} \in \mathbb{R}^+ \; \forall i \in S \\ o^{ij} \in \{0,1\} \; \forall (i < j) \in V \cap S^2 \end{cases} \tag{3.4}$$

Given a partial schedule $s = \{(x^i, y^i, t^i) \; \forall i \in I\}$, in order to schedule a new task $j \in I$, we solve sub-problem (3.4) for $S = \{i \in I \mid x^i = 1\} \cup \{j\}$. If there is no solution to the sub-problem, $j$ has to be dropped. After a task is scheduled, all remaining tasks in $A$ can be tested for dropping using this procedure.

Sub-problem (3.4) can also be treated with a MIP solver; however this requires finding the order and execution times of all tasks in $S$ at every transition. This is inefficient, because with this transition model, when we move from a partial schedule $s$ to a new one $s'$ by adding a task $i$, the execution times in $s$ are already optimal with regard to the tasks scheduled in $s$. This means that if for example $i$ can be placed at its desired execution time without interfering with already scheduled tasks, then there is no need to recompute the execution times of these tasks. To exploit this and a number of other optimizations, we implement a custom solver for sub-problem (3.4) which we do not detail here due to space limitations: the key idea is to recursively generate and evaluate permutations of the scheduled tasks while limiting the number of generated permutations by exploiting temporal constraints. Note that this way, we can check the availability of a task faster, by interrupting the recursion as soon as we find a feasible permutation.

Our transition model allows adding tasks in any order by delegating part of the optimization—ordering and time setting—to the environment. Crucially, for any schedule $s$ reached through this transition model, the execution times are optimal given the tasks scheduled in $s$. (We can also limit the number of generated permutations to lower the computation time, although we then lose the optimality guarantee.) This opens new

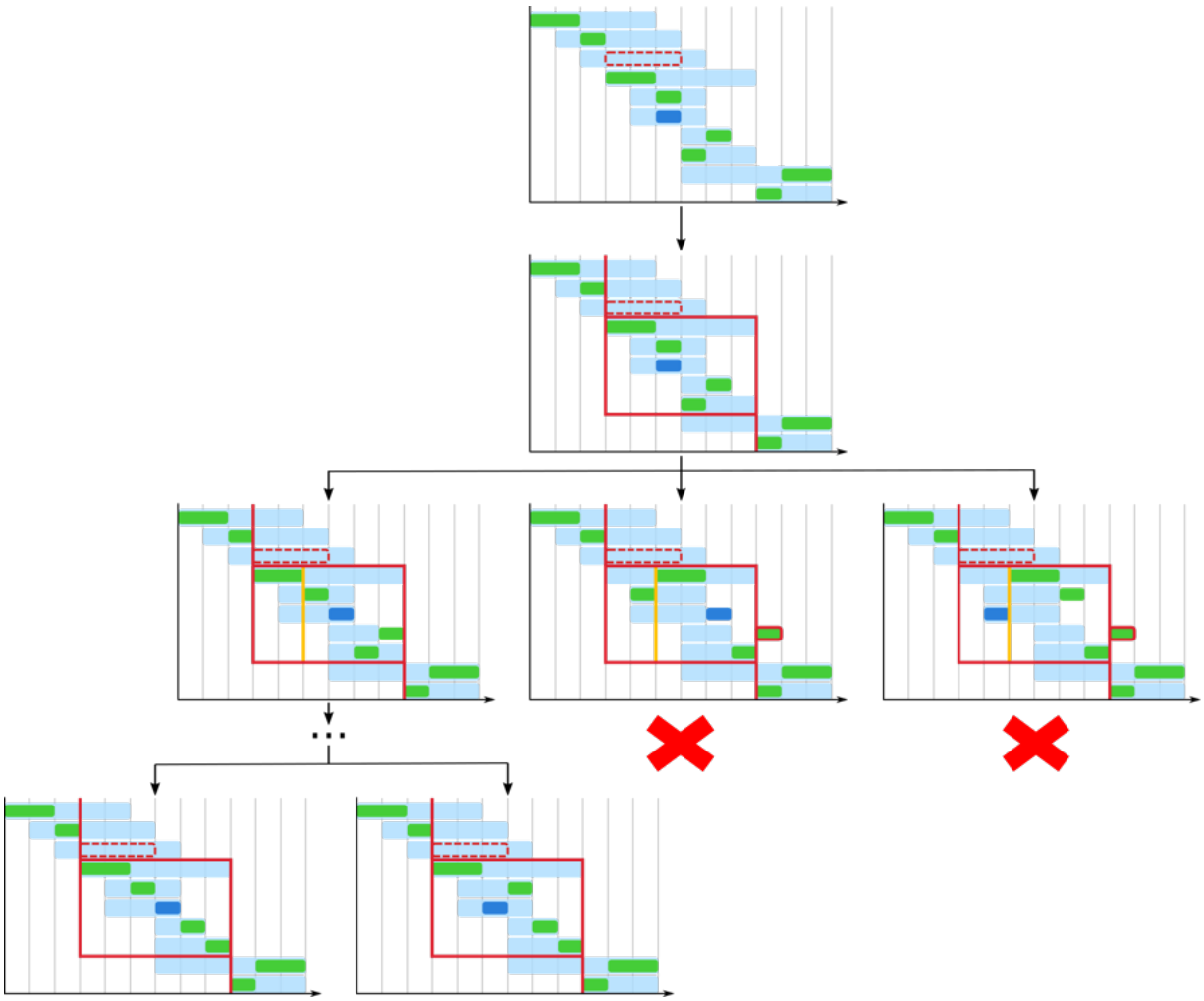

Figure 3.4: Example of a transition processed by the custom solver.

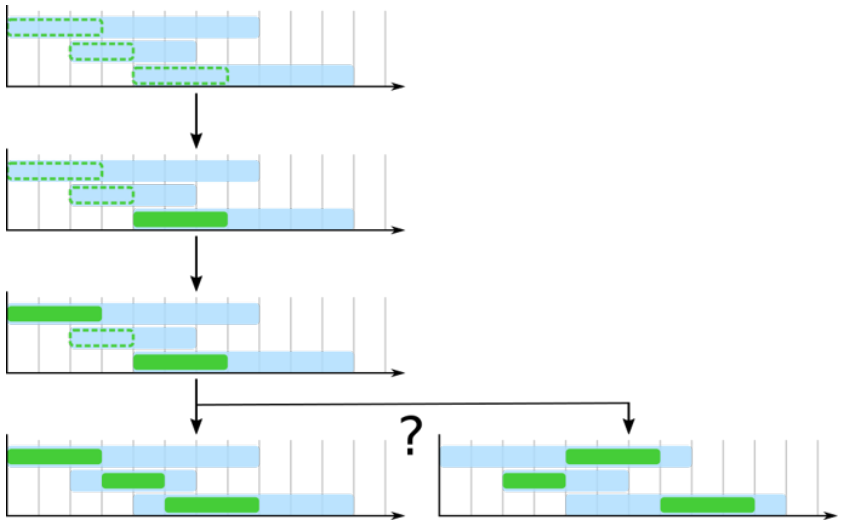


Figure 3.5: Solving the time setting and ordering problem.

possibilities for the choice of the decision algorithm, whose role is to solve the hardest part of the problem: inclusion.

### 3.1.3 Task selection methods

#### HCLR

The first decision algorithm we propose is a heuristic that we call highest cost-length ratio first (HCLR), which is similar to some knapsack problem heuristics. It consists in sorting tasks by their ratio between drop cost and length, then scheduling them in decreasing order (if they are still available when their turn comes). Equivalently, at each step, we choose the following action:

$$a = \text{argmax}_{a \in A} \frac{C_{dr}^{i}}{L^{i}}$$

This heuristic empirically performs better than scheduling the task with the highest drop cost first, because it accounts for situations where for example two shorter, lower-priority tasks have a higher total drop cost than one longer, higher-priority task which they are incompatible with. A similar criterion has been proposed for radar resource management by Qu et al. (2019); however, the authors used the ratio in a task selection phase that preceded the scheduling itself, which is based on EST. Using our transition model, we can instead directly select and schedule each task in turn.

**MCTS**

Our next decision algorithm is a version of Monte Carlo Tree Search (MCTS) adapted to task scheduling, which is based in large part on the version of Gaafar et al. (2019). MCTS uses a search tree where nodes represent MDP states and branches correspond to actions taken in the parent node (Browne et al., 2012). The tree is constructed by successive rollouts: starting from the root node, which corresponds to the initial state, we select an action, apply the transition function to get the next node (which is created if needed), and repeat. Once a terminal state is reached, we can compute its total cost $C$, then backpropagate $C$ up the path we just followed to update the best cost reached from each state-action pair: $C(s,a) \leftarrow \min\{C(s,a), C\}$. This value is then used in the computation of the upper-confidence bound $U(s,a)$ which determines which actions are chosen in the selection phase: $a = \text{argmax}_{a'}\ U(s,a')$ with $U(s,a) = \frac{P(s,a)}{C(s,a)^{\tau}(1+N(s,a))}$ where $\tau$ is a temperature, $N(s,a)$ is the number of rollouts where action $a$ was taken in state $s$, and $P(s,a)$ is a prior probability function over actions. This selection rule is designed to balance exploration of the search tree and exploitation around the best solutions found so far. The prior, especially, plays a prominent role in steering exploration; a simple way to parameterize it is to sort the $m$ available tasks according to a criterion, then assign them a respective prior probability of $p(1-p)^m$ for $m$ ranging from 0 (for the first task) to $m-1$. For our experiments, we set $\tau = 2$ and $p = 0.6$, similarly to (Gaafar et al., 2019), but we diverge by using HCLR as our sorting criterion instead of EST.

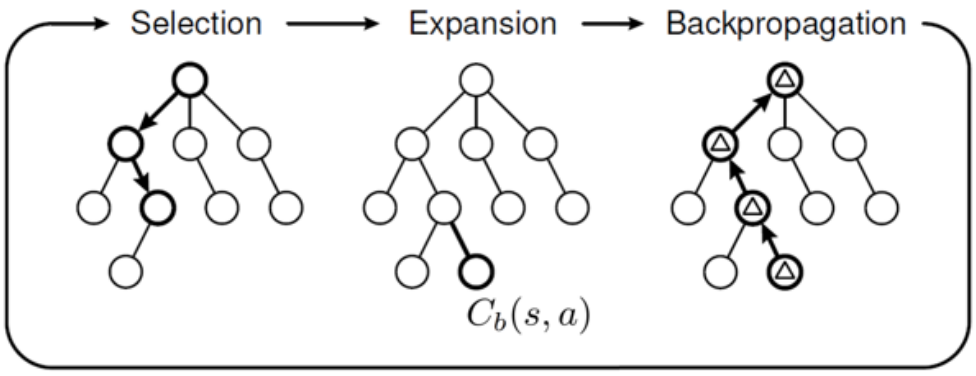


Figure 3.6: The steps of MCTS (adapted from Browne et al. (2012)).

One issue is that when using our transition model, since tasks can be scheduled in any order, it is possible to reach the same terminal state via multiple different rollouts. In order to prevent this, we structure our search tree similarly to a branch-and-bound

(B&B) tree : when a new action is taken in node $s$, leading to a new node $s'$, all actions explored in $s$ in previous rollouts are made unavailable in $s'$ and its descendants. This makes sure there is only one path to each terminal state in the search tree. However, it also means that we may reach terminal nodes that are not complete schedules, when some tasks can still be scheduled, but have all been made unavailable by the previous rule. To limit the number of such situations, at each visit of a node $s$, we check if there exists a terminal node $s'$ descended from $s$ such that all tasks that are unexplored in $s$ are scheduled in $s'$; if so, these tasks are made unavailable in $s$. Additionally, to avoid performing the same rollout twice, if a node has no more available actions, the action that led to it is also made unavailable in the parent node, as in (Gaafar et al., 2019). We call this algorithm B&B-MCTS.

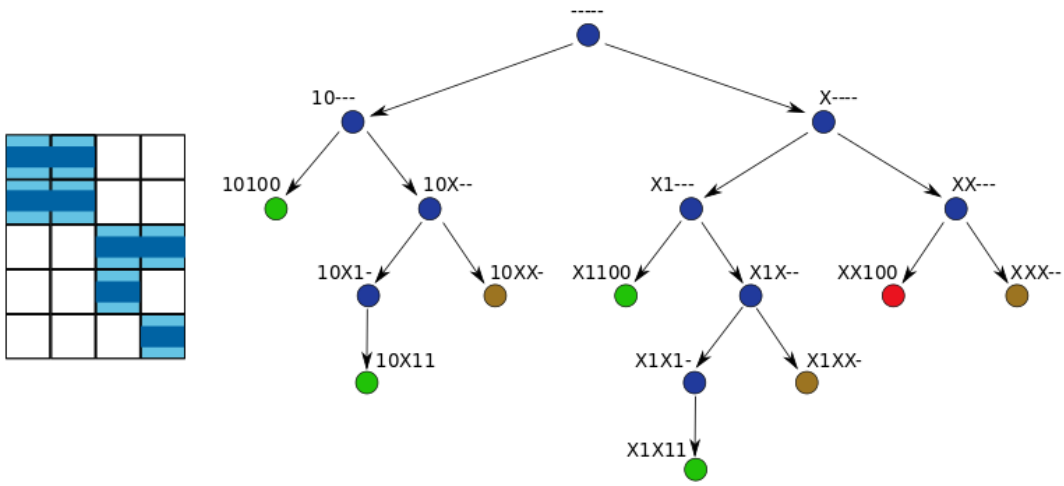


Figure 3.7: Example of a B&B MCTS execution on the 5-task instance shown on the left. Node label: 1 if the task is scheduled, 0 if dropped, dash if available, red if disallowed. Solutions are shown in green. Brown nodes can be pruned based on solutions already obtained, which restricts the search tree.

Usually, in MCTS, when we reach a new state, we want to know which actions are available. With B&B-MCTS, this allows making the most effective use of pruning, according to the above rules. However, with our transition model, it requires partially solving (3.4) for each a priori available task to check if it has to be dropped. In larger instances, this involves significant computation, which reduces the number of rollouts that can be carried out in a given time. We propose two variants of B&B-MCTS: a "safe" one that checks for unavailable tasks to drop when a new node is created, and a "rash" one that attempts to schedule tasks without prior verification, and drops them

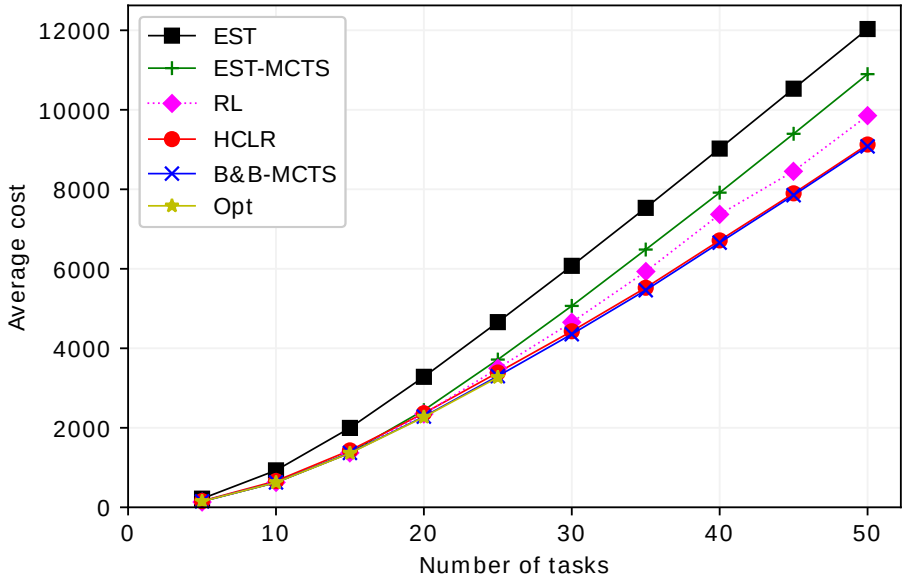


Figure 3.8: Cost plotted against instance size for identical start and due times.

only if the attempt fails.

### 3.1.4 Results

To enable comparisons, we run experiments on instances from the same distribution as in (Gaafar et al., 2019): $L^i \sim U(2, 15)$, $T_s^i \sim U(0, T_{max} - 12)$, $T_{du}^i = 0$, $T_{dr}^i - T_s^i \sim U(2, 12)$, $C_{dr}^i \sim U(100, 500)$, $C_{de}^i \sim U(1, 15)$, $T_{max} = 100$. The difficulty of an instance mostly depends on the density of tasks; by keeping $T_{max}$ fixed, the difficulty can be controlled by setting the number of tasks.

We compare our two methods, HCLR and B&B-MCTS, with the EST heuristic and with the version of MCTS proposed in (Gaafar et al., 2019) (which we term EST-MCTS). The run time of both versions of MCTS is limited to 1 second. All these algorithms are implemented in Python. We compute the optimal solution using MIP on instances where this resolution can be performed in a reasonable amount of time. These five algorithms are run on the same instances, 1000 per number of tasks. We also compare our results to those reported by (Gaafar et al., 2019) for a reinforcement learning-based extension of EST-MCTS inspired by AlphaZero (Silver et al., 2017a).

Our results show that our approach provides a significant performance improvement over EST-MCTS, and even surpasses the reinforcement learning algorithm of (Gaafar et al., 2019), which used deep learning in conjunction with MCTS. Strikingly, the per-

Table 3.1: Detailed statistics on three different instance types.

| Algorithm | Avg. cost | Cost std. dev. | Dropped tasks (%) | Optimal solutions (%) | Avg. runtime (milliseconds) | Avg. number of rollouts |
|---|---|---|---|---|---|---|
| Number of tasks = 25, identical start and due times | | | | | | |
| EST | 4658.69 | 634.65 | 59.06 | 0.0 | 1.93 | nan |
| EST-MCTS | 3716.55 | 506.0 | 52.15 | 0.87 | 1002.88 | 727.06 |
| HCLR | 3390.6 | 551.68 | 48.2 | 21.35 | 13.12 | nan |
| B&B-MCTS | 3299.39 | 526.16 | 48.44 | 57.73 | 975.32 | 96.12 |
| MIP-optimal | 3268.43 | 523.85 | 48.65 | 100.0 | 1730.82 | nan |
| Number of tasks = 50, identical start and due times | | | | | | |
| EST | 12031.8 | 857.18 | 77.64 | nan | 2.32 | nan |
| EST-MCTS | 10897.03 | 758.17 | 73.23 | nan | 1004.79 | 587.63 |
| HCLR | 9130.98 | 799.9 | 63.62 | nan | 36.7 | nan |
| B&B-MCTS | 9080.49 | 780.86 | 63.79 | nan | 991.72 | 30.1 |
| Number of tasks = 25, distinct start and due times | | | | | | |
| HCLR | 3301.15 | 550.93 | 47.53 | 22.0 | 19.98 | nan |
| B&B-MCTS | 3219.61 | 524.51 | 47.61 | 56.5 | 994.69 | 70.96 |
| MIP-optimal | 3185.86 | 520.0 | 47.88 | 100.0 | 2219.44 | nan |

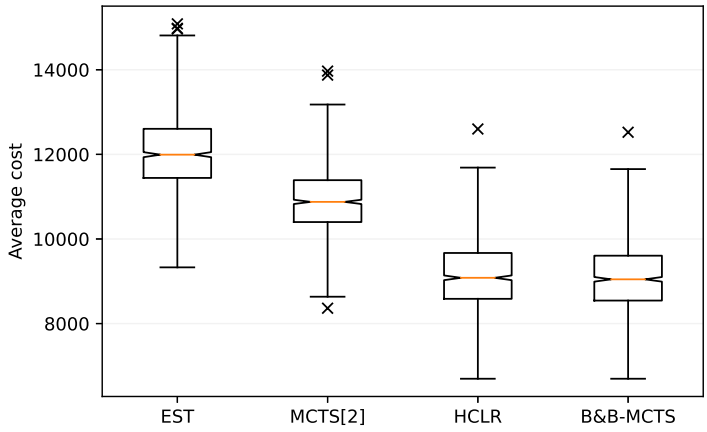


Figure 3.9: 50 tasks

formance gain of B&B-MCTS over HCLR proves minimal. Moreover, with our implementation, HCLR's run time averages 36 milliseconds on 50-task instances, which would presumably make it more suitable for radar use cases than MCTS-based methods (Vincent et al., 2021).

Unlike in (Gaafar et al., 2019), our approach allows due times distinct from start times, thus reflecting radar requirements more closely. We also run experiments on a similar task distribution but with $T^i_{du} \sim U(1,4)$ and $T^i_{dr} - T^i_s \sim U(1,8)$; our results match those of the first distribution very closely.

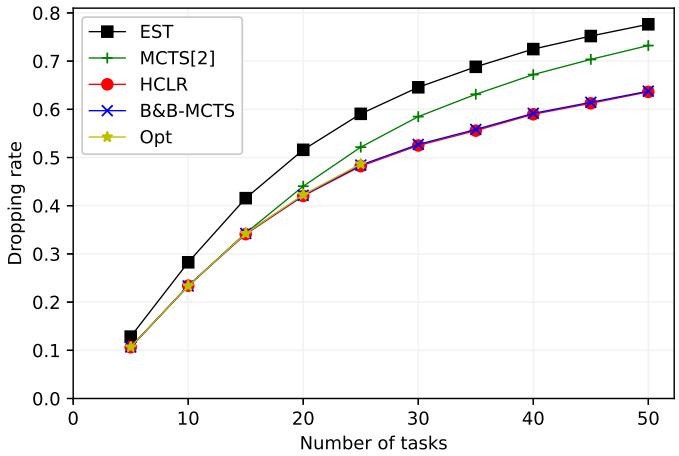


Figure 3.10: 50 tasks

## 3.2 Reinforcement learning for active tracking

Electronically-scanned array antennas give multi-function radars more degrees of freedom over the parameters of the dwells they can form. In particular, digital beamforming offers more possibilities to vary the direction and shape of the beam, conferring to the radar a form of beam agility. Dynamic beamwidth adaptation, especially, could help to track more challenging targets, for example when hypermaneuverability and hypervelocity are involved: increasing the beamwidth during maneuvers, for example, would increase the robustness of the track by making the target less likely to go out of beam and avoid detection. However, the ability to vary beamwidth from one dwell to another has so far been underutilized. The reason is that the most common active tracking parameter selection methods assume a fixed beamwidth in order to determine the required update rate of the track. Therefore, if we want to perform dynamic beamwidth adaptation, we also need to make decisions for the update rate. The beamwidth used also has an impact on the dwell's signal-to-noise ratio (SNR), which larger beamwidths decrease, lowering the detection probability and measurement accuracy. To compensate for lower SNR, the duration of the dwell should be increased. More adaptive active tracking resource management methods should therefore control these three parameters of beamwidth, update rate and waveform duration concurrently to optimize track maintenance.

In this chapter we approach this problem through a reinforcement learning lens. After examining existing methods for active tracking parameter selection, we describe

the simulator we implemented to train and test our agents. We first apply our method to single-target tracking, where we do not take into account the radar's global state. We then study a more holistic approach to radar resource management, where we attempt to manage both search and multi-target tracking: in this setting, we uncover limitations of standard reinforcement learning methods, which motivates the development of new algorithms in the next chapter.

### 3.2.1 Existing methods

Track update rate is a crucial part of target tracking. The track update rate can equivalently be expressed as the revisit interval, which is the duration between update attempts. The problem of setting track update rates is also known in the literature as beam scheduling, not to be confused with task scheduling which is the placement of the dwell requests on the antenna timeline by the scheduler as studied in section 3.1. Longer revisit intervals mean lower antenna time utilization, but are only suitable for non-maneuvering targets with predictable trajectories. Maneuvering targets, on the other hand, require shorter revisit intervals as predictions on their movement must be updated often. Most algorithms select the update rate based on either the innovation sequence of the tracking filter (Baek et al., 2010) or on the predicted error covariance matrix (van Keuk and Blackman, 1993; Daeipour et al., 1994).

These methods are computationally efficient and can be deployed in current radars. However, they are model-sensitive: they especially depend on target models which can be unknown or simplified compared to real target dynamics. Additionally, they determine the update rate on a per-track basis. In practice MFRs may have to track hundreds of targets in parallel: if the radar is overloaded, the prioritization and scheduling modules must arbitrate for which tracks the update must be delayed. Yet this limits the adaptability of track updates and results in a lack of coordination between the different tracks.

The alternative is to consider performing multi-target resource allocation. This problem has in particular been studied as a type of restless bandit problem. Restless bandits are an extension of $k$-armed bandits where each action is associated to a process that evolves over time. Processes have a default (or passive) transition function, which differs from the active transition that takes place when the process is selected. Unlike in

previous methods, the idea isn't to set the revisit interval of each target after its update; instead, this revisit interval emerges from the choice of updating or not a target at each step. Resolution methods for bandit formulations of multi-target tracking use variants of dynamic programming. Krishnamurthy and Evans (2001) develop optimal and sub-optimal algorithms for optimal beam scheduling in electronically scanned array tracking systems, where the problem is formulated as a multi-arm bandit problem with hidden Markov models. La Scala and Moran (2006) study adaptive beam scheduling and propose an algorithm to minimize target tracking error with phased array radars. The bandit formulation can also be adapted to cover the update of both search and tracking tasks, as proposed in (Wintenby, 2003). This setting removes the requirement for prioritization and scheduling modules distinct from the dwell generation phase.

Bandit formulations for multi-target resource management is a promising approach as it could allow joint optimization over all radar tasks, foregoing coordination issues with common beam scheduling methods. However, current resolution methods for these formulations use dynamic programming, which is accompanied by high computational complexity owing to the curse of dimensionality, making it impractical to implement in real radar systems. They also require to make a number of theoretical assumptions that do not take into account some of the physical constraints exerted on radar systems.

Given the limitations of current active tracking methods, reinforcement learning provides an appealing alternative. If applied to single-target tracking, it holds the promise of increased adaptability in terms of environment models, target models, and simultaneous optimization of more dwells parameters than previously possible, especially regarding to beamwidth. If applied to multi-target tracking, deep reinforcement learning would allow global resource management optimization while lifting the hurdles posed by dynamic programming methods proposed previously. Pulkkinen et al. (2021) pioneered the application of reinforcement learning to adaptive update rate selection for single-target tracking. Specifically, a tabular Q-learning algorithm with epsilon-greedy exploration is used to adaptively select the revisit interval of the track after each update, and the reward to maximise is defined as the negative ratio between the update's observation time and previous revisit interval. If the target is lost, a negative track loss cost is given as reward instead and the episode ends. Different state spaces are tested: they include either the mode probabilities of the IMM or the innovation, i.e. the change in the filter's angular accuracy after the update; either state can be augmented with the revisit

interval selected at the previous step. The performance of the proposed approach is favorably benchmarked against the prediction error covariance matrix-based solution from (Daeipour et al., 1994) in terms of mean and peak tracking loss and position prediction error values. However, the approach is trained and evaluated on the same data, which is a set of just six trajectories, which limits the paper's claims in terms of generalization. Furthermore, the use of a tabular algorithm limits the scalability of this method to more informative state spaces or to multi-target scenarios. Finally, this approach is focused only on update rates, whereas our goal is to allow adaptability in beamwidth as well, which forces us to select observation time simultaneously.

### 3.2.2 Radar simulator

In order to train and evaluate RL methods, we need to implement an environment that simulates the interaction of a radar with its surroundings. This must include trajectory generation as well as a complete radar pipeline, from detection to track management to dwell generation. Most importantly, we need an antenna model that reflects the effects of the dwell parameters we are interested, especially beamwidth and observation time. We do not aim to maximize physical accuracy as this would severely impact performance and needlessly slow down the learning process. Instead we need a behavioral simulator where our experiments can provide a proof of concept. The environment must then wrap this simulator and provide an interface for reinforcement learning algorithms to interact with it.

We build our simulator as an extension to the Stone Soup framework. Stone Soup provides a testbed for tracking and state estimation algorithms. It is written in Python and focused on flexibility to allow the user to build an environment suited to their use through the combination of components corresponding to various radar modules. Standard implementation of various components are provided, which researchers can substitute for their own to develop new algorithms and compare them with existing ones according to metrics. To support this modularity, Stone Soup clearly defines the external interface of each type of components, which gives it the extensibility we need to wrap a RL environment around it.

The architecture of our simulator is outlined in fig. 3.11. Stone Soup provides groundtruth simulators, which generate trajectories based on one or more kinematic

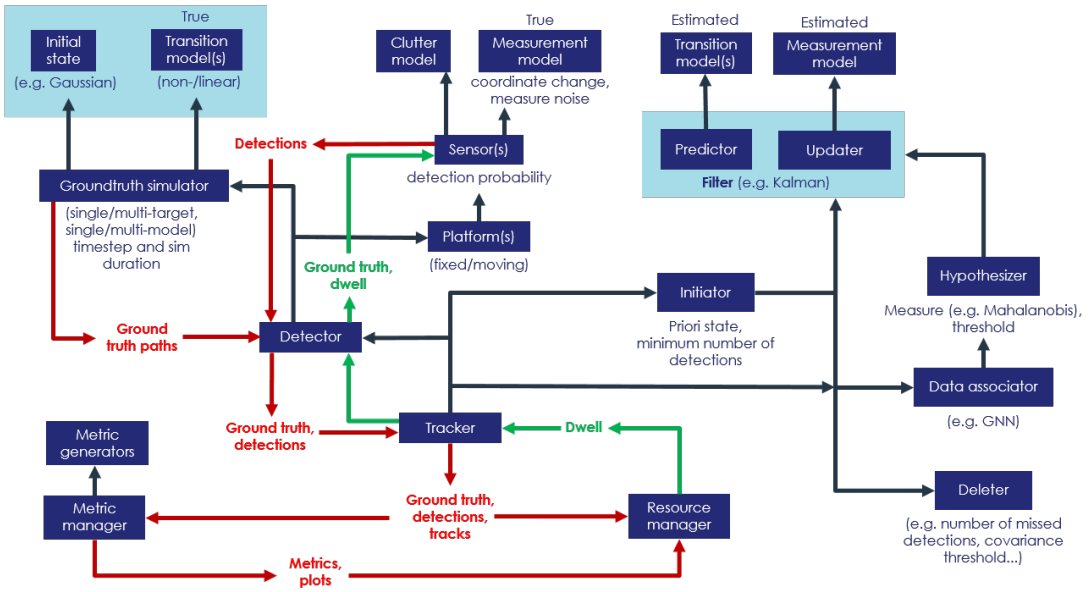


Figure 3.11: Architecture of our simulator based on Stone Soup. Black arrows represent composition relationships between modules. Red arrows represent the original Stone Soup data flow. Green arrows represent the feedback loop added to our simulator.

models. A sensor component models the detection probability of target and returns measurements originating from it based on its measurement model. A detector component coordinates groundtruth generation and the detections resulting from them. The detections are fed to a tracker component that covers the whole management of the tracks' life cycles from initialization to deletion. The detections go through a data association phase and are then used to update the associated track via a filtering algorithm; a Kalman filter and variants thereof are provided. The groundtruth, detections, and tracks are collected to establish metrics.

We extend Stone Soup's framework in a number of ways to make it suitable for our experiments. First, we implement an Interacting Multiple Model (IMM), which is the type of filtering algorithm most adapted to the tracking of maneuvering targets. We also develop a sensor component that models an electronically scanned array antenna along the lines of the description given in chapter 1. Most crucially, we add a resource management component that establishes a feedback loop between the tracks and the dwells generated at the next step. Stone Soup only implements a one-way data flow, from groundtruth to tracks. By contrast, the resource manager takes the current tracks as input and can generate dwells which are transmitted to the sensor The resource manager serves as the interface with which the reinforcement learning algorithms can

interact to receive observations corresponding to the tracks and to take actions which will be translated into dwells to send to the antenna. These dwells affect the detection probability and measurement model of the antenna. Finally, we implement an interactive plotter based on Plotly to analyze the results of runs in terms of tracks and metrics.

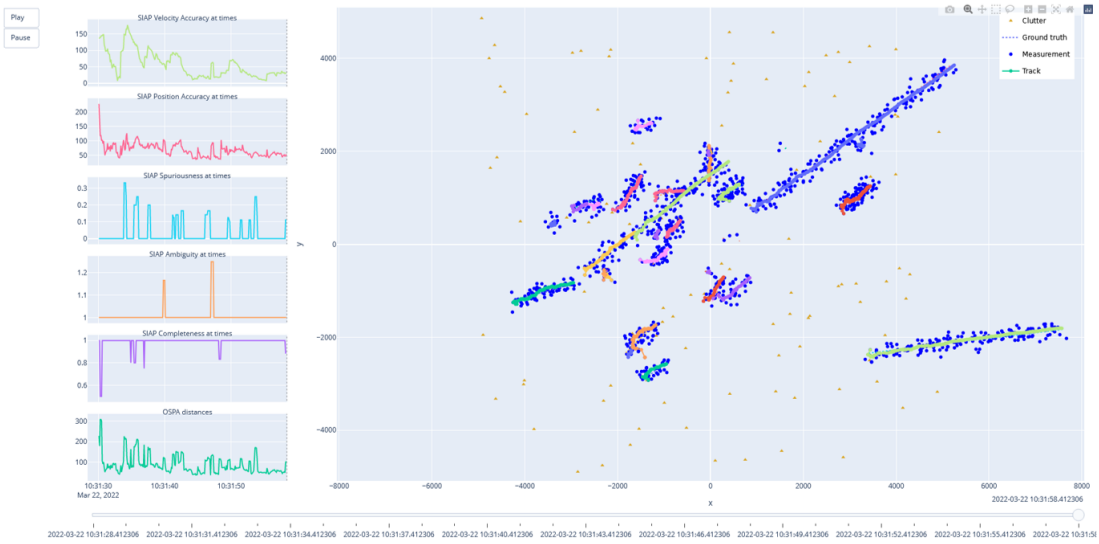


Figure 3.12: Example of the rendering of a run in the simulator.

### 3.2.3 Single-target tracking

We now turn to our objective to optimize the joint selection of beamwidth, update rate, and observation time for active tracking. We simplify the setting by assuming a single-target scenario. We also ignore the elevation to consider a 2D simulation. Each episode starts with the track begin initialized at the target's initial state; we do not study the role of the search function in the track's initialization as it happens in real systems. The trajectory of the target is generated online with a maneuvering target kinematic model; the target is constrained to remain in a 40km radius around the radar. The target's state consists in its position, velocity, and acceleration in Cartesian coordinates $[x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}]^\top$. The antenna is characterized by a reference signal-to-noise ratio $SNR^{ref} = 15$dB at 50km for a reference dwell with observation time $T_{obs}^{ref} = 432\mu$s and beamwidth $\theta_{3dB}^{ref} = 4.12°$, and a probability of false alarm $P_{fa} = 1 \times 10^{-5}$. The filter algorithm used for the simulation is an IMM estimator that takes three linear Kalman filters, one per target kinematic models: these are a constant velocity, a constant turn, and a constant acceleration models, all with noise coefficient $q = 0.01$; the constant turn

rate is $\omega = 0.3$rad. The transition probability matrix between the three models is given by:

$$p = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.3 & 0.7 & 0.0 \\ 0.2 & 0.2 & 0.6 \end{bmatrix} \tag{3.5}$$

The IMM's target model mirrors the one used to generate the trajectories. Finally, an episode is interrupted if the track is lost, or ends after a time limit of 200s. A track is considered lost if the IMM's standard deviation of the estimated target azimuth exceeds 3° or after 5 consecutive missed detections.

In terms of interaction between the environment and the RL agent, the observations exposed to the agent consist in the state of the target as estimated by the IMM, the IMM's current mode probabilities, and the previous action. The target state information contains the estimated position and velocity in range and azimuth, as well as their estimated variance. Along the previous action, the observation includes information about whether a detection occurred at the previous time step and about the accuracy of this detection in terms of variance in azimuth. We use a discrete action space where each action corresponds to a combination of an observation time and a beamwidth. The available observation times are $\mathcal{A}_T = \{1, 2, 3, 4\} \times T_{obs}^{ref}$ while the available beamwidths are $\mathcal{A}_\theta = \{0.25, 0.5, 0.75, 1, 1.25\} \times \theta_{3dB}^{ref}$. The agent's action is converted to a dwell according to these parameters. We also include a no-op action for which no dwell is generated, which gives us a total of 21 possible actions. An environment step corresponds to 1 second of in-simulation time, to mimic the constraints of a antenna rotating at 60 rotations per minute. The update rate is therefore not considered a settable parameter, but emerges implicitly from the alternation of no-ops and dwell actions. The transition function is implemented by the radar simulator based on Stone Soup outlined above. The reward function takes into account the satisfaction of an accuracy window $[\sigma_{min}, \sigma_{max}]$, defined in terms of angular standard deviation in azimuth $\sigma_{az}$ as computed by the IMM; if the agent takes an action other than no-op, it receives a penalty proportional to the dwell's observation time $T_{obs}$ weighted by a coefficient $c_{obs}$ to discourage increased radar load; in case of track loss, a more significant penalty $c_{loss}$ is incurred. The reward can therefore be described as:

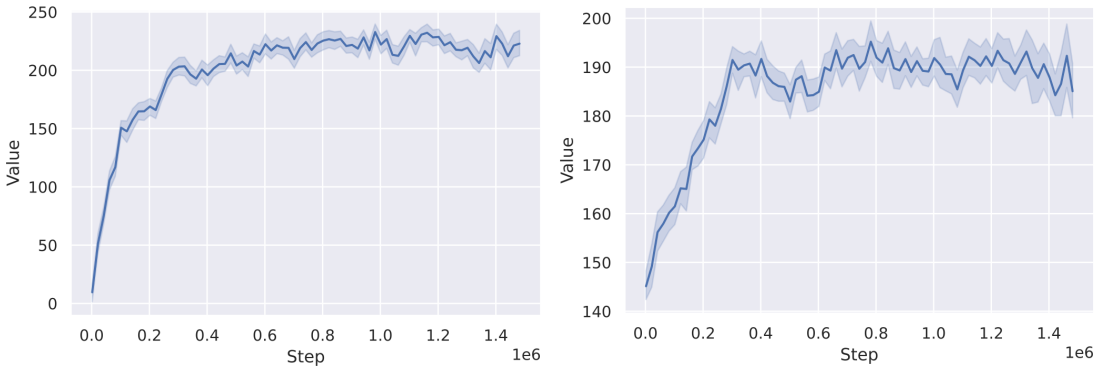


Figure 3.13: Mean and 95% confidence interval of reward (left) and episode length (right) over 5 training runs.

$$r = \begin{cases} -c_{loss} & \text{if track is lost} \\ \mathbb{1}[\sigma_{min} \leqslant \sigma_{az} \leqslant \sigma_{max}] - \frac{T_{obs}}{\max \mathcal{A}_T} & \text{otherwise} \end{cases} \tag{3.6}$$

In our experiments we set $\sigma_{min} = 1°$, $\sigma_{max} = 2°$, $c_{obs} = 1$, and $c_{loss} = 50$. The choice of these parameters determines the trade-off between track accuracy, track loss, and tracking load that the agent is expected to maintain.

We train our agent with PPO. We use two separate neural networks to output the policy and value function. The policy network is a multi-layer perceptron (MLP) with 2 layers of size 64, while the value network is a 2-layer MLP with size 256; both use tanh activation functions. The clipping parameter $\epsilon$ is set to 0.25, and the generalized advantage estimation parameter $\lambda$ to 0.9. Training takes place over 1.5 million steps; training updates happen every 2000 steps, where each training update is repeated for 10 epochs with minibatch size 256. The learning rate is $3 \times 10^{-4}$.

To evaluate our algorithm, we consider the track loss rate and the mean tracking load, which is the ratio between the accumulated observation time over all dwells and the duration of the episode. When evaluated, our RL agent demonstrates a track loss rate of 12.9% for a mean tracking load of $1.96 \times 10^{-3}$. As a point of comparison, we implement the standard update rate selection strategy from (van Keuk and Blackman, 1993). This method uses a fixed beamwidth, and schedules dwells such that the angular accuracy of the track doesn't exceed one sixth of this beamwidth. Using the reference beamwidth $\theta^{ref}_{3dB}$ for the Blackman-van Keuk method, we obtain a track loss rate of 29.1% for a mean tracking load of $3.02 \times 10^{-3}$. As we can see, our RL agent greatly reduces

the risk of track loss all while maintaining lower antenna utilization.

### 3.2.4 Multi-target tracking

The single-target approach does not provide a satisfactory answer to all the problems surrounding current resource management systems. The main shortcoming of current methods is that they optimize each task separately according to its own criteria, before instructing the scheduler to arbitrate the execution of task requests according to their respective priorities. This creates two problems in case of overload: on the one hand, the dwell parameters do not take into account or imperfectly take into account the global state of the system, which can lead to vicious circles; on the other hand, it makes the behavior of the system dependent on the prioritization of the tasks while this prioritization is not necessarily adapted to the objectives of the radar.

These limitations lead us to question the conventional resource management architecture to address the problem on a larger scale, by applying reinforcement learning directly at the level of the entire resource management module. The modeling of the problem would be as follows: the state space would correspond to the list of all radar search areas and tracks, with associated information (time since last update, estimated track accuracy); the action would be to choose which of these tasks to update and what dwell parameters to update it with; the reward could be defined based on the desired tracking accuracy thresholds and the actual accuracy on all targets present in the system, including undetected ones, which would encourage maintaining search (alternatively we could include a term in the reward that is directly related to the quality of the search). This represents an extension to the restless bandit framework, where the choice of task to update is further parameterized. Additionally, we note that our problem is better described as a multi-objective, with one reward function per task: by weighing each reward, we could reflect the priority level of each task. To our knowledge, no existing reinforcement learning method combines the combinatorial and multi-objective aspects required by our use case; this motivates the search for such methods in the next chapter. We hope, with such an approach, to be able to take full advantage of the flexibility of MFRs to enable system-wide optimization.

## 3.3 Conclusion

In this chapter, We studied two components of radar resource management: scheduling and active tracking.

Through a reformulation of the problem of radar task scheduling, we were able to implement an efficient framework whose algorithmic applications can return quasi-optimal solutions in a limited amount of time. We see potential improvements for this framework, both by optimizing its run time to make it usable by real-world radars, and by increasing its flexibility; for example, being able to remove a task from a partial schedule could allow exploring the solution space more effectively. Furthermore, our B&B-MCTS algorithm could be extended with an AlphaZero-style reinforcement learning procedure which would make it able to adapt to various task distributions.

We apply model-free reinforcement learning to single-target active tracking to learn to adjust the update rate, beamwidth and duration of dwells dynamically. The extension of this type of method to multi-target tracking requires the development of multi-objective algorithms that can deal with combinatorial state spaces, which we approach in th enext chapter.

# Chapter 4

# Challenges in multi-objective reinforcement learning

The difficulty of resource management in an MFR is to maintain a balance between several competing sub-objectives: each function must be able to be carried out while limiting its use of resources, so as not to interfere with other sub-objectives. These functions are subdivided into tasks: for the search function, a task can correspond to the surveillance of a given area; for the tracking function, it corresponds to one track. To apply end-to-end reinforcement learning to the whole radar resource management module, we therefore need to account for both the multi-objective and the combinatorial aspects of the problem. More specifically, we need algorithms that can tackle environments where the number of objectives may vary between episodes. Moreover, some of the radar's objectives may have strictly higher priorities over others: for example, it is common to specify a minimum required total search time ratio, or to have tracks corresponding to threatening targets take priority over non-threatening ones. These preferences cannot be readily expressed with a linear scalarization over criteria: therefore we are interested in non-linear multi-objective reinforcement learning methods.

In this chapter, we address each of these points to open perspectives toward a fully reinforcement learning-controlled radar resource manager. We review current methods, detail their limitations, and propose novels algorithms for each case.

## 4.1 Multi-objective factored MDPs

Many potential applications of reinforcement learning involve complex, structured environments. Radar resource management is no exception in this regard, as in our framework, it involves a selection phase where the agent must choose the next radar dwell from a list where each dwell updates a given task. Some of these problems can be analyzed as factored MDPs, where the dynamics are decomposed into locally independent state transitions and the reward is rewritten as the sum of local rewards. However, in some scenarios, these rewards may represent conflicting objectives, so that the problem is better interpreted as a multi-objective one, with a weight associated to each reward. To deal with such *multi-objective factored MDPs*, we propose a method which combines the use of graph neural networks, to process structured representations, and vector-valued Q-learning. We show that our approach empirically outperforms methods that directly learn from the scalarized reward and demonstrate its ability to generalize to different weights and number of entities.

Most research in the field of deep reinforcement learning initially focused on end-to-end learning, where the agent starts out with no prior on the task, for example having to learn to play Atari games from pixels (Mnih et al., 2015). Standard RL algorithms thus make minimal assumptions about the structure of the task: they typically take monolithic observations as input, which have to be disentangled internally. However, for complex problems, we often have information about the structure of the environment at our disposal. This information can be used to inject priors into the learning process. Before the advent of deep RL, researchers tackled large Markov decision processes (MDP) by decomposing the states and rewards into locally conditioned elements (Guestrin et al., 2003b); such factored MDPs were more amenable to tabular approaches, which were hardly directly applicable to large state spaces. Even though in recent times function approximation by deep neural networks has alleviated these issues, accounting for the structure of the state space still helps solving complex environments, for example using relational learning. Relational deep learning aims to embed inductive biases into deep learning architectures in order to leverage assumptions on structured representations composed of entities and their relations (Battaglia et al., 2018). These representations can be processed by graph neural networks (GNN), which have been successfully applied to challenging reinforcement learning environments, enabling agents to generalize to

varying numbers of entities, for example in real-time strategy games like StarCraft and combinatorial puzzle games like Sokoban (Zambaldi et al., 2018; Vinyals et al., 2019; Janisch et al., 2021).

In factored MDPs, rewards typically have an additive structure: they represent the sum of locally-scoped rewards. However, some tasks are better described as striking a compromise between several possibly contradicting objectives. Multi-objective reinforcement learning treats such problems where multiple reward functions are used, each associated with a different objective (Roijers et al., 2013). The overall goal is given by a utility function that depends on these rewards and on their associated weights, which reflect their level of priority. This setting allows easier definitions of the desired compromises between the competing objectives; it also makes it possible to train an agent that adapts to utility functions that may change over time, e.g. when using a linear scalarization whose weights are not constant. Many promising applications of reinforcement learning benefit from a multi-objective specification. For example, self-driving cars should minimize travel time and maximize the comfort and safety of passengers as well as the safety of nearby pedestrians and drivers at the same time (Li and Czarnecki, 2019). Multi-objective reinforcement has also been used to train a video game-playing agent that can adopt a wide variety of play-styles, opening the way to automated game testing (Le Pelletier de Woillemont et al., 2021).

In this section, we study the case where the problem combines a structured environment with a multi-objective specification. We argue that this is a natural extension of the concept of factored MDP that holds broad applicative implications, including for radar systems. The radar must regularly update these tracks and surveillance areas, but it can only update one of these tasks at a time due to signal processing and antenna constraints. We can view tracks and surveillance areas as tasks with limited mutual dependence. These tasks are given priority levels according to the mission and threat model of the radar. The antenna time and power constraints under which the radar operates may force the system to make compromises between tasks when under heavy load. Approaching this problem from a RL perspective requires designing an agent which can deal with variable state and action spaces (as the number of targets can vary) and takes into account the priority levels of the different tasks it has to fulfill.

Given the characteristics of this type of problems, we claim that they are best approached within both the factored MDP and multi-objective paradigms. Although sig-

nificant work has been carried out in each field, to our knowledge their intersection has not received scrutiny yet. We therefore term our framework *multi-objective factored MDPs* and set out to analyze how to best solve these problems. To do so, we propose a multi-objective version of DQN (Mnih et al., 2015) that can be applied to structured observations by making use of recent advances in graph neural networks proposed by (Janisch et al., 2021). We discuss how our algorithm, factored multi-objective DQN (FMODQN), can be adapted according to the type of action space and to the characteristics of the factored MDP. We test our approach on a multi-objective version of SysAdmin, a classic example of factored MDP (Guestrin et al., 2003b), and on a novel multi-objective task featuring multiple independent entities. Comparisons are made with (Janisch et al., 2021)'s algorithm, that learns from the scalarized reward. We show that our algorithm outperforms this baseline, can generalize to different reward weight vectors and number of entities, and is able to deal with hundreds of objectives.

### 4.1.1 Structured representations

#### Factored MDPs

In the default MDP formulation, no information about the structure of the environment is given, forcing the agent to learn from *tabula rasa*. Yet there are many problems which can be decomposed into smaller parts whose evolution is largely independent of each other: typically when modeling a production line composed of a number of machines, we would know that the next state of each machine is only influenced by its immediate neighbors. Priors about this type of environment structure motivate us to define compact representations that can help learning agents to generalize more easily over complex states.

Factored MDPs are a notable framework for representing structured MDPs compactly via the exploitation of the conditional independence structures of transitions and rewards (Guestrin et al., 2003b; Osband and Van Roy, 2014). More formally, a state $s \in \mathcal{S}$ is reinterpreted as a set of $n$ state variables $(s_1, s_2, ..., s_n)$. From there, we can define a scoped variable as a subset of the state variables, so that for a scope $Z \subseteq \{1, ..., n\}$, the corresponding scoped variable is $s[Z] = (s_i)_{i \in Z}$. We write $s[i]$ instead of $s[\{i\}]$ or $s_i$ for clarity. Given $n$ scopes $Z_1, ..., Z_n$, the transition probability function can then be factored into $n$ locally-scoped transition functions in the form

$$T(s' \mid s, a) = \prod_{i=1}^{n} T_i(s'[i] \mid s[Z_i], a) \tag{4.1}$$

Similarly, the reward function can be factored over $m$ scopes $Z_1^r, ..., Z_m^r$, for example as a sum $R(s,a) = \sum_{i=1}^{m} R_i(s[Z_i^r], a)$.

SysAdmin (Guestrin et al., 2003b) is a classic example of factored MDP. This problem is defined by a directed graph between $n$ nodes that represents a network of computers. Each computer $i$ is characterized by a single state variable $s_i$ with value 1 if the computer is online and 0 if it is offline. The scope of each state variable covers the state variable itself and its parents in the graph. At each time step, the agent can either wait or force a single computer to go (or stay) online; the other computers may randomly shutdown if they are online, based on their dependencies, and may reset when offline. The probability for a computer to be online at the next step if it is not acted on at this step is:

$$T(s_{t+1}[i] = 1) = \begin{cases} 0.9\frac{\sum_{j \in Z_i} s_t[j]}{|Z_i|} & \text{if } s_t[i] = 1 \\ 0.04 & \text{if } s_t[i] = 0 \end{cases} \tag{4.2}$$

All computers are online at the beginning of an episode. The agent receives a reward that depends on the number of online computers and includes a penalty $p < 0$ for manually rebooting a computer:

$$R(s_t, a_t) = \sum_{1 \leqslant i \leqslant n} s_t[i] + p \cdot \mathbb{1}(a_t \neq noop) \tag{4.3}$$

### Graph neural networks

Factored MDPs provide structured representations of complex environments. The next question is how an agent can efficiently process such representations. In machine learning, structured representations generally refer to graph data showing the relations between a set of entities, for example between the atoms of a molecule or the bodies of a physical system. These representations have been approached through the prism of graph neural networks (GNN) (Battaglia et al., 2018; Zhou et al., 2021). GNNs take a tuple $(V, E, \mathbf{g})$ as input, where $V = \{\mathbf{v}_i\}_i$ is a set of node attributes, $E = \{(\mathbf{e}_k, r_k, s_k)\}_k$ a set of directed edges characterized by attributes $\mathbf{e}_k$ and indices of receiver $r_k$ and sender $s_k$ nodes, and $\mathbf{g}$ represents the global context. The nodes, edges, and context of the graph are updated by

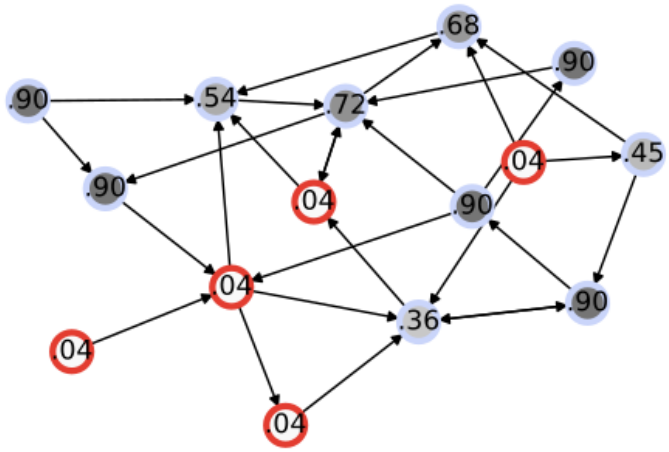


Figure 4.1: A visualization of a SysAdmin instance. Offline nodes are red, online ones are blue. The probability of a node going online at the next step (unless it is manually reset) is shown in each node. The probability of a node going offline increases with the number of offline parent nodes.

a series of network passes. GNNs impose strong inductive biases, since they can process arbitrary relations between entities, are invariant to node permutations, and support combinatorial generalization to different numbers of nodes and edges. They have been successfully applied to reinforcement learning tasks involving reasoning about interacting entities (Hamrick et al., 2018; Wang et al., 2018; Zambaldi et al., 2018).

### Related work

Structured sequential decision problems have long been studied in the frameworks of factored MDPs and relational MDPs (Boutilier et al., 2002). In most works on the subject, the environment is assumed to have known dynamics, so that it can be described in an influence diagram language like RDDL (Sanner, 2011) or PPDDL (Younes et al., 2005), for which there is a rich literature of planning methods (Bonet and Geffner, 2003; Keyder and Geffner, 2008; Kolobov et al., 2009, 2012). Several methods aim to perform generalized planning by constructing plans that transfer across instances (Fern et al., 2003; Guestrin et al., 2003a; Natarajan et al., 2011; Bonet et al., 2019). A series of recent works (Toyer et al., 2017; Bajpai et al., 2018; Garg et al., 2019, 2020; Sharma et al., 2022) combines planning and neural networks in order to enable transfer learning.

Methods that operate in a pure RL setting without assuming known dynamics are less studied. GRL (Karia and Srivastava, 2022) uses tabular Q-learning where the Q-values are initialized by a neural network trained to predict the final Q-values on different

problem sizes, meaning it cannot deal with continuous state spaces; it also cannot perform zero-shot adaptation to problems of different sizes without retraining its tabular Q-values. The method most closely related to ours in the literature is SR-DRL (Janisch et al., 2021), which combines message-passing neural networks and auto-regressive policies into a neural network trained with A2C (Mnih et al., 2016) that is applicable to problems with variable state and action spaces. The method is tested on three domains, including SysAdmin, and it is shown that the agent is able to generalize to different problem sizes after training on a single problem size.

In contrast to related work, our method, factored multi-objective DQN (FMODQN), can solve multi-objective factored MDPs with model-free reinforcement learning. We hereafter describe it and detail how it can be adapted to different types of multi-objective factored MDPs.

### 4.1.2 Problem statement

In the context of factored MDPs, the reward structure is decomposed into a number of locally-scoped rewards. The overall reward can then be retrieved by an aggregation of these local rewards, typically through a simple sum. We argue that in many cases however, each local reward can be interpreted as a different objective. These objectives might not all be of equal importance, with a notion of priority dictating compromises between them. For example, it is not a stretch to suppose that some computers in the SysAdmin scenario might play a more critical role than others in the network's infrastructure. We therefore modify the factored MDP formalism to account for this multi-objective component. More specifically, we alter the agent's objective to maximize the scalarization of expected returns over a vector of locally-scoped rewards $\mathbf{R}(s_t, a_t) = (R(s_t[Z_i^r], a_t))_{1 \leqslant i \leqslant m}$. In this section, we restrict ourselves to linear scalarizations; therefore, we can rewrite equation 2.39 to express the agent's objective as:

$$\max_{\pi \in \Pi} \sum_{i=1}^{m} w_i \mathop{\mathbb{E}}_{\pi, T} [\sum_{t=0}^{\infty} \gamma^t R(S_t[Z_i^r], A_t)] \tag{4.4}$$

Our focus is on combinatorial generalization, that is, on environments where the number of state variables and local rewards may vary between episodes, and on agents that can generalize to these variations: as a consequence, we only consider cases where the local transitions and rewards are defined the same for all scopes. This is the case

in SysAdmin for example, where the evolution of each computer follows the same rules, and where we can associate the same local reward function to each computer, i.e. 1 if the computer is online and 0 otherwise. This is a key difference compared to most of the multi-objective reinforcement learning literature, where the number of objectives is usually a constant of the problem, since each reward corresponds to an objective of a different kind, like maximizing safety and minimizing travel time for self-driving cars. By contrast, in the type of multi-objective factored MDP that we focus on, there is actually a single reward function that gets applied to different local scopes: for example, to reuse the example of the production line, we would define a single reward function that could return a measure of performance of any machine to which it is applied. Encoding the priority of each reward as a weight rather than as the magnitude of the reward has several advantages: all rewards will by definition have the same scale, allowing the agent to generalize to any number of rewards, and we will then be able to condition the agent's neural network on the current reward weights so that it can learn a policy adapted to different preferences over rewards. Besides, since the weights of all rewards in an instance of the problem sum to 1, the reward weights tend to be lower as the number of rewards increases: this makes the priority levels implicitly linked to the number of rewards.

### 4.1.3 Algorithm

The general approach in multi-objective value-based methods is to train (using eq. 2.42) a Q-value estimator that returns a matrix $\mathbf{Q}(s, \cdot, \mathbf{w}) \in \mathcal{R}^{|\mathcal{A}| \times m}$ from which we can then retrieve the scalarized action-state value $Q(s, a, \mathbf{w}) = \mathbf{Q}(s, a, \mathbf{w})^\top \cdot \mathbf{w}$. The policy is derived by choosing the action that maximizes this scalarized Q-value. The question is how to design an agent that can generate this Q-matrix on problems where the number of rewards $m$ (and possibly the number of actions) may vary from episode to episode.

**Feature extraction** Inspired by (Janisch et al., 2021), we use GNNs to process factored state representations. To do so, we need to represent the whole factored MDP state as a graph. For a given factored MDP decomposed into $n$ state variables and $m$ rewards, we define a directed graph $G$ with $n$ *state* nodes $(c_1, ..., c_n)$, where each state node's attribute consists of the associated state variable $s[i]$ and where an edge $c_i \rightarrow c_j$ exists only if $i \in Z_j$. This graph, containing only state information, is the input model proposed by (Janisch et al., 2021) to process the state and retrieve features used to

compute the action logits and the state value as required by policy-based algorithms. However, in our case, we need to predict the Q-values for each reward, so we have to integrate information about the reward structure as well. Therefore, we add $m$ additional *reward* nodes $(c_1^r, ..., c_m^r)$ to the graph such that a reward node attribute contains the weight linked to this reward and an edge $c_i \to c_j^r$ exists only if $i \in Z_j^r$. State and reward nodes are distinguished by an additional type feature. With this structure, a GNN is able to predict the influence of each state variable on the rewards that depend on this state variable. Once this graph has been passed through a GNN, we only need to retrieve the updated reward nodes from the output so as to use them as features to compute the Q-value associated with each reward. Using this scheme we can condition the network on the current reward weights and stay invariant to the number of state variables and rewards/weights. A special case of interest is when there is one reward per state variable: in this situation we fuse each pair of state and reward nodes.

**Action space** Once we have extracted the features of each reward node, we can compute the Q-values. How we do this depends on the action space. If the action space is fixed and doesn't vary in size with the number of state variables, then we can apply a position-wise multi-layer perceptron (MLP) $f$ that maps each reward node output $v_i^{r\prime}$ to the Q-values of all actions for this reward, so that for all $i \in \{1, ..., m\}$, $f(v_i^{r\prime}) = (Q_i(s, a, \mathbf{w}))_{a \in \mathcal{A}}$.

A more difficult case appears when the action space is variable. We focus on the case where there is one action per state variable, for example when one entity has to be selected at each time step. In this case we need to obtain $\mathbf{Q}(s, a, \mathbf{w})$ for each action $a$. To do so, we augment each state node attribute with an action embedding with value 1 if the action corresponding to this state variable is chosen, 0 otherwise. For a given action $a$, this augmented graph $G_a$ is passed through a GNN; similarly to before, we apply a position-wise MLP $f$ to each reward node output $v_i^{r\prime}$ that gives the Q-value for action $a$ and reward $r_i$, i.e. $f(v_i^{r\prime}) = Q_i(s, a, w_i)$. This way for action $a$ we obtain $\mathbf{Q}(s, a, \mathbf{w})$. This operation is applied over $n$ augmented graphs, one for each action, to compute $(\mathbf{Q}(s, a, \mathbf{w}))_{a \in \mathcal{A}}$. The processing can be parallelized by batching the augmented graphs. We call this version of our algorithm Batch-FMODQN. One potential issue with this approach is its time complexity, which is quadratic in the number of nodes: since the time complexity of a message-passing step is $O(kn)$ for a graph with $n$ nodes and

a maximal degree $k$, repeating the operation over $n$ graphs yields $O(kn^2)$ complexity. This may limit the applicability of our method on large instances.

**Complete independence** However, we note that this complexity can be improved depending on the graph topology. In particular, in some environments the state variables may be entirely independent from each other (i.e. there are no edges in the graph). This type of structure can be seen for example in some radar scenarios where each surveillance area is an independent task that remains unaffected by the update of other areas. In this case, assuming we have one action $a_i$ and one reward $r_i$ per state variable $s[i]$, we can process each entity separately by applying a position-wise MLP to the list of state variables. This MLP has two outputs for each state variable $s[i]$: the action-state value relative to reward $r_i$ that corresponds to choosing the associated action $Q_i(s[i], a_i, w_i)$, and the same action-state value corresponding to any other action $Q_i(s[i], A \neq a_i, w_i)$. Our reasoning is that since each pair of state variable and reward is independent from the others, the only relevant information to predict the evolution of a given reward is whether the corresponding action is chosen at the present time step. We can then recover the scalarized Q-value of each action via the weighted sum of the local Q-values: $Q(s, a_i, \mathbf{w}) = w_i Q_i(s[i], a_i, w_i) + \sum_{j \neq i} w_j Q_j(s[j], A \neq a_i, w_j)$. This way we do not have to directly compute the whole Q-matrix since it would be mostly redundant; instead, the time complexity of the algorithm is $O(n)$. We call this variant of our algorithm Split-FMODQN.

**Dynamic weights** By conditioning the network on the current reward weights, we aim to make our agent perform well in the dynamic weights setting where the weights vary between episodes. On this side, we expect to experience fewer issues with reward weight variation compared to standard multi-objective reinforcement learning. Indeed, one of the challenges in multi-objective DQN is its tendency to forget the policies it has learned for other reward weight vectors over the course of training as it tends to overfit to the current weight vector instead: (Abels et al., 2019) in particular identifies this issue and proposes to modify the loss function and replay buffer to alleviate it. We argue that in the context of multi-objective factored MDPs, this problem will be less present, as there is actually one reward function applied over the different reward scopes. Since the Q-value associated with each reward will rarely depend on the whole reward weight

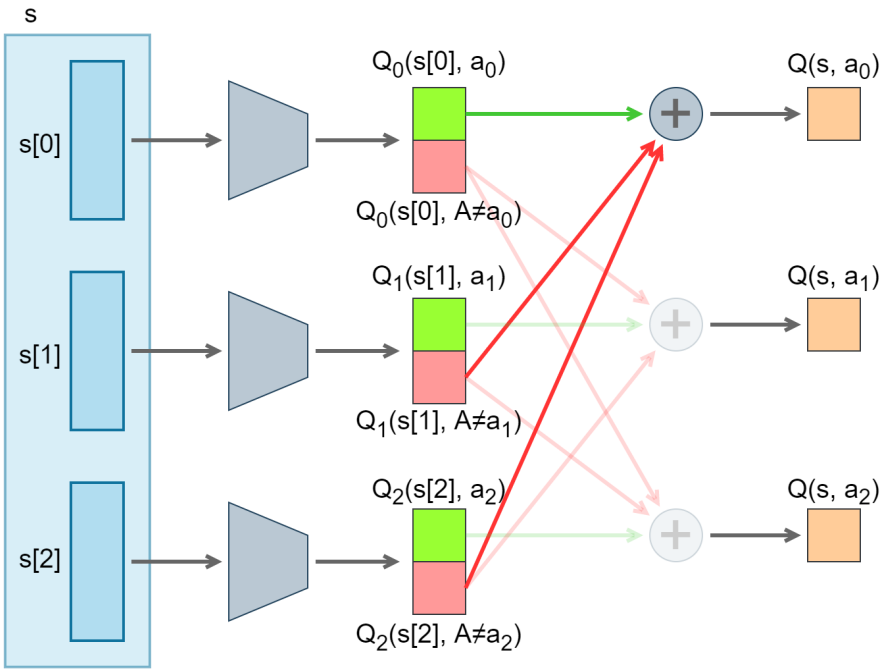


Figure 4.2: High-level view of the Split-FMODQN architecture. The same MLP is applied in parallel to each state variable: the outputs are recombined to obtain the scalarized Q-value of each action. For clarity, on this figure, reward weights are omitted from the input of the MLP and from the scalarization.

vector, our understanding is that the Q-value will generalize more easily to different weights.

### 4.1.4 Experiments

To validate our framework, we test it on two environments: we apply Batch-FMODQN to a multi-objective version of SysAdmin, and Split-FMODQN to a custom environment where all state variables are independent. Our choice of domains was guided by our long-term goal of applying these methods to air surveillance radar resource management, which involves a task maintenance scheduling problem where numerous tasks must be updated sequentially while accounting for their relative importance. We show that our algorithms can generalize to different reward weights and different instance sizes (Vincent et al., 2023).

**Training details** We train the network on GPU using multi-objective DQN loss (equation 2.42). To gather experience, we run a batch of 256 environments distributed over 8 CPU cores; training takes place over 5000 iterations, so the amount of experience is

| Algorithm | N | Score | Normalized | Time (s) |
|---|---|---|---|---|
| B-FMODQN | 5 | 76.8 ± 0.5 | 1.01 | 0.042 |
| | 10 | 58.6 ± 0.5 | 1.18 | 0.099 |
| | 20 | 42.8 ± 0.4 | 1.28 | 0.283 |
| | 40 | 33.5 ± 0.2 | 1.34 | 0.997 |
| | 80 | 27.0 ± 0.4 | 1.31 | 3.021 |
| | 160 | 22.1 ± 0.2 | 1.20 | 12.213 |
| SR-DRL | 5 | 76.0 ± 0.5 | 1.00 | 0.019 |
| | 10 | 54.3 ± 0.6 | 1.09 | 0.025 |
| | 20 | 38.9 ± 0.4 | 1.16 | 0.030 |
| | 40 | 30.0 ± 0.2 | 1.20 | 0.042 |
| | 80 | 23.6 ± 0.1 | 1.15 | 0.069 |
| | 160 | 19.9 ± 0.1 | 1.08 | 0.130 |

Table 4.1: Results of evaluation on multi-objective SysAdmin over 1000 runs in the dynamic weights setting for different problem sizes $N$. Both algorithms were trained on instances of size $N = 10$. Score is the average cumulative reward with 95% confidence interval. We also normalize the average score against the baseline score. Resolution time per instance in seconds is shown in the last column.

the same as that used by (Janisch et al., 2021), since at each iteration, we advance one step in each environment and minimize the loss with one mini-batch of 64 transitions uniformly sampled from the replay buffer. We limit the maximal gradient norm to 3. We use a soft update rate of 0.005 for the target network. The replay buffer size is $10^5$; learning starts once the buffer contains 1000 transitions. We use $\epsilon$-greedy exploration with $\epsilon$ linearly annealed to 0.1 after 4000 iterations. On all the environments we test, we set the discount factor to 0.99. For Batch-FMODQN, the GNN used to extract features is identical to that of (Janisch et al., 2021), with 5 message-passing steps and embedding size 32. We use Adam optimizer with a learning rate of $10^{-4}$. Our algorithms are implemented in the SaLinA framework (Denoyer et al., 2021). All tests were performed with AMD Ryzen 3600X CPU, 16 GB of RAM and nVidia GTX 1660 Ti GPU.

**Multi-objective SysAdmin** We test our proposal on a multi-objective version of SysAdmin. The overall reward is defined as a weighted sum of the rewards corresponding to each computer: the local reward of each computer is 1 if the computer is online and 0 otherwise. Since our focus is on the weighing of the different computers, we remove the penalty for rebooting a computer and consequently also remove *noop* from the action space since it would always be sub-optimal. The scalarized reward is therefore:

$$R(s_t, a_t) = \sum_{1 \leqslant i \leqslant n} w_i \cdot s_t[i] \tag{4.5}$$

Similarly to (Janisch et al., 2021), we randomly generate between 1 and 3 dependencies for each computer at the beginning of an episode. Episodes end after 100 steps.

Since in SysAdmin there is one reward per state variable, we model the state as a graph of $n$ nodes, each of whose attribute contains the state of the corresponding machine (1 or 0 if online or offline respectively) and the weight for this machine's reward. There is one possible action per machine and machines exhibit dependencies, so we will use Batch-FMODQN. We train a Q-matrix as described above, and reuse the GNN structure proposed by (Janisch et al., 2021). We compare our approach to the SR-DRL algorithm of (Janisch et al., 2021), using the same set of hyperparameters as the ones given in (Janisch et al., 2021); our only modifications to their algorithm are the presence of the reward weights in the node attributes, so that the network can condition on them, and the absence of a *noop* action. We normalize our results with a simple baseline policy that selects the offline computer with highest reward weight to reboot.

We test our approach in the dynamic weights setting: a new reward weight vector is generated at the beginning of each episode, both in training and evaluation. The agents are trained with problems of size 10 and evaluated on other problem sizes. Our results are presented in table 4.1. We also train agents in the fixed weights settings; each agent is trained on problems of constant size. This way we test generalization to both problem size and reward weight; these results are presented in table 4.2. Training times are 30/35/60/120 minutes for B-FMODQN versus 15/20/30/50 minutes for SR-DRL, for N=5/10/20/40 respectively.

Our first observation is that we incur no performance loss in the dynamic weights setting compared to fixed weights, demonstrating our algorithm's ability to adapt to different reward weights. Similarly, as was already observed with SR-DRL (Janisch et al., 2021), agents trained on problems with 10 nodes and evaluated on a different number of nodes $N$ attain performance levels comparable or even higher than that of agents trained directly on $N$ nodes. We also see that Batch-FMODQN clearly outperforms both the baseline and SR-DRL on this task, gaining between 9 and 16 percentage points in normalized score relative to SR-DRL for $N \geqslant 10$. This improvement in performance comes at a cost in computation time, which reflects the time complexity and grows

| Algorithm | N | Score | Normalized |
|---|---|---|---|
| B-FMODQN | 5 | 77.2 ± 0.5 | 1.02 |
| | 10 | 59.1 ± 0.5 | 1.19 |
| | 20 | 42.7 ± 0.4 | 1.27 |
| | 40 | 33.1 ± 0.3 | 1.32 |
| SR-DRL | 5 | 75.2 ± 0.6 | 0.99 |
| | 10 | 54.2 ± 0.6 | 1.09 |
| | 20 | 35.9 ± 0.3 | 1.07 |
| | 40 | 26.8 ± 0.2 | 1.07 |

Table 4.2: Results of evaluation on multi-objective SysAdmin over 1000 runs in the fixed weights setting for different problem sizes $N$. $N$ is the problem size used both for training and evaluation. Score is the average cumulative reward with 95% confidence interval. We also normalize the average score against the baseline score.

quadratically in the number of nodes, unlike SR-DRL which has linear computation time. We note however that even on the largest instances tested, the resolution time for a problem instance remains in the seconds, while alternative algorithms based on probabilistic planning may require several minutes to solve a single problem (Janisch et al., 2021).


Figure 4.3: Evolution of average cumulative reward during training on multi-objective SysAdmin with dynamic weights, $N = 10$. Mean with standard deviation over 4 runs.

**Plate Spinning** Next we propose a toy environment where we can study the case where the agent must process a set of entities that are wholly independent from each other: in terms of factored MDPs, this is equivalent to the case where the scope of each state variable is limited to that same state variable, i.e. $Z_i = \{i\}$ for all state variable indices $i$. This type of problem is similar to having to maintain a number of plates spinning on their poles. Let us suppose that the agent can only correct the balance of

one plate at a time to keep it spinning, and that the plates have different values, as if some were made of porcelain and others of terracotta. A penalty is inflicted only when a plate loses its balance and crashes on the ground. When many plates are at risk of falling off, the agent should prioritize the most valued ones. The presence of numerous sparse rewards makes this a challenging problem.

More formally, we define the problem as consisting of $n$ plates. Each plate $i$ has two features: a balance level $b_i$ and a degradation rate $d_i$. There are $n$ available actions, one for each plate. At the beginning of an episode, a plate starts with a balance level of 1; at each time step, this balance level is reduced by $d_i$, unless the agent selects this plate, in which case the balance level is reset to 1. There is one reward per plate: if the balance level of the plate reaches 0, a reward of $-1$ is given for this plate, and a new plate of same value replaces it with initial balance level 1; otherwise the reward is 0. Again, since we are interested in the multi-objective aspect, we associate a weight to each reward, so that the global reward is the weighted sum of the local rewards. To facilitate comparisons between the scores for different problem sizes $N$, we limit degradation rates so that $\sum_i d_i = N/10$. Degradation rates are uniformly generated with a maximal factor of 10 between the highest and lowest rates. Episodes last for 100 steps.

Our training procedure is the same as for multi-objective SysAdmin, except that we use Split-FMODQN to obtain Q-values, since each entity is independent. The network consists of a single MLP with 5 hidden layers of size 32. The agent's observation is the list of state attributes, containing the current level of balance, the degradation rate, and the weight of the corresponding reward. Again, we compare to SR-DRL, which we train with the same set of hyperparameters as for SysAdmin. The agents are trained with problems of size 10 and evaluated on other problem sizes, all in the dynamic weights setting. Training time with N=10 is 5 minutes for Split-FMODQN and 7 minutes for SR-DRL. Resolution time per episode reaches 17ms for Split-FMODQN and 42ms for SR-DRL with N=160.

Our results from table 4.3 show that it is difficult for SR-DRL to learn an effective policy on this environment: for $N \geqslant 20$, it performs similarly to a random policy. By contrast, our method deals efficiently with this type of problem, with noticeable performance increase over SR-DRL on all the problem sizes we tested. The adapted architecture also allows linear scaling in the number of entities, meaning that even the largest instances can be solved under 0.1 ms, similarly to SR-DRL. Overall, our experiments confirm the

| N | Random policy | SR-DRL | Split-FMODQN |
|---|---|---|---|
| 10 | -4.85 ± 0.04 | -3.52 ± 0.03 | -0.43 ± 0.01 |
| 20 | -6.64 ± 0.03 | -6.32 ± 0.03 | -3.23 ± 0.02 |
| 40 | -7.76 ± 0.02 | -7.68 ± 0.02 | -5.58 ± 0.02 |
| 80 | -8.37 ± 0.01 | -8.37 ± 0.01 | -7.17 ± 0.01 |
| 160 | -8.71 ± 0.01 | -8.72 ± 0.01 | -8.07 ± 0.01 |

Table 4.3: Results of evaluation on spinning plates over 1000 runs in the dynamic weights setting for different problem sizes $N$. Both algorithms were trained on instances of size $N = 10$. Score is the average cumulative reward with 95% confidence interval. We include results from a random policy as a baseline.

efficiency of our approach on this specific type of problems.

### 4.1.5 Discussion

In this section, we proposed to study multi-objective factored MDPs, a domain so far left unexplored, through which many real-world problems can be analyzed. We presented a family of methods that can solve different types of such problems. We distinguished these methods according to the characteristics of the target problem, and proposed benchmarks on which we showed how our approach outperformed existing algorithms.

To extend our approach to more complex domains such as resource management for air surveillance radars we need to allow parameterization of the selected action, e.g. by choosing the antenna parameters most adapted to the selected radar task. Two research directions seem crucial to us in order to develop these applications. The first one is to make our approach compatible with more complex action spaces of the type seen in parameterized MDPs (Hausknecht and Stone, 2016). This might be pursued using auto-regressive policies. Auto-regressive policies represent a relatively minor modification in policy-based algorithms (Janisch et al., 2021; Vinyals et al., 2019); however we need to combine it with value-based learning in order to maintain our method's advantage in dealing with multi-objective problems. Despite having been first proposed as a modification to DQN by (Metz et al., 2017), auto-regressive policies have seen little use in value-based methods so far, to the point of being ignored in discussions on auto-regressive structures (Fu et al., 2022); this may be due to existing auto-regressive adaptations of DQN being complex to implement properly, which warrants further research into how to simplify such adaptations. The second direction would rely on advances in multi-objective RL with non-linear scalarization, for which a few approaches have been proposed (Dornheim,

2022) even if issues remain on theoretical and practical levels (Vamplew et al., 2022).

## 4.2 Non-linear scalarization in stochastic multi-objective MDPs

Many sequential decision problems of interest are best described in relation to multiple, possibly conflicting objectives. However, most existing multi-objective reinforcement learning algorithms only apply in cases where the utility over the different criteria is linear. Yet it is well known in decision theory that in general linear scalarization poorly reflects users' preferences. Non-linear scalarization poses a number of issues in reinforcement learning, with so far little study and no definite solution. In particular, current methods struggle to deal with the most general setting of multi-objective reinforcement learning, where the environment is stochastic and the scalarization of the objectives is non-linear under the scalarization of expected returns criterion. This is an impediment in our case, since radar operators' preferences are typically better represented with non-linear utility. We approach this issue from a theoretical side and present a value-based algorithm that provides encouraging preliminary results for this type of problems.

### 4.2.1 Issues with non-linear scalarization

Current MORL methods face a number of limitations. Recent research on the subject has focused on adaptations of DQN where the preferences are given by a linear scalarization of the vector-valued reward (Abels et al., 2019; Nguyen, 2018; Mossalam et al., 2016). These methods do provide advantages: their learned Q-values can be scalarized; and they are off-policy, meaning they can learn the optimal policies for different scalarization weights over one training. Unfortunately, value-based methods restrict their solutions to deterministic policies, yet when using non-linear preferences, the optimal solution may require a stochastic policy (Chatterjee et al., 2006). Such a policy can be obtained by defining a mixture of deterministic policies; however finding the optimal mixture given a set of Pareto-optimal policies is a NP problem, which requires the use of heuristics at execution (Reymond and Nowé, 2019). Multi-objective policy search algorithms either approximate the Pareto front and suffer from the same limitation or are constrained in the form of their preferences (Uchibe and Doya, 2007; Shelton, 2001). In addition, when

the environment is stochastic, multi-objective value-based methods can even fail to find the optimal deterministic policy, as demonstrated by Vamplew et al. (2022).

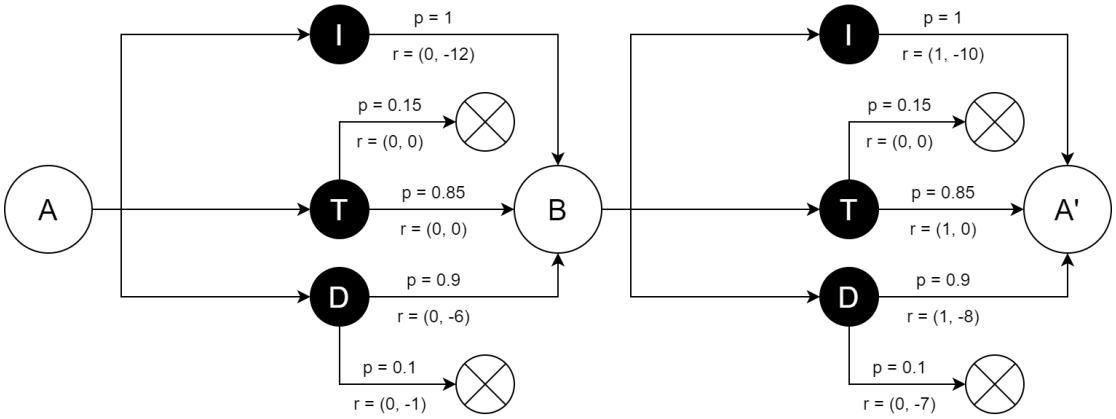


Figure 4.4: The Space Traders environment, adapted by Vamplew et al. (2022). Displayed next to each transition from a state-action pair to a new state are the reward vector $r$ and the probability $p$ of the transition.

Vamplew et al. (2022) introduce a new environment, Space Traders, consisting of 3 states, 3 actions, and 2 objectives, as depicted in fig. 4.4. The first objective reflects the success of the agent's mission, who must go from state $A$ to state $A^{'}$: the associated reward is 1 when reaching $A^{'}$, 0 elsewhere. The second objective is to the time spent to reach the goal state: the associated reward is negative for all transitions. Crucially, the environment is stochastic: for example, when taking action $T$ in state $A$, there is a probability $p = 0.85$ that the agent reaches state $B$, and a probability $p = 0.15$ that the episode terminates prematurely. Moreover, the agent's objective is not given by a linear scalarization. Instead, the thresholded lexicographic order over expected returns is used, with a threshold of 0.88 over the first objective. This means that a policy whose expected return for the first objective is above this threshold is always preferred to a policy where the threshold isn't reached. If both policies are on the same side of the threshold, the one which maximizes the last objective is preferred. Under this criterion, the optimal deterministic policy is $DI$, where $D$ is the action taken in state $A$ and $I$ is taken in state $B$, as shown in fig. 4.5.

The usual idea to apply Q-learning to a non-linear scalarization $u$ is to define the policy derived from the Q-value as $\pi(s_t) \doteq \arg\max_{a \in \mathcal{A}} u(\sum_{i=0}^{t-1} \gamma^i \boldsymbol{r}(s_i, a_i) + \mathbf{Q}(s_t, a))$, that is, to take into account the rewards accumulated since the beginning of the episode in the scalarization. However, the resulting algorithm does not reflect the SER objec-

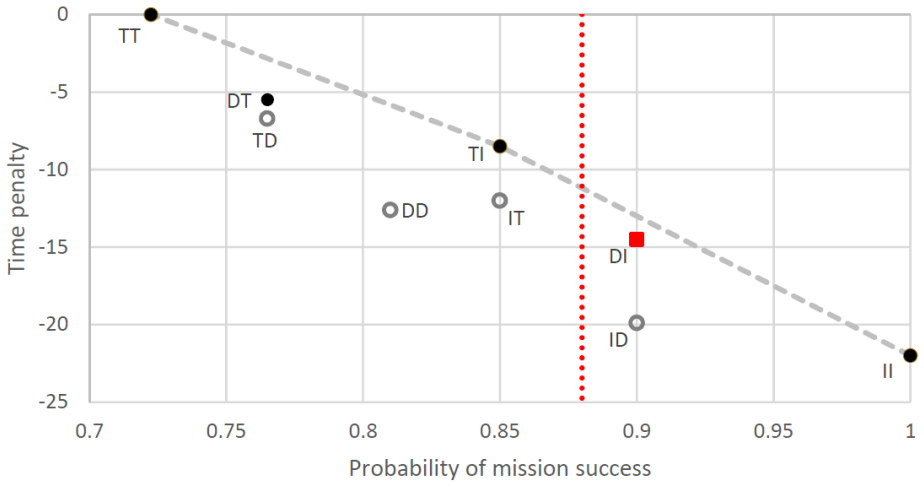


Figure 4.5: Values associated to each objective for every possible deterministic policy. The gray dotted line represents the convex front. Policies represented as a hollow point are Pareto-dominated. DI is the optimal deterministic policy when optimizing under thresholded lexicographic order with threshold 0.88, shown as the red dotted line. (Reproduced from Vamplew et al. (2022).)

tive when the environment is stochastic, which leads the multi-objective Q-learning to converge to the sub-optimal policy $ID$ in the Space Traders environment, as shown in fig. 4.6. In detail, since all transitions from $A$ to $B$ give reward 0 for the first objective, the choice of action in state $B$ is independent of the previous action and only depends on the Q-values in $B$. These Q-values are equal to the mean action values since the next state $A'$ is terminal. $D$ is then preferred as it meets the threshold (unlike $T$) and gives a higher average reward than $I$ on the second objective. From there, since the algorithm assumes $D$ to be the best action in state $B$, it then converges to chose action $I$ in state $A$ as the other two possible actions in $A$ would not allow to meet the threshold. The crux of the issue is that the algorithm chooses its action in $B$ under the assumption that the transition from $A$ to $B$ always succeeds, whereas there is actually a possibility that the episode terminates prematurely, failing to reach the threshold, if $T$ or $D$ are chosen in $A$. Therefore, there is need for methods that can account for both non-linear scalarizations and stochastic environments.

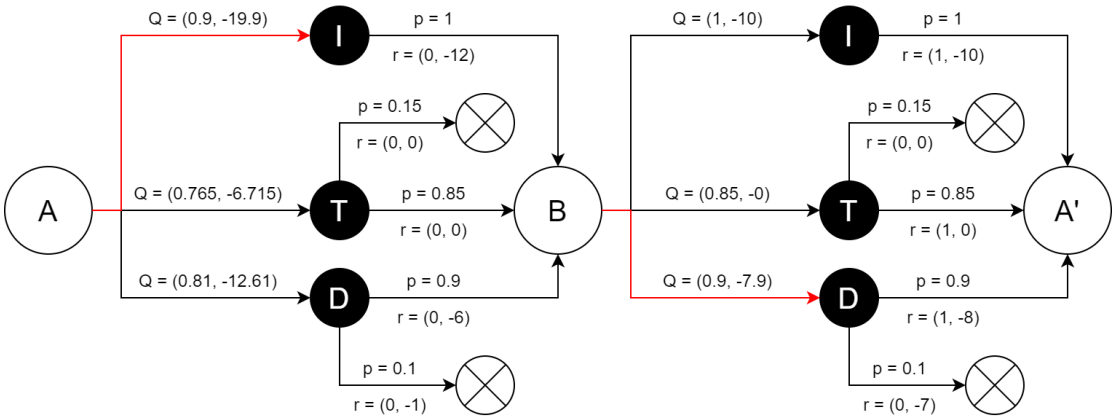


Figure 4.6: Policy to which multi-objective Q-learning converges on Space Traders. The Q-value vector learned for each state-action pair is displayed. Red arrows represent the action taken by the policy in a given state.

### 4.2.2 K-learning

#### A new notion of value function for SER

We focus on the most common formulation of MORL, where the general objective is the scalarization of expected returns (SER):

$$\max_{\pi} u\left(\mathop{\mathbb{E}}_{\mu,\pi,P}\left[\sum_{i=0}^{\infty}\gamma^{i}\boldsymbol{r}(s_i,a_i)\right]\right) \tag{4.6}$$

where $u : \mathbb{R}^n \to \mathbb{R}$ is the scalarization function and $\boldsymbol{r}(s,a) \in \mathbb{R}^n$ is a vector-valued reward, with $n$ the number of objectives. We write $\mu$ for the initial state distribution and $P$ for the transition probability function. When an agent finds itself in a given state of a stochastic environment, in order to take the action that maximizes SER with a non-linear scalarization, it needs to take into account the sum of all rewards, both already received so far and expected on the rest of the episode. Since in SER we apply the scalarization over the expectation of rewards, intuitively the rewards received so far in the episode should be re-weighted to reflect the probability of the trajectory followed so far in the episode. Here we will note a trajectory as $\tau_t = \langle s_0^\tau, a_0^\tau, s_1^\tau, ..., a_{t-1}^\tau, s_t^\tau \rangle$. Since we are looking for a value-based method, we only consider deterministic policies taking trajectories as input $\pi(\tau)$. Let $\Pi_D$ be the space of trajectory-based deterministic policies. As usual, we can define the various value functions:

$$\mathbf{V}^{\pi} \doteq \mathop{\mathbb{E}}_{\mu,\pi,P}\left[\sum_{i=0}^{\infty}\gamma^{i}\boldsymbol{r}(s_i,a_i)\right] \tag{4.7}$$

$$\mathbf{V}^{\pi}(\tau_t) \doteq \mathop{\mathbb{E}}_{\pi,P}\left[\sum_{i=t}^{\infty}\gamma^{i}\boldsymbol{r}(s_i,a_i)\,\middle|\,\tau_t\right] \tag{4.8}$$

$$\mathbf{Q}^{\pi}(\tau_t,a_t) \doteq \mathop{\mathbb{E}}_{\pi,P}\left[\sum_{i=t}^{\infty}\gamma^{i}\boldsymbol{r}(s_i,a_i)\,\middle|\,\tau_t,a_t\right] \tag{4.9}$$

As shown in section 4.2.1, if we want to find a value-based policy that optimizes for SER, we need to define an adequate notion of value function. Our insight here is that $\mathbf{Q}^{\pi}(s_t,a_t)$ is the expected future cumulative reward when following policy $\pi$ — *except in state $s_t$ where action $a_t$ is chosen.* This amounts to following a sort of *adjusted policy*, which we formally define as:

$$\rho_{\pi}(\tau_t;\hat{\tau}_i,a_i^{\hat{\tau}}) \doteq \begin{cases} a_t^{\hat{\tau}} \text{ if } t \leqslant i \text{ and } \tau_t = \hat{\tau}_t \\ \pi(\tau_t) \text{ else} \end{cases} \tag{4.10}$$

We can then rewrite equivalently $\mathbf{Q}^{\pi}(\tau_t,a_t) = \mathbb{E}\left[\sum_{i=t}^{\infty}\gamma^{i}\boldsymbol{r}(s_i,a_i)\,\middle|\,\tau_t,\rho_{\pi}(\cdot\,;\tau_t,a_t)\right]$. Note that if $\hat{\tau}_i \sim \pi$ and $a_i^{\hat{\tau}} = \pi(\hat{\tau}_i)$, then $\rho_{\pi}(\tau_t;\hat{\tau}_i,a_i^{\hat{\tau}})$ simplifies to $\pi(\tau_t)$.

In order to take into account the rewards received so far, we modify this formulation of the Q-value to remove the conditioning on the current trajectory $\tau_t$ and take the expectation of the sum of rewards over whole episodes. We call this the K-value, which we formally define given a policy $\pi$ as:

$$\mathbf{K}^{\pi}(\tau_t,a_t) \doteq \mathop{\mathbb{E}}_{\mu,P}\left[\sum_{i=0}^{\infty}\gamma^{i}\boldsymbol{r}(s_i,a_i)\,\middle|\,\rho_{\pi}(\cdot\,;\tau_t,a_t)\right] \tag{4.11}$$

The greedy policy associated to a K-value $\mathbf{K}$ is naturally defined as maximizing the scalarization of this K-value:

$$\pi_u^{\mathbf{K}}(\tau) \doteq \arg\max_{a\in\mathcal{A}} u(\mathbf{K}(\tau,a)) \tag{4.12}$$

Furthermore, the optimal K-value for deterministic policies is:

$$\mathbf{K}_u^{*}(\tau_t,a_t) \doteq \arg\max_{\mathbf{K}} \max_{\pi\in\Pi_D} u(\mathbf{K}^{\pi}(\tau_t,a_t)) \tag{4.13}$$

Indeed, if we apply $\pi_u^{\mathbf{K}_u^*}$ (thereafter simplified as $\pi_u^*$), then we always have $\tau_t \sim \pi_u^*$, and $a_t$ is chosen by $\pi_u^*$, so $\rho_{\pi_u^*}(\cdot\,; \tau_t, a_t)$ simplifies to $\pi_u^*$, so that $\pi_u^*$ is optimal for the original objective.

The K-value forms the basis of the K-learning algorithm which we present later in this chapter. However, it is worth exploring why it is a useful generalization of existing value function definitions.

### Links to existing approaches

We first show that the K-value can be rewritten as a function of $\mathbf{Q}^\pi$ and $\mathbf{V}^\pi$. In all our demonstrations we assume a discrete state space. First we note that $\mathbf{K}^\pi$ can be separated into two terms based on the probability that the initial state is the trajectory's first state $s_0^\tau$:

$$\begin{aligned}\mathbf{K}^\pi(\tau_t, a_t) = &\sum_{s \neq s_0^\tau} \mu(s) \mathop{\mathbb{E}}_{P}\left[\sum_{i=0}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, s_0 = s,\, \rho_\pi(\cdot\,; \tau_t, a_t)\right] \\ &+ \mu(s_0^\tau) \mathop{\mathbb{E}}_{P}\left[\sum_{i=0}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, s_0 = s_0^\tau,\, \rho_\pi(\cdot\,; \tau_t, a_t)\right] \end{aligned} \tag{4.14}$$

In the first term, we can see that since $s \neq s_0^\tau$, then $\rho_\pi(\cdot\,; \tau_t, a_t)$ simplifies to $\pi$, so that this term can be expressed as a function of $\mathbf{V}^\pi$:

$$\sum_{s \neq s_0^\tau} \mu(s) \mathop{\mathbb{E}}_{P}\left[\sum_{i=0}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, s_0 = s,\, \rho_\pi(\cdot\,; \tau_t, a_t)\right] = \sum_{s \neq s_0^\tau} \mu(s) \mathop{\mathbb{E}}_{P}\left[\sum_{i=0}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, s_0 = s,\, \pi\right] \tag{4.15}$$

$$= \sum_{s \neq s_0^\tau} \mu(s) \mathbf{V}^\pi(s) \tag{4.16}$$

$$= \sum_{s} \mu(s) \mathbf{V}^\pi(s) - \mu(s_0^\tau) \mathbf{V}^\pi(s_0^\tau) \tag{4.17}$$

$$= \mathbf{V}^\pi - \mu(s_0^\tau) \mathbf{V}^\pi(s_0^\tau) \tag{4.18}$$

The second term can be rewritten as $\mu(s_0^\tau) \mathbf{K}_0^\pi(\tau_t, a_t)$, where for $j \leqslant t$:

$$\mathbf{K}_j^\pi(\tau_t, a) \doteq \mathop{\mathbb{E}}_{P}\left[\sum_{i=j}^{\infty}\gamma^{i-j}\boldsymbol{r}(s_i,a_i)\,\middle|\,\tau_j,\ \rho_\pi(\cdot\,;\tau_t,a)\right] \tag{4.19}$$

This value can then be written recursively for $j < t$:

$$\mathbf{K}_j^\pi(\tau_t,a_t) = \mathbb{E}[\boldsymbol{r}(s_j^\tau,a_j^\tau)] + \gamma\sum_s P(s|s_j^\tau,a_j^\tau)\mathop{\mathbb{E}}_{P}\left[\sum_{i=j+1}^{\infty}\gamma^{i-j-1}\boldsymbol{r}(s_i,a_i)\,\middle|\,\langle\tau_j,a_j^\tau,s\rangle,\ \rho_\pi(\cdot\,;\tau_t,a_t)\right] \tag{4.20}$$

$$\begin{aligned}= \mathbb{E}[\boldsymbol{r}(s_j^\tau,a_j^\tau)] &+ \gamma\sum_{s\neq s_{j+1}^\tau} P(s|s_j^\tau,a_j^\tau)\mathop{\mathbb{E}}_{P}\left[\sum_{i=j+1}^{\infty}\gamma^{i-j-1}\boldsymbol{r}(s_i,a_i)\,\middle|\,\langle\tau_j,a_j^\tau,s\rangle,\ \rho_\pi(\cdot\,;\tau_t,a_t)\right]\\ &+ \gamma P(s_{j+1}^\tau|s_j^\tau,a_j^\tau)\mathop{\mathbb{E}}_{P}\left[\sum_{i=j+1}^{\infty}\gamma^{i-j-1}\boldsymbol{r}(s_i,a_i)\,\middle|\,\langle\tau_j,a_j^\tau,s_{j+1}^\tau\rangle,\ \rho_\pi(\cdot\,;\tau_t,a_t)\right]\end{aligned} \tag{4.21}$$

$$= \mathbb{E}[\boldsymbol{r}(s_j^\tau,a_j^\tau)] + \gamma\sum_{s\neq s_{j+1}^\tau} P(s|s_j^\tau,a_j^\tau)\mathbf{V}^\pi(\langle\tau_j,a_j^\tau,s\rangle) + \gamma P(s_{j+1}^\tau|s_j^\tau,a_j^\tau)\mathbf{K}_{j+1}^\pi(\tau_t,a_t) \tag{4.22}$$

$$= \mathbb{E}[\boldsymbol{r}(s_j^\tau,a_j^\tau)] + \gamma\mathop{\mathbb{E}}_{s\sim P}\mathbf{V}^\pi(\langle\tau_j,a_j^\tau,s\rangle) + \gamma P(s_{j+1}^\tau|s_j^\tau,a_j^\tau)(\mathbf{K}_{j+1}^\pi(\tau_t,a_t) - \mathbf{V}^\pi(\tau_{j+1})) \tag{4.23}$$

$$= \mathbf{Q}^\pi(\tau_j,a_j^\tau) + \gamma P(s_{j+1}^\tau|s_j^\tau,a_j^\tau)(\mathbf{K}_{j+1}^\pi(\tau_t,a_t) - \mathbf{V}^\pi(\tau_{j+1})) \tag{4.24}$$

Now, noticing that $\mathbf{K}_t^\pi(\tau_t,a_t) = \mathbf{Q}^\pi(\tau_t,a_t)$, we can apply this relation recursively (replacing $a_t$ with $a_t^\tau$ for notational convenience):

$$\mathbf{K}_0^\pi(\tau_t,a_t^\tau) = \mathbf{Q}^\pi(\tau_0,a_0^\tau) + \sum_{i=1}^{t}\gamma^i P(\tau_i)(\mathbf{Q}^\pi(\tau_i,a_i^\tau) - \mathbf{V}^\pi(\tau_i)) \tag{4.25}$$

where $P(\tau_i) \doteq \prod_{k=0}^{i-1} P(s_{k+1}^\tau|s_k^\tau,a_k^\tau)$.

Equations 4.18 and 4.25 can now be used to simplify equation 4.14:

$$\mathbf{K}^{\pi}(\tau_t, a_t^{\tau}) = \mathbf{V}^{\pi} + \mu(s_0^{\tau}) \left( \mathbf{Q}^{\pi}(\tau_0, a_0^{\tau}) - \mathbf{V}^{\pi}(\tau_0) + \sum_{i=1}^{t} \gamma^i P(\tau_i)(\mathbf{Q}^{\pi}(\tau_i, a_i^{\tau}) - \mathbf{V}^{\pi}(\tau_i)) \right) \tag{4.26}$$

$$= \mathbf{V}^{\pi} + \sum_{i=0}^{t} \gamma^i P_{\mu}(\tau_i)(\mathbf{Q}^{\pi}(\tau_i, a_i^{\tau}) - \mathbf{V}^{\pi}(\tau_i)) \tag{4.27}$$

where $P_{\mu}(\tau_i) \doteq \mu(s_0^{\tau})P(\tau_i)$. As we can see, eq. (4.27) shows that $\mathbf{K}^{\pi}(\tau_t, a_t^{\tau})$ can be expressed as the sum of the policy value $\mathbf{V}^{\pi}$ and of the advantages associated to the actions taken in the trajectory $\tau_t$, weighted by the probability of reaching these actions as dictated by the environment's stochasticity.

From here, we can establish equivalences between the notion of K-value and the more common uses of the Q-value in multi-objective settings. These equivalences show up when either of the conditions that brought us to define the K-value are absent, that is: when the scalarization is linear or when the environment is deterministic.

First we consider the case where the scalarization $u$ is linear. We can then isolate the Q-value corresponding to action $a_t$ when scalarizing $\mathbf{K}^{\pi}(\tau_t, a_t)$:

$$\begin{aligned} u(\mathbf{K}^{\pi}(\tau_t, a_t)) = u(\mathbf{V}^{\pi} + \sum_{i=1}^{t-1} \gamma^i P_{\mu}(\tau_i)(\mathbf{Q}^{\pi}(\tau_i, a_i^{\tau}) - \mathbf{V}^{\pi}(\tau_i)) - \gamma^t P_{\mu}(\tau_t)\mathbf{V}^{\pi}(\tau_t)) \\ + \gamma^t P_{\mu}(\tau_t) u(\mathbf{Q}^{\pi}(\tau_t, a_t)) \end{aligned} \tag{4.28}$$

As a consequence, in this case, maximizing the scalarized K-value is equivalent to maximizing the scalarized Q-value, as has been done previously (Mossalam et al., 2016; Abels et al., 2019):

$$\arg\max_{a_t} u(\mathbf{K}^{\pi}(\tau_t, a_t)) = \arg\max_{a_t} u(\mathbf{Q}^{\pi}(\tau_t, a_t)) \tag{4.29}$$

Now we consider the case where the environment is deterministic. If the transitions are deterministic, since $\tau_t$ has a non-zero probability of occurring, then equation 4.22 simplifies to:

$$\mathbf{K}_j^{\pi}(\tau_t, a_t) = \mathbb{E}[\boldsymbol{r}(s_j^{\tau}, a_j^{\tau})] + \gamma \mathbf{K}_{j+1}^{\pi}(\tau_t, a_t) \tag{4.30}$$

Since $\mu$ is also deterministic, $s_0^\tau$ is the only possible initial state, so $\mathbf{K}^\pi(\tau_t, a_t) = \mathbf{K}_0^\pi(\tau_t, a_t)$. Applying equation 4.30 recursively, we get:

$$u(\mathbf{K}^\pi(\tau_t, a_t)) = u\left(\sum_{i=0}^{t-1} \gamma^i \, \mathbb{E}[\boldsymbol{r}(s_i^\tau, a_i^\tau)] + \gamma^t \mathbf{Q}^\pi(\tau_t, a_t)\right) \tag{4.31}$$

which is equivalent to the common solution when rewards are deterministic (Vamplew et al., 2022).

In both cases, if the MDP is fully observable, since the transition function is Markovian, the Q-value depends only on the most recent state, i.e. $\mathbf{Q}^\pi(\tau_t, a_t) = \mathbf{Q}^\pi(s_t, a_t)$.

### Optimality operator for the K-value

Equation 4.27 could be used as the basis of a control algorithm, where the K-value could be approximated based on a learned Q-value and state value. However, this would require knowing or learning the initialization and transition probabilities of the MDP, which can be difficult to scale up. Instead, we set out to find an algorithm which learns to approximate the K-value itself. To do so, we first need to define $H(h \,|\, \tau_t)$ the improper probability of deviating from trajectory $\tau_t$ after $h \in [0, t]$ steps while following policy $\rho_\pi(\cdot\,; \tau_t, a_t^\tau)$ (when the trajectory doesn't deviate, we consider that $h = t$):

$$H(h \,|\, \tau_t) \doteq prob(\forall i < h, s_i = s_i^\tau \wedge (h = t \vee s_h \neq s_h^\tau) \,|\, s_0 \sim \mu, \forall j \; s_j \sim P(\cdot \,|\, s_{j-1}, a_{j-1}^\tau)) \tag{4.32}$$

Note that we have $h = 0$ if $s_0 \neq s_0^\tau$. The K-value can now be expressed as:

$$\mathbf{K}^\pi(\tau_t, a_t^\tau) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, s_i \sim (\mu, P), a_i = \rho_\pi(\cdot\,; \tau_t, a_t^\tau)\right] \tag{4.33}$$

$$= \mathbb{E}\left[\sum_{i=0}^{h-1} \gamma^i \boldsymbol{r}(s_i, a_i) + \sum_{i=h}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, h \sim H(\cdot\,; \tau_t), s_i \sim (\mu, P), a_i = \rho_\pi(\cdot\,; \tau_t, a_t^\tau)\right] \tag{4.34}$$

$$= \mathbb{E}\left[\sum_{i=0}^{h-1} \gamma^i \boldsymbol{r}(s_i^\tau, a_i^\tau) + \gamma^h \mathbf{Q}^\pi(\tau^{s_h}, a_h) \,\middle|\, h \sim H(\cdot\,; \tau_t), s_h \sim (\mu, P), a_h = \rho_\pi(\tau^{s_h}\,; \tau_t, a_t^\tau)\right] \tag{4.35}$$

where $\tau^{s_h} = \langle s_0^\tau, a_0^\tau, s_1^\tau, ..., a_{h-1}^\tau, s_h \rangle$. Note that we have $a_h = a_t^\tau$ if $h = t$ and $s_h = s_h^\tau$ (i.e. there was no deviation), and $a_h \sim \pi$ otherwise. This way, the probability function $H$ allows us to rewrite the K-value as an expectation over the sum of rewards up to a deviation from the trajectory $\tau_t$ at step $h$, plus the Q-value after this deviation.

Recalling that $\pi_u^{\mathbf{K}}$ is the greedy policy w.r.t. value $\mathbf{K}$ and scalarization $u$, our proposed optimality operator, defined over a value space $\mathcal{Q}^2$, where $\mathcal{Q} = \bigcup_{n=1}^{\infty}((\mathbb{R}^m)^{\mathcal{S}\times\mathcal{A}})^n$, $m$ is the number of objectives and $n$ the length of the trajectory, is therefore $\mathcal{T}(\mathbf{K}, \mathbf{Q}) \doteq (\mathcal{T}\mathbf{K}, \mathcal{T}\mathbf{Q})$ for $(\mathbf{K}, \mathbf{Q}) \in \mathcal{Q}^2$, where:

$$\mathcal{T}\mathbf{K}(\tau_t, a_t^\tau) \doteq \mathbb{E}\left[\sum_{i=0}^{h-1} \gamma^i \boldsymbol{r}(s_i^\tau, a_i^\tau) + \gamma^h \mathbf{Q}(\tau^{s_h}, a_h) \,\middle|\, h \sim H(\cdot\,; \tau_t), s_h \sim (\mu, P), a_h = \rho_{\pi_u^{\mathbf{K}}}(\tau^{s_h}\,; \tau_t, a_t^\tau)\right] \tag{4.36}$$

$$\mathcal{T}\mathbf{Q}(\tau_t, a_t^\tau) \doteq \mathbb{E}\left[\boldsymbol{r}(s_t^\tau, a_t^\tau) + \gamma \mathbf{Q}(\tau^{s_{t+1}}, a_{t+1}) \,\middle|\, s_{t+1} \sim P, a_{t+1} = \pi_u^{\mathbf{K}}(\tau^{s_{t+1}})\right] \tag{4.37}$$

In order to prove the convergence of this operator, we define a value metric:

$$d((\mathbf{K}, \mathbf{Q}), (\mathbf{K}', \mathbf{Q}')) = \max\{\|\mathbf{K} - \mathbf{K}'\|, \|\mathbf{Q} - \mathbf{Q}'\|\} \tag{4.38}$$

where $\|\mathbf{Q}\| = \max_{\tau,a} \|\mathbf{Q}(\tau, a)\|^\infty$ for $\mathbf{Q} \in \mathcal{Q}$. We now prove that the operator we defined is a contraction on the metric space $(\mathcal{Q}, d)$. Let $K_{(i)}$ be the value of $\mathbf{K}$ for the $i$th objective. Then:

$$\|\mathcal{T}\mathbf{K} - \mathcal{T}\mathbf{K}'\| = \max_{i,\tau_t,a_t^\tau} \left|\mathcal{T}K_{(i)}(\tau_t, a_t^\tau) - \mathcal{T}K'_{(i)}(\tau_t, a_t^\tau)\right| \tag{4.39}$$

$$= \max_{i,\tau_t,a_t^\tau} \left|\mathbb{E}\left[\gamma^h Q_{(i)}(\tau^{s_h}, a_h) - \gamma^h Q'_{(i)}(\tau^{s_h}, a_h) \,\middle|\, H, \mu, P, \rho_{\pi_u^{\mathbf{K}}}(\cdot\,; \tau_t, a_t^\tau)\right]\right| \tag{4.40}$$

$$\leqslant \max_{i,\tau_t} \max_{h,s_h,a_h} \left|\gamma^h Q_{(i)}(\tau^{s_h}, a_h) - \gamma^h Q'_{(i)}(\tau^{s_h}, a_h)\right| \tag{4.41}$$

$$\leqslant \max_{i,h,\tau,a} \left|\gamma^h Q_{(i)}(\tau, a) - \gamma^h Q'_{(i)}(\tau, a)\right| \tag{4.42}$$

$$\leqslant \max_{i,\tau,a} \left|Q_{(i)}(\tau, a) - Q'_{(i)}(\tau, a)\right| \tag{4.43}$$

$$= \|\mathbf{Q} - \mathbf{Q}'\| \tag{4.44}$$

Similarly, we can show that our operator constitutes a contraction over $\mathbf{Q}$:

$$\|\mathcal{T}\mathbf{Q} - \mathcal{T}\mathbf{Q}'\| = \max_{i,\tau_t,a_t^\tau} \left|\mathcal{T}Q_{(i)}(\tau_t, a_t^\tau) - \mathcal{T}Q'_{(i)}(\tau_t, a_t^\tau)\right| \tag{4.45}$$

$$= \max_{i,\tau_t,a_t^\tau} \left|\mathbb{E}\left[\gamma Q_{(i)}(\tau^{s_{t+1}}, a_{t+1}) - \gamma Q'_{(i)}(\tau^{s_{t+1}}, a_{t+1}) \,\middle|\, P, \pi_u^{\mathbf{K}}\right]\right| \tag{4.46}$$

$$\leqslant \gamma \max_{i,\tau_t,a_t^\tau} \max_{s_{t+1},a_{t+1}} \left|Q_{(i)}(\tau^{s_{t+1}}, a_{t+1}) - Q'_{(i)}(\tau^{s_{t+1}}, a_{t+1})\right| \tag{4.47}$$

$$\leqslant \gamma \max_{i,\tau,a} \left|Q_{(i)}(\tau, a) - Q'_{(i)}(\tau, a)\right| \tag{4.48}$$

$$= \gamma \|\mathbf{Q} - \mathbf{Q}'\| \tag{4.49}$$

By using the two previous inequalities, we see that $\mathcal{T}^2(\mathbf{K}, \mathbf{Q})$ is a contraction:

$$d(\mathcal{T}^2(\mathbf{K}, \mathbf{Q}), \mathcal{T}^2(\mathbf{K}', \mathbf{Q}')) = \max\{\|\mathcal{T}^2\mathbf{K} - \mathcal{T}^2\mathbf{K}'\|, \|\mathcal{T}^2\mathbf{Q} - \mathcal{T}^2\mathbf{Q}'\|\} \tag{4.50}$$

$$\leqslant \max\{\|\mathcal{T}\mathbf{Q} - \mathcal{T}\mathbf{Q}'\|, \gamma\|\mathcal{T}\mathbf{Q} - \mathcal{T}\mathbf{Q}'\|\} \tag{4.51}$$

$$\leqslant \max\{\gamma\|\mathbf{Q} - \mathbf{Q}'\|, \gamma^2\|\mathbf{Q} - \mathbf{Q}'\|\} \tag{4.52}$$

$$= \gamma\|\mathbf{Q} - \mathbf{Q}'\| \tag{4.53}$$

$$\leqslant \gamma \max\{\|\mathbf{K} - \mathbf{K}'\|, \|\mathbf{Q} - \mathbf{Q}'\|\} \tag{4.54}$$

$$= \gamma d((\mathbf{K}, \mathbf{Q}), (\mathbf{K}', \mathbf{Q}')) \tag{4.55}$$

We then show that the fixed point of this operator is the optimal value for $(\mathbf{K}, \mathbf{Q})$.

$$\mathcal{T}\mathbf{Q}_u^*(\tau_t, a_t^\tau) = \mathbb{E}\left[\boldsymbol{r}(s_t^\tau, a_t^\tau) + \gamma \mathbf{Q}_u^*(\tau^{s_{t+1}}, a_{t+1}) \,|\, s_{t+1} \sim P, a_{t+1} = \pi_u^*(\tau^{s_{t+1}})\right] \tag{4.56}$$

$$= \mathbb{E}\left[\boldsymbol{r}(s_t^\tau, a_t^\tau) + \gamma \mathop{\mathbb{E}}_{\pi_u^*, P}\left[\sum_{i=t+1}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, \tau^{s_{t+1}}, a_{t+1}\right] \,\middle|\, s_{t+1} \sim P, a_{t+1} = \pi_u^*(\tau^{s_{t+1}})\right] \tag{4.57}$$

$$= \mathop{\mathbb{E}}_{\pi_u^*, P}\left[\boldsymbol{r}(s_t^\tau, a_t^\tau) + \gamma \sum_{i=t+1}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, \tau^t, a_t^\tau\right] \tag{4.58}$$

$$= \mathop{\mathbb{E}}_{\pi_u^*, P}\left[\sum_{i=t}^{\infty} \gamma^i \boldsymbol{r}(s_i, a_i) \,\middle|\, \tau_t, a_t^\tau\right] \tag{4.59}$$

$$= \mathbf{Q}_u^*(\tau_t, a_t^\tau) \tag{4.60}$$

Similarly, the fixed point for $\mathbf{K}$ is $\mathbf{K}^*_u$, which can be shown be replacing $\mathbf{K}^\pi$, $\mathbf{Q}^\pi$, $\pi$ by $\mathbf{K}^*_u$, $\mathbf{Q}^*_u$, $\pi^*_u$ respectively in equation 4.35.

This concludes our proof that $\mathcal{T}(\mathbf{K}, \mathbf{Q})$ is an optimality operator.

### Practical algorithm

Based on our optimality operator, we can define an algorithm that learns both $\mathbf{Q}$ and $\mathbf{K}$ concurrently. This method, which we call K-learning, is outlined below. We write $\mathbb{1}_T(s)$ for the characteristic function indicating that state $s$ is terminal.

**Algorithm 1** K-learning

**Require:** learning rate $\alpha$, exploration probability $\epsilon$
  **repeat** $N$ **times**
    Generate a trajectory $\tau_t$ using an $\epsilon$-greedy policy over $u(\mathbf{K})$
    **for** $i$ from 0 to $t-2$ **do**
      $\mathbf{Q}(\tau_i, a^\tau_i) \leftarrow (1-\alpha)\mathbf{Q}(\tau_i, a^\tau_i) + \alpha(\boldsymbol{r}_i + \gamma\mathbf{Q}(\tau_{i+1}, \pi^{\mathbf{K}}_u(\tau_{i+1})))$
    **end for**
    $\mathbf{Q}(\tau_{t-1}, a^\tau_{t-1}) \leftarrow (1-\alpha)\mathbf{Q}(\tau_{t-1}, a^\tau_{t-1}) + \alpha\boldsymbol{r}_{t-1}$
    $s_0 \sim \mu(\cdot)$
    $h \leftarrow -1$
    **for** $i$ from 0 to $t-1$ **do**
      **if** $h = -1$ and $s_i \neq s^\tau_i$ **then**
        $h \leftarrow i$
      **end if**
      **if** $h > -1$ **then**
        $\mathbf{K}(\tau_i, a^\tau_i) \leftarrow (1-\alpha)\mathbf{K}(\tau_i, a^\tau_i) + \alpha(\sum_{j=0}^{h-1}\gamma^j\boldsymbol{r}_j + \gamma^h(1-\mathbb{1}_T(s_h))\mathbf{Q}(\tau_h, \pi^{\mathbf{K}}_u(\tau_h)))$
      **else**
        $\mathbf{K}(\tau_i, a^\tau_i) \leftarrow (1-\alpha)\mathbf{K}(\tau_i, a^\tau_i) + \alpha(\sum_{j=0}^{i-1}\gamma^j\boldsymbol{r}_j + \gamma^i\mathbf{Q}(\tau_i, a^\tau_i))$
        $s_{i+1} \sim P(\cdot \mid s_i, a^\tau_i)$
        $r_i \sim R(\cdot \mid s_i, a^\tau_i)$
      **end if**
    **end for**
  **end**

$\mathbf{Q}$ is learned in a very similar way to Q-learning. $\mathbf{K}$, on the other hand, is learned via a two-step process. We first generate a reference trajectory $\tau_t$ (which is used to learn $\mathbf{Q}$). Then, a second trajectory is generated, that attempts to replicate the reference trajectory by selecting the same actions as it. $\mathbf{K}$ is updated based on the rewards received while on the reference trajectory and the Q-value thereafter. If at any step $h$ the new trajectory deviates from the reference, only the rewards up to $h$ are considered. This way we approximate the expectation over $H$ present in the optimality operator.

However, this algorithm has many disadvantages. First, it only applies to discrete

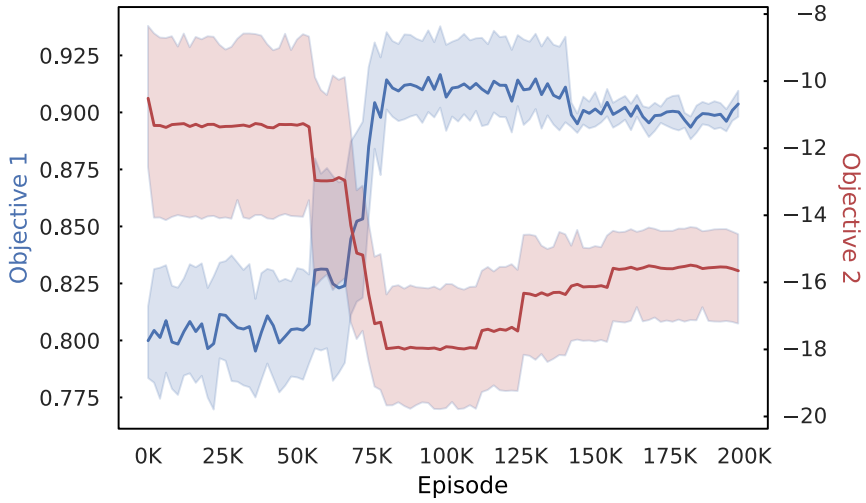


Figure 4.7: Mean and 95% confidence interval of the returns on both objectives for Space Traders over 10 training runs of K-learning.

state spaces (even though it could potentially be adapted to continuous state spaces by using approximate state comparisons). It is also inapplicable in infinite-horizon settings, with no clear equivalent. Moreover, the learned K-value could have high variance, especially the more stochastic the environment is, which would make it non-trivial to learn. Additionally, both the K-value and Q-value in this scheme depend on the whole trajectory instead of the last state: for tabular implementations, their representation would take memory space exponential in the number of states, while if using neural networks for function approximation, we would probably have to use RNNs, which would make training even more difficult. Finally, one limitation that we had to accept as soon as restricted ourselves to value-based methods is that we can only obtain deterministic policies, even though the optimal policy under SER can be stochastic.

### 4.2.3 Experiments

We test our method on the Space Traders environment. In order to do this, we implement a tabular version of K-learning, which is possible for this environment given its small state space. In our experiments, we set the learning rate to 0.001 and epsilon to 0.5 and generate $N = 100{,}000$ reference trajectories, which means training lasts for 200,000 episodes. Similarly to Vamplew et al. (2022), we apply the thresholded lexicographic order on the two objectives, with a threshold of 0.88 on the first one. Therefore, the optimal deterministic policy we expect to converge to is $DI$, with a mean return of 0.9

and $-14.5$ on the first and second objectives respectively.

The learning curves for each objective are shown in fig. 4.7. As can be seen on the left, the agent reliably converges to a policy that respects the threshold on the first objective with a mean return of 0.9. On the other hand, the mean return over 10 episodes for the second objective is $-15.66$, slightly offset from the optimal value $-14.5$, as the agent will sometimes converge to policy $ID$ instead, whose mean return on the second objective is $-19.9$.

Overall, we experimentally confirm that K-learning can converge to the optimal deterministic policy in stochastic environments even when the scalarization function is non-linear.

# Perspectives and conclusion

Throughout this dissertation, we have explored a variety of perspectives for future multi-function radar resource management techniques through the prism of reinforcement learning.

In chapter 3, we developed practical applications for two components of resource management: task scheduling and active tracking dwell optimization. In our study of radar task scheduling, we built upon previous work that applied Monte Carlo Tree Search (MCTS) to find low-cost schedules. Through a reformulation of the scheduling problem, we highlighted the crucial role of the choice of transition function when modeling the problem as an MDP, and implemented an efficient transition function. The use of this transition function in MCTS required a number of adjustments inspired by branch-and-bound algorithms. We tested this MCTS variant as well as a new heuristic allowed by our transition function, and demonstrated improved performance compared to previous approaches: in particular, our algorithms scale better as the size of the scheduling problem increases and uncover lower-cost solutions faster than existing methods, even when these implemented AlphaZero-style learning on top of MCTS.

We see several avenues of improvement on the topic of radar scheduling. In particular, the problem formulation we studied is actually relatively restricted. For example, it doesn't allow for dwell interleaving. During a dwell, the antenna performs an emitting phase and a receiving phase; between these two phases it has to wait for the signal to travel to the target and back. Dwell interleaving takes advantage of this idle phase to schedule the emitting or receiving phase of another dwell in the meantime. This way antenna utilization can be further increased. Another idea is to consider more variable parameters for the dwells. In our dissertation, we only attempted to set the execution time of each dwell, yet the scheduler could also be tasked with dynamically adapting

more dwell parameters, especially the duration of the dwells, which we considered fixed in our work. This could attenuate the difficulties posed by the separate optimization of each track. While these extensions of the problem have already been studied and heuristics have been developed for them, adapting MCTS techniques to these cases has the potential to increase the performance and adaptivity of radar schedulers.

There is also room for improvement for radar scheduling in terms of reinforcement learning techniques. One clear extension is to apply algorithms like AlphaZero, which combine MCTS with the learning of a value network and a policy network. These networks are trained with the experience gathered by execution of the tree search, and are used to guide the search, which leads to the mutual optimizing of the search process and the networks. This learning procedure could also help adapt the search to different distributions of tasks. Although it did not provide improvement for the version of the scheduling problem we studied, AlphaZero-style methods could prove useful in addressing more intricate scheduling problems as outlined above. In addition to model-based methods like AlphaZero, combinatorial optimization problems have also been addressed with model-free reinforcement learning algorithms coupled with neural encoders like pointer networks and transformers, with promising results (Mazyavkina et al., 2020). This perspective has so far seen little study in the case of radar scheduling, apart from (George et al., 2022). Yet it is compelling as the resulting algorithms could be able to output high-quality solutions at a fraction of the run time required by search methods, while still allowing for adaptation to different task distributions.

However, all methods focused solely on radar scheduling suffer from a more practical disadvantage in that they require to define the costs associated to each task in order to evaluate the possible schedules. These costs can be derived from the priority levels of the dwells as determined in the prioritization phase, yet this prioritization introduces an intermediate objective that can be difficult to align with the desired radar behavior correctly in practice. This limitation could be circumvented by adopting holistic approaches to radar resource management, as we discuss below.

Our second area of study in chapter 3 was active tracking dwell optimization. We observed that despite the beam agility of electronically scanned array antennae, the beamwidth parameter was underutilized by current resource management techniques as it was usually fixed to a constant value on which the algorithm for update rate selection depended. Given the interplay between beamwidth, revisit interval, and dwell duration

in determining the quality of the measurements required to maintain tracks, we set out to apply model-free reinforcement learning to obtain an agent that could dynamically adapt these parameters. We implemented a simulator that reflected the high-level dynamics of the interactions between the radar and its environment and that could interface with reinforcement learning algorithms. Having restricted ourselves to single-target tracking on radars with rotating antennae, we showed that the agent we trained boasted both a lower track loss rate and lower antenna time utilization compared to a fixed-beamwidth method, confirming the benefits for beamwidth adaptivity.

This single-target agent fits into the currently common architectural choice of per-track dwell optimization. However, a compelling alternative is to develop a multi-target agent that would manage the allocation of antenna time between the different tracks established by the system while also adjusting the same dwell parameters as previously, especially beamwidth. This method could be extended to handle all the radar's tasks, that would mainly include tracks and search tasks. Such an extension would represent a holistic approach, which unlike current architectures would not separate between dwell parameter selection, task prioritization, and task scheduling, and would therefore avoid their pitfalls, where a lack of coordination between these different components may lead to ungraceful degradation in situations of overload. This framework has previously been studied in terms of restless multi-armed bandits, but was hindered by the computational limitations of the algorithms used to solve this problem, which were based on dynamic programming. To apply modern reinforcement techniques, we not only need to deal with the combinatorial aspect of the problem, as the radar's tasks routinely number in the hundreds and the number of tracks varies over time; it is also desirable to introduce multi-objective learning in order for the agent to be able to adapt to the different levels of priority between scenarios and between tasks. To develop the tools necessary to this vision, we had to address more fundamental questions in reinforcement learning.

In chapter 4, we investigated two reinforcement learning topics which hold high potential for radar resource management: the combination of multi-objective learning with factored MDPs, and the use of non-linear scalarizations in multi-objective reinforcement learning (MORL).

On the topic of MORL in combinatorial problems, we identified that factored MDPs as a suitable framework for the optimization of strategies in temporally-extended prob-

lems with combinatorial state and action spaces, as the ones faced by multi-function radars. We modified the formulation of factored MDPs to include a multi-objective aspect, where each of the problem's entities have an associated reward and weight, from which the general objective is derived as a linear, weighted scalarization of these rewards. We leverage recent advances in the use of graph neural networks (GNN) to deal with the combinatorial aspect of the problem, and combine this with a multi-objective version of DQN. Starting from the most general case, we detail how the structure of the network can be simplified based on the characteristics of the problem. Finally, on two custom benchmark environments chosen for their similarity to the our radar use case, we show that our algorithms can handle problems with variable numbers of entities and rewards and that they compare favorably to an existing GNN-based method, with better performance and generalization to dynamic reward weights.

Although encouraging, this approach inherit a number of disadvantages from its value-based aspect. The most pressing issue is that in the most general case, our method has to compute the Q-value of each objective for each possible action: when the number of objectives and the number of actions are both tied to the number of entities in the problem, our method suffers from a quadratic time complexity at each action selection step, which limits its scalability to radar resource management problems. Moreover, unlike the policy-based method against which we made comparisons, our value-based method cannot handle complex action spaces: in particular, it is not suitable for parameterized actions, which policy-based algorithms can manage trivially via auto-regressive networks. This is especially relevant if we have the multi-target scenario in mind, since we want our agent to perform both task selection (choosing a track to update at the present step) and task parameterization (setting the parameters of dwell used to update the track). The last limitation is shared with existing multi-objective value-based methods: only linear scalarizations can be used, since non-linear scalarizations break a number of key assumptions in reinforcement learning. Again, this has implications in terms of applicability to radar systems, as the radar's mission is often specified via strict priority levels: for example, it may be required that the proportion of time spent on search tasks remain above a given threshold to guarantee adequate surveillance even during overload; a dangerous target may be given absolute priority over all the other less threatening targets. These preferences can be difficult, sometimes impossible, to encode using linear scalarizations.

The limitations of linear scalarizations motivate our last contribution, centered on the use of MORL with non-linear scalarizations. It had been established previously that one of the most common approaches to MORL in the scalarization of expected returns (SER) setting, multi-objective DQN, could fail to converge to the optimal solution in stochastic environments under a non-linear scalarization. Given the inherently stochastic nature of the radar's observations, solving this issue could open new perspectives for applications of reinforcement learning in radar systems. By virtue of being off-policy, value-based methods can adapt to varying reward weights and so have been a focus in MORL. Therefore, we attempt to develop a value-based algorithm adapted to non-linear scalarization. To do so we extend the classic notion of Q-value: the resulting K-value can be learned by an algorithm, which we term K-learning, and for which we provide a proof of convergence showing that it can reach the optimal deterministic policy even in stochastic environments. We successfully test our method on a toy problem that was designed to highlight the issues with non-linear scalarizations. Despite its theoretical guarantees, K-learning suffers from many disadvantages that limit its applicability to large-scale problems.

In future work, we hope to apply MORL methods such as those investigated in chapter 4 to develop agents that can optimize multi-function radar resource management end-to-end. With the rewards defined at the task level and expressive, non-linear scalarization to translate the radar's intended behavior, we expect to obtain strategies that can efficiently coordinate dwell optimization and antenna scheduling. We also anticipate that this will allow better coordination between search and tracking. Search dwells can be used to update tracks via an process called track-while-scan (TWS). Current systems decide on a per-track basis whether to update the track by search or active tracking dwells depending on e.g. the threat level or priority level of the track. This choice could be delegated to our end-to-end resource management agent, which could decide for each search dwell it selects whether to use it for track updates as well: this would allow to adjust TWS more dynamically.

The idea of an end-to-end resource management agent could be taken further still by giving it control over the beam direction. In our work, we assumed that the role of the agent would be to select a task to update; whether it corresponds to a track or a search zone, the update dwell would be given a beam direction beyond the control of the

agent, as this direction can be determined based on the estimated position of the target according to the filtering algorithm in the tracking case, or based on a pre-computed search pattern in the search case. However, we could alternatively remove task selection from the agent's action space, and instead make this action space correspond only to the choice of the dwell's parameters, including the direction of the beam. The agent would then have to also optimize its search pattern. Although this formulation would be harder to solve, it may result in a more versatile agent that would learn new, more agile forms of coordination between search and tracking.

Throughout our work, we restricted ourselves to the case of a single multi-function radar with rotating antenna, which is a common use case nowadays. However, future generations of radar systems will go beyond this model, be it through the use of multiple-input multiple-output (MIMO) antennae or of radar networks. Yet the capabilities of these new systems will pose a challenge in terms of resource management. MIMO radars, for example, will be able to generate several dwells in parallel: this puts into question methods that assume a linear utilization of antenna time. Radars working in network will have to apportion tracks among themselves depending on the disposition of the targets and the load of each radar: the coordination of the radars could be tackled via multi-agent reinforcement learning, with each radar learning to communicate and adapt to the behavior of the other radars. Multi-agent reinforcement learning could also be applied to the tracking of autonomous reactive targets. Although most targets can detect when they are being illuminated by a radar, so far only human pilots are susceptible to react and attempt maneuvers to cause the radar to lose their track. In the future, as reactive unmanned targets are bound to be developed, one way to adapt radar strategies would be to train radar resource management agents in environments where the target is also modeled as an agent that learns to escape the radar's sight. Given the host of evolutions to come in radar systems, we hope that the insights gained from the application of reinforcement learning to multi-function radars may be transferred and expanded on to build adaptive, reliable systems.

Finally, we believe multi-objective methods holds broader implications for reinforcement learning in general. We see MORL as a crucial tool to help with one of the most common practical impediments to real-life applications of reinforcement learning, which is reward specification. The expressiveness of non-linear scalarizations opens new possibilities for reward shaping that could feel more intuitive to practitioners. Furthermore,

we think non-linear scalarizations like the thresholded lexicographic order could be repurposed to implement a form of curriculum learning: indeed, with this approach, the agent would have to start by learning to reach a first threshold on the expected return for the most important objective, and once this objective is fulfilled, try to reach the second threshold, etc. If the objectives are ordered by increasing difficulty, this could help guide the agent's learning without requiring more complex curriculum methodologies. One disadvantage for this approach is that under SER, each threshold is checked against the expectation of the return for this objective, which doesn't correspond to the intuitive notion of the threshold as a level that the agent can reach at (almost) all episodes. In this context, it could be interesting to take inspiration from distributional reinforcement learning (Dabney et al., 2018). By learning a distributional value function, we can derive risk-sensitive policies: roughly speaking, we could have the agent consider not the average return, but the mean of the bottom 10% of returns, which would take us closer to putting a threshold on the worst cases. In the long term, MORL also represents a promising perspective towards AI safety and alignment, as it allows practitioners to express complex preferences over conflicting objectives. We hope further progress on multi-objective methods will bolster this vision and permit the development of safe AI systems.

# Bibliography

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, Banff, Alberta, Canada, July 2004. Association for Computing Machinery. ISBN 978-1-58113-838-2. doi: 10.1145/1015330.1015430.

F. B. Abdelaziz and H. Mir. An Optimization Model and Tabu Search Heuristic for Scheduling of Tasks on a Radar Sensor. *IEEE Sensors Journal*, 2016. doi: 10.1109/JSEN.2016.2587730.

A. Abels, D. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher. Dynamic Weights in Multi-Objective Deep Reinforcement Learning. In *International Conference on Machine Learning*, pages 11–20, Long Beach, California, USA, May 2019. PMLR.

J. Achiam and S. Sastry. Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning. *arXiv:1703.01732*, Nov. 2016.

J. Achiam, H. Edwards, D. Amodei, and P. Abbeel. Variational Option Discovery Algorithms. *arXiv:1807.10299 [cs]*, July 2018.

S.-i. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10 (2):251–276, Feb. 1998. ISSN 0899-7667. doi: 10.1162/089976698300017746.

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight Experience Replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5048–5058. Curran Associates, Inc., 2017.

M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement Learning

through Asynchronous Advantage Actor-Critic on a GPU. *arXiv:1611.06256 [cs]*, Mar. 2017.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 1726–1734, San Francisco, California, USA, Feb. 2017. AAAI Press.

S. H. Baek, H. Seok, K. H. Park, and J. Chun. An adaptive update-rate control of a phased array radar for efficient usage of tracking tasks. In *2010 IEEE Radar Conference*, pages 1214–1219, May 2010. doi: 10.1109/RADAR.2010.5494433.

A. Bajpai, S. Garg, and Mausam. Transfer of deep reactive policies for MDP planning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 10988–10998, Red Hook, NY, USA, Dec. 2018. Curran Associates Inc.

G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap. Distributed Distributional Deterministic Policy Gradients. In *International Conference on Learning Representations*, Feb. 2018.

D. K. Barton. *Radar System Analysis and Modeling*. Artech House Publishers, Boston, MA, har/cdr edition edition, Oct. 2004. ISBN 978-1-58053-681-3.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*, Oct. 2018.

M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1471–1479. Curran Associates, Inc., 2016.

M. G. Bellemare, W. Dabney, and R. Munos. A Distributional Perspective on Reinforcement Learning. In *International Conference on Machine Learning*, pages 449–458, July 2017.

S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House Publishers, Boston, Mass., unabridged édition edition, July 1999. ISBN 978-1-58053-006-4.

B. Bonet and H. Geffner. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. pages 12–31, Jan. 2003.

B. Bonet, G. Francès, and H. Geffner. Learning Features and Abstract Actions for Computing Generalized Plans. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:2703–2710, July 2019. doi: 10.1609/aaai.v33i01.33012703.

C. Boutilier, R. Reiter, and B. Price. Symbolic Dynamic Programming for First-Order MDPs. *IJCAI International Joint Conference on Artificial Intelligence*, Feb. 2002.

Y. Briheche. *Optimization of Search Patterns for Fixed-Panel Tridimensional Scanning Radars*. Theses, Ecole Centrale de Nantes (ECN), Nov. 2017.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, Mar. 2012. ISSN 1943-0698. doi: 10.1109/TCIAIG.2012.2186810.

J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. *arXiv:1807.01675 [cs, stat]*, June 2019.

Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-Scale Study of Curiosity-Driven Learning. In *International Conference on Learning Representations*, Sept. 2018.

J. Butler, A. Moore, and H. Griffiths. Resource Management For A Rotating Multifunction Radar. In *Radar 97 (Conf. Publ. No. 449)*, pages 568–572, Oct. 1997.

N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu. Boltzmann Exploration Done Right. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6284–6293. Curran Associates, Inc., 2017.

A. Charlish and F. Katsilieris. Array Radar Resource Management. Oct. 2017. ISBN 978-1-61353-227-0. doi: 10.1049/SBRA512F.

K. Chatterjee, R. Majumdar, and T. A. Henzinger. Markov Decision Processes with Multiple Objectives. In B. Durand and W. Thomas, editors, *STACS 2006*, Lecture Notes in Computer Science, pages 325–336, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-32288-7. doi: 10.1007/11672142_26.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4754–4765. Curran Associates, Inc., 2018.

I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel. Model-Based Reinforcement Learning via Meta-Policy Optimization. In *Conference on Robot Learning*, Sept. 2018.

A. V. Clemente, H. N. Castejón, and A. Chandra. Efficient Parallel Methods for Deep Reinforcement Learning. *arXiv:1705.04862 [cs]*, May 2017.

C. Colas, P. Fournier, M. Chetouani, O. Sigaud, and P.-Y. Oudeyer. CURIOUS: Intrinsically Motivated Modular Multi-Goal Reinforcement Learning. In *International Conference on Machine Learning*, pages 1331–1340, May 2019.

E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5027–5038. Curran Associates, Inc., 2018.

W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit Quantile Networks for Distributional Reinforcement Learning. In *International Conference on Machine Learning*, pages 1096–1105, July 2018.

E. Daeipour, Y. Bar-Shalom, and X. Li. Adaptive beam pointing control of a phased array radar using an IMM estimator. In *Proceedings of 1994 American Control Conference - ACC '94*, volume 2, pages 2093–2097 vol.2, June 1994. doi: 10.1109/ACC.1994.752445.

P. Dayan and G. E. Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 271–278, San Francisco, CA, USA, Nov. 1992. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-274-8.

P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1):19–67, Feb. 2005. ISSN 1572-9338. doi: 10.1007/s10479-005-5724-z.

T. Degris, M. White, and R. S. Sutton. Off-Policy Actor-Critic. In *International Conference on Machine Learning*, Edinburgh, United Kingdom, June 2012.

M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 465–472, Bellevue, Washington, USA, June 2011. Omnipress. ISBN 978-1-4503-0619-5.

L. Denoyer, A. de la Fuente, S. Duong, J.-B. Gaya, P.-A. Kamienny, and D. H. Thompson. SaLinA: Sequential Learning of Agents. Oct. 2021. doi: 10.48550/arXiv.2110.07910.

S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks. *arXiv:1605.07127 [cs, stat]*, Mar. 2017.

T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13(1):227–303, Nov. 2000. ISSN 1076-9757.

J. Dornheim. gTLO: A Generalized and Non-linear Multi-Objective Deep Reinforcement Learning Approach, Apr. 2022.

D. Driess, I. Schubert, P. Florence, Y. Li, and M. Toussaint. Reinforcement Learning with Neural Radiance Fields, June 2022.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1329–1338, New York, NY, USA, June 2016a. JMLR.org.

Y. Duan, J. Schulman, X. Chen, P. Bartlett, I. Sutskever, and P. Abbeel. $RL^2$: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv.org*, Nov. 2016b.

A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-Explore: A New Approach for Hard-Exploration Problems. *arXiv:1901.10995 [cs, stat]*, May 2019.

A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. First return then explore. *arXiv:2004.12919 [cs]*, May 2020.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *arXiv:1802.01561 [cs]*, June 2018.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is All You Need: Learning Skills without a Reward Function. *arXiv:1802.06070 [cs]*, Oct. 2018.

G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson. TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning. *arXiv:1710.11417 [cs, stat]*, Mar. 2018.

V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. *arXiv:1803.00101 [cs, stat]*, Feb. 2018.

A. Fern, S. Yoon, and R. Givan. Approximate Policy Iteration with a Policy Language Bias. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1126–1135, Sydney, NSW, Australia, Aug. 2017. JMLR.org.

C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic Goal Generation for Reinforcement Learning Agents. *arXiv:1705.06366 [cs]*, July 2018.

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy Networks for Exploration. *International Conference on Learning Representations*, 2017.

R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-Level Discovery of Deep Options. *arXiv:1703.08294 [cs]*, Oct. 2017.

V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. *An Introduction to Deep Reinforcement Learning*. now publishers Inc, Dec. 2018. ISBN 978-1-68083-538-0.

W. Fu, C. Yu, Z. Xu, J. Yang, and Y. Wu. Revisiting Some Common Practices in Cooperative Multi-Agent Reinforcement Learning, Aug. 2022.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. *International Conference on Machine Learning*, Oct. 2018.

M. Gaafar, M. Shaghaghi, R. S. Adve, and Z. Ding. Reinforcement Learning for Cognitive Radar Task Scheduling. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1653–1657, Nov. 2019. doi: 10.1109/IEEECONF44664.2019.9048892.

Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. *International Conference on Machine Learning*, 2016.

S. Garg, A. Bajpai, and Mausam. Size Independent Neural Transfer for RDDL Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29:631–636, 2019. ISSN 2334-0843. doi: 10.1609/icaps.v29i1.3530.

S. Garg, A. Bajpai, and Mausam. Symbolic network: Generalized neural policies for relational MDPs. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, pages 3397–3407. JMLR.org, July 2020.

M. Gemici, C.-C. Hung, A. Santoro, G. Wayne, S. Mohamed, D. J. Rezende, D. Amos, and T. Lillicrap. Generative Temporal Models with Memory. *arXiv:1702.04649 [cs, stat]*, Feb. 2017.

T. George, K. Wagner, and P. Rademacher. Deep Q-Network for Radar Task-Scheduling Problem. In *2022 IEEE Radar Conference (RadarConf22)*, pages 1–5, Mar. 2022. doi: 10.1109/RadarConf2248738.2022.9764230.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, Massachusetts, Nov. 2016. ISBN 978-0-262-03561-3.

A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, and R. Munos. The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning. *International Conference on Learning Representations*, 2017.

C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 1003–1010, San Francisco, CA, USA, Aug. 2003a. Morgan Kaufmann Publishers Inc.

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, Oct. 2003b. ISSN 1076-9757. doi: 10.1613/jair.1000.

A. Guez, T. Weber, I. Antonoglou, K. Simonyan, O. Vinyals, D. Wierstra, R. Munos, and D. Silver. Learning to Search with MCTSnets. *arXiv:1802.04697 [cs, stat]*, July 2018.

A. Guez, M. Mirza, K. Gregor, R. Kabra, S. Racaniere, T. Weber, D. Raposo, A. Santoro, L. Orseau, T. Eccles, G. Wayne, D. Silver, and T. Lillicrap. An Investigation of Model-Free Planning. In *International Conference on Machine Learning*, pages 2464–2473, May 2019.

D. Ha and J. Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2450–2462. Curran Associates, Inc., 2018.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement Learning with Deep Energy-Based Policies. *International Conference on Machine Learning*, July 2017.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *International Conference on Machine Learning*, Aug. 2018.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning Latent Dynamics for Planning from Pixels. *arXiv:1811.04551 [cs, stat]*, June 2019.

J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, and P. W. Battaglia. Relational inductive bias for physical construction in humans and machines, June 2018.

N. Hansen. The CMA Evolution Strategy: A Tutorial. *arXiv:1604.00772 [cs, stat]*, Apr. 2016.

M. Hausknecht and P. Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *Association for the Advancement of Artificial Intelligence*, 2015.

M. Hausknecht and P. Stone. Deep Reinforcement Learning in Parameterized Action Space, Feb. 2016.

K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an Embedding Space for Transferable Robot Skills. In *International Conference on Learning Representations*, Feb. 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015.

N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning Continuous Control Policies by Stochastic Value Gradients. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2944–2952. Curran Associates, Inc., 2015.

M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. Multi-task Deep Reinforcement Learning with PopArt. *arXiv:1809.04474 [cs, stat]*, Sept. 2018.

T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep Q-learning from Demonstrations. *arXiv:1704.03732 [cs]*, Nov. 2017.

J. Ho and S. Ermon. Generative adversarial imitation learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4572–4580, Barcelona, Spain, Dec. 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9 (8):1735–1780, Nov. 1997.

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed Prioritized Experience Replay. *International Conference on Learning Representations*, Mar. 2018.

A. Izquierdo-Fuente and J. Casar-Corredera. Optimal radar pulse scheduling using a neural network. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 7, pages 4588–4591 vol.7, June 1994. doi: 10.1109/ICNN.1994.375014.

A. Jabri, K. Hsu, B. Eysenbach, A. Gupta, S. Levine, and C. Finn. Unsupervised Curricula for Visual Meta-Reinforcement Learning. *arXiv:1912.04226 [cs]*, Dec. 2019.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. *International Conference on Learning Representations*, 2017.

J. Janisch, T. Pevný, and V. Lisý. Symbolic Relational Deep Reinforcement Learning based on Graph Neural Networks, July 2021.

V. Jeauneau, T. Guenais, and F. Barbaresco. Scheduling on a fixed multifunction radar antenna with hard time constraint. *2013 14th International Radar Symposium (IRS)*, 2013.

S. Kakade. A natural policy gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 1531–1538, Vancouver, British Columbia, Canada, Jan. 2001. MIT Press.

K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics. *arXiv:1706.04317 [cs]*, Aug. 2017.

S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*, Sept. 2018.

R. Karia and S. Srivastava. Relational Abstractions for Generalized Reinforcement Learning on Symbolic Problems. arXiv, 2022. doi: 10.48550/ARXIV.2204.12665.

M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data. *arXiv:1605.06432 [cs, stat]*, Mar. 2017.

S. Kelly and M. I. Heywood. Emergent Solutions to High-Dimensional Multitask Reinforcement Learning. *Evolutionary Computation*, 26(3):347–380, 2018. ISSN 1530-9304.

E. Keyder and H. Geffner. The HMDP Planner for Planning with Probabilities. *3rd International Probabilistic Planning Competition (IPPC-ICAPS'08)*, Jan. 2008.

S. Khadka and K. Tumer. Evolution-guided policy gradient in reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 1196–1208, Montréal, Canada, Dec. 2018. Curran Associates Inc.

A. Kolobov, M. Mausam, and D. Weld. ReTrASE: Integrating Paradigms for Approximate Probabilistic Planning. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1746–1753, Jan. 2009.

A. Kolobov, P. Dai, M. Mausam, and D. Weld. Reverse Iterative Deepening for Finite-Horizon MDPs with Large Branching Factors. *Proceedings of the International Conference on Automated Planning and Scheduling*, 22:146–154, May 2012. ISSN 2334-0843. doi: 10.1609/icaps.v22i1.13523.

V. Konda and J. Tsitsiklis. Actor-Critic Algorithms. In *SIAM Journal on Control and Optimization*, pages 1008–1014. MIT Press, 2000.

I. Kostrikov, D. Yarats, and R. Fergus. Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels, Mar. 2021.

V. Krishnamurthy and R. Evans. Hidden Markov model multiarm bandits: A methodology for beam scheduling in multitarget tracking. *IEEE Transactions on Signal Processing*, 49(12):2893–2908, Dec. 2001. ISSN 1941-0476. doi: 10.1109/78.969499.

R. G. Krishnan, U. Shalit, and D. Sontag. Deep Kalman Filters. *arXiv:1511.05121 [cs, stat]*, Nov. 2015.

T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-Ensemble Trust-Region Policy Optimization. In *International Conference on Learning Representations*, Feb. 2018.

B. F. La Scala and B. Moran. Optimal target tracking with restless bandits. *Digital Signal Processing*, 16(5):479–487, Sept. 2006. ISSN 1051-2004. doi: 10.1016/j.dsp.2006.04.008.

P. Le Pelletier de Woillemont, R. Labory, and V. Corruble. Configurable Agent With Reward As Input: A Play-Style Continuum Generation. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, Copenhagen, Denmark, Aug. 2021. IEEE Press. doi: 10.1109/CoG52621.2021.9619127.

Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. *The handbook of brain theory and neural networks*, 1995.

J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, June 2011. ISSN 1063-6560.

S. Levine. Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review. *arXiv:1805.00909 [cs, stat]*, May 2018.

S. Levine and P. Abbeel. Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1071–1079. Curran Associates, Inc., 2014.

S. Levine and V. Koltun. Guided Policy Search. In *International Conference on Machine Learning*, pages 1–9, Feb. 2013.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, Apr. 2016.

C. Li and K. Czarnecki. Urban Driving with Multi-Objective Deep Reinforcement Learning, Feb. 2019.

S. Li, X. Puig, C. Paxton, Y. Du, C. Wang, L. Fan, T. Chen, D.-A. Huang, E. Akyürek, A. Anandkumar, J. Andreas, I. Mordatch, A. Torralba, and Y. Zhu. Pre-Trained Language Models for Interactive Decision-Making, Oct. 2022.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699.

N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement Learning for Combinatorial Optimization: A Survey. *arXiv:2003.03600 [cs, math, stat]*, Dec. 2020.

L. Metz, J. Ibarz, N. Jaitly, and J. Davidson. Discrete Sequential Prediction of Continuous Actions for Deep RL. *arXiv:1705.05035 [cs, stat]*, 2017.

H. Mir and A. Guitouni. Variable Dwell Time Task Scheduling for Multifunction Radar. *IEEE Transactions on Automation Science and Engineering*, 2014. doi: 10.1109/TASE.2013.2285014.

N. Mishra, P. Abbeel, and I. Mordatch. Prediction and Control with Temporal Segment Models. *arXiv:1703.04070 [cs, stat]*, July 2017.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, Dec. 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 1476-4687. doi: 10.1038/nature14236.

V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *International Conference on Machine Learning*, 48:1928–1937, June 2016.

P. W. Moo and Z. Ding. *Adaptive Radar Resource Management*. Elsevier, 2015.

H. Mossalam, Y. M. Assael, D. M. Roijers, and S. Whiteson. Multi-Objective Deep Reinforcement Learning. *arXiv:1610.02707 [cs]*, Oct. 2016.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Bridging the Gap Between Value and Policy Based Reinforcement Learning. *Advances in Neural Information Processing Systems*, Nov. 2017.

O. Nachum, S. Gu, H. Lee, and S. Levine. Data-Efficient Hierarchical Reinforcement Learning. *arXiv:1805.08296 [cs, stat]*, Oct. 2018a.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Trust-PCL: An Off-Policy Trust Region Method for Continuous Control. *International Conference on Learning Representations*, Feb. 2018b.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, May 2018. doi: 10.1109/ICRA.2018.8463189.

A. Nagabandi, C. Finn, and S. Levine. Deep Online Learning via Meta-Learning: Continual Adaptation for Model-Based RL. *arXiv:1812.07671 [cs, stat]*, Jan. 2019.

S. Natarajan, S. Joshi, P. Tadepalli, K. Kersting, and J. Shavlik. Imitation Learning in Relational Domains: A Functional-Gradient Boosting Approach. pages 1414–1420, Jan. 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-239.

A. Y. Ng and S. J. Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, June 2000. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-707-1.

A. Y. Ng, D. Harada, and S. J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth Interna-*

*tional Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, June 1999. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-612-8.

T. Nguyen. A Multi-Objective Deep Reinforcement Learning Framework. *ArXiv*, 2018. doi: 10.1016/j.engappai.2020.103915.

T. Nguyen-Tang, S. Gupta, and S. Venkatesh. Distributional Reinforcement Learning with Maximum Mean Discrepancy. *ArXiv*, July 2020.

B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih. Combining policy gradient and Q-learning. *International Conference on Learning Representations*, Apr. 2017.

J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games. *arXiv:1507.08750 [cs]*, Dec. 2015.

J. Oh, S. Singh, and H. Lee. Value Prediction Network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6118–6128. Curran Associates, Inc., 2017.

J. Oh, Y. Guo, S. Singh, and H. Lee. Self-Imitation Learning. *arXiv:1806.05635 [cs, stat]*, June 2018.

J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering Reinforcement Learning Algorithms. *arXiv:2007.08794 [cs]*, July 2020.

A. J. Orman, C. N. Potts, A. K. Shahani, and A. R. Moore. Scheduling for a multi-function phased array radar system. *European Journal of Operational Research*, 90(1): 13–25, Apr. 1996. ISSN 0377-2217. doi: 10.1016/0377-2217(95)00307-X.

I. Osband and B. Van Roy. Near-optimal Reinforcement Learning in Factored MDPs, Oct. 2014.

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep Exploration via Bootstrapped DQN. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4026–4034. Curran Associates, Inc., 2016.

G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2721–2730, Sydney, NSW, Australia, Aug. 2017. JMLR.org.

P.-Y. Oudeyer and F. Kaplan. What is Intrinsic Motivation? A Typology of Computational Approaches. *Frontiers in Neurorobotics*, 1, Nov. 2007. ISSN 1662-5218. doi: 10.3389/neuro.12.006.2007.

P.-Y. Oudeyer and F. Kaplan. How can we define intrinsic motivation ? *Proc. of the 8th Conf. on Epigenetic Robotics*, 5:29–31, 2009.

S. Parisi, A. Rajeswaran, S. Purushwalkam, and A. Gupta. The Unsurprising Effectiveness of Pre-Trained Vision Models for Control, Aug. 2022.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pages 1043–1049, Denver, Colorado, USA, July 1998. MIT Press. ISBN 978-0-262-10076-2.

R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia. Learning model-based planning from scratch. *arXiv:1707.06170 [cs, stat]*, July 2017.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2778–2787, Sydney, NSW, Australia, Aug. 2017. JMLR.org.

F. Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv:1712.06567 [cs]*, Apr. 2018.

A. Pourchot and O. Sigaud. CEM-RL: Combining evolutionary and gradient-based methods for policy search. *arXiv:1810.01222 [cs, stat]*, Feb. 2019.

P. Pulkkinen, T. Aittomäki, A. Ström, and V. Koivunen. Time Budget Management in

Multifunction Radars Using Reinforcement Learning. In *2021 IEEE Radar Conference (RadarConf21)*, pages 1–6, May 2021. doi: 10.1109/RadarConf2147009.2021.9455344.

Z. Qu, Z. Ding, and P. Moo. A Machine Learning Task Selection Method for Radar Resource Management (Poster). In *2019 22th International Conference on Information Fusion (FUSION)*, pages 1–6, July 2019.

S. Racanière, T. Weber, D. P. Reichert, L. Buesing, A. Guez, D. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5694–5705, Long Beach, California, USA, Dec. 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

M. Reymond and A. Nowé. Pareto-DQN: Approximating the Pareto front in complex multi-objective decision problems. *Proceedings of the Adaptive and Learning Agents Workshop 2019 (ALA-19) at AAMAS*, May 2019.

D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48:67–113, Oct. 2013. ISSN 1076-9757. doi: 10.1613/jair.3987.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864 [cs, stat]*, Sept. 2017.

S. Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description. Jan. 2011.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1312–1320, Lille, France, July 2015. JMLR.org.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. *International Conference on Learning Representations*, Feb. 2016.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari,

Go, Chess and Shogi by Planning with a Learned Model. *arXiv:1911.08265 [cs, stat]*, Feb. 2020.

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust Region Policy Optimization. *International Conference on Machine Learning*, 2015.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *International Conference on Learning Representations*, 2016.

J. Schulman, X. Chen, and P. Abbeel. Equivalence Between Policy Gradients and Soft Q-Learning. *arXiv:1704.06440 [cs]*, 2017a.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, Aug. 2017b.

M. Shaghaghi and R. S. Adve. Machine learning based cognitive radar resource management. *2018 IEEE Radar Conference (RadarConf18)*, pages 1433–1438, Apr. 2018. doi: 10.1109/RADAR.2018.8378775.

M. Shaghaghi, R. S. Adve, and Z. Ding. Multifunction cognitive radar task scheduling using Monte Carlo tree search and policy networks. *IET Radar, Sonar & Navigation*, 12(12):1437–1447, 2018. ISSN 1751-8792. doi: 10.1049/iet-rsn.2018.5276.

M. Shaghaghi, R. S. Adve, and Z. Ding. Resource Management for Multifunction Multichannel Cognitive Radars. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1550–1554, Nov. 2019. doi: 10.1109/IEEECONF44664.2019.9049014.

V. Sharma, D. Arora, F. Geißer, Mausam, and P. Singla. SymNet 2.0: Effectively handling Non-Fluents and Actions in Generalized Neural Policies for RDDL Relational MDPs. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 1771–1781. PMLR, Aug. 2022.

E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own Reward: Self-Supervision for Reinforcement Learning. *arXiv:1612.07307 [cs]*, Mar. 2017.

C. Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives*. PhD thesis, Aug. 2001.

X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 802–810. Curran Associates, Inc., 2015.

O. Sigaud and F. Stulp. Policy search in continuous action domains: An overview. *Neural Networks: The Official Journal of the International Neural Network Society*, 113:28–40, May 2019. ISSN 1879-2782. doi: 10.1016/j.neunet.2019.01.011.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395, Beijing, China, June 2014. JMLR.org.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016. ISSN 1476-4687. doi: 10.1038/nature16961.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, Oct. 2017a. ISSN 1476-4687. doi: 10.1038/nature24270.

D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris. The Predictron: End-To-End Learning and Planning. *arXiv:1612.08810 [cs]*, July 2017b.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through

self-play. *Science*, 362(6419):1140–1144, Dec. 2018. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aar6404.

M. I. Skolnik. *Radar Handbook*. McGraw-Hill Professional, New York, 3 edition, Mar. 2008. ISBN 978-0-07-148547-0.

K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, Apr. 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202.

A. Stooke and P. Abbeel. Accelerated Methods for Deep Reinforcement Learning. *arXiv:1803.02811 [cs]*, 2018.

S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *International Conference on Learning Representations*, Feb. 2018.

R. S. Sutton. The reward hypothesis. http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.ht

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, Cambridge, Massachusetts, 2nd edition edition, 2018. ISBN 978-0-262-03924-6.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, Aug. 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1.

R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, pages 761–768, Taipei, Taiwan, May 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9826571-6-4.

I. Szita and A. Lörincz. Learning Tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, Dec. 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.12.2936.

T. Tajmajer. Modular Multi-Objective Deep Reinforcement Learning with Decision Values. *arXiv:1704.06676 [cs]*, Feb. 2018.

A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. Value Iteration Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2154–2162. Curran Associates, Inc., 2016.

H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. *arXiv:1611.04717 [cs]*, Dec. 2017.

A. Tavakoli, F. Pardo, and P. Kormushev. Action Branching Architectures for Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. ISSN 2374-3468.

Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust Multitask Reinforcement Learning. *arXiv:1707.04175 [cs, stat]*, July 2017.

E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306 vol. 1, June 2005. doi: 10.1109/ACC.2005.1469949.

S. Toyer, F. Trevizan, S. Thiebaux, and L. Xie. Action Schema Networks: Generalised Policies With Deep Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, Sept. 2017. doi: 10.1609/aaai.v32i1.12089.

E. Uchibe and K. Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. In *2007 IEEE 6th International Conference on Development and Learning*, pages 163–168, July 2007. doi: 10.1109/DEVLRN.2007.4354030.

P. Vamplew, C. Foale, and R. Dazeley. A Demonstration of Issues with Value-Based Multiobjective Reinforcement Learning Under Stochastic State Transitions. *Neural Computing and Applications*, 34(3):1783–1799, Feb. 2022. ISSN 0941-0643, 1433-3058. doi: 10.1007/s00521-021-05859-1.

H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *Association for the Advancement of Artificial Intelligence*, 2016.

G. van Keuk and S. Blackman. On phased-array radar tracking and parameter control. *IEEE Transactions on Aerospace and Electronic Systems*, 29(1):186–194, Jan. 1993. ISSN 1557-9603. doi: 10.1109/7.249124.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 3540–3549, Sydney, NSW, Australia, Aug. 2017. JMLR.org.

M. Vincent, A. El Fallah Seghrouchni, V. Corruble, N. Bernardin, R. Kassab, and F. Barbaresco. Monte Carlo Tree Search for Multi-function Radar Task Scheduling. In *Conference on Artificial Intelligence for Defense*, Rennes, France, Nov. 2021.

M. Vincent, A. El Fallah Seghrouchni, V. Corruble, N. Bernardin, R. Kassab, and F. Barbaresco. Reinforcement Learning for Multi-objective Factored MDPs using Graph Neural Networks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, London, UK, May 2023.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, Nov. 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z.

C. Wang, X. Luo, K. Ross, and D. Li. VRL3: A Data-Driven Framework for Visual Deep Reinforcement Learning, Feb. 2022a.

S. Wang, M. Reymond, A. A. Irissappane, and D. M. Roijers. Near On-Policy Experience Sampling in Multi-Objective Reinforcement Learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, pages 1756–1758, Richland, SC, May 2022b. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-9213-6.

T. Wang, R. Liao, J. Ba, and S. Fidler. NerveNet: Learning Structured Policy with Graph Neural Networks. In *International Conference on Learning Representations*, 2018.

Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *International Conference on Machine Learning*, Apr. 2016.

Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample Efficient Actor-Critic with Experience Replay. *International Conference on Learning Representations*, July 2017.

Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents, Feb. 2023.

M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally Linear Latent dynamics model for control from raw images. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2746–2754, Montreal, Canada, Dec. 2015. MIT Press.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.

J. Wintenby. Resource Allocation in Airborne Surveillance Radar. 2003.

É. Winter and P. Baptiste. On scheduling a multifunction radar. 2007. doi: 10.1016/J.AST.2007.01.006.

Y. Wu, E. Mansimov, L. Shun, A. Radford, and J. Schulman. OpenAI Baselines: ACKTR & A2C, Aug. 2017.

D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, and T. Liu. Fully Parameterized Quantile Function for Distributional Reinforcement Learning, Aug. 2020.

R. Yang, X. Sun, and K. Narasimhan. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation, Nov. 2019.

H. Younes, M. Littman, D. Weissman, and J. Asmuth. The First Probabilistic Track of the International Planning Competition. *J. Artif. Intell. Res. (JAIR)*, 24:851–887, July 2005. doi: 10.1613/jair.1880.

V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals, and P. Battaglia. Relational Deep Reinforcement Learning. *arXiv:1806.01830 [cs, stat]*, June 2018.

H. Zhang, J. Xie, J. Ge, Z. Zhang, and B. Zong. A hybrid adaptively genetic algorithm for task scheduling problem in the phased array radar. *European Journal of Operational Research*, 272(3):868–878, Feb. 2019. ISSN 03772217. doi: 10.1016/j.ejor.2018.07.012.

J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph Neural Networks: A Review of Methods and Applications, Oct. 2021.

## Reinforcement Learning for Multi-Function Radar Resource Management

**Abstract**

In the wake of recent advances in the field of machine learning, much progress has been accomplished in one of its sub-fields, reinforcement learning, whose aim is to solve sequential decision problems under uncertainty. Radar resource management seems to represent an ideal application case for this type of technique. Indeed, a radar emits signals, called dwells, whose echoes are used to measure the state of surrounding objects; these dwells vary according to numerous parameters (duration, beam width...) and must be executed sequentially. The surveillance strategy of a multi-function radar thus consists in continuously selecting the dwells to perform, with the aim of searching the surrounding space while tracking already detected targets. The methods currently used to address this problem are largely heuristic, and are likely to run into difficulties in a range of complex situations involving hyper-velocity or hyper-maneuvering targets.

First, we propose applications of reinforcement learning techniques adapted to the current architecture of multi-function radars. These contributions focus on two aspects : dwell scheduling on the antenna using *model-based* methods, and active tracking dwell optimization using *model-free* methods. Secondly, we highlight the limitations of current resource management architectures, which leads us to consider an alternative architecture for which we propose new reinforcement learning algorithms designed to address the problems it raises. These contributions focus both on the multi-objective aspect, which is useful in multi-function radars to reflect the trade-offs to be made between different functions, and on the combinatorial aspect, which is due to the large number of tasks that the radar must carry out in parallel.

**Keywords:** reinforcement learning, radar, scheduling, multi-objective

---

**Résumé**

Dans le sillage des avancées récentes dans le champ de l'apprentissage automatique, de nombreux progrès ont été réalisés dans l'un de ses sous-domaines, l'apprentissage par renforcement, dont le but est de résoudre des problèmes de décision séquentielle dans l'incertain. La gestion de ressources radar semble représenter un cadre d'application propice pour ce type de techniques. En effet, un radar émet des signaux, appelés pointages, dont l'écho permet de mesurer l'état des objets alentour ; ces pointages varient selon de nombreux paramètres (durée, largeur de faisceau...) et doivent être exécutés séquentiellement. La stratégie de surveillance d'un radar multi-fonctions revient ainsi à sélectionner en continu les pointages à effectuer dans le but de surveiller l'espace environnant tout en pistant les cibles déjà détectées. Les méthodes utilisées actuellement pour répondre à cette problématique sont en grande partie heuristiques et risquent d'être mises en difficulté dans une gamme de situations complexes impliquant des cibles hyper-véloces ou hyper-manœuvrantes.

Dans un premier temps, nous proposons des applications de techniques d'apprentissage par renforcement adaptées à l'architecture courante des radars multi-fonction. Ces contributions portent sur deux aspects : l'ordonnancement des pointages sur l'antenne par méthodes *model-based* et l'optimisation des pointages de poursuite active par méthodes *model-free*. Dans un second temps, nous mettons en avant les limites des architectures de gestion de ressources actuelles, ce qui nous amène à envisager une architecture alternative pour laquelle nous proposons de nouveaux algorithmes d'apprentissage par renforcement destinés à répondre aux problèmes qu'elle soulève. Ces contributions portent à la fois sur un aspect multi-objectif, utile dans les radars multi-fonctions pour refléter les compromis à réaliser entre les différentes fonctions, et sur l'aspect combinatoire qui est dû au grand nombre de tâches que le radar doit mener à bien en parallèle.

**Mots clés :** apprentissage par renforcement, radar, ordonnancement, multi-objectif

---

**LIP6**
Sorbonne Université – Campus Pierre et Marie Curie – 4 place Jussieu – 75005 Paris – France