UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques
i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

Lexicographic Proximal Policy Optimization for Ethical Multi-Agent Learning

Autora: Núria Torquet Luna

Directors: Dra. Maite López Sánchez i
Dr. Manel Rodríguez Soto
Realitzat a: Departament de Matemàtiques
i Informàtica

Barcelona, 4 de juny de 2024

Contents

Introduction **iv**

1 Introduction **1**

2 Background **3**
2.1 Reinforcement Learning 4
2.2 Markov Decision Processes 5
2.3 Expected Discounted Returns 6
2.4 Value Functions 7
2.5 Optimal Policy 8
2.6 Value Based Methods 10
2.6.1 Temporal-Difference Learning 10
2.7 Policy Based Learning 11
2.8 Proximal Policy Optimization 11
2.8.1 Policy Gradient Methods 12
2.8.2 Overview of the algorithm 17
2.8.3 Return 21
2.8.4 Advantage Estimates 21
2.8.5 Policy Network's Objective Function L^{CLIP} 22
2.9 Lexicographic Proximal Policy Optimization 24
2.9.1 Multi-Objective Markov Decision Process 26
2.9.2 Lexicographic RL 26
2.9.3 Policy-based algorithm 27
2.10 Models of Multi-Agent Interaction 29
2.10.1 Normal Form Games 29
2.10.2 Stochastic Games 30
2.11 Solution Concepts for Games 31
2.11.1 Joint Policy 31
2.11.2 Best Response 32
2.11.3 Nash Equilibrium 32

2.12 Independent Learning 32

3 State of the art **34**

4 Description of the environment **36**
4.1 Environment characteristics 37
4.2 Hyper-parameters 39
4.2.1 ILPPO hyper-parameters 40
4.3 Desired Ethical Policies 40

5 Experiments **42**
5.1 Objectives 42
5.2 Empirical Setup 42
5.2.1 Comparison with IPPO in a small environment 44

6 Conclusion and Future Work **55**

A Hyper-parameters for the Learning Approach **56**

B Simulation results separated by seed **58**
B.1 Simulation results for IPPO algorithm 58
B.2 Simulation results for ILPPO algorithm 58
B.3 T-Test Results Between IPPO and ILPPO Metrics 58

List of Tables

1 Nomenclature (Part 1) vii
2 Nomenclature (Part 2) viii

5.1 Environment Setup 45
5.2 Comparison of IPPO and ILPPO Metrics 52

A.1 Hyper-parameters for the Learning Approach 57

B.1 IPPO Performance Metrics (Part 1) 58
B.2 IPPO Performance Metrics (Part 2) 59
B.3 ILPPO Performance Metrics (Part 1) 59
B.4 ILPPO Performance Metrics (Part 2) 60
B.5 ILPPO Performance Metrics (Part 3) 61
B.6 T-test Results Between PPO and LPPO Metrics 61

List of Figures

2.1 Classification of methods to find optimal policies 13
2.2 Table with separate cases for the function L^{CLIP} 23
2.3 Parts of a MARL problem 31

4.1 Graphical representation of the tiny environment. 37
4.2 Evolution of the number of apples collected by each agent (A_1 and A_2) and the apples in the donation box over time. 41

5.1 Evolution of the accumulated reward for the efficient (a) and inefficient (b) agent during ILPPO training 48
5.2 Evolution of the separate accumulated rewards for Agent-1 during ILPPO training 49
5.3 Evolution of the separate accumulated rewards for Agent-2 during ILPPO training 49
5.4 Evolution of the average accumulated reward during ILPPO Training 50
5.5 Comparison of the cumulative rewards between IPPO (orange) and ILPPO (blue) for both agents 50
5.6 Comparison of the average cumulative reward between IPPO (orange) and ILPPO (blue) 51
5.7 IPPO (a) and IPPO (b) evolution of the number of apples collected by Agent 1, Agent 2, and the Donation Box over time. The shaded regions represent the interquartile range for each agent. 54

Abstract

This Bachelor's Degree Final Project studies the integration of ethical principles into multi-objective, multi-agent reinforcement learning (MOMARL) through the implementation and evaluation of the Independent Lexicographic Proximal Policy Optimization (ILPPO) algorithm. In multi-agent reinforcement learning (MARL) the dynamic interactions between agents can make ethical learning particularly complex. ILPPO addresses these challenges by prioritizing ethical decision-making via a lexicographic ordering of multiple objectives, ensuring that the ethical objectives are met before addressing other individual objectives.
We start by presenting the necessary background on a multi-objective Markov Decision Process (MOMDP), the Proximal Policy Optimization (PPO) algorithm and the lexicographic RL framework, which forms the basis for LPPO algorithm. Then, we extrapolate the three elements of this framework (MOMDP, PPO and LPPO) in the context of multi-agent Markov games, where each agent learns independently. Once we develop the Independent LPPO (ILPPO), we evaluate it in the Ethical Gathering Game, an environment where agents learn to behave in alignment with the moral value of beneficence. Our experiments demonstrate that ILPPO can learn optimal ethical policies aligned with ethical values, similar to the ones obtained with Independent PPO (IPPO). This study concludes that ILPPO provides a robust framework for embedding ethical considerations into MOMARL, offering new insights and paving the way for future research in more complex environments.

Resum

Aquest Treball de Fi de Grau estudia la integració de principis ètics en l'aprenentatge per reforç multiobjectiu i multiagent (MOMARL) mitjançant la implementació i avaluació de l'algoritme Independent Lexicographic Proximal Policy Optimization (ILPPO). En l'aprenentatge per reforç multiagent (MARL), les interaccions dinàmiques entre agents poden fer que l'aprenentatge ètic sigui particularment complex. ILPPO aborda aquests reptes prioritzant la presa de decisions ètica mitjançant una ordenació lexicogràfica de múltiples objectius, assegurant que els objectius ètics es compleixin abans d'abordar altres objectius individuals.
Comencem presentant els antecedents necessaris sobre un procés de decisió de Markov multiobjectiu (MOMDP), l'algoritme Proximal Policy Optimization (PPO) i el marc lexicogràfic d'aprenentatge per reforç, que forma la base de l'algoritme LPPO.
A continuació, extrapolem els tres elements d'aquest marc (MOMDP, PPO i LPPO)

en el context dels jocs de Markov multiagent, on cada agent aprèn de manera independent.
Un cop desenvolupat l'Independent LPPO (ILPPO), l'avaluem en el "Ethical Gathering Game", un entorn on els agents aprenen a comportar-se d'acord amb el valor moral de la beneficència. Els nostres experiments demostren que ILPPO pot aprendre polítiques ètiques òptimes alineades amb valors ètics, similars a les obtingudes amb Independent PPO (IPPO). Aquest estudi conclou que ILPPO proporciona un marc sòlid per incorporar consideracions ètiques en MOMARL, oferint noves perspectives i obrint camí per a futures investigacions en entorns més complexos.

Resumen

Este Trabajo de Fin de Grado estudia la integración de principios éticos en el aprendizaje por refuerzo multiobjetivo y multiagente (MOMARL) mediante la implementación y evaluación del algoritmo Independent Lexicographic Proximal Policy Optimization (ILPPO). En el aprendizaje por refuerzo multiagente (MARL), las interacciones dinámicas entre agentes pueden hacer que el aprendizaje ético sea particularmente complejo. ILPPO aborda estos retos priorizando la toma de decisiones éticas mediante una ordenación lexicográfica de múltiples objetivos, asegurando que los objetivos éticos se cumplan antes de abordar otros objetivos individuales.
Comenzamos presentando los antecedentes necesarios sobre un proceso de decisión de Markov multiobjetivo (MOMDP), el algoritmo Proximal Policy Optimization (PPO) y el marco lexicográfico de aprendizaje por refuerzo, que forma la base del algoritmo LPPO.
A continuación, extrapolamos los tres elementos de este marco (MOMDP, PPO y LPPO) en el contexto de los juegos de Markov multiagente, donde cada agente aprende de manera independiente.
Una vez desarroyado el Independent LPPO (ILPPO), lo evaluamos en el "Ethical Gathering Game", un entorno donde los agentes aprenden a comportarse de acuerdo con el valor moral de la beneficencia. Nuestros experimentos demuestran que ILPPO puede aprender políticas éticas óptimas alineadas con valores éticos, similares a las obtenidas con Independent PPO (IPPO). Este estudio concluye que ILPPO proporciona un marco sólido para incorporar consideraciones éticas en MOMARL, ofreciendo nuevas perspectivas y abriendo camino para futuras investigaciones en entornos más complejos.

2020 Mathematics Subject Classification. 68T01, 68T05, 68T07, 68T20, 68T37, 68T42

Agraïments

La realització d'aquest treball de final de grau no hauria estat possible sense l'ajuda i suport de moltes persones, a les quals vull expressar el meu agraïment.

En primer lloc, vull agrair a la Dra. Maite López per donar-me l'oportunitat d'iniciar-me en el món de la investigació, per les explicacions i correccions, i per les revisions de la memòria, que han donat coherència i cohesió al treball.

També vull donar les gràcies al Dr. Manel Rodríguez per proposar un tema d'investigació tan interessant i pel temps dedicat a revisar aquest treball, estant sempre obert a resoldre els dubtes que anaven sorgint.

Al Dr. Juan Antonio Rodríguez per la seva implicació i encoratjament al llarg d'aquests mesos, proposant solucions quan es presentaven problemes.

A l'Arnau Mayoral per l'explicació i codi del IPPO, així com per resoldre tots els dubtes que tenia al respecte.

I finalment, a la meva família, per donar consells valuosos i ensenyar-me a relativitzar els problemes.

Nomenclature

The following table describes the meaning of various abbreviations and acronyms used throughout the document. Time-steps will be denoted as sub-indexes, while agents will be denoted as super-indexes in a variable. E.g., π_t^i, policy at time-step t for agent i.

Symbol	Description
RL	Reinforcement Learning
MDP	Markov Decision Process
MOMDP	Multi-Objective Markov Decision Process
MARL	Multi-Agent Reinforcement Learning
$\mathcal{S}$	Set of states
$\overline{\mathcal{S}}$	Subset of terminal states
$\mathcal{A}$	Set of actions
$\mathcal{A}(s)$	Set of actions that can be taken from state s
$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$	Reward function for single agent MDP
$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$	Transition function for single agent MDP
$\mu : \mathcal{S} \to [0, 1]$	Initial state distribution
$\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$	Policy
Max_t	Maximum number of time-steps in a trajectory
τ	Trajectory
$\mathbb{E}$	Expectation operator
$V^\pi(s)$	Value function of state s under policy π
$Q^\pi(s, a)$	Action-value function of the pair state-action under policy π
$V^*(s)$	Optimal value function
$Q^*(s, a)$	Optimal action-value function
Π_*	Set of all the optimal policies
Ω	Finite set of observations in a POMDP
$\mathcal{O} : \mathcal{A} \times \mathcal{S} \times \Omega \to [0, 1]$	Observation transition probability function in a POMDP
$\Gamma := (I, \{\mathcal{A}^i\}_{i \in I}, \{R^i\}_{i \in I})$	Normal-form game
$\langle \cdot \rangle$	Concatenation operator

Table 1: Nomenclature (Part 1)

Symbol	Description
PGM	Policy Gradient Methods
SGA	Stochastic Gradient Ascent
MSE	Mean Squared Error
LPPO	Lexicographic Proximal Policy Optimization
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
IQL	Independent Q-Learners
IPPO	Independent Proximal Policy Optimization
ILPPO	Independent Lexicographic Proximal Policy Optimization
ϵ	Tolerance value in lexicographic optimization
β^i	Learning rate for ith objective in lexicographic optimization
η^i	Weight assigned to difference between the most recent loss and the average loss in lexicographic optimization
c_t^i	Coefficient for ith objective at time-step t
A_t	Advantage estimate at time-step t
L^{CLIP}	Clipped objective function in PPO
H	Entropy bonus
L^V	Loss function for the value function network
$L^{\text{CLIP+H+V}}$	Combined objective function for policy and value function networks with entropy bonus
u_t	Discounted expected return
V_t^{target}	Target value function at time-step t
g^{PGR}	Policy gradient estimator in REINFORCE algorithm
g^{VR}	Variance reduced policy gradient estimator
g^A	Policy gradient estimator in Actor-Critic methods
$L_i(\theta, \lambda)$	Lagrangian function for ith objective
Γ_θ	Projection operator for policy parameters θ
Γ_λ	Projection operator for Lagrange multipliers λ

Table 2: Nomenclature (Part 2)

Chapter 1

Introduction

In recent years, as Artificial Intelligence systems become increasingly integrated in our everyday activities, their ethical behaviour has never been more critical. One of the most promising branches of AI are reinforcement learning (RL) algorithms. Within RL, multi-agent reinforcement learning (MARL) studies how multiple agents can learn to make decisions by interacting with each other and their environment to maximize cumulative rewards. This field stands out for its potential to address complex scenarios involving multiple agents. However, the dynamics generated between agents in a MARL introduce additional layers of complexity, making ethical learning particularly challenging.

One common approach to integrating multiple objectives in RL is through scalarized reward functions, where various objectives are combined into a single scalar value. This method simplifies the optimization process, but often fails to adequately represent the hierarchical nature of ethical decision-making. Scalarized reward functions may lead to misrepresentations that do not align with ethical priorities, as they blend multiple goals into a single objective value, without considering the relative importance of each goal.

This Bachelor's Degree Final Project focuses on addressing this challenge by introducing the Independent Lexicographic Proximal Policy Optimization (ILPPO) algorithm. ILPPO is designed to incorporate ethical decision-making into multi-objective MARL (MOMARL) by leveraging lexicographic ordering of multiple objectives. This approach ensures that higher-priority ethical objectives are satisfied before addressing other goals, thereby promoting ethical behavior in multi-agent environments.

The project is structured as follows:

- **Theoretical Foundations**: We begin by establishing the theoretical groundwork necessary for understanding ILPPO. This includes defining the Multi-Objective Markov Decision Process (MOMDP), which extends the traditional

Markov Decision Process to handle multiple objectives. We also provide an explanation of the Proximal Policy Optimization (PPO) algorithm. This algorithm will serve as the basis for ILPPO. Additionally, we introduce the lexicographic RL framework, which prioritizes objectives lexicographically for a single agent, and we extend this definition to multi-agent systems, where each agent learns independently.

- **Algorithm Development**: Using the principles of Lagrangian relaxation, we study a policy-based approach to solve MOMARL environments. This approach ensures that the optimization process respects the lexicographic ordering of objectives, focusing on higher-priority goals first. We document the modifications made to the standard IPPO algorithm to create ILPPO, emphasizing how it integrates multiple objectives in a multi-agent environment.

- **Implementation and Experiments**: To validate the effectiveness of ILPPO, we implement it in an environment called Ethical Gathering Game. This environment models a multi-agent scenario where multiple agents learn to behave in alignment with the ethical principle of beneficence. We conduct a series of experiments to compare ILPPO with the standard IPPO algorithm, evaluating its performance in terms of ethical behaviour and overall efficiency.

- **Results and Analysis**: Our experimental results demonstrate that with ILPPO agents learn an ethical optimal policy, where the efficient agent learns to display beneficence and helps the inefficient agent to survive. We present detailed analyses of the agents' behaviours, convergence metrics and ethical compliance, highlighting the advantages of ILPPO algorithm.

- **Conclusions and Future Work**: The project concludes summarizing the main findings and contributions. We outline possible directions for future research, including scaling ILPPO to more complex environments.

Through this project, we aim to provide an algorithm that guarantees that all agents in the system learn to behave ethically in a multi-agent setting. By addressing the limitations of scalarized reward functions and employing a lexicographic approach, ILPPO represents a significant advancement in aligning agent behaviour with ethical principles.

Chapter 2

Background

This chapter will provide the necessary preliminary knowledge to understand how the Proximal Policy Optimization algorithm works and how to apply a lexicographic order to the policies generated by each reward. Section 2.1 will provide a brief definition of Reinforcement Learning and introduce its main pillars: what is an agent, a policy and a reward. We will begin by presenting the model we will work with, the Markov Decision Process, and how to optimize policies with expected discounted rewards and value functions. With this knowledge, we will define Q-learning, an algorithm that learns optimal policies by computing value functions.
Section 2.8 will delve into Proximal Policy Optimization (PPO), a key algorithm in reinforcement learning. We will explain the motivation behind PPO, its underlying principles, and its advantages over previous policy gradient methods. The section will cover the mathematical formulation of PPO, including the objective functions and clipping strategies that stabilize training. We will also discuss the practical aspects of implementing PPO, such as the choice of neural network architectures and hyper-parameter settings.
Following the introduction to PPO, Section 2.9 will introduce the concept of lexicographic optimization in reinforcement learning. This section will explain how multiple objectives can be prioritized and how the lexicographic Proximal Policy Optimization (LPPO) algorithm integrates this prioritization into the training process. We will describe the formulation of LPPO, the role of the Lagrangian relaxation technique in handling constraints, and how LPPO ensures that higher-priority objectives are optimized before considering lower-priority ones.
Once the foundation has been established for single agent environments, we will proceed to define the model for multi-agent environments in Sections 2.10 and 2.11, with a particular emphasis on game theory. We will furnish definitions for various types of games, based on the number of agents, possible actions, and the

number of iterations. The necessary information for writing sections 2.1 to 2.7 has been obtained from the book on Multi-Agent Reinforcement Learning by Albrecht et al. [1].

2.1 Reinforcement Learning

In Reinforcement Learning (RL), an agent learns to behave by interacting with the environment. Each action has its corresponding reward or punishment. The objective of the agent is to learn a behaviour that maximizes its gain of rewards.

A helpful way to grasp these abstract concepts is through an example:
Consider the scenario of training a dog to get a stick. Since the dog does not understand our language, it is impossible to communicate what we want it to do verbally. Instead, we adopt a different strategy. We create a situation (throwing the stick), and the dog attempts various responses. If the dog's response aligns with our desired behavior (get it as soon as possible), we reward it with snacks. Consequently, the next time the dog encounters a similar situation, it tends to repeat the action faster and more efficiently, anticipating more food. This exemplifies learning how to behave, striving to maximize the reward of snacks. Similarly, dogs learn what actions to avoid when faced with negative experiences.

That's how Reinforcement Learning works in a broader sense:

- The dog is the agent, that is interacting with the environment. This environment could be a physical space, such as a room or a yard, where the dog performs actions and receives feedback.
- The situation the dog encounters is analogous to a state. An example of a state could be the dog standing and staring at you impatiently as you keep the stick in your hands. Or the state in which the stick is no longer in your hands, and the dog has to pick it up.
- Our agent reacts to a state by performing an action, causing the environment to transition from one state to another. In our example, when the dog catches the stick and brings it back to you, it is performing an action to change between states. The environment, once the dog action is applied, changes to a new "stick-retrieved" state.
- After the transition, they may receive a reward or penalty in return. You can give them a treat or a "No" as a penalty.
- The policy is the strategy of choosing an action given a state in expectation of better outcomes, or snacks.

In the following section, we will formally define these concepts within a framework called the Markov decision process.

2.2 Markov Decision Processes

The standard model used in single agent RL to define a sequential decision process is called the Markov decision process:

Definition 2.1 (Markov decision process)**.** *A finite Markov decision process (MDP) consists of:*

- *A finite set of states $\mathcal{S}$, with a subset of terminal states $\overline{\mathcal{S}} \subset \mathcal{S}$,*
- *a finite set of actions $\mathcal{A}$,*
- *a reward function $R : S \times A \times S \to \mathbb{R}$,*
- *a state transition probability function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ such that*

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A} : \sum_{s' \in \mathcal{S}} T(s, a, s') = 1, \tag{2.1}$$

- *and an initial state distribution $\mu : \mathcal{S} \to [0, 1]$ such that*

$$\sum_{s \in S} \mu(s) = 1 \quad \textit{and} \quad \forall s \in \overline{\mathcal{S}} : \mu(s) = 0. \tag{2.2}$$

An MDP starts at an initial state s_0 which is stochastically drawn from the so-called initial state distribution, denoted mathematically as $s_0 \sim \mu(s)$. This function μ assigns probability values to the elements in $\mathcal{S}$, indicating how probable it is for any given state to be chosen as an initial state. At time t, the agent observes the current state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$ with probability given by its decision-making strategy, called policy.
A policy is map

$$\begin{aligned} \pi : \mathcal{A} \times \mathcal{S} &\to [0, 1] \\ \pi(a, s) &= Pr(\mathcal{A}_t = a \mid \mathcal{S}_t = s) \end{aligned} \tag{2.3}$$

that gives the probability of taking action a when in state s. It is often written as a conditioned probability to highlight the fact that the action depends on the specific state. Different states may have different allowed actions to take.
Given the state s_t and the action a_t sampled with the policy, the MDP transitions into the next state s_{t+1} with probability given by the state transition probability

function $T(s_t, a_t, s_{t+1})$. Once transitioned to the new state, the agent receives a reward $r_{t+1} = R(s_{t+1} \mid s_t, a_t)$, a scalar feedback that indicates how well the agent is doing at time-step t. The agent's objective is to maximize the total reward it receives over the long run.
These steps are repeated until the process reaches a terminal state $s_t \in \overline{S}$ or after completing a maximum number of time-steps Max_t, in which the process terminates. Each independent run of the above sequence of interactions between the agent and the environment is called an episode. The set of states, actions and rewards carried out in an episode is called a trajectory

$$\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{Max_t-1}, a_{Max_t-1}, r_{Max_t}\}. \tag{2.4}$$

Dealing with an MDP has the following implication: an agent only needs information about the current state to choose optimal actions. This comes from the Markov property of all MDP, which states that the future states and rewards are conditionally independent of past states and actions, given the current state and action

$$\Pr(s_{t+1}, r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = \Pr(s_{t+1}, r_{t+1} \mid s_t, a_t). \tag{2.5}$$

2.3 Expected Discounted Returns

To measure the overall performance of a policy π, we can compute the (expected) discounted sum of rewards that the agent obtains by following it

$$u_t = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}]. \tag{2.6}$$

The expectation operator allows us to compute the average reward that we expect to receive when following a certain policy over a sequence of actions. And the discount factor $\gamma \in [0, 1)$ makes it possible to compare the performance of policies in MDPs without terminal states.
The discount factor determines the "weight" the agent gives to the reward: a reward received $k+1$ time-steps in the future is only worth γ^k times what it would be worth if it were received now. For $\gamma < 1$ and $\{r_{t+k+1}\}_{k=0}^{\infty}$ bounded, the infinite sum in Equation 2.6 has a finite value.
For $\gamma \approx 0$, the agent is only concerned with maximizing immediate rewards; its objective is to learn how to choose a_t in order to maximize r_{t+1}. As $\gamma \approx 1$, the return objective takes future rewards into account more strongly.

2.4 Value Functions

Almost all reinforcement learning algorithms involve estimating value functions, which estimate "how good" it is for the agent to be in a given state. The term "how good" here refers to the future rewards the agent expects to receive if it follows the policy from the current state s onwards. Or, more precisely, the expected return it expects to receive.
From the Markov property defined in Equation 2.5, we can deduce that future rewards are independent of past rewards. Thus, the expected discounted returns no longer must be defined for the entire trajectory, but can be defined for each individual state $s \in S$. This operation is formalised by the so-called value function, defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \mathcal{S}_t = s]. \tag{2.7}$$

Denoted by $V^{\pi}(s)$, is the expected return when starting in s and following the current policy π. Note that the value of any terminal state is always zero. Being tightly connected to the concept of value functions are the so-called action-value functions $Q^{\pi}(s, a)$, which give the expected return once selected action a in state s and then following policy π to navigate to future states

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \mathcal{S}_t = s, \mathcal{A}_t = a]. \tag{2.8}$$

The value and action-value functions can be estimated from experience. If an agent that follows a policy π maintains an average of the expected returns for each encountered state, then the average will converge to the state's value $V^{\pi}(s)$ as the number of times that state is explored approaches infinity.
Similarly, if separate averages are kept for each action taken in each state, then these averages will similarly converge to the action-values $Q^{\pi}(s, a)$. Nevertheless, when dealing with numerous states, it is impractical to maintain averages for each state individually. Instead, the agent can store Q^{π} and V^{π} as parameterized functions and adjust the parameters to more accurately align with the encountered rewards.
Value functions are used, one way or another, in almost every RL algorithm. An important property of value functions is that they satisfy a recursive relationship. Given any policy π and any state s, the following condition holds between the

value of s and the value of its successor states

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \mathcal{S}_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \mid \mathcal{S}_t = s] \\ &= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s, a)[r(s, a, s') + \gamma \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid \mathcal{S}_{t+1} = s']] \\ &= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s, a)[r(s, a, s') + \gamma V^{\pi}(s')]. \end{aligned} \tag{2.9}$$

The recursive formula given in Equation 2.9 is called the Bellman equation. It expresses a relationship between the value of the current state s and the value of its successor states.

Analogous of recursive value functions, we can define recursive action-value functions $Q^{\pi}(s, a)$

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \\ &= \sum_{s' \in S} P(s' \mid s, a)[r(s, a, s') + \gamma V^{\pi}(s')] \\ &= \sum_{s' \in S} P(s' \mid s, a)[r(s, a, s') + \gamma \sum_{a' \in A} \pi(a' \mid s) Q^{\pi}(s', a')]. \end{aligned} \tag{2.10}$$

2.5 Optimal Policy

In a MDP environment, there are different value functions according to different policies. The optimal value function is one that yields maximum value compared to all other value functions. Solving a MDP means finding the optimal value function. Mathematically, optimal value functions can be expressed as:

$$V^*(s) = \max_{\pi} V^{\pi}(s). \tag{2.11}$$

Where V^* tell us the maximum reward we can get from the system. Thus, maximizing the expected return in a MDP is equal to maximizing the expected return in each possible state $s \in S$.

Likewise, the optimal action-value function indicates the maximum reward attainable in state s upon selecting action a and subsequently continuing from that point onward.

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a). \tag{2.12}$$

Now that we have defined the basic components for optimality, we can define what is meant by optimal policy. Observe that value functions define a partial ordering over policies. A policy π is better than other policies π' if its expected return is greater or equal to that of π' for all states.

$$\pi \geq \pi' \text{ if } V^{\pi}(s) \geq V^{\pi'}(s), \forall s. \tag{2.13}$$

There is always at least one deterministic optimal policy for every finite MDP. Note that there can be more than one optimal policy in a MDP. But all optimal policies achieve the same optimal value functions and optimal action-value functions. We denote all the optimal policies Π_*.
We find an optimal policy maximizing over our action-value function $Q^*(s,a)$. The idea is to solve $Q^*(s,a)$ for each possible action and then pick the one that gives us the greatest reward.

$$\pi_*(a \mid s) = \begin{cases} 1, & \text{if } a = arg\max_{a \in A} Q^*(s,a) \\ 0, & \text{otherwise} \end{cases} \tag{2.14}$$

For state s, we select an action with probability 1 if it returns the largest value of $Q^*(s,a)$. So, in order to identify an optimal policy, we simply have to focus on finding $Q^*(s,a)$. How can we find these $Q^*(s,a)$? This is where Bellman equation (2.9) will be handy.
The Bellman Optimallity equation is identical to the Bellman equation, with the exception that instead of choosing the average of the actions our agent can perform, we select the action with the highest value. Specifically, the best action taken in that state.

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^*(s,a) \\ &= \max_{a} \sum_{s' \in S} P(s' \mid s,a)[r(s,a,s') + \gamma V^*(s')]. \end{aligned} \tag{2.15}$$

Similarly, the Bellman optimality equation for $Q^*(s,a)$ is

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma \max_{a'} Q^*(s',a') \mid \mathcal{S}_{t+1} = s', \mathcal{A}_t = a'] \\ &= \sum_{s' \in S} P(s' \mid s,a)[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')]. \end{aligned} \tag{2.16}$$

The Bellman Optimallity equations define a system of non-linear equations, one for each state. So, if there are N states, then there are N equations with N unknowns. The non-linearity is due to the max-operator used in the equations.
If the dynamics of the environment are known ($P(s' \mid s,a), R(s,a,s')$) one can solve this system of equations for V^* using a convenient method for solving non-linear

equations. One can solve a related set of equations for Q^{π_*}.
Once we know an optimal action-value function Q^*, an optimal policy π_* can be derived by choosing actions with maximum value, as described in Equation 2.14. Written as a a shorthand,

$$\pi_*(s) = arg\max_{a \in A} Q^*(s, a). \tag{2.17}$$

Which assigns a probability 1 to the action that maximizes the action-value function at state s.

2.6 Value Based Methods

Value-based methods are a class of algorithms in reinforcement learning (RL) that focus on estimating the value functions to derive optimal policies. These methods are based on the Bellman equations, which provide a recursive decomposition of value functions, facilitating their computation. Among the value-based methods, Temporal-Difference (TD) learning is particularly important for its ability to learn directly from raw experience without a model of the environment.

2.6.1 Temporal-Difference Learning

Temporal-Difference (TD) learning is a family of RL algorithms that learn value functions and optimal policies from past interactions with the environment. TD learning does not require complete knowledge of the Markov Decision Process (MDP). Instead, the information needed is collected solely by interacting with the environment.
TD algorithms update action-value functions using the following rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\chi - Q(s_t, a_t)]. \tag{2.18}$$

where χ is the target, and $\alpha \in (0, 1]$ is the learning rate. The target χ is constructed from interactions with the environment $(s_t, a_t, r_{t+1}, s_{t+1})$[1] . Depending on how we construct the target value, we can implement different TD algorithms, which impact how they learn.

Q-learning

Q-learning is a TD algorithm that uses the information tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered from the environment to compute the following χ:

$$\chi = r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a'). \tag{2.19}$$

[1]Note that the reward function when transitioning from state s_t and applying the action a_t returns the reward r_{t+1}.

This leads to the complete Q-learning update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s_t, a_t)]. \tag{2.20}$$

Q-learning is guaranteed to converge to the optimal policy π_* provided that all state-action pairs $(s, a) \in S$ are visited infinitely often and the learning rate α is decayed to zero over time. Unlike other TD methods, Q-learning does not require the policy used to interact with the environment to be adjusted gradually toward the optimal policy. Instead, it directly uses the maximum possible reward of the next state, disregarding the current policy.

2.7 Policy Based Learning

Suppose you are in a new town and you want to reach downtown. The only problem is that you have neither a map nor a GPS. You can try to assess your current position relative to your destination, as well as the effectiveness (value) of each direction you take. Or you can ask a local, who will tell you to go straight, and when you see a fountain, turn to the left and continue until you reach downtown. This is a policy to follow. Clearly, in this case, following a policy is much easier than computing the value function on your own.
So the main idea is to be able to determine in each state s which action to take in order to maximize the expected return. The way to achieve this objective is to fine-tune a vector of parameters θ in order to select the best action to take for policy π. The policy is noted as $\pi(a \mid s, \theta) = Pr(a \mid s, \theta)$, which means that the policy π is the probability of taking the action a when at state s and the parameters are θ. These policy-based learning algorithms have the important advantage that they can directly learn the action probabilities of policies based on gradient-ascent techniques. As we will see, these algorithms achieve interesting convergence properties when varying the action probabilities.

2.8 Proximal Policy Optimization

The perception and representation of the environment are one of the key problems that must be solved before the agent can decide to select an optimal action to take. In RL tasks, a human expert provides features of the environment based on his knowledge of the task. However, this causes issues with the environment scalability and is limited to low-dimensional problems[10].
This is where Neural Networks fit in. Neural Networks are function approximates, which are particularly useful when the state space or action space are too

large or unknown. A NN can be used to approximate a value function, or a policy function. That is, NN can learn to map states to values, or state-action pairs to state-action values. Instead of using a lookup table to store all possible states and their values, we can train a NN on samples of the state or action space to learn to predict how valuable those are in relation with our objective.
NNs use coefficients (θ) to approximate the function relating inputs and outputs. Their learning process involves iteratively adjusting these coefficients (weights) along gradients to minimize error. Given a set of pixels that represents a state in a Super Mario game, a NN can rank the possible actions to perform in that state. For example, it might predict that running will have a return of 5 points, jumping of 7 and staying in place none.
At the beginning of the training, the NN coefficients are initialized stochastically. Feedback from the environment is then used to adjust these weights, improving the NN's ability to interpret state-action pairs and optimize actions accordingly [4].
In Deep Reinforcement Learning (DRL), NNs are employed to approximate either the value function or the policy function, or sometimes both. By approximating these functions, DRL enables agents to learn intricate strategies in environments with vast state-action spaces, such as Super Mario games.
However, these algorithms are highly sensitive to hyper-parameter settings and initialization. For example, if the learning rate is too high, it can push the policy network into a parameter space region where the collected data is inadequate for the current policy, making it difficult or impossible for the system to recover or converge [8].
To tackle these challenges, Schulman et al. (2017) developed a new algorithm called Proximal Policy Optimization (PPO) [8]. The primary aim of PPO is to achieve a balance between simplicity of implementation, efficiency in using batches of data, and ease of tuning.

2.8.1 Policy Gradient Methods

PPO is based on the principles of policy gradient methods (PGMs). In the last sections, we saw two methods to find an optimal policy π_*:

- In **value-based methods**, we want to learn a value function. By minimizing the loss between predicted and target value functions, we can approximate the true action-value functions. And from this approximation, we can obtain an optimal value function, which leads to an optimal policy π_* if we choose actions that maximize the action-values at each state. We have a policy, but it's implicit since it is generated directly from the value function.

- In **policy-based methods**, we directly learn to approximate π_* without having to learn a value function. The idea is to parameterize the policy using the weights and biases of a neural network θ. This policy π_θ will output a probability distribution over actions (stochastic policy).

$$\pi_\theta(a_t \mid s_t) = P(a_t \mid s_t; \theta). \tag{2.21}$$

  Our objective is to maximize the expected cumulative reward following a parameterized policy using gradient ascent. To do that, we control the parameter θ that will affect the distribution of actions over a state.

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]. \tag{2.22}$$

  where τ is a sampled trajectory[2] of the policy and $R(\tau)$ is the cumulative reward following the trajectory.

$$R(\tau) = \sum_{t=0}^{Max_t - 1} R(s_t, a_t). \tag{2.23}$$

Consequently, thanks to policy-based methods, we can directly optimize our policy π_θ to output a distribution over actions $\pi_\theta(a \mid s)$ that leads to the best cumulative reward. To do that, we define an objective function $J(\theta)$, the expected cumulative reward, and find the value θ that maximizes this objective function.

Policy-gradient methods are a subclass of policy-based methods. The difference between these two methods lies on how we optimize the parameter θ.

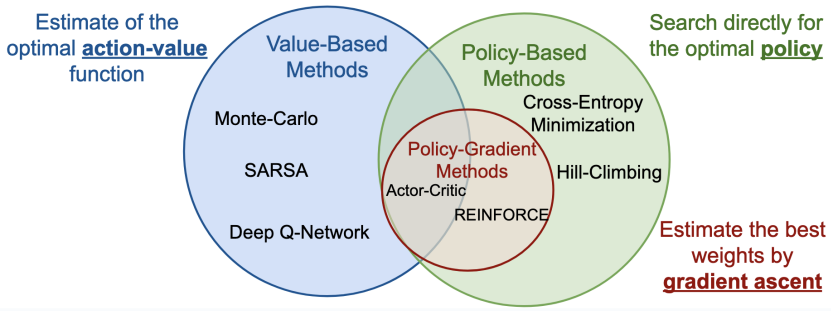


Figure 2.1: Classification of methods to find optimal policies

- In **policy-based methods** we search directly for the optimal policy. But we do it optimizing indirectly the parameter θ by maximizing the local approximation of the objective function with techniques like Hill Climbing or Cross Entropy Minimization.

[2] State-action sequence with horizon Max_t, $s_0, a_0, s_1, a_1, ..., s_{Max_t - 1}, a_{Max_t - 1}$.

- In **policy-gradient methods** we also search directly for the optimal policy. But we optimize the parameter θ directly by performing the gradient ascent on the performance objective function $J(\theta)$.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t). \tag{2.24}$$

The idea behind policy-gradient is to control the probability distribution of actions. Actions that maximize the cumulative reward must be sampled more frequently in the future. So, each time the agent interacts with the environment, we can tweak the parameters such that "good actions" will be sampled more likely.
We are going to let the agent interact with the environment during an episode. If the agent accumulates a high reward in the episode, we want to increase the probability of each state-action pair leading to this success. Conversely, we want to decrease the probability of state-action pairs that led to lower rewards.
To measure the performance of our policy, we use the objective function defined above, $J(\theta)$, which outputs the expected cumulative reward.

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \\ &= \sum_\tau P(\tau;\theta) R(\tau) \\ &= \sum_\tau [\prod_{t=0}^{Max_t - 1} P(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)] R(\tau). \end{aligned} \tag{2.25}$$

Our objective is to maximize the expected cumulative reward by finding the θ that outputs the action probability distribution that leads to higher rewards.

$$\max_\theta J(\theta) = \max_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]. \tag{2.26}$$

As it is a maximization problem, we need to use the gradient-ascent, seeking the direction of the steepest increase of $J(\theta)$. The update step for gradient ascent is

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{2.27}$$

The derivative of $J(\theta)$ can be computed applying the Policy Gradient Theorem, that helps to reformulate the objective function into a differentiable function.

Theorem 2.2 (Policy Gradient Theorem)**.** *The derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy* π_θ

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) R(\tau)]. \tag{2.28}$$

Now, expanding the definition of π_θ

$$\pi_\theta(\tau) = \mu(s_0) \prod_{t=0}^{Max_t - 1} \pi_\theta(a_t \mid s_t) P(s_{t+1}, r_{t+1} \mid s_t, a_t). \tag{2.29}$$

Equivalently, taking the log probabilities, we have

$$\begin{aligned} \log \pi_\theta(\tau) &= \log \mu(s_0) + \sum_{t=0}^{Max_t - 1} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{Max_t - 1} \log P(s_{t+1}, r_{t+1} \mid s_t, a_t). \\ \nabla_\theta \log \pi_\theta(\tau) &= \sum_{t=0}^{Max_t - 1} \nabla_\theta \log \pi_\theta(a_t \mid s_t). \end{aligned} \tag{2.30}$$

So, substituting the equivalence of the derivatives of the log probabilities obtained in Equation 2.30 into the derivative of the objective function 2.28 we obtain

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)(\sum_{t=0}^{Max_t - 1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)))]. \tag{2.31}$$

This is a very useful result, as it tells us that we do not need to know the transition probability distribution (and the reward function) associated with the Markov decision process (MDP) in order to find the parameter θ that converges to a value that maximizes $J(\theta)$. These types of algorithms are called model-free, because they do not model the transition nor the reward function of the environment [10]. An immediate implication of the fact that the gradient is reduced to an expectation is that we can approximate the gradient with the empirical estimate for m sample trajectories under policy π_θ as follows:

$$g^{PG} = \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{Max_t - 1} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}), \tag{2.32}$$

where i iterates over m sample trajectories, $R(\tau^{(i)})$ is the cumulative reward following the trajectory $\tau^{(i)}$ and $a_t^{(i)}, s_t^{(i)}$ are the action and state obtained at time-step t under trajectory $\tau^{(i)}$. This approximation of the gradient g^{PG} is the computed gradient for policy-based methods.

The REINFORCE algorithm

Given the stochasticity of the environment and the policy, the same initial state can lead to very different cumulative rewards $R(\tau)$, which can lead to high variance in the estimator g^{PG}. Because of this, the cumulative reward starting at the

same state can vary significantly across episodes [10]. Instead, we can use the return u_t defined in Equation 2.6 recalling the Markov property that future rewards are independent of past rewards. Hence, if we replace $R(\tau)$ by the discounted expected return u_t, we arrive at the REINFORCE Policy Gradient algorithm with the following policy gradient estimator (PGR) [11]:

$$g^{PGR} = \nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[(\sum_{t=0}^{Max_t-1} u_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)))]. \quad (2.33)$$

However, we have not yet solved the problem of variance in the sampled trajectories. In order to reduce the variance of the policy gradient estimator g^{PGR} we can opt for subtracting a baseline estimate b_t from the expected return u_t from Equation 2.33 leading to the following variance reduced estimator (g^{VR}) of $\nabla_\theta J(\theta)$

$$g^{VR} = \mathbb{E}_{\tau\sim\pi_\theta}[(\sum_{t=0}^{Max_t-1} (u_t - b_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t)))]. \quad (2.34)$$

Using a baseline, in both theory and practice reduces the variance while keeping the gradient unbiased. As we will see in the following section, a good baseline would be to use the value function.

Actor-Critic Methods

Actor-Critic methods lie at the intersection of value-based and policy-gradient methods, as shown in Figure 2.1. These methods leverage the strengths of both approaches to stabilize and improve learning. In Actor-Critic methods, finding a good baseline is essential but challenging. We approximate this baseline using the parameters ω[3] (weights and biases) of a Neural Network (NN), which generates the value functions $V_\omega(s_t)$. This means that the baseline b_t is approximated by $V_\omega(s_t)$ [10].
It is important to note that Actor-Critic methods involve two neural networks: one for the actor and one for the critic. The actor's neural network is parameterized by θ and is responsible for selecting actions, while the critic's neural network is parameterized by ω and evaluates those actions by estimating value functions. This dual-network setup allows the actor and the critic to work in opposition: the actor seeks to improve its actions, while the critic assesses the quality of those actions [2].
The term u_t represents an estimate of the action-value function $Q(s_t, a_t)$, which is associated with taking action a_t in state s_t. This relation can be expressed as

[3] As we will specify later, we will use ω for the parametrization of the critic's neural network, and θ for the actor's neural network.

$u_t \approx Q(s_t, a_t)$, and consequently, $u_t - b_t \approx Q(s_t, a_t) - V_\omega(s_t)$. The difference $u_t - b_t$ is known as the advantage estimate, defined as:

$$A_t = Q(s_t, a_t) - V_\omega(s_t). \tag{2.35}$$

Intuitively, A_t indicates how much better or worse it was to perform action a_t in state s_t. This is measured by comparing $u_t \approx Q(s_t, a_t)$ to the expected value $V_\omega(s_t)$ for being in state s_t under the policy π_θ [2].
This leads to the following gradient estimator:

$$g^A = \mathbb{E}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) A_t]. \tag{2.36}$$

In Actor-Critic methods, we bootstrap the gradient using the learnable $V_\omega(s)$. Bootstrapping involves updating estimates based on other estimates, allowing the critic to provide a more stable training signal to the actor. Here, the actor represents the policy π_θ, deciding which actions to take, while the critic is the state value network $V_\omega(s)$, assessing the quality of those actions. The critic's role is to predict the value of the current state, providing a baseline against which the advantage is calculated. The actor's role is to improve its policy based on the feedback from the critic. This creates a dynamic where the actor and critic are in opposition, with the actor seeking to improve its actions and the critic evaluating those actions [10]. One potential issue with policy-gradient methods is that too large updates to the policy's parameters θ can move the parameter vector away from a local maximum of the objective function $J(\theta)$. This risk is heightened when performing multiple epochs[4] of parameter updates on the same set of freshly collected training data. To mitigate this, Proximal Policy Optimization (PPO) uses a clipped objective function. The clipped objective function prevents large changes to the policy by limiting the update size, ensuring that the new policy does not deviate excessively from the old one. This helps maintain stability and improves convergence during training [8].

2.8.2 Overview of the algorithm

Proximal Policy Optimization (PPO) is a Deep Reinforcement Learning (DRL) algorithm from the class of policy gradient methods (PGMs). PPO aims for improving upon PGMs sample efficiency by employing an objective function that allows multiple epoch of updates of its trainable parameters θ based on the same training data (batch). Moreover, PPO is an on-policy algorithm, meaning it evaluates and improves the same policy that is used to select actions. Since it is a

[4] In the context of Deep Learning (DL), an epoch refers to a complete pass through the entire batch of training data, both forward and backward through the neural network.

PGM, PPO is also a model-free algorithm. A model-free algorithm does not require a model of the environment, such as the transition probabilities and reward functions, to make decisions. Instead, it learns directly from interactions with the environment, making it more flexible and easier to apply to a wide range of problems where the environment's dynamics are unknown or complex [8].
Consider a PPO agent whose task is to map for each observed state $s_t \in S$ a single action $a_t \in A$ to be performed in s_t. Actions are selected by means of a policy π_θ where θ denotes the policy's trainable parameters, the weights and biases of a Neural Network. The NN inside PPO's policy is used to generate the parametrization for some probability distribution that is used to sample the action. Action a_t is selected with probability $\pi_\theta(a_t \mid s_t)$ given the current set of parameters θ. Upon executing action a_t, the agent transitions into state s_{t+1} and receives a reward r_{t+1}. During training, the set of parameters θ is repeatedly updated in incremental steps using SGA in a way such that an approximation of the expected return $J(\theta)$ gets maximized.
In more detail, generating action a_t through the policy π_θ requires three steps, dividing the policy into two separate portions: the stochastic portion of the policy π^s_θ and the deterministic portion of the policy π^d_θ.

1. **State Representation:** A PPO agent receives a state representation s_t. This state representation is passed through the policy network (the NN parameterized by θ), constituting the deterministic portion of the agent's policy π^d_θ.

2. **Parameter Computation:** This results in a set of parameters ϕ_t being computed in the policy network's output layer. This set of parameters ϕ_t is then used to parameterize a probability distribution[5] δ_t which is defined over the agent's action space A.

3. **Action Selection:** Finally, an action a_t is sampled from the action space A in accordance with δ_t in the stochastic portion of π^s_θ.

While PPO main objective is to train the agent's deterministic policy network π^d_θ, training the agent actually involves training two networks concurrently as described in the context of REINFORCE. The first network is the policy network π^d_θ, while the second is the value function network V_ω. The value function is used to reduce the variance in the numeric estimates in which the policy network is trained [8].
Training the agent means repeatedly alternating between two steps:

[5]Such as the multinomial distribution, where k is the number of actions that can be taken from state s and each one has its corresponding probability parameterized by $\phi_t = (p_1, \ldots, p_k)$.

- **Data collection step**, or rollout: In which the PPO agent is made to interact with the environment during m episodes, each of Max_t time-steps. In all this episodes the agent uses its most up-to-date state of the policy network, which is held fixed during the data collection step. For each experienced state transition, the corresponding state s_t, next state s_{t+1}, action a_t, probability of selecting action a_t in s_t given π_θ denoted as $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$, and the corresponding reward r_{t+1} get stored in a buffer of the form $o_t = (s_t, a_t, \pi_{\theta_{\text{old}}}(a_t \mid s_t), s_{t+1}, r_{t+1})$ in order to be used in the update step. For each observed state transition, two additional target values are computed and appended to the buffer. The first one is the expected return V_t^{target}. The second is the advantage A_t as described in the Actor-Critic methods 2.8.1. The computations of V_t^{target} and A_t will be described in detail in sections 2.8.3 and 2.8.4.
- **Data generation step**, or update: After all the freshly training data has been collected and stored in the buffer, the trainable parameters of the policy network π_θ and the value function network V_ω get updated using multiple epochs for the same batch in the buffer.

The objective function used by PPO has been designed to avoid destructive large weight updates of the policy network and perform multiple epoch of weight updates over the same batch of training data. As explained in REINFORCE section 2.8.1, this is meant to increase data efficiency compared to vanilla PGMs. The data used to update the weights is exclusively the one stored in the buffer from the rollout step. The clipped objective function is defined as follows:

$$J(\theta) = L^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t]. \tag{2.37}$$

where $\mathbb{E}_t$ performs an average over the finite set of training data in the batch. Note that the subscript t has two meanings, depending on which training step we are. Subscript t denotes the index of a randomly sampled training observation taken from the batch during the update step. While, during the rollout step, t denotes the time-step inside the environment when the information contained in an observation o_t has been observed.
The term $r_t(\theta)$ in Equation 2.37 refers to the probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{2.38}$$

where $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ is the probability of action a_t in state s_t with the policy used during the rollout step. And $\pi_\theta(a_t \mid s_t)$ refers to the probability of a_t in s_t given the most up-to-date state of the policy, the one we want to refine. The $\min(\cdot,\cdot)$ operator is used to return the minimum of its two input values. Whereas the

clip operator clips its input value $r_t(\theta)$ between the range $[1-\epsilon, 1+\epsilon]$. If the probability ratio falls outside the range $[1-\epsilon, 1+\epsilon]$ the clip function will output the value in the border of the interval. Finally, ϵ is a hyper-parameter that helps us define this clip range. For reference, in the paper [8], the ϵ of the clip range is assigned a value of $\epsilon = 0.2$. We will explain further its individual terms and why this definition of the objective function let us perform multiple epoch based on the same batch in section 2.8.5.

In theory, once we have defined our policy network's objective function L^{CLIP} we can optimize it (i.e. maximize it) using stochastic gradient ascent. However, standard deep learning libraries used to train DRL only support SGD, not SGA. Therefore, it is a widespread practice use SGD to minimize $-L^{\text{CLIP}}$, treating the objective function as a loss function to be minimized.

In order to encourage exploratory behaviour of the policy network π_θ during training, we can add an entropy bonus H to the policy network's objective function L^{CLIP} and its weighted factor h. This factor allow us to control the contribution of the entropy bonus to the overall function. Adding this weighted entropy bonus to our clipped objective function results in:

$$L^{\text{H}} = L^{\text{CLIP}} + hH. \tag{2.39}$$

Which is then maximized using SGA, or can converted into $-L^{\text{H}}$ in order to minimize it using SGD.

The value function network V_ω is trained minimizing the mean square error (MSE) over the multiple observations in the batch. The predicted value that we want to approximate is $V_\omega(s_t)$, the output of the critic's network. The target value used to approximate it is denoted V_t^{target} and will be further explained in section 2.8.3. The corresponding loss function of the MSE for training the value function network V_ω is defined as:

$$L^V = \mathbb{E}_t[(V_\omega(s_t) - V_t^{\text{target}})^2]. \tag{2.40}$$

where V_t^{target} refers to the expected return collected during the rollout with the critic's NN old parameters ω_{old}. And $V_\omega(s_t)$ refers to the predicted expected return from the critic's NN updated parameters ω. Finally, we can define the overall objective function taking into account all the factors involved:

$$L^{\text{CLIP+H+V}} = L^{\text{CLIP}} + hH - vL^V. \tag{2.41}$$

where, just as the scalar h described for the entropy, the value v is a weighting factor for the value function loss.

The goal is to optimize $L^{\text{CLIP+H+V}}$ using stochastic gradient ascent. Or, equivalently, minimize $-L^{\text{CLIP+H+V}}$ using stochastic gradient descent. Commonly, the Adam optimizer algorithm is used to perform SGD to minimize $-L^{\text{CLIP+H+V}}$.

2.8.3 Return

The return is essential for evaluating the performance of actions taken by the agent, providing a measure of the cumulative reward that can be expected from any given state. This information is crucial for updating both the value function and the policy [10].
One method to compute this returns is through a target value function V_t^{target}, which represents the discounted cumulative reward of taking action a_t in state s_t. First an agent is made to interact with the environment for a given maximal trajectory length Max_t. When the trajectory ends, the expected returns are computed for every state s_t experienced during the trajectory following the equation:

$$V_t^{\text{target}} = \sum_{k=0}^{Max_t-1} \gamma^k r_{t+k+1}. \tag{2.42}$$

where V_t^{target} is the target value function associated with a given state s_t, r_{t+1} is the reward received after executing action a_t in state s_t and $\lambda \in (0, 1]$ is the discount factor. This recursive formula is applied backwards on multiple episodes, and its values are stored in the observations batch during the rollout step [10].
In the context of training a PPO agent, target value functions are used in two different parts: they are used to train the value function network V_ω. As well as to compute advantage estimates A_t for training the policy network.

2.8.4 Advantage Estimates

Advantage estimates are crucial for improving the efficiency and stability of policy gradient methods. They help the agent understand the relative value of actions taken, allowing it to focus on actions that yield better outcomes compared to others[10].
As described in the REINFORCE subsection of Policy Gradient Methods 2.8.1, the advantage estimate A^t determines how much better or worse the observed outcome of choosing certain action at a given state was compared to the state's estimated value predicted by the value function network. Intuitively, it expresses how much better is the action chosen than the other possible actions on average.
In order to compute the advantage A^t, we need two things:

- Discounted sum of rewards, or return: measured by the target value function V^{target}, computed as described in the above subsection 2.8.3.
- A baseline estimate: evaluation of the value function in state s_t predicted by the value function network during the rollout step (with old weights and biases θ_{old}).

The equation for calculating an advantage estimate A_t, associated with having taken action a_t in state s_t as experienced during some trajectory of maximal length Max_t is given by:

$$A_t = V_t^{\text{target}} - V_{\omega_{\text{old}}}(s_t). \tag{2.43}$$

2.8.5 Policy Network's Objective Function L^{CLIP}

The objective function employed by PPO L^{CLIP}, is designed to allow multiple epoch of weight updates under the same set of training data (batch) [8].

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t]. \tag{2.44}$$

Roughly speaking, it limits the extend to which the current state of the policy (π_θ) can be changed compared to the old state $(\pi_{\theta_{\text{old}}})$ used in the rollout step. Let's study each part to understand how it works:

- Ratio function:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}}(a_t \mid s_t). \tag{2.45}$$

  The probability ratio acts as an approximation of divergence between π_θ and $\pi_{\theta_{\text{old}}}$. If an action a_t at state s_t becomes more unlikely under the current policy π_θ than it used to be under the old policy $\pi_{\theta_{\text{old}}}$, the probability ratio will decrease towards positive 0 since π_θ shrinks compared to $\pi_{\theta_{\text{old}}}$. When actions become more likely, the probability ratio will increase towards positive infinity. In cases where the probability is comparatively similar under both policies, the ratio will tend to 1. Note that this divergence measure does not give us enough information to assess how different the two policies are across all possible actions in all possible states. It only evaluates how much the behaviour of the policy has changed with respect to the old policy for each training observation in a given batch.

- Unclipped objective:

$$r_t(\theta)A_t. \tag{2.46}$$

  This ratio can replace the log probability we use in the Actor-Critic objective function 2.36.

- Clipped objective:

$$\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t. \tag{2.47}$$

  Which restricts the range of values that the ratio can take to the interval $[1-\epsilon, 1+\epsilon]$.

The minimum operator takes the minimum of the unclipped and clipped objective. This design has two effects:

- It yields a pessimistic estimate of the policy's performance.
- It avoids destructively large updates in the direction of the probability of re-selecting some action a_t in a given state s_t.

Let's look case-wise how PPO's objective function L^{CLIP} behaves when varying its input arguments' values. In cases 1 and 2 the clipping does not apply since the

	$p_t(\theta) > 0$	A_t	**Return Value of** min	**Objective is Clipped**	**Sign of Objective**	**Gradient**
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
3	$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
4	$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
5	$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
6	$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

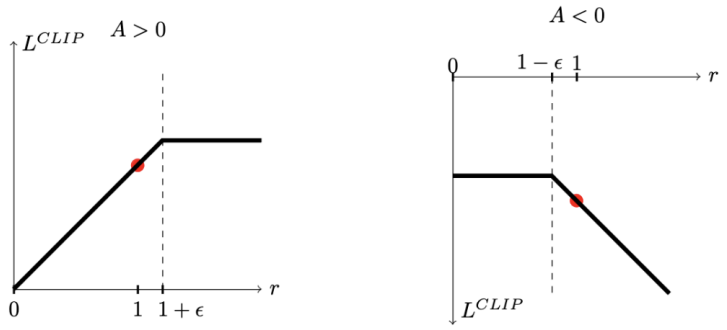


Figure 2.2: Table with separate cases for the function L^{CLIP}

ratio is between the range $[1-\epsilon, 1+\epsilon]$. In case 1, we have a positive advantage: the action taken is better than the average of all the actions in that state. We should encourage the current policy to increase the probability of taking that action in that state. In case 2, we have a negative advantage. The action taken is worse than the average action. Therefore, we should discourage the policy from taking that action in that state. Since the ratio in both cases is between the interval, we can increase/decrease the probability that our policy takes that action at that state.
In cases 3 and 4, the ratio is below the range $1-\epsilon$. This means that the probability of taking that action at that state is much lower than with the old policy. If we are in case 3, the advantage estimate A_t is positive. Thus, we seek to increase the probability of taking that action at that state. However, if the advantage estimate is negative (case 4), we do not want to decrease further the probability of taking that action at that state. Therefore, the gradient is set to 0 (looking at figure 2.2,

we are in a flat line). We do not update our weights.
In cases 5 and 6, the ratio is above the range $1 + \epsilon$. The probability of taking that action at that state in the current policy is much higher than with the old policy. If the advantage is positive (case 5), we do not want to act greedy. We already have a higher chance of taking that action at that state. Therefore, the gradient is set to 0 (looking at figure 2.2, we are in a flat line). We do not update our weights. If the advantage is negative (case 6) we want to decrease the probability of taking that action at that state.
In brief, we update our policy only if one of the following conditions is met:

- Ratio is in the range $[1 - \epsilon, 1 + \epsilon]$
- Ratio is outside the range, but the advantage makes it come closer to the range
  - Below the range, but the advantage is positive
  - Above the range, but the advantage is negative

2.9 Lexicographic Proximal Policy Optimization

Typically, the reward function is what indirectly dictates how the agent will end up behaving. If we want the agent to learn to pick up sticks (remember the example of the dog from section 2.1), the environment will provide a reward each time the agent completes the task successfully. However, the objective is not always so straightforward. What if the agent has to take into account two objectives simultaneously? Even worse, what if they are in conflict with one another at some points?

Example 2.3. Consider an agent in an environment where it has two objectives: gathering apples and donating apples to a charity box. The agent receives a reward for each apple it gathers, which satisfies its primary objective of maximizing its own apple collection. However, the agent also receives a reward for each apple it donates to the charity box, which fulfills a secondary objective of benefiting others. Here's the conflict: if the agent focuses solely on gathering apples, it neglects the beneficence objective. Conversely, if it donates too many apples, it compromises its own collection goal. To manage this, we need to prioritize the objectives.

Let's suppose that we at least have clarity on which objective we want to prioritize, in other words, a lexicographic order. In a lexicographic order, we rank the objectives based on their priority. This means that the agent first seeks to fulfill the highest-priority objective before addressing lower-priority ones, even if they

occasionally conflict. If we have a lexicographic order, there has been a body of literature in RL working on this problem. One notable approach is outlined in the paper "Lexicographic Multi-Objective Reinforcement Learning" by Skalse et al. (2022) [9]. In their work, Skalse et al. introduce several techniques for handling multiple objectives in a lexicographic manner within the RL framework. In particular, we will use the lexicographic version of PPO presented in the paper, which is considered the state-of-the-art in this field.
Their approach involves two key steps:

- First, they define two reward functions, each corresponding to a different objective. For example, one reward function might measure the agent's success in donating apples to a charity box, while the other measures its success in gathering apples.
- Then, they ensure that the agent's policy focuses on maximizing the primary reward function first. Once the agent has optimized its behavior to achieve this primary goal, it then shifts focus to maximizing the secondary reward function. This means that the agent first ensures it donates enough apples to fulfill its ethical obligation. After meeting this primary objective, it then gathers any additional apples for itself.

Example 2.4. To better understand lexicographic optimization, consider the following example with vectors in $\mathbb{R}^2$. Suppose we have four vectors representing different outcomes based on two objectives: R_1 (donating apples) and R_2 (gathering apples):

$$\mathbf{v}_1 = (8,2), \quad \mathbf{v}_2 = (5,5), \quad \mathbf{v}_3 = (10,1), \quad \mathbf{v}_4 = (8,4)$$

We want to determine which vector is the best lexicographically. We start by comparing the first objective, R_1:

$$\mathbf{v}_3 = (10,1) \quad \text{(highest value for } R_1\text{)}$$

Since $\mathbf{v}_3$ has the highest value for the primary objective R_1, it is prioritized. Therefore, $\mathbf{v}_3$ is considered the best lexicographically because it maximizes the primary objective, regardless of the values for the secondary objective R_2. If two or more vectors had the same value for R_1, we would then compare their values for R_2 to determine the best vector. For instance, both $\mathbf{v}_1$ and $\mathbf{v}_4$ have the same value for R_1. We would then compare their values for R_2:

$$\mathbf{v}_4 = (8,4) \quad \text{(higher value for } R_2\text{)}$$

Thus, $\mathbf{v}_4$ would be chosen over $\mathbf{v}_1$ because it has a higher value for the secondary objective.

In the following sections, we will delve deeper into the implementation details of Lexicographic PPO, exploring how it integrates multiple reward functions and manages the learning process to ensure optimal performance across all prioritized objectives.

2.9.1 Multi-Objective Markov Decision Process

In a standard Markov Decision Process (MDP), the return of the reward function is typically a scalar value. However, in many real-world scenarios, an agent must consider multiple objectives simultaneously. This leads to the concept of a Multi-Objective Markov Decision Process (MOMDP).
In an MOMDP, the reward function is modified to return a vector of rewards, each component corresponding to a different objective. Formally, the reward function is defined as:

$$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^m \tag{2.48}$$

Here, the reward function returns an m-dimensional vector $R(s_t, a_t, s_{t+1})$ at each time step, where each dimension represents a different objective. The value function of a policy π in this context is defined as:

$$V_\pi = \mathbb{E}_\pi \left[\sum_{t=0}^{Max_t} R(s_t, a_t, s_{t+1}) \mid \mathcal{S}_t = s, \mathcal{A}_t = a \right] \tag{2.49}$$

This value function represents the expected sum of vector-valued rewards obtained by following policy π. It encapsulates the performance of the policy across all objectives, considering the trade-offs between them. In the context of MOMDPs, different solutions can be obtained depending on how we prioritize the various objectives. One possible approach is to search for a policy that maximizes the lexicographic order among objectives. This means that we first optimize the highest-priority objective, and only after achieving the best possible outcome for this objective do we move on to optimize the next priority, and so on. In the following section, we will explain how to implement this approach.

2.9.2 Lexicographic RL

To address the challenge of prioritizing multiple objectives in a MOMDP, we employ Lexicographic Reinforcement Learning (RL), which ensures that higher-priority objectives are optimized before considering lower-priority ones. We use the following steps:

- **Start with all policies:** We begin with the set Π_0^ϵ, which includes all possible policies.

- **Evaluate primary objective:** For each policy in Π_0^ϵ, we evaluate the objective function J_1, corresponding to the highest-priority reward R_1.
- **Select top policies:** We retain only those policies that perform within a margin of ϵ_1 of the best-performing policy for R_1. This new set of policies is denoted as Π_1^ϵ.
- **Iterate for next objectives:** We repeat this process iteratively for the remaining objectives. For the ith objective:
  - Evaluate the objective function J_i for all policies in the current set Π_{i-1}^ϵ.
  - Retain the policies that perform within a margin of ϵ_i of the best-performing policy for R_i.
  - The resulting set of policies is Π_i^ϵ.
- **Final set of policies:** The final set Π_m^ϵ includes policies that are nearly optimal for all objectives within the specified tolerances ϵ.

Here, Π_i^ϵ represents the set of policies that are approximately optimal for the first i objectives. Each iteration refines this set by ensuring that only the policies meeting the ϵ_i criteria for the current objective are retained, progressively narrowing down to policies that consider all objectives effectively.
We can formalize this idea with the following definition:

Definition 2.5 (Lexicograpically ϵ-optima)**.** *Given a MOMDP $\mathcal{M}$ with m rewards, we say that a policy π is (globally) lexicographically ϵ-optima if $\pi \in \Pi_m^\epsilon$, where $\Pi_0^\epsilon = \Pi$ is the set of all policies in $\mathcal{M}$, $\Pi_{i+1}^\epsilon := \{\pi \in \Pi_i^\epsilon \mid \max_{\pi' \in \Pi_i^\epsilon} J_i(\pi') - J_i(\pi) \leq \epsilon_i\}$ and $\mathbb{R}^m \ni \epsilon \gg 0$.*

2.9.3 Policy-based algorithm

Once we have defined the lexicographic solution for a multi-objective problem, we need to understand how to find such a solution using a policy-based approach. In this section, we introduce the Lexicographic Proximal Policy Optimization (LPPO) algorithm, which optimizes multiple objectives in a prioritized manner. We will define for each reward R_i its corresponding objective function J_i. Our goal is to update the parameters θ of π_θ using a multi-timescale approach, following the lexicographically ϵ-optima definition. The process involves the following steps:

- **Initial Optimization:** First, we optimize θ using J_1.

- **Subsequent Optimizations:** Then, at a lower timescale, we optimize θ while ensuring that the loss with respect to the first reward remains bounded by some tolerance value. This procedure is repeated for the following objectives.

To solve these problems, we apply the Lagrangian relaxation technique [3]. This technique is used to solve constrained optimization problems by transforming them into unconstrained problems. It introduces Lagrange multipliers to incorporate the constraints into the objective function, allowing us to find the optimal solution by solving for a saddle point where the gradients of the Lagrangian function with respect to the variables and the multipliers are zero.

- Suppose that θ' is optimized lexicographically with respect to $J_1, \ldots, J_{i-1}$, and now we wish to lexicographically optimize θ with respect to J_i.
- We define $j_k := J(\theta')$ for each $k \in \{1, \ldots, i-1\}$ as the value of the objective function of the kth reward in the updated policy parameters θ' by J_i.
- We aim to solve the constrained optimization problem given by:

$$\begin{aligned} &\text{maximize} \quad J_i(\theta), \\ &\text{subject to} \quad J_k(\theta) \geq j_k - \tau, \quad \forall k \in \{1, \ldots, i-1\} \end{aligned} \tag{2.50}$$

  where $\tau > 0$ is the tolerance parameter ϵ_i that appears in the lexicographically ϵ-optima definition. While learning, this parameter is set to decay such that $\tau_t \to 0$ as $t \to \infty$.

- This constrained optimization problem can be translated into finding a saddle point of the following function:

$$L_i(\theta, \lambda) = J_i(\theta) + \sum_{k=1}^{i-1} \lambda_k (J_k(\theta) - j_k + \tau) \tag{2.51}$$

  where the critical point (θ_*, λ_*) is a relative maximum along the θ-dimension and a relative minimum along the λ-dimension.

$$\begin{aligned} \nabla_\theta L_i(\theta_*, \lambda_*) &= \nabla_\theta J_i(\theta_*) = 0 \\ \nabla_\lambda L_i(\theta_*, \lambda_*) &= \sum_{k=1}^{i-1} (J_k(\theta_*) - j_k + \tau) = 0 \end{aligned} \tag{2.52}$$

- A valid solution would be to solve each optimization problem L_i separately. This would lead to a correct solution, but the process would be slow and sample-inefficient. Instead, we solve this problem synchronously, guaranteeing convergence to a lexicographically optimal solution.

Using learning rates β^i and η^i with respect to the i-th objective, we can compute a saddle point solution to each L_i via the following gradient updates:

$$\begin{aligned} \theta &\leftarrow \Gamma_\theta[\theta + \beta_t^i(\nabla_\theta \hat{J}_i(\theta) + \sum_{k=1}^{i-1} \lambda_k \nabla_\theta \hat{J}_k(\theta))], \\ \lambda_k &\leftarrow \Gamma_\lambda[\lambda_k + \eta_t(\hat{j}_k - \tau_t - \hat{J}_k(\theta))] \;\; \forall k \in \{1, \ldots, i-1\} \end{aligned} \tag{2.53}$$

where $\Gamma_\lambda(.) = \max(., 0)$ and Γ_θ projects θ to the nearest point in the space of permitted values $\Theta \in \mathbb{R}^x$. The symbol $\hat{}$ denotes the estimated value obtained from experience. The terms involved in the updates of θ can be simplified by the following update:

$$\theta \leftarrow \Gamma_\theta[\theta + \nabla \hat{J}(\theta)]$$

where

$$\hat{J}(\theta) = \sum_{i=1}^{m} c_t^i \hat{J}_i(\theta) \quad \text{and} \quad c_t^i = \beta_t^i + \lambda_i \sum_{k=i+1}^{m} \beta_t^k$$

assuming $\sum_{k=m+1}^{m} \beta_t^k = 0$. Thus, to compute a solution to a lexicographic optimization problem from a collection of objective functions, we just need to update the coefficients c_t^i at each time-step and linearly combine them with the objective functions. Recall that policy-based algorithms not only need an optimization function, but also a critic V_i to estimate each $\hat{J}_i$. Its parameters ω_i can be updated following the update rule for V_i given by $\omega_i \leftarrow \omega_i + \alpha_t(\delta_t^i \nabla_{\omega_i} V_i)$. Where α_t is the learning rate and δ_t^i is the MSE for V_i.

2.10 Models of Multi-Agent Interaction

The classical definition of a MDP (def. 2.1) typically revolves around a single agent who controls all the actions. However, in a multi-agent setting, additional complexities arise. Here, the consequences of actions and the rewards obtained depend not only on the individual agent's decision but also on the actions taken by other agents. Moreover, these agents might even act as adversaries, striving to minimize our rewards.

2.10.1 Normal Form Games

To delve deeper into multi-agent environments, we first examine normal-form games. These games are single-shot (non-sequential and stateless) games where a group of agents each has to execute an action simultaneously. However, the rewards received by each agent depend on the collective actions chosen by all the participants.

Definition 2.6 (Normal-form game)**.** *A normal-form game consists of:*

- *A finite set of agents* $I = 1, ..., n$.
- *For each agent* $i \in I$*:*
  - *a finite set of actions* $\mathcal{A}^i$,
  - *a reward function* $R^i : \mathcal{A} \to \mathbb{R}$, *where* $\mathcal{A} = \mathcal{A}^1 \times ... \times \mathcal{A}^n$.

Each agent $i \in I$ selects a policy $\pi^i : \mathcal{A}^i \to [0, 1]$. This policy function assigns probabilities to the available actions $\mathcal{A}^i$ for the agent i, ensuring that the sum of probabilities for all the actions equals 1. Each agent then samples an action $a^i \in \mathcal{A}^i$ according to the probabilities specified by its policy function. The collective actions chosen by all the agents then constitute a joint action, denoted as $a = (a^1, ..., a^n)$. Finally, each agent i receives a reward denoted as r^i, determined by its specific reward function and the resulting joint action a, expressed as $r^i = R^i(a)$.

2.10.2 Stochastic Games

Normal-form games, while valuable for analyzing agent interactions, lack the concept of an environment state influenced by agents' actions. In contrast, stochastic games bridge this gap by introducing a state-based environment wherein the state evolves dynamically over time due to agents' actions and probabilistic state transitions.

Definition 2.7 (Stochastic game)**.** *A stochastic game consists of:*

- *A finite set of agents I = 1, ..., n,*
- *a finite set of states* $\mathcal{S}$, *with subset of terminal states* $\bar{\mathcal{S}} \subset \mathcal{S}$.
- *For each agent* $i \in I$*:*
  - *a finite set of actions* $\mathcal{A}^i$,
  - *a reward function* $R^i : S \times \mathcal{A} \times S \to \mathbb{R}$ *where* $\mathcal{A} = \mathcal{A}^1 \times ... \times \mathcal{A}^n$.
- *A state transition probability function:* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *such that*

$$\forall s \in \mathcal{S}, a \in \mathcal{A} : \sum_{r' \in S} T(s, a, s') = 1, \tag{2.54}$$

- *an initial state distribution:* $\mu : \mathcal{S} \to [0, 1]$ *such that*

$$\sum_{s \in \mathcal{S}} \mu(s) = 1 \text{ and } \forall s \in \bar{\mathcal{S}} : \mu(s) = 0. \tag{2.55}$$

A stochastic game proceeds as follows: the game starts in an initial state s_0 drawn from a distribution μ. At each time-step t, every agent $i \in I$ observes the current state s_t and selects an action $a_t^i \in \mathcal{A}^i$ according to its policy $\pi^i(a_t^i \mid h_t)$. This leads to the joint action $a_t = (a_t^1, ..., a_t^n)$. The policy is conditioned on the state-action history $h_t = (s_0, a_0, s_1, a_1, ..., s_t)$, containing the current state, previous states, and previous joint actions. The history is observable by all agents, denoted as full observability. Subsequently, given the state s_t and joint action a_t, the game transitions to the next state s_{t+1} with probability determined by $P(s_t, a_t, s_{t+1})$, and each agent i receives reward $r_t^i = R^i(s_t, a_t, s_{t+1})$. These steps iterate until reaching a terminal state s_t in $\bar{\mathcal{S}}$ or after completing a maximum number of Max_t time-steps, after which the game concludes. Alternatively, the game may continue indefinitely if it is non-terminating.

2.11 Solution Concepts for Games

Section 2.10 laid the groundwork by introducing basic game models to formalize multi-agent environments and interactions. In this section, we will introduce a series of solution concepts. This involves determining what constitutes a solution to a game, which specifies when a collection of agent policies constitutes a stable or desirable outcome. Together, a game model and its associated solution concept form the basis of a learning problem in Multi-Agent Reinforcement Learning (MARL).

In general, a solution to a game comprises a joint policy consisting of one policy

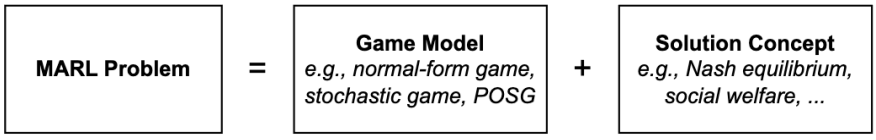


Figure 2.3: Parts of a MARL problem

for each agent, satisfying certain properties related to expected returns for each agent and their interactions. Our definition of solution concepts will assume finite game models. In particular, we will assume finite state, action, and observation spaces, as well as a finite number of agents.

2.11.1 Joint Policy

A solution to a game is a joint policy $\pi = (\pi^1, ..., \pi^n)$ satisfying certain criteria outlined by the chosen solution concept. These criteria typically revolve around

the expected discounted return, V_{π}^{i}, garnered by each agent i under the joint policy.

2.11.2 Best Response

Suppose that π is a joint policy. A best response policy for agent i is a policy that maximizes player i's expected return against the other players' policies π_{-i}. There may be many best responses, and we denote the set of such best responses as

$$BR^{i}(\pi^{-i}) = arg\max_{\pi^{i}} V^{i}_{\langle \pi^{i}, \pi^{-i} \rangle}. \tag{2.56}$$

2.11.3 Nash Equilibrium

Following the definition of best response, in a multi-agent learning environment each agent aims to find a policy that maximizes their own expected return. Together, the overall goal is to find a joint policy that gathers the most reward for each agent. This goal applies to both competitive and collaborative situations. Agents can find policies that best counter or complement others.
We call this optimal policy the Nash equilibrium.

Definition 2.8 (Nash equilibrium)**.** *A joint policy $\pi = (\pi^{1}, ..., \pi^{n})$ is a Nash equilibrium if*

$$\forall i, \hat{\pi}^{i} : V^{i}_{\langle \hat{\pi}^{i}, \pi^{-i} \rangle} \leq V^{i}_{\pi}. \tag{2.57}$$

Each agent's policy in Nash equilibrium is the best response to the other agents' optimal policies. No agent is incentivized to change their policy because doing so gives them less reward. In other words, all the agents are at standstill, i.e., $\pi^{i} \in BR^{i}(\pi^{-i}) \quad \forall i \in I$.
To give an example, imagine a competitive game between two robots. During each round, they have to choose a number between one and ten, and whoever selects the higher number wins. As expected, both pick the ten every time because they do not want to risk losing. If robot A were to choose any other number, he would risk losing against robot B's optimal policy of always choosing ten and vice versa. They are both in equilibrium.

2.12 Independent Learning

In the context of MARL, independent learning refers to a strategy where each agent independently learns and optimizes its policy without explicitly considering the learning processes of other agents. This approach decomposes a n-agent MARL problem into n single-agent problems, each handled separately by individual agents. Each agent's objective is to optimize its policy based on its own

experiences, observations and rewards. This decomposition allows agents to learn simultaneously without needing to coordinate their learning processes explicitly. Moreover, each agent constructs its value function independently.
In independent learning, the observations available to each agent can vary. Some agents may have observations that include explicit information about the actions of other agents, while others may not have this information. The effects of the agents' actions are treated as part of the environment, which can introduce non-stationarity into the learning process. Since each agent is continually learning and updating its policy, the environment from any agent's perspective is changing. This non-stationarity can make it difficult for agents to identify the consequences of their actions, and can lead to instability during training. That also imply that the Markov property is lost.
Independent Proximal Policy Optimization (IPPO) is an independent learning approach where each agent optimizes its policy using PPO independently. IPPO has been shown to achieve results comparable or better than state-of-the-art centralized learning approaches in the popular multi-agent benchmark SMAC [12], thanks to the robustness of PPO to non-stationarity.
Independent Lexicographic Proximal Policy Optimization (ILPPO), extends the principles of IPPO to handle multiple objectives in a lexicographic manner. This approach maintains the independent learning framework.

Chapter 3

State of the art

Rodriguez-Soto et al. (2023) introduced in their research an innovative ethical embedding process designed to integrate ethical considerations into reinforcement learning environments [7]. This process combines traditional reward functions with an ethical reward function, resulting in a single-objective environment. The primary advantage of this approach is that it guarantees convergence to an ethical-optimal policy, which maximizes rewards while respecting the ethical values encoded in the reward function.
In multi-agent environments, the ethical embedding process is implemented using independent Q-learners (IQL). Each agent independently learns its Q-function without considering the actions and strategies of other agents. Despite this isolation, agents using IQL can successfully learn an ethical-optimal policy.
The same research also studied the implementation of lexicographic IQL (LIQL) without the need of using the ethical embedding process, just placing the agents directly into the Ethical MOMG. This was performed ensuring that the ethical objective was met before optimizing the individual objective.
However, initial implementations faced significant challenges. The experiments conducted on a tiny map with only two agents under conditions of full observability revealed several issues. These included slow convergence, resulting in a prolonged training time of up to 24 hours, and problems with scalability when expanding the system to larger and more complex environments.
To address these challenges, Mayoral Macau's (2023) [6] research explored the implementation of the Independent Proximal Policy Optimization (IPPO) algorithm, where each agent operates its own instance of Proximal Policy Optimization (PPO). Mayoral Macau also applied the ethical embedding process, similar to Rodriguez-Soto et al. (2023), but with IPPO as the base algorithm instead of IQL. The IPPO implementation has been tested in larger environments and has effectively resolved scalability issues, demonstrating that it can handle the complexity

of multi-agent systems without the prohibitive training times and dimensionality problems previously encountered.
Through these advancements, the combination of the ethical embedding algorithm and IPPO offers a robust solution for developing ethical AI in multi-agent settings. This approach not only facilitates the convergence to ethical-optimal policies but also ensures that such solutions are scalable and efficient.
Building on this foundation, a new implementation using the Independent Lexicographical Proximal Policy Optimization (ILPPO) algorithm has been proposed. ILPPO leverages the principles of the lexicographic decision rule, which prioritizes objectives based on their importance. In ILPPO, agents are designed to prioritize ethical considerations above other objectives. This ensures that ethical compliance is non-negotiable, effectively eliminating any trade-offs between ethicality and other objectives.

Chapter 4

Description of the environment

The Ethical Gathering Game (Rodriguez-Soto et al. 2023 [7]) is an ethical variation of the Gathering Game introduced in Leibo et al. (2017) article [5], where agents manage to compensate social inequalities by learning to behave in alignment with the moral value of beneficence. Beneficence, in this context, refers to the principle of acting with the intent to benefit others, promoting their well-being, and preventing or removing harm. In the Ethical Gathering Game, agents should make decisions that not only guarantee their survival but also consider the welfare of other agents in the environment. We expect agents to learn to behave ethically, compensating the forced inequalities using the donation box.
The agents live together in a grid world, with apples spawning in various locations. Each agent needs k apples to survive, but they have different gathering capabilities. This means that when two agents try to pick the same apple, only the most efficient one will get it. The donation box allows agents to contribute their surplus apples to a communal resource that can be accessed by other agents who struggle to gather enough apples. To ensure a realistic setup, the donation box has a limited capacity, storing up to c apples. Once the donation box reaches its capacity, any additional apples donated will not be accepted.
Agents in the Ethical Gathering Game deal with two reward functions: the individual and the ethical one. The ethical reward function aligns agent behaviors with ethical principles based on the value of beneficence. Actions are classified as ethical (praiseworthy) or unethical (blameworthy) according to this value. When an agent has enough apples to survive and decides to take an apple from the donation box, it receives a penalty. This penalty discourages selfish behavior and reinforces the importance of using the donation box solely for supporting agents in need. Alternatively, agents that act in accordance with beneficence, such as donating surplus apples when their own needs are met, receive a positive reward. The individual reward function, on the other hand, focuses on the agent's personal

survival and resource accumulation. Agents receive positive rewards for gathering apples or taking them from the donation box, and receive negative rewards for donating them.
During training, agents learn to recognize when their own needs are met and when they have excess apples that could benefit others.
Figure 4.1 shows a graphical representation of our environment. It is a grid consisting of 3x4 cells. Agents are represented as red (inefficient) or yellow (efficient) squares, while the apples are depicted in green. The empty cells are shown in black. The counter at the top of the image indicates the number of apples collected by each agent and the number of apples stored in the donation box.

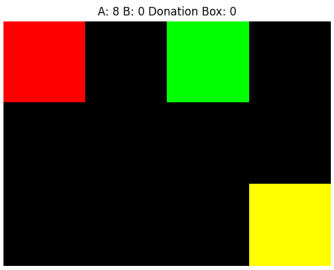


Figure 4.1: Graphical representation of the tiny environment.

4.1 Environment characteristics

The Ethical Gathering Game, as an ethical MOMG, is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}^{i=1,\dots,n}, (R_0, R_\mathcal{N} + R_e)^{i=1,\dots,n}, T \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}^i$ is the action sets of agent i, T is the transition function of the game and $R^i = (R_0^i, R_\mathcal{N}^i + R_e^i)$ is the reward function of agent i. R_0^i corresponds to its individual reward, whereas $R_\mathcal{N}^i + R_e^i$ corresponds to its ethical reward, separated into the normative and evaluative reward [1]. We will simplify the ethical reward treating the sum as a single term, $R_E^i = R_\mathcal{N}^i + R_e^i$.

We define the simplified state for each agent as s = (map, agent_apples, donation_box_apples), where:

- **map**: binary matrix of size MxN.

[1] See Rodriguez-Soto et al. 2023 [7] Section 6.4.2, for a more detailed explanation of each reward component.

- **agent_apples**: array with the abstract number of apples held by each agent (0: no apples, 1: not enough to survive, 2: enough to survive, 3: surplus).

- **donation_box_apples**: number of apples in the donation box, normalized by the donation box capacity (e.g. If the donation box has 3 apples and its maximum capacity is 5, $donation\_box\_apples = 3/5$).

This abstraction allows us to significantly reduce the complexity of the state space. For example, on a tiny grid map, we can compute the total number of states by considering the limited values each component of the state can take, thus making the learning process more efficient and scalable [6].

Agents have seven different actions available, $|\mathcal{A}| = 7$: movement actions (MOVE UP, MOVE DOWN, MOVE LEFT, and MOVE RIGHT) are possible as long as they stay within the bounds of the grid. Interaction actions with the donation box include DONATE (if the donation box is not full) and TAKE DONATION (if available). Additionally, the agent can choose to STAY in its current position, not moving to any adjacent cell. These actions allow agents to navigate the grid, gather resources, and interact with the donation box [7].

The reward system is designed to promote both individual survival and ethical behavior among agents. The reward function $R^i(s, a)$ for agent i is determined by the current state s of the environment and the action a taken by the agent. It returns a separate scalar value for each objective. The first component focuses on the agent's individual survival (R_0^i), while the second focuses on the agent's ethical behavior, such as contributing to the donation box or taking a donation when its not needed it (R_E^i). Two constants are used to define the reward specification:

- **Survival threshold**: sets the minimum number of apples an agent needs to survive. Agents must gather at least this many apples to avoid penalties related to survival.

- **Donation box capacity**: indicates the maximum number of apples that the donation box can hold. If the donation box is full, agents are not penalized for not donating additional apples.

The complete specification of the reward function, divided into two components, has been taken from Mayoral Macau (2023) [6]:

$$\begin{aligned} \text{hunger} &\leftarrow \text{agent apples} < \text{survival\_threshold} \\ \text{full\_box} &\leftarrow \text{apples in the donation box} == \text{donation\_box\_capacity} \end{aligned} \tag{4.1}$$

$$R_0^i(s,a) = \begin{cases} -1 & \text{if Hunger} \\ +1 & \text{if agent gathered an apple from the ground by performing a} \\ -1 & \text{if agent donated an apple} \\ 0 & \text{otherwise} \end{cases}$$

$$R_E^i(s,a) = \begin{cases} -1 & \text{if agent takes or tries to take donation while not Hunger} \\ +0.7 & \text{if agent donates while not Hunger and not full\_box} \\ 0 & \text{if agent takes donation while Hunger} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

In the context of the Ethical Gathering Game, efficiency refers to how likely an agent is to successfully gather an apple when it attempts to do so. If an agent has an efficiency of 0.7, it means that 7 out of 10 times it will successfully gather the apple. In either case, the apple is removed from the map after the attempt [7]. If two agents step on the same cell that has an apple, the agent with higher efficiency will have priority over it.

4.2 Hyper-parameters

We use a comprehensive set of hyper-parameters to fine-tune the performance of the algorithms. These hyper-parameters are classified into four groups:

- **Environment setup**: Defines the configuration of the environment used for training.
- **Training parameters**: Specifies the general aspects of the learning process.
- **IPPO parameters**: Crucial for ensuring stable and efficient learning. These parameters help in keeping the variance low and achieving convergence of the metrics.
- **ILPPO parameters**: Designed to prioritize ethical considerations within the learning process (see 4.2.1).

A detailed explanation of each hyper-parameter of Table A.1 (see Appendix A) can be seen in Mayoral Macau's final master project [6] (Section 4.2.2). We will only focus on the ILPPO-specific hyper-parameters, which have not been explained yet.

4.2.1 ILPPO hyper-parameters

The parameters used to fine-tune the ILPPO algorithm are the following ones:

- **we-reward0**: Determines the weight used for the individual reward.
- **we-reward1**: Determines the weight used for the ethical reward.
- **prioritize-performance-over-safety**: Lets you decide if you want to first optimize lexicographically the individual reward or the ethical reward. Controls the order in which the rewards are stored. By default, it is set to first optimize the ethical reward (ILPPO safety).
- **eta-value**: Scalar used to weight the difference between the most recent loss and the average loss of the actor. This weighted value is then used with the *beta-values* to weight the first advantage (associated with the first objective, the ethical one) in the next update. It corresponds to the η_k variable in the lexicographic gradient update 2.53.
- **beta-values**: List of two float values. The first one, *beta-value-0*, is summed with the *eta-value* from the previous update to weight the first advantage (A, see subsection 2.8.4 and equation 2.43), associated with the ethical reward. The second one, *beta-value-1*, is used to weight the second advantage, associated with the individual reward. Then we compute the weighted sum of both advantages to proceed with the PPO algorithm using a scalarized single advantage. It corresponds to the β_t^i variables in the he lexicographic gradient update 2.53.

4.3 Desired Ethical Policies

In our reinforcement learning environment, the known and desired Nash equilibrium represents the ethical-optimal policy we aim to achieve. The behavior for two agents, A_1 and A_2, where A_1 is more efficient than A_2, follows these phases:

1. **Initial Gathering Phase:** Agent A_1 gathers more apples than A_2 due to higher efficiency, reaching the survival threshold first. In Figure 4.2, this phase is seen at the start of the timeline where A_1's apple count increases and reaches the dotted red line (steps 0 to 120).
2. **Donation Phase:** A_1 donates extra apples to the donation box after reaching the survival threshold, while A_2 takes apples from the donation box to reach its own survival threshold. Figure 4.2 shows this phase as an interval from steps 120 to 220, where A_2 (orange line) starts increasing its apple count.

3. **Filling the Donation Box:** Both agents donate any additional apples to the donation box until it is full. This is illustrated in Figure 4.2 where the donation box line (green) starts increasing until stabilizing into a horizontal line at the donation box capacity (dotted black line).

4. **Continued Gathering:** With the donation box full, both agents continue gathering apples for themselves. Due to its efficiency, A_1 will naturally gather more apples than A_2. In Figure 4.2 we can see that A_1 and A_2 continue to gather apples, with A_1 accumulating apples at a faster rate.

These desired ethical policies ensure that both agents cooperate to reach a state where their basic needs are met before accumulating surplus resources. This promotes a fair distribution of resources, aligning with the ethical principles embedded in the environment [7].
Figure 4.2 illustrates the evolution of the number of apples collected by (A_1 and A_2) and the apples in the donation box over time in a single run, demonstrating these four phases.

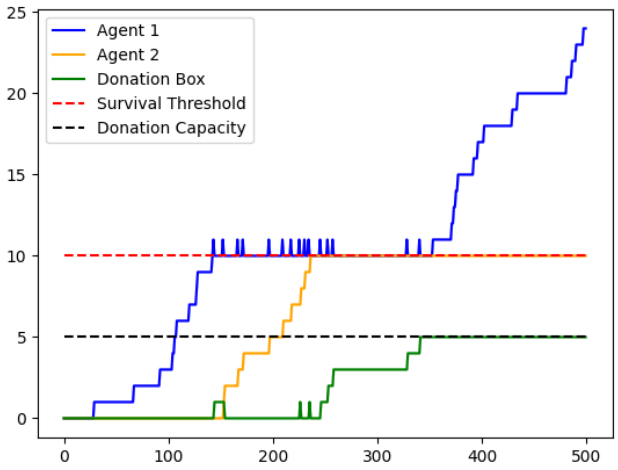


Figure 4.2: Evolution of the number of apples collected by each agent (A_1 and A_2) and the apples in the donation box over time.

Chapter 5

Experiments

5.1 Objectives

The primary objectives of our experiments are to validate the Independent Lexicographic Proximal Policy Optimization (ILPPO) algorithm in a multi-agent environment, ensuring that each agent is learning an ethical-optimal policy, and compare its performance with the Independent Proximal Policy Optimization (IPPO) algorithm 2.12, a state of the art algorithm that ensures its agents learn ethical-optimal policies. Specifically, we aim to:

- **Validation of ILPPO in a Multi-Agent Environment:** Evaluate the effectiveness of ILPPO in a setting with multiple agents, a scenario that has not been previously studied [9].
- **Ethical Objective Prioritization:** Determine if agents using ILPPO learn to prioritize the ethical objective over the individual objective [6].
- **Minimize the number of sub-optimal and non-ethical actions:** Ensure that the agents are learning an ethical-optimal policy studying the actions chosen, making sure that it only chooses those actions that have more probability to return ethical rewards.

By addressing these objectives, we aim to highlight the strengths and weaknesses of ILPPO in ensuring ethical decision-making in multi-agent environments and its correct implementation by comparing it to a state of the art algorithm, IPPO.

5.2 Empirical Setup

In this section, we present our experimental setup and results for comparing the performance of agents using ILPPO and IPPO in a controlled environment.

The tiny environment serves as a test-bed to illustrate the differences in agent behavior and objective prioritization between the two algorithms. To facilitate the monitoring and visualization of the training process, we utilized TensorBoard[1]. We monitored the most important metrics for each agent under both IPPO and ILPPO algorithms. These metrics include:

- **Actor Loss:** Measures the performance of the policy network in selecting actions.
- **Critic Loss:** Measures the accuracy of the value network in predicting the expected rewards.
- **Accumulated Reward:** Tracks the total reward accumulated by each agent over time.

Additionally, we logged: the step at which each agent survived; if they survived; if they managed to fill the donation box; and a study about how many unethical, missed ethical, and sub-optimal actions they did on average for each simulation.
For version control, we used a Git repository, ensuring that all changes and developments were systematically tracked. The implementation was carried out in Python using PyTorch. We began with a pre-existing environment programmed for IPPO and made the necessary modifications to ensure full functionality of ILPPO in a multi-agent, vectorized-reward context within the same environment. The IPPO implementation was studied using the code from Mayoral Macau (2023) [6]. The lexicographic implementation is inspired by the Trust Region Policy Optimization (TRPO) single-agent implementation from Skalse et al. (2022) [9], adapted to PPO in a multi-agent environment. However, the available code did not produce the results reported in the paper, and the agent was not obtaining rewards. Additionally, parts of the code were based on deprecated libraries, such as OpenAI Gym. These issues could not be resolved through communication with the authors. The complete implementation and experimental setup can be found in our Git repository[2].
The experiments were conducted on a computing cluster comprising 11 "CPU-only" nodes. Each node is equipped with 2 Intel Xeon CPUs at 2.2 GHz, 10 cores per CPU and 92 GB of RAM. For each experiment, we utilized one node at a time to ensure consistent and comparable results.

[1] TensorBoard allows us to track and compare key metrics in real-time, providing valuable insights into the performance of each agent

[2] The GitHub repository can be found at `https://github.com/ntorqulu/lexicographic_ppo`

5.2.1 Comparison with IPPO in a small environment

In this section, we provide a detailed comparison between the performance of the Independent Lexicographical Proximal Policy Optimization (ILPPO) algorithm and the Independent Proximal Policy Optimization (IPPO) algorithm in a simplified environment. The primary objective is to evaluate the effectiveness of ILPPO in ensuring ethical decision-making while maintaining efficient learning in a multi-agent setting.

Description of the Tiny Environment

The tiny environment used for our experiments is the same version of the Ethical Gathering Game used in Rodriguez-Soto et al. (2023) [7]. This environment features two agents learning to navigate and gather resources within a 3x4 grid. The key characteristics of this environment include:

- **Full Observability**: Both agents have general information about the environment. They both know the position of other agents, the number of apples on the ground and the state of the donation box. However, the number of apples held by each agent is abstracted in a label (see section 4.1).
- **Two Agents**: The environment is populated with two agents, each with different gathering efficiencies. The efficiency of an agent determines the probability of successfully gathering an apple once the agent is located in the same cell as the apple.
- **Small Ethical Gathering Environment**: The compact size of the grid and the limited number of agents make this environment more manageable and simplify the comparison of different learning algorithms.

Table 5.1 shows the parameters of the tiny environment used in our experiments.

Figure 4.1 provides a visualization of the tiny environment, illustrating the initial setup with agents, apples, and the donation box. The expected behavior for the agents once trained in the tiny environment and having learned their policies can be found in section 4.3. In summary, Agent 1 (more efficient) is expected to gather enough apples for survival and then donate its surplus apples to support Agent 2 (less efficient).

IPPO and ILPPO Training Hyper-parameters

The training hyper-parameters for the IPPO algorithm are consistent with those outlined in section 5.3.1 of Mayoral Macau's final master project [6]. These

Environment Setup		
Parameter	**Value**	**Explanation**
n-agents	2	Number of agents in the environment
efficiency of agents	Agent 1 = 0.85 Agent 2 = 0.2	85% probability of gathering an apple 20% probability of gathering an apple
map-size	tiny	Size of the grid (3x4)
we (ethical weights w_e)	[1, 10]	Weights for individual (1) and ethical (10) rewards
donation-capacity	5	Maximum capacity of the donation box
survival-threshold	10	Minimum apples required for agent survival
partial-observability	False	Agents have general observability of the environment (apples held by each agent are abstracted)
visual-radius	$\emptyset$	No visual radius limitation
apple-regen	0.05	Probability of apple regeneration per step
inequality-mode	loss	Mode to make gathering success probabilistic and dependent on each agent's efficiency

Table 5.1: Environment Setup

include parameters such as actor learning rate, critic learning rate, discount factor (γ), and entropy coefficient.
For ILPPO, we performed an exhaustive hyper-parameter optimization, focusing on the β and η values of equations 2.53. The goal was to find the combination of these values that maximized the expected reward and behaved similarly to IPPO in terms of the ethical metrics in Table 5.2.

Since we are working in an environment with two agents, we need to fine tune the η_0 and (β_t^0, β_t^1) variables of equation 2.53 for each time-step t. We are dealing with two objectives: the individual objective and the ethical objective. Our goal is to lexicographically optimize these objectives by prioritizing the ethical objective first, and only maximizing the individual objective when the ethical objective is stabilized.
To achieve this, we introduce a condition based on the loss function of the ethical objective. The step-by-step process is detailed as follows:

1. **Track the ethical loss function:** After each epoch in the update method of the ILPPO algorithm, we store the loss obtained for the ethical objective in an array, called *recent_losses*.

2. **Evaluate loss stability:** Once all epochs of the same data batch have been processed, we compute the difference between the mean of the *recent_losses* values and the most recent loss (the last element appended in the *recent_losses* array).

3. **Adjust λ_0:**

   - **Large difference:** If the difference between the last recent loss and the mean of *recent_losses* is large, it indicates that the ethical objective is not being optimized properly, as the updates are fluctuating significantly. In this case, λ_0 will store this large difference, weighted by η_0.
   - **Small difference or negative value:** If the difference is a small number or is negative, the ethical objective is stable. Therefore, we can focus on optimizing the individual objective. Here, λ_0 is set to zero.

   By controlling λ_0, we ensure that if the ethical objective is not being optimized correctly, λ_0 will gain more weight. Consequently, in the following update, the ethical objective will be prioritized to ensure it is maximized. Otherwise, if the ethical objective is stable, we can shift our focus to the individual objective.

4. **Adjust (β_t^0, β_t^1):** These variables are used to scalarize the advantages (A_0, A_1) of the ethical and individual rewards, respectively. Rather than using a simple weighted sum of the β vector and the advantages estimates, we perform the following calculation:

$$weighted\_advantages = (\beta_t^0 + \lambda_0) * A_0 + \beta_t^1 * A_1 \tag{5.1}$$

   This method ensures that the ethical advantage A_0 receives more weight when its loss function is not being correctly minimized due to the influence of λ_0.

In summary, the fine-tuning process involves dynamically adjusting λ_0 based on the stability of the ethical objective's loss function. If λ_0 is large, we need to focus on optimizing the ethical objective. When λ_0 is small or zero, the ethical objective is stabilized, and we can focus on optimizing the individual objective. The combination of weights $(\beta_t^0 + \lambda_0)$ and β_t^1 ensures that the ethical objective is prioritized lexicographically.

We found that the combination of $\eta_0 = 2.5$ (the weight used for the λ_0 variable) and $\beta = [1.0, 0.5]$ values had the most significant impact on optimizing the ethical and individual objectives. A high η_0 value is desirable because it amplifies the difference stored in λ_0, assigning more weight to the ethical advantage when there are fluctuations in the ethical objective's loss function, helping correct this fluctuations immediately in the following updates. Vector β is set to 1 for the ethical advantage A_0 and 0.5 to the individual advantage A_1 to ensure that the ethical objective is given more weight even when its loss function is minimized correctly, ensuring that the algorithm gives greater emphasis on the ethical advantage.
The training process demonstrated that when η_0 was set to a high value, it effectively corrected sub-optimal policies by emphasizing immediate performance improvements. On the other hand, when the differences in the ethical recent losses were smaller, the learning process directed the agent towards maximizing individual rewards. This approach ensures a balanced alignment with the ethical objectives of the environment.

Action Selection and Hyper-parameter Adjustment

Following Mayoral Macau's PPO implementation [6], the ILPPO algorithm uses the following approach for action selection:

- In **training mode**, the SoftmaxActionSelection is used to encourage exploration and learning. It performs a multinomial draw (like a weighted random choice) over a given probability distribution over actions, and returns the index of the selected action.
- During **policy evaluation**, the FilterSoftmaxActionSelection is used to filter out actions with probabilities below a certain threshold (0.1). It then applies softmax to the remaining probabilities, and then selects an action using the adjusted probabilities. This approach maintains stochastic behavior while eliminating actions deemed sub-optimal.

We also decreased the entropy coefficient and the learning rates of both the actor and critic networks. Lowering the entropy coefficient reduces exploration in favor of more stable and ethical policies, while adjusting the learning rates helps stabilize training by preventing drastic updates.

Experimental Results and Analysis

Our experiments are conducted across 20 seeds, each training lasting 25M steps, corresponding to 50,000 episodes of 500 steps each. The following aspects are evaluated:

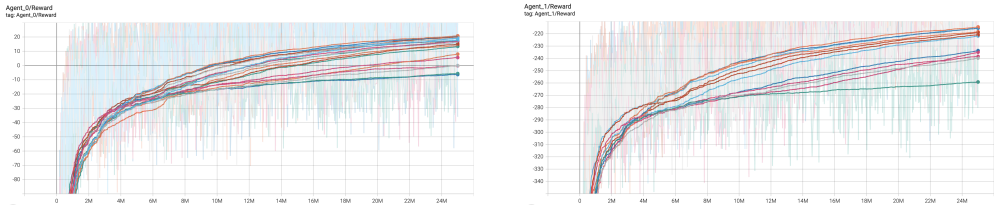
- **Agent_x/Reward**: represents the **accumulated reward** for agent x across the training steps. It corresponds to the mean of all reward components (Reward_0 and Reward_1) received by the agent x during the training process. The rewards are accumulated for each episode and then averaged over the number of episodes in the batch.

- **Agent_x/Reward_0**: represents the **ethical accumulated reward** for agent x from the ethical reward component (Reward_0). It is the sum of all rewards corresponding to the first reward component received by agent x during the training process, normalized over the number of episodes in the batch.

- **Agent_x/Reward_1**: represents the **individual accumulated reward** for agent x from the individual reward component (Reward_1). It is computed the same as the previous metric.

- **Training/Avg Reward**: represents the **average accumulated reward** across both agents during the training process. It is the mean of the Agent_1/Reward and Agent_2/Reward at each step, weighted equally at 0.5 for each agent.

Figures 5.1(a) and 5.1(b) show the evolution of the accumulated reward obtained by each agent during training with ILPPO. Each line represents a specific seed, but all experiments were conducted with the same parameters to ensure consistency. To reduce the variance in the visualizations, each step is smoothed using an exponential moving average with $\omega = 0.999$. The x-axis indicates the step in the training, whereas the y-axis is the accumulated reward obtained. Both

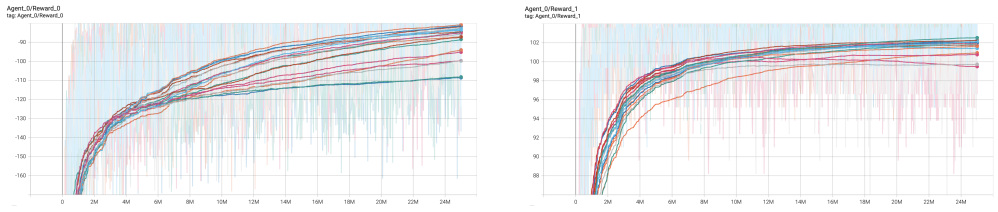


(a) Efficient agent (Agent-1) (b) Inefficient agent (Agent-2)

Figure 5.1: Evolution of the accumulated reward for the efficient (a) and inefficient (b) agent during ILPPO training

agents show an improvement in their accumulated rewards as training progresses. The efficient agent (a) obtains more rewards and reaches convergence around step 20M. The inefficient agent (b) shows more instability from steps 0 to 8M. At step 8M it starts to stabilize and continue receiving rewards with a less steep slope. This could be due to its inefficient nature. Because it gathers fewer rewards, Agent-2

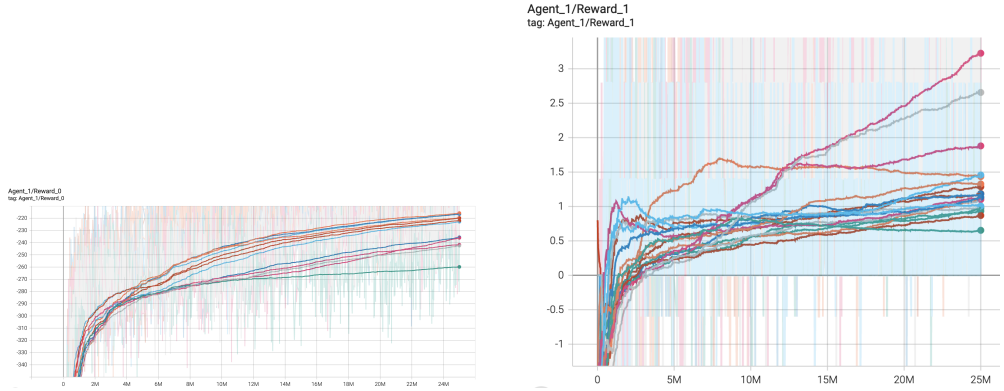
faces more significant challenges in optimizing its objectives, contributing to the higher fluctuation in its performance metrics.



(a) Individual accumulated reward (b) Ethical accumulated reward

Figure 5.2: Evolution of the separate accumulated rewards for Agent-1 during ILPPO training

Examining the separate accumulated rewards of the efficient agent, we see that the ethical accumulated reward 5.2(b) converges faster than the individual reward 5.2(a), reaching a point with minimal slope at step 14M. This indicates that Agent-1 learns to optimize the ethical objective faster and, once optimized, proceeds to optimize the individual accumulated reward.



(a) Individual accumulated reward (b) Ethical accumulated reward

Figure 5.3: Evolution of the separate accumulated rewards for Agent-2 during ILPPO training

For the inefficient agent, the graph of the ethical accumulated reward looks quite different. Due to its inefficient nature, Agent-2 has fewer opportunities to donate apples to the donation box, making it more difficult for it to learn from experience. Figure 5.3(b) still presents a increasing slope, but with many more fluctuations. Looking at Agent-2's individual accumulated reward, 5.3(a) we see that it learns correctly to accept donations from the donation box, increasing steadily

its individual reward.

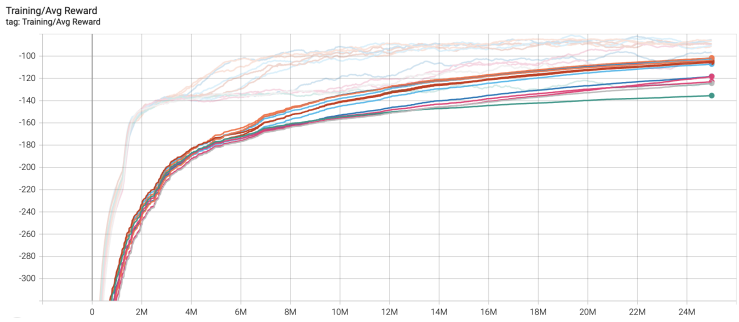


Figure 5.4: Evolution of the average accumulated reward during ILPPO Training

The upward trend in the average accumulated rewards of Figure 5.4 reflects improvements in both individual and ethical rewards across different training runs. The x-axis represents the number of steps in the training process, ranging from 0 to 24M. The y-axis represents the average accumulated reward over the accumulated rewards of Agent_1 and Agent_2 as explained at the beginning of the subsection. The faint lines correspond to the actual values, whereas the more vibrant lines are the smoothed values using the exponential moving average of $\omega = 0.999$.
Figures 5.5(a), 5.5(b) and 5.6 compare the evolution of the cumulative rewards between seed 1 of IPPO (orange) and ILPPO (blue).

(a) Agent 1 (b) Agent 2

Figure 5.5: Comparison of the cumulative rewards between IPPO (orange) and ILPPO (blue) for both agents

We can observe three different stages in the ILPPO training:

- From steps 0M to 8M, ILPPO agents receive lower cumulative rewards compared to IPPO agents. This could be because, initially, ILPPO agents focus on optimizing the ethical objective, giving more weight to the ethical advantage. However, at the beginning of the training process, the most optimal actions are the ones that act greedy, striving to collect apples. If the agents

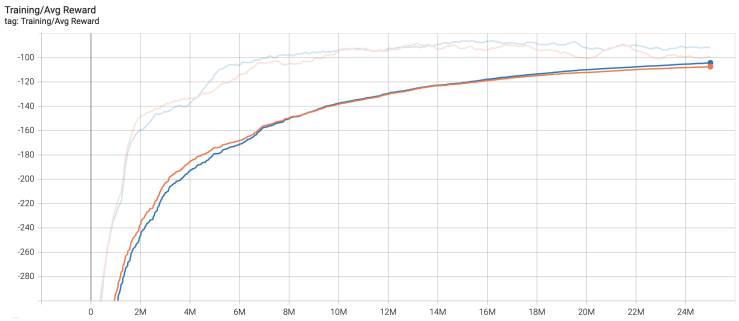


Figure 5.6: Comparison of the average cumulative reward between IPPO (orange) and ILPPO (blue)

are more encouraged to donate but they do not have apples, it is natural that they obtain lower rewards.

- From steps 8M to 10M the behaviour of ILPPO and IPPO are very similar. In ILPPO the advantage weights should have shifted once the ethical loss function is stabilized, giving priority to the individual objective.

- From steps 10M to 24M, ILPPO slightly obtains higher cumulative rewards than IPPO, probably due to the stabilization of the ethical objective.

The Figure 5.6 displays the average cumulative reward between IPPO and ILPPO for seed 1. It follows a similar trend to that in figure 5.6, but the gaps between the two graphs are closer, due to the similar behaviour of the inefficient agent 5.5(b). Looking at the average cumulative reward, it follows a similar trend to figures 5.5(a) and 5.5(b).

Numerical Analysis of Ethical Policies

In this subsection, we provide a numerical analysis of the ethical policies learned in the tiny environment under the ILPPO algorithm. The objective of these experiments is to determine how close the learned policies align with those obtained with the IPPO algorithm, which has been tested and proven to provide ethically optimal policies [6]. We conducted 1000 simulations for each of the 20 seeds, and evaluated its results under the ILPPO and IPPO algorithms by analyzing the following key metrics:

- **Step to survival**: The average step at which each agent survives, averaged over all simulations.

- **Times not survived**: The number of simulations in which the agent did not survived (collected enough apples to reach the survival threshold) across all 20000 simulations.
- **Accumulated reward**: The accumulated reward obtained by each agent at the end of each simulation, averaged over all simulations.
- **Individual reward and Ethical reward**: The accumulated individual and ethical rewards for each agent at the end of each simulation, averaged over all simulations.
- **DB not filled**: The number of times the donation box was not full at the end of the simulations.
- **Step DB full**: The average step at which the agents managed to fill the donation box, averaged over all simulations.
- **R'$_E$ actions**: The average number of missed ethical actions over all simulations. That is, missed opportunities where agents could have donated apples but did not.
- **R'$_N$ actions**: The average number of unethical actions over all simulations. Instances where agents took apples from the donation box despite having enough apples to survive (acting greedily).
- **Sub-optimal actions**: The average number of instances where agents donated apples to the donation box despite not having enough apples to survive, across all simulations.

Table 5.2 summarizes the numerical comparison of these metrics.

Table 5.2: Comparison of IPPO and ILPPO Metrics

Metric	ILPPO Agent 0	ILPPO Agent 1	IPPO Agent 0	IPPO Agent 1
Step to Survival	107.63 ± 30.61	209.33 ± 41.92	110.71 ± 32.18	206.65 ± 42.35
Times not Survived	13/20000	15/20000	18/20000	24/20000
Accumulated Reward	24.73 ± 36.44	-206.76 ± 42.51	20.81 ± 38.81	-204.29 ± 43.12
Individual Reward	-77.60 ± 35.19	-208.25 ± 42.35	-	-
Ethical Reward	102.33 ± 5.05	1.50 ± 3.16	-	-
DB not Full	18/20000	18/20000	30/20000	30/20000
Step DB Full	271.56 ± 49.41	271.56 ± 49.41	268.52 ± 50.55	268.52 ± 50.55
R'$_E$ Actions	0.05 ± 0.22	0.00 ± 0.06	0.03 ± 0.20	0.39 ± 4.27
R'$_N$ Actions	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Sub-optimal Actions	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Let's analyse table 5.2, grouped into several key areas:

Survival Metrics: Both agents using ILPPO and IPPO have similar mean times to survival, with ILPPO Agent 0 slightly faster than IPPO Agent 0 and ILPPO Agent 1 slightly slower than IPPO Agent 1. The standard deviations indicate variability, with ILPPO Agent 0 showing slightly less variability compared to IPPO Agent 0. ILPPO agents have fewer non-survival simulations, suggesting that ILPPO may provide more consistent survival performance. Note that the "Times not Survived" metric is measured as a sum over the 20,000 simulations.

Reward Metrics: ILPPO Agent 0 achieved a slightly higher mean total reward compared to IPPO Agent 0. Both agents 1 have negative total rewards, but ILPPO Agent 1 has a slightly lower mean total reward compared to IPPO Agent 1. The standard deviations indicate a similar level of variability between the algorithms. Regarding the ethical and individual cumulative reward, we couldn't compare this metrics because the reward function in PPO is scalarized. However, we can ensure that ILPPO Agent 0 received a significantly higher ethical reward compared to ILPPO Agent 1, suggesting a focus on ethical behavior for Agent 0.

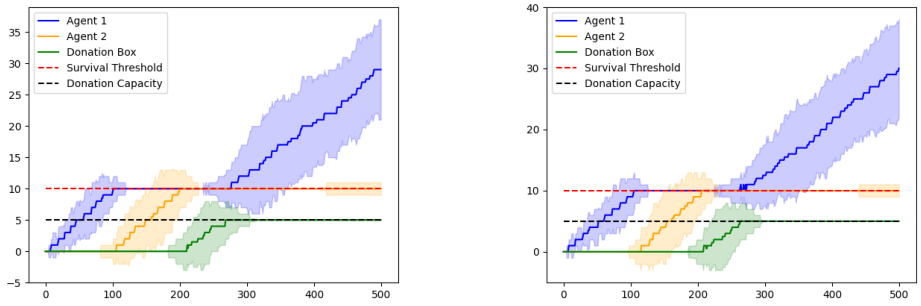
Donation Box Metrics: ILPPO had fewer instances where the donation box was not full, indicating better performance in managing the donation box. The average step in which the donation box is filled is similar for both ILPPO and IPPO agents.

Action Metrics: Both ILPPO and IPPO agents had zero sub-optimal and unethical actions recorded. ILPPO agents exhibited slightly higher missed ethical actions for Agent 0, but lower missed ethical action for Agent 1.

Overall, the ethical policies obtained with the ILPPO approach are very similar to the ethical-optimal policies obtained with IPPO. The ILPPO algorithm shows a slight edge in terms of survival consistency for both agents and ethical behavior for Agent 1, while maintaining similar performance to IPPO in most other metrics. Furthermore, ILPPO does not require to compute the weights of the individual and ethical rewards in the ethical embedding process, making it simpler to implement and potentially more adaptable to different environments or scenarios without the need for extensive parameter tuning. Note that, although the differences between the metrics are small, the T-test results indicate significant differences between ILPPO and IPPO across all comparable key metrics (see Table B.3).

Evaluation of the learnt ethical policy

The provided figure 5.7(a) illustrates the evolution of apple collection by Agent 1 (efficient, blue line) and Agent 2 (inefficient, orange line) as well as the apples in the donation box (green line) over an aggregation of 100 simulations of 500 steps. Agent 1, being 0.85 efficient, quickly gathers apples and reaches the survival threshold (red dashed line) around step 100. Agent 2 gathers apples more slowly. After reaching the survival threshold, Agent 1 begins donating excess apples to the donation box, and Agent 2 begins accepting donations to reach the survival threshold (step 200). Once both agents have ensured their survival, they start to donate its surplus of apples to the donation box. The green line increases steadily until the donation box reaches its capacity (black dashed line) around step 270. With the donation box full, both agents resume gathering apples for themselves. Agent 1, due to its higher efficiency, accumulates apples faster, as shown by the steep rise in the blue line. The figure 5.7(a) demonstrates that the



(a) ILPPO (b) IPPO

Figure 5.7: IPPO (a) and IPPO (b) evolution of the number of apples collected by Agent 1, Agent 2, and the Donation Box over time. The shaded regions represent the interquartile range for each agent.

agents successfully follow the desired ethical policies. Agent 1 helps Agent 2 survive by donating apples, both agents fill the donation box, and they continue gathering apples for themselves once the donation box is full.
Moreover, comparing the ethical policy of ILPPO 5.7(a) versus the one obtained with IPPO 5.7(b) we observe that both policies behave very similar.

Chapter 6

Conclusion and Future Work

This Bachelor's Degree Final Project explored the integration of ethical values into RL through the ILPPO algorithm. By implementing a lexicographic ordering of multiple objectives, ILPPO ensures that higher-priority ethical objectives are met before addressing other goals. Theoretical foundations include Multi-Objective Markov Decision Process (MOMDP), Proximal Policy Optimization, lexicographical policy based learning and stochastic games.

We implemented ILPPO in the Ethical Gathering Game, where agents learn to behave in alignment with the moral value of beneficence, managing to compensate for inequality by gathering and sharing apples. Our experiments showed that ILPPO effectively learns ethical optimal policies similar to those generated with IPPO. Furthermore, ILPPO outperforms IPPO in promoting ethical behaviour in some key metrics such as agent survival and donation box filling, while maintaining to zero the sub-optimal actions and missed ethical actions.

The results of this research are encouraging, and we already envision several areas for future work to further test and enhance ILPPO. Future research should scale ILPPO into large environments, testing how it handles more agents and larger state spaces. Moreover, extensive experiments to fine-tune the β and η parameters could be conducted. These parameters play a crucial role in balancing ethical objectives and individual rewards, and optimizing them can enhance the algorithm's performance.

Appendix A

Hyper-parameters for the Learning Approach

Environment Setup	
max-steps	Max number of steps per episode
n-agents	Number of agents
map-size	Size of the map
efficiency	Efficiency picking an apple
donation-capacity	Donation box capacity
survival-threshold	Survival threshold
Training hyper-parameters	
seed	Seed of the experiment
batch-size	Steps between policy updates
tot-steps	Total time-steps of the experiment
IPPO hyper-parameters	
actor-lr	Actor learning rate
critic-lr	Critic learning rate
anneal-lr	Toggles annealing learning rates
clip	Surrogate clipping coefficient
gamma	Discount factor
gae-lambda	Gae lambda
ent-coef	Entropy coefficient
anneal-entropy	Toggles annealing entropy
concavity-entropy	Determines the curvature of the function that adjusts the entropy
v-coef	Value function coefficient
n-epochs	Number of update epochs
norm-adv	Toggles advantages normalization
max-grad-norm	Maximum norm for gradient clipping
h-size	Layers size
ILPPO hyper-parameters	
we-reward0	Weight given to the individual reward
we-reward1	Weight given to the ethical reward (beneficence)
beta-values	Weights assigned to each advantage to compute its weighted sum. Corresponding to the β_t^i in the 2.53 formula.
eta-value	Weight assigned to past loss actor values. Corresponding to the η_t in the 2.53 formula.
prioritize-performance-over-safety	train the agent to first optimize the individual reward (True) or the ethical reward (False). By default is set to False.

Table A.1: Hyper-parameters for the Learning Approach

Appendix B

Simulation results separated by seed

B.1 Simulation results for IPPO algorithm

Seed	Mean Time to Survival		Times not Survived		Mean Total Reward		Mean Time to Full	Times not Full
	Agent 0	Agent 1	Agent 0	Agent 1	Agent 0	Agent 1	Time	Count
1	112.51 ± 33.55	209.33 ± 43.51	0	0	19.30 ± 37.13	-206.18 ± 43.66	270.92 ± 50.00	0
2	112.91 ± 33.37	209.36 ± 45.36	3	3	16.89 ± 45.86	-207.47 ± 47.05	270.89 ± 53.20	4
3	102.01 ± 29.39	193.36 ± 38.64	0	0	36.72 ± 32.38	-192.02 ± 38.66	243.38 ± 42.72	0
4	110.38 ± 31.14	205.07 ± 39.61	0	0	20.93 ± 34.53	-201.62 ± 39.65	270.11 ± 47.89	0
5	110.83 ± 30.80	206.00 ± 40.70	1	1	23.52 ± 37.43	-204.90 ± 41.33	262.87 ± 47.27	1
6	107.61 ± 30.00	203.61 ± 40.36	1	2	22.83 ± 36.80	-200.99 ± 41.80	266.59 ± 48.60	2
7	109.38 ± 31.04	203.70 ± 40.06	1	2	22.20 ± 37.89	-201.44 ± 41.45	267.24 ± 48.94	2
8	113.55 ± 32.22	209.50 ± 41.12	0	0	15.36 ± 35.86	-205.86 ± 41.42	276.65 ± 50.75	1
9	108.61 ± 32.43	206.08 ± 42.44	0	0	23.04 ± 35.98	-203.08 ± 42.55	269.20 ± 48.76	0
10	115.60 ± 32.83	213.09 ± 42.21	0	0	13.91 ± 36.05	-209.52 ± 42.65	276.36 ± 48.83	0
11	114.62 ± 33.18	213.31 ± 42.99	1	1	15.15 ± 39.84	-210.42 ± 43.59	277.83 ± 51.28	2
12	121.20 ± 35.91	224.88 ± 47.67	1	2	8.56 ± 42.12	-223.66 ± 48.29	288.60 ± 56.19	3
13	111.93 ± 32.96	206.48 ± 41.25	0	0	17.83 ± 36.24	-201.94 ± 41.65	273.91 ± 49.87	0
14	109.24 ± 33.03	201.90 ± 41.70	0	0	24.46 ± 36.13	-199.42 ± 41.76	262.91 ± 49.23	0
15	110.92 ± 32.13	205.74 ± 40.78	0	0	24.39 ± 35.47	-204.16 ± 40.86	262.26 ± 46.22	0
16	109.22 ± 32.37	205.80 ± 47.72	10	13	14.05 ± 61.18	-208.78 ± 53.37	270.76 ± 58.22	13
17	108.19 ± 30.49	205.97 ± 40.61	0	0	22.03 ± 33.99	-202.31 ± 40.84	270.74 ± 51.87	2
18	106.10 ± 30.23	197.46 ± 39.95	0	0	31.93 ± 33.43	-196.13 ± 40.01	248.70 ± 44.80	0
19	109.85 ± 31.18	204.37 ± 39.36	0	0	21.93 ± 34.34	-201.12 ± 39.47	267.94 ± 46.77	0
20	109.58 ± 30.16	208.00 ± 40.48	0	0	21.20 ± 33.72	-204.68 ± 40.82	272.58 ± 48.95	0
Average	**110.71 ± 32.18**	**206.65 ± 42.35**	**18**	**24**	**20.81 ± 38.81**	**-204.29 ± 43.12**	**268.52 ± 50.55**	**30**

Table B.1: IPPO Performance Metrics (Part 1)

B.2 Simulation results for ILPPO algorithm

B.3 T-Test Results Between IPPO and ILPPO Metrics

Seed	Mean R'_E Actions		Mean R'_N Actions		Mean Suboptimal Actions	
	Agent 0	Agent 1	Agent 0	Agent 1	Agent 0	Agent 1
1	0.04 ± 0.20	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
2	0.03 ± 0.16	0.00 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
3	0.02 ± 0.14	1.05 ± 6.59	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
4	0.01 ± 0.11	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
5	0.04 ± 0.20	3.00 ± 11.66	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
6	0.03 ± 0.16	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
7	0.03 ± 0.18	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
8	0.04 ± 0.19	0.00 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
9	0.02 ± 0.13	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
10	0.05 ± 0.25	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
11	0.05 ± 0.23	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
12	0.04 ± 0.23	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
13	0.03 ± 0.17	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
14	0.07 ± 0.26	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
15	0.06 ± 0.27	2.89 ± 11.39	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
16	0.03 ± 0.17	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
17	0.04 ± 0.21	0.00 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
18	0.03 ± 0.18	0.95 ± 6.29	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
19	0.03 ± 0.16	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
20	0.03 ± 0.20	0.00 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Average	**0.03 ± 0.20**	**0.39 ± 4.27**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**

Table B.2: IPPO Performance Metrics (Part 2)

Seed	Mean Time to Survival		Times not Survived		Mean Total Reward		Mean Time to Full	Times not Full
	Agent 0	Agent 1	Agent 0	Agent 1	Agent 0	Agent 1	Time	Count
1	108.32 ± 30.76	210.85 ± 42.47	0	0	24.23 ± 34.22	-207.85 ± 42.87	271.69 ± 49.81	1
2	106.70 ± 30.41	208.19 ± 41.49	0	0	25.88 ± 33.65	-205.71 ± 41.63	271.72 ± 48.71	0
3	106.62 ± 30.46	207.78 ± 40.38	0	0	25.30 ± 33.66	-204.07 ± 40.41	269.15 ± 48.37	0
4	107.01 ± 29.51	209.54 ± 42.54	3	3	23.99 ± 43.18	-207.71 ± 44.19	269.33 ± 49.39	3
5	106.31 ± 30.35	205.82 ± 40.49	0	0	26.70 ± 34.03	-202.50 ± 40.67	269.01 ± 47.21	0
6	109.45 ± 30.78	209.71 ± 40.87	0	0	24.24 ± 34.20	-207.07 ± 40.99	271.57 ± 47.04	0
7	103.54 ± 29.83	201.37 ± 41.80	7	7	25.85 ± 54.43	-201.47 ± 45.94	263.35 ± 51.37	7
8	108.91 ± 30.43	210.99 ± 39.80	0	0	23.29 ± 33.77	-208.25 ± 39.82	274.80 ± 47.95	0
9	108.72 ± 30.45	210.72 ± 43.06	0	0	24.47 ± 33.74	-208.07 ± 43.27	273.39 ± 49.86	1
10	104.24 ± 30.19	201.24 ± 39.74	0	0	29.15 ± 33.18	-197.96 ± 40.14	261.63 ± 47.20	0
11	108.80 ± 30.76	212.68 ± 42.52	1	2	22.68 ± 37.35	-210.86 ± 43.72	273.74 ± 48.73	2
12	109.33 ± 30.94	214.55 ± 43.20	0	0	23.32 ± 34.68	-211.97 ± 43.27	279.04 ± 51.86	1
13	105.32 ± 30.45	208.79 ± 41.42	0	0	28.29 ± 33.95	-206.26 ± 41.53	270.49 ± 48.67	0
14	108.28 ± 31.55	212.49 ± 43.02	0	0	24.78 ± 35.26	-210.06 ± 43.25	277.28 ± 51.44	0
15	107.24 ± 29.41	210.77 ± 39.86	1	1	25.00 ± 36.48	-208.36 ± 40.52	274.98 ± 48.77	1
16	105.83 ± 30.23	205.00 ± 40.70	0	0	27.12 ± 33.67	-201.59 ± 40.77	264.91 ± 46.43	0
17	107.17 ± 29.77	207.26 ± 42.56	1	2	24.30 ± 36.90	-205.18 ± 43.85	269.60 ± 50.65	2
18	107.07 ± 29.38	207.09 ± 39.41	0	0	25.19 ± 32.66	-203.72 ± 39.45	267.51 ± 45.98	0
19	116.03 ± 33.90	223.30 ± 46.43	0	0	15.22 ± 37.49	-221.34 ± 46.48	287.70 ± 53.46	0
20	107.70 ± 30.33	208.53 ± 40.95	0	0	25.58 ± 33.78	-205.18 ± 41.25	270.39 ± 48.07	0
Aggregate	**107.63 ± 30.61**	**209.33 ± 41.92**	**13**	**15**	**24.73 ± 36.44**	**-206.76 ± 42.51**	**271.56 ± 49.41**	**18**

Table B.3: ILPPO Performance Metrics (Part 1)

Seed	Mean Individual Reward		Mean Ethical Reward		Mean R′_E Actions	
	Agent 0	Agent 1	Agent 0	Agent 1	Agent 0	Agent 1
1	-78.12 ± 33.86	-209.37 ± 42.51	102.35 ± 4.30	1.53 ± 3.26	0.01 ± 0.09	0.00 ± 0.00
2	-77.14 ± 33.34	-206.85 ± 41.55	103.02 ± 3.71	1.14 ± 2.77	0.05 ± 0.24	0.00 ± 0.00
3	-76.40 ± 33.42	-206.25 ± 40.44	101.70 ± 4.51	2.18 ± 3.69	0.04 ± 0.20	0.00 ± 0.00
4	-77.77 ± 39.40	-209.63 ± 43.94	101.76 ± 7.11	1.93 ± 3.48	0.05 ± 0.23	0.00 ± 0.00
5	-75.19 ± 33.28	-204.30 ± 40.54	101.89 ± 4.77	1.81 ± 3.54	0.05 ± 0.22	0.00 ± 0.00
6	-78.64 ± 33.73	-208.32 ± 40.94	102.88 ± 3.87	1.25 ± 2.84	0.03 ± 0.17	0.00 ± 0.00
7	-75.50 ± 47.62	-203.45 ± 45.60	101.35 ± 9.55	1.97 ± 3.50	0.04 ± 0.23	0.00 ± 0.00
8	-79.28 ± 33.20	-209.50 ± 39.88	102.57 ± 4.25	1.25 ± 2.94	0.04 ± 0.22	0.01 ± 0.19
9	-78.20 ± 33.50	-209.31 ± 43.13	102.68 ± 3.95	1.23 ± 2.89	0.05 ± 0.24	0.00 ± 0.00
10	-73.06 ± 33.00	-199.69 ± 39.84	102.21 ± 4.24	1.74 ± 3.33	0.04 ± 0.20	0.00 ± 0.00
11	-79.75 ± 35.86	-212.23 ± 43.46	102.44 ± 5.68	1.36 ± 3.03	0.06 ± 0.25	0.00 ± 0.07
12	-79.42 ± 34.00	-213.12 ± 43.27	102.74 ± 4.09	1.14 ± 2.82	0.04 ± 0.21	0.00 ± 0.00
13	-74.54 ± 33.50	-207.39 ± 41.49	102.84 ± 3.87	1.13 ± 2.74	0.04 ± 0.19	0.00 ± 0.09
14	-78.05 ± 34.88	-211.06 ± 43.09	102.83 ± 3.81	1.00 ± 2.55	0.07 ± 0.28	0.01 ± 0.16
15	-77.49 ± 34.71	-209.86 ± 40.40	102.49 ± 5.21	1.50 ± 3.22	0.03 ± 0.18	0.00 ± 0.00
16	-75.19 ± 33.28	-204.30 ± 40.54	101.89 ± 4.77	1.81 ± 3.54	0.05 ± 0.22	0.00 ± 0.00
17	-77.66 ± 35.12	-206.77 ± 43.63	101.96 ± 6.36	1.59 ± 3.28	0.05 ± 0.21	0.00 ± 0.00
18	-76.57 ± 32.17	-205.46 ± 39.46	101.77 ± 4.65	1.74 ± 3.35	0.04 ± 0.18	0.00 ± 0.03
19	-88.20 ± 37.13	-221.98 ± 46.48	103.42 ± 3.41	0.64 ± 2.12	0.06 ± 0.25	0.00 ± 0.00
20	-76.10 ± 33.33	-207.01 ± 41.03	101.68 ± 4.80	1.83 ± 3.49	0.08 ± 0.28	0.00 ± 0.03
Aggregate	**-77.60 ± 35.19**	**-208.25 ± 42.35**	**102.33 ± 5.05**	**1.50 ± 3.16**	**0.05 ± 0.22**	**0.00 ± 0.06**

Table B.4: ILPPO Performance Metrics (Part 2)

Seed	Mean Suboptimal Actions	
	Agent 0	Agent 1
1	0.00 ± 0.00	0.00 ± 0.00
2	0.00 ± 0.00	0.00 ± 0.00
3	0.00 ± 0.00	0.00 ± 0.00
4	0.00 ± 0.00	0.00 ± 0.00
5	0.00 ± 0.00	0.00 ± 0.00
6	0.00 ± 0.00	0.00 ± 0.00
7	0.00 ± 0.00	0.00 ± 0.00
8	0.00 ± 0.00	0.00 ± 0.00
9	0.00 ± 0.00	0.00 ± 0.00
10	0.00 ± 0.00	0.00 ± 0.00
11	0.00 ± 0.00	0.00 ± 0.00
12	0.00 ± 0.00	0.00 ± 0.00
13	0.00 ± 0.00	0.00 ± 0.00
14	0.00 ± 0.00	0.00 ± 0.00
15	0.00 ± 0.00	0.00 ± 0.00
16	0.00 ± 0.00	0.00 ± 0.00
17	0.00 ± 0.00	0.00 ± 0.00
18	0.00 ± 0.00	0.00 ± 0.00
19	0.00 ± 0.00	0.00 ± 0.00
20	0.00 ± 0.00	0.00 ± 0.00
Aggregate	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$

Table B.5: ILPPO Performance Metrics (Part 3)

Metric	**t-statistic**	**p-value**
Survival Times (Agent 0)	9.8151	0.0000
Survival Times (Agent 1)	-6.3681	0.0000
Total Rewards (Agent 0)	-10.4105	0.0000
Total Rewards (Agent 1)	5.7750	0.0000
Missed Ethical Actions (Agent 0)	-5.0107	0.0000
Missed Ethical Actions (Agent 1)	13.0413	0.0000
Non-Ethical Actions (Agent 0)	nan	nan
Non-Ethical Actions (Agent 1)	nan	nan
Suboptimal Actions (Agent 0)	nan	nan
Suboptimal Actions (Agent 1)	nan	nan
Donation Full Times	-6.0861	0.0000

Table B.6: T-test Results Between PPO and LPPO Metrics

Bibliography

[1] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL: https://www.marl-book.com.

[2] Vijay R. Konda and John N. Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 2000, pp. 1008–1014. URL: https://proceedings.neurips.cc/paper/2000/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

[3] *Lagrangian Methods for Constrained Optimization*. https://en.wikipedia.org/wiki/Lagrange_multiplier. Accessed: 2024-05-21.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[5] Joel Z. Leibo et al. "Multi-agent Reinforcement Learning in Sequential Social Dilemmas". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2017)* (2017). URL: https://arxiv.org/abs/1702.03037.

[6] Arnau Mayoral Macau. "Ethical Considerations in Multi-Agent Reinforcement Learning: A Lexicographic Approach". TFM, Accessed: 2024-05-19. MA thesis. Universidad Politecnica de Madrid, 2023. URL: https://oa.upm.es/75349/1/TFM_ARNAU_MAYORAL_%20MACAU.pdf.

[7] Manel Rodriguez-Soto, Maite Lopez-Sanchez, and Juan A. Rodriguez-Aguilar. "Multi-objective reinforcement learning for designing ethical multi-agent environments". In: *Neural Computing and Applications* (2023). Published: 23 August 2023, Received: 13 October 2022, Accepted: 14 July 2023. DOI: 10.1007/s00521-023-08898-y. URL: https://doi.org/10.1007/s00521-023-08898-y.

[8] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[9] Joar Skalse et al. "Lexicographic Multi-Objective Reinforcement Learning". In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. IJCAI-2022. International Joint Conferences on Artificial Intelligence Organization, July 2022. DOI: 10.24963/ijcai.2022/476. URL: http://dx.doi.org/10.24963/ijcai.2022/476.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[11] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[12] Christian Schröder de Witt et al. "Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?" In: *CoRR* abs/2011.09533 (2020). arXiv: 2011.09533. URL: https://arxiv.org/abs/2011.09533.