TU WIEN Informatics

Normative Compliance in Lexicographic Multi-Objective Reinforcement Learning Agents

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Bernhard Schiehl, BSc.
Matrikelnummer 11778260

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ezio Bartocci
Zweitbetreuung: Univ.Prof.in Dr.in Agata Ciabattoni

Wien, 5. Dezember 2023

Bernhard Schiehl	Ezio Bartocci

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Normative Compliance in Lexicographic Multi-Objective Reinforcement Learning Agents

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Bernhard Schiehl, BSc.
Registration Number 11778260

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Ezio Bartocci
Second advisor: Univ.Prof.in Dr.in Agata Ciabattoni

Vienna, December 5, 2023

Bernhard Schiehl | Ezio Bartocci

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Bernhard Schiehl, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 5. Dezember 2023

Bernhard Schiehl

Kurzfassung

Autonome, auf maschinellem Lernen basierte KI-Agenten haben begonnen, Teil unseres Alltags zu werden, von selbstfahrenden Autos bis hin zu Chatbots als persönliche Assistenten. Viele dieser Agenten werden mit Hilfe von Reinforcement Learning trainiert, wobei ein System von Belohnungen oder Bestrafungen eingesetzt wird, um bestimmte Verhaltensweisen zu fördern oder zu unterbinden. In jüngster Zeit hat der rasche Fortschritt in diesen Bereichen Sicherheitsbedenken aufgeworfen, sodass manche Experten sogar einen vorübergehenden Entwicklungsstopp forderten. Wenn wir KI-Agenten in unsere Gesellschaft integrieren wollen, sollten wir sicherstellen, dass sie ethischen, rechtlichen und sozialen Normen unterliegen, ähnlich wie Menschen.

Diese Arbeit befasst sich mit der Frage, wie wir sicherstellen können, dass Agenten, die mit Reinforcement Learning trainiert wurden, Normen einhalten, ohne an Nutzen zu verlieren. Unser Ansatz erweitert bestehende Techniken mit Algorithmen für lexikographische Mehrzielprobleme. Bei diesen sind die Ziele nach Priorität geordnet und werden unter der Bedingung optimiert, dass vorherige Ziele bereits optimiert wurden. Mit einem externen Theorembeweiser für deontische Logik - die Logik der Verpflichtungen und Erlaubnisse - bestrafen wir den Agenten für die Verletzung von Normen. Indem er zunächst diese normativen Strafen minimiert und dann seine anderen Ziele optimiert, lernt der Agent, sein Ziel zu erreichen und dabei eine Vielzahl von Normen einzuhalten.

Wir evaluieren diesen Ansatz experimentell, indem wir ihn mit verschiedenen Agenten testen, die das Arcade-Spiel Pac-Man spielen. In einer vereinfachten Version des Spiels lernten die Agenten, die Normen nicht zu verletzen und gewannen die meisten ihrer Testspiele. Sie waren jedoch nicht in der Lage, das gleiche Leistungsniveau in einer komplexeren Umgebung zu erreichen.

Abstract

Autonomous AI agents based on machine learning have started to commonly aid us in our everyday lives, from self-driving cars to personal assistant chatbots. Many of these agents are trained using reinforcement learning, utilizing a system of rewards or punishments to encourage or discourage certain behaviors. Recently, the rapid progress in these fields raised safety concerns, with some experts even calling for a temporary development stop. If we want to integrate AI agents into our society they should be subjected to ethical, legal and social norms, just like humans.

This thesis is concerned with the question of how to ensure normative compliance in reinforcement learning agents while conserving their usefulness. Our approach extends existing techniques from normative reinforcement learning with algorithms for lexicographic multi-objective problems, where objectives are ordered by priority and optimized subject to the constraint that prior objectives have already been optimized. Using an external reasoning module for Deontic logic - the logic of obligations and permissions - we penalize the agent for violating norms. By first minimizing these normative penalties and then optimizing its other objectives the agent learns to reach its goal while complying with a variety of norms.

We evaluate this approach experimentally by testing it with various agents playing the arcade game Pac-Man. The agents won most of their test games while complying with the normative system in a simplified version of the game. However, they were unable to reach the same level of performance in a more complex environment.

Contents

CHAPTER 1

Introduction

1.1 Motivation

"In an age characterized by the relentless march of technological progress, artificial intelligence (AI) has become an inseparable companion in our daily lives, reshaping the way we live, work, and interact. From personalized recommendations on streaming platforms to the predictive text on our smartphones, AI's influence is ubiquitous, often operating seamlessly in the background. Take, for instance, the algorithms powering social media platforms, curating content tailored to individual preferences. While these advancements offer unprecedented convenience and efficiency, they also raise profound ethical concerns. The impact of AI extends beyond mere convenience; it can significantly affect individuals and communities. For example, in the realm of hiring processes, AI-driven algorithms may inadvertently perpetuate biases present in historical data, potentially leading to discriminatory outcomes. As we witness AI's pervasive integration into daily affairs, the imperative to establish robust AI ethics becomes increasingly evident." This is the response that ChatGPT 3.5 gave to the following prompt: "Write an introduction where you first describe the impact of artificial intelligence in daily life and then go on to argue the importance of AI ethics. Give specific examples to illustrate your point."

It is clear that large language models (LLMs) like OpenAI's GPT series are now available to use for everyone with an internet connection. This creates several ethical, legal and safety challenges as LLMs can be used to generate a variety of harmful content. In their GPT-4 technical report [Ope23] OpenAI state that in the early stages of development, GPT-4 could "generate instances of hate speech, discriminatory language, incitements to violence, or content that is then used to either spread false narratives or to exploit an individual." Moreover, while some high-level theoretical guidelines for trustworthy AI already exist (e.g. [int19]), the question of how to enforce these guidelines is unsolved.

This explains why establishing techniques for safe and ethical AI agents is currently widely viewed as one of the world's most pressing problems.

Rapid progress in the field of machine learning is one of the reasons why AI is nowadays used for numerous applications. *Reinforcement Learning* (RL) is a paradigm of machine learning capable of training autonomous agents to take intelligent actions in a dynamic environment. Past successes of RL include superhuman performance in games such as Chess, Go, Shogi [Sil+17] and Dota 2 [Ope+19], as well as potential applications for autonomous driving [Sal+17]. These state of the art algorithms belong to the area of Deep Reinforcement Learning because they harness the power of artificial neural networks.

1.2 Problem statement and research questions

Constrained RL is an established research field concerned with developing techniques for limiting the actions of a RL agent. Much of the literature in constrained RL focuses on simple safety constraints and neglects scenarios where the constraints conflict with each other and complying with the safety specification is not possible. The goal of this thesis is to overcome these challenges with a more general approach: We want to design RL agents capable of complying with a *normative system*, regardless if the normative system contains ethical, legal or social norms.

While normative compliance is the primary goal we must not forget about the original objective that we wanted the RL agent to attain. After all, an autonomous vehicle that does not move will surely not violate any norms by colliding with other traffic participants, but it will also be unable to reach its destination. That leads us to the following research question: **How can we ensure that RL agents fulfill the objective they were trained for, while also complying with a normative system as much as possible?** Training RL agents for this purpose is a tight balancing act because we need to teach the agent normative compliance while not completely avoiding situations where norm violations are possible in order to retain the usefulness of the agent.

Our attempt at answering the above research question utilizes the normative RL technique of norm-guided RL (NGRL) [Neu22], which relies on automated normative reasoning with deontic logic, the branch of formal logic concerned with obligation and related notions. NGRL is able to train certain types of RL agents to comply with certain types of normative systems. However, it suffers from scaling issues as it is too computationally expensive for practical application in complex RL environments. Combining NGRL with deep RL would greatly expand its applicability but currently these two techniques are incompatible with each other. Thus, a second, more specific research question of the present thesis is: **How can we extend NGRL with deep RL methods?** Our proposed solution uses algorithms for lexicographic multi-objective RL (LMORL) problems [Ska+22], which are problems involving multiple objectives where the goal is to optimize for the first objective, and subject to this constraint also optimize for the second objective, and so on.

1.3 Outline

The main part of the thesis consists of chapters 2, 3, 4 and 5. Chapter 2 gives a brief introduction to value-based RL, multi-objective RL, normative reasoning and deontic logic. In chapter 3 we review existing approaches to safe and normative RL and explain the unique contributions of the present work. Chapter 4 introduces norm-guided lexicographic multi-objective RL (NGLMORL) and the setting of its experimental evaluation. The results of the experiments are presented in chapter 5. Finally, chapter 6 concludes with a summary and points out directions for future research.

CHAPTER 2

Preliminaries

In this chapter, we will cover the essential foundational knowledge required to grasp the subsequent content of this thesis. Firstly, an introduction to reinforcement learning is given, with a special focus on value-based reinforcement learning and multi-objective reinforcement learning. Secondly, we explore the main types of norms and unique challenges in normative reasoning. Thirdly, the formal logic of normative reasoning used for this thesis is defined.

2.1 Reinforcement Learning

The field of machine learning is often divided into three broad categories which mainly differ in the type of feedback given to the learning algorithm:

- In supervised learning, each training example is manually labelled with the desired output. By computing an error function from model predictions and the labels, the algorithm adjusts its parameters so that future predictions match the labels better.
- Unsupervised learning comprises several machine learning techniques that take unlabelled data as input. A common goal for such algorithms is finding hidden patterns in the input.
- A reinforcement learning (RL) algorithm teaches agents to achieve a certain goal by rewarding or punishing them for their interactions with an environment. If the inner workings of the environment are unknown to the agent, it discovers through trial and error which actions should be taken in a certain environment state to maximize its rewards.

Formally, a reinforcement learning environment is described by Markov decision process (MDP):

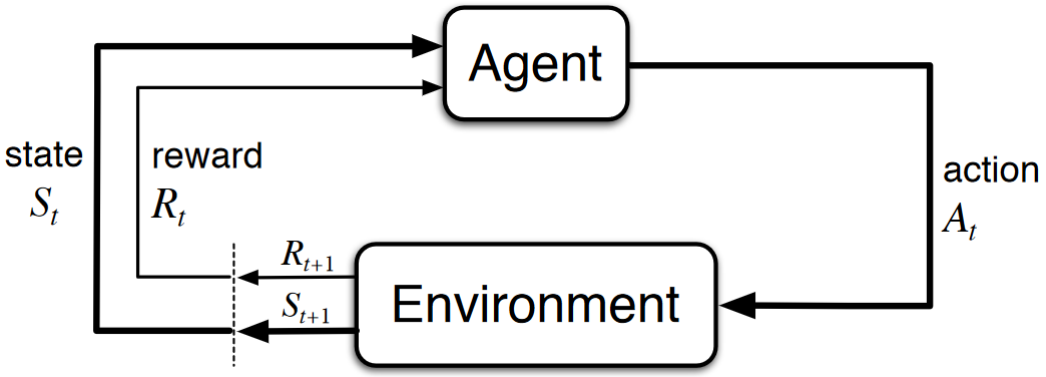

Figure 2.1: The agent-environment interaction in a Markov decision process ([SB18], Chapter 3.1). Based on the state S_t and reward R_t the agent selects an action A_t, which induces a state transition in the environment.

Definition 2.1.1 *A Markov Decision Process is a 4-tuple*

$$\langle S, A, P, R \rangle$$

where S is a set of states, A is a function $A: S \to 2^{Act}$ from states to sets of possible actions (where Act is the set of actions available to the agent), $R : S \times Act \to \mathbb{R}$ is a scalar reward function over states and actions and $P : S \times Act \times S \to [0, 1]$ is a function that gives the probability P(s, a, s') of transitioning from state s to state s' after performing action a.

A reinforcement learning algorithm seeks to find an optimal policy $\pi^* : S \to Act$ such that

$$v^{\pi^*}(s) = \max_{\pi \in \Pi} v^{\pi}(s)$$

where Π is the set of all policies over the MDP and

$$v^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+i} \mid s_i = s\right]$$

The function v^{π} takes as input a state s and outputs the expected discounted cumulative reward $r_t = R(s_t, \pi(s_t))$ obtained from following the policy π from s. The discount factor $\gamma \in (0, 1)$ ensures that the infinite sum converges and that immediate rewards are weighted higher than rewards in the far future.

We also define the action-value function q^{π}:

$$q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+i} \mid s_i = s, a_i = a\right]$$

$q^{\pi}(s, a)$ gives the expected discounted cumulative reward when following policy π from state s and taking action a.

Model-based RL algorithms need access to (or learn) a model of the environment, i.e. a function that predicts state transitions and rewards. For most applications of RL, a full description of the underlying MDP is not available, therefore a model has to be learned from exploration of the environment, which can be challenging. The benefits of this approach are that once a model is available, the optimal policy can be efficiently computed through dynamic programming methods like value iteration and policy iteration ([SB18], Chapter 4). A potential downside is that any bias in the learned model can result in an agent that performs well with respect to the learned model, but poorly in the real environment. This thesis will focus purely on model-free methods.

2.1.1 Value-based reinforcement learning

In value-based reinforcement learning, the algorithm finds the optimal policy π^* by learning an action-value function Q such that $\pi^*(s) \in \text{argmax}_{a \in A} Q(s, a)$ for all $s \in S$. This is most often accomplished using *temporal difference* (TD) learning techniques. Assuming a model-free setting, the algorithm starts off by picking random actions to explore the environment, building a sequence of states and actions called *trajectories* or *episodes*. A Monte Carlo method would wait until the end of the episode to update its policy. In contrast, TD methods "update estimates based in part on other learned estimates, without waiting for a final outcome (they *bootstrap*)" ([SB18], Chapter 6). This means that TD methods can update the value function whenever a state transition (s, a, s') occurs. For example, the TD algorithm Sarsa uses the following rule to update its Q-function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a) + \gamma Q(s', a') - Q(s, a)]$$

where α is the learning rate, γ is the discount factor, s' is the state resulting from s when choosing action a and a' is the action chosen in state s'.

An alternative to value-based RL are policy-based methods (also called policy gradient methods), where the policy depends on and is differentiable with respect to some parameters θ. The parameters can then be updated such that the policy is optimal according to some objective.

Tabular Q-learning

The Sarsa update rule is called *on-policy* because it estimates the future return assuming that the current policy continues to be followed. However, an early breakthrough in reinforcement learning involved the discovery of an *off-policy* TD-algorithm known as *Q-learning* [Wat89], given by algorithm 2.1. It updates its value estimations based on the rule

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a) + \gamma \max_{a' \in A(s)} Q(s', a') - Q(s, a)] \tag{2.1}$$

This rule estimates future returns based on the greedy policy despite the fact that the current policy might not be the greedy policy.

Algorithm 2.1: Tabular Q-learning

Input: learning rate $\alpha \in (0, 1]$, small $\epsilon > 0$, discount $\gamma \in (0, 1)$

foreach *episode* **do**
  Initialize s
  while s *is not terminal* **do**
    Choose a from s using policy derived from Q (e.g. ϵ-greedy)
    Take action a, observe $R(s, a)$ and s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a) + \gamma \max_{a' \in A(s)} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  end
end

The action-value function Q learned by algorithm 2.1 has been shown to converge to the optimal action-value function q^{π^*} in the limit under certain conditions. One of these conditions is that each state-action pair must be visited infinitely often. Therefore we cannot simply use a greedy policy that selects $\text{argmax}_{a \in A} Q(s, a)$ in all states s, otherwise we risk getting stuck in a suboptimal policy. A popular solution in value-based RL is the *epsilon-greedy* strategy, that in each state selects a random action with probability ϵ (*exploration*) and the greedy action with probabilty $1 - \epsilon$ (*exploitation*).

Deep Q-Networks

Algorithm 2.1 and similar methods are called *tabular*, because they store Q-values explicitly in a look-up table, where neighboring entries do not necessarily have to be correlated. This quickly becomes infeasible in use cases with complex environments that might be continuous or inhabited by multiple agents. For such applications, function approximation has become indispensible. Using function approximation, we no longer update only one Q-value at a time, but updating one Q-value also affects the values at similar states. This gain in efficiency makes RL in complex environments possible.

One possible approach is linear function approximation, where the Q-function is approximated as a weighted sum of environment *features* $f_i(s, a) : S \times Act \to \mathbb{R}$ that tell the agent something useful (e.g. how far away it is from the goal). However, this introduces the problem of *feature engineering*: Choosing or learning features that are computable from state descriptions and helpful for approximating Q-values.

Deep reinforcement learning is another approach that leverages the ability of *artificial neural networks* (ANNs) to approximate arbitrary functions. Policy gradient methods use ANNs to directly approximate the optimal policy, while value-based deep RL algorithms use them to approximate the Q-function. In the following we will not go into further detail regarding policy gradient methods. Instead, we will focus on *Deep Q-Networks* (DQN), a value-based deep RL algorithm. Deep Q-Networks are trained using algorithm 2.2. Its basic structure is identical to algorithm 2.1, but we have to make some adjustments to

Algorithm 2.2: Deep Q-Network

Input: learning rate $\alpha \in (0, 1]$, small $\epsilon > 0$, discount $\gamma \in (0, 1)$, target net update rate τ, buffer size N, batch size n, neural network hyperparameters

foreach *episode* **do**
  Initialize s, policy network, target network, replay buffer
  while s *is not terminal* **do**
    Choose a from s using policy derived from Q (e.g. ϵ-greedy)
    Take action a, observe $R(s, a)$ and s'
    $d \leftarrow 1$ if s' is terminal else 0
    Add $(s, a, R(s, a), s', d)$ to replay buffer
    Sample minibatch of transitions $(\vec{s}, \vec{a}, \vec{r}, \vec{s'}, \vec{d}) = ((s_1, ..., s_n)^T, (a_1, ..., a_n)^T, (R(s_1, a_1), ..., R(s_n, a_n))^T, (s'_1, ..., s'_n)^T, (d_1, ..., d_n)^T)$ from replay buffer
    Compute $Q(\vec{s}, \vec{a}) = (Q(s_1, a_1), ..., Q(s_n, a_n))^T$ from policy network
    Compute $\max_{\vec{a'} \in A(\vec{s'})} Q(\vec{s'}, \vec{a'})$ from target network
    $\vec{y} \leftarrow \vec{r} + \gamma \max_{\vec{a'} \in A(\vec{s'})} Q(\vec{s'}, \vec{a'}) \cdot (\vec{1} - \vec{d})$
    Compute $loss(Q(\vec{s}, \vec{a}), \vec{y})$
    Perform a gradient descent step on the policy network
    foreach *network parameter* p **do**
      $p_t \leftarrow p$ in target network
      $p_p \leftarrow p$ in policy network
      $p_t \leftarrow \tau \cdot p_p + (1 - \tau) \cdot p_t$
    end
    $s \leftarrow s'$
  end
end

accommodate the function approximation with ANNs.

The biggest advantage of function approximation is also one of its biggest drawbacks: When the value at one state is updated, there is a risk of inappropriately updating the values of other states, for instance the state whose value estimate is used in the target of the Q-learning update (rule 2.1). This has negligible effects when the values used for these bootstrapping updates are updated as often as they are used [SMW15]. However, if the learning algorithm is off-policy, this is not guaranteed, possibly leading to unwanted feedback loops, convergence to a poor local minimum or even divergence of the parameters [TV97]. For this reason, the combination of function approximation, bootstrapping and off-policy training has been dubbed "the deadly triad" ([SB18], Chapter 11.3). As the DQN algorithm includes all three components of the deadly triad, it was extended with techniques to minimize the danger of harmful learning dynamics arising, namely *experience replay* and a *target network* [Mni+15].

The experience replay (also called replay buffer) is a data structure $D_t = \{e_1, ..., e_t\}$ storing state transitions $e_t = (s_t, a_t, R(s_t, a_t), s_{t+1})$ for each time step t. In practice it is mainly implemented as a fixed-size queue, so once it is full, the oldest entry gets replaced by a new entry. In the inner loop of algorithm 2.2 a fixed number of experiences (a minibatch) is sampled from the replay buffer and used for Q-learning updates. This has several advantages:

- A state transition can be used in multiple updates, increasing data efficiency.
- Learning from state transitions in a randomized order breaks up the strong correlations between consecutive samples, reducing the variance of the updates.
- Local maxima can more easily be avoided because the policy updates are averaged over many previous states and the current policy does not influence the future training samples as much.

In addition to the policy network that is used for approximating the Q-function and selecting actions, we also use a separate target network for generating the targets $\vec{y}$ of the Q-learning update. This target network is an "outdated" copy of the policy network designed to add stability to the update targets. Without a target network, an update to $Q(s_t, a_t)$ would immediately also change $Q(s_{t+1}, a)$ for all a and therefore the target which was just used for the update, potentially leading to oscillations and divergence of the policy. To update the target network it can be set equal to the policy network after a certain amount of time steps, or target network weights can be slowly moved in the direction of the policy network weights after every policy update (like in algorithm 2.2).

2.1.2 Multi-Objective Reinforcement Learning

In *Multi-Objective Reinforcement Learning* (MORL) the goal is to compute an optimal policy with respect to multiple, possibly competing objectives. This setting is formalised by a multi-objective MDP (MOMDP):

Definition 2.1.2 *A multi-objective Markov Decision Process is a 4-tuple* $\langle S, A, P, \vec{R} \rangle$ *where* S, A *and* P *are defined as in regular MDPs and* $\vec{R} = (R_1, ..., R_m)^T, R_i : S \times Act \to \mathbb{R}, 1 \leq i \leq n$ *is a vector of* m *reward functions.*

Each of the m rewards returned by $\vec{R}$ in a MOMDP corresponds to one objective. Analogously to regular MDPs we can therefore define a vector of action-value functions $\vec{q^\pi} = (q_1^\pi, ..., q_m^\pi)^T$ in a MOMDP, where each $q_i^\pi(s, a), 1 \leq i \leq m$ gives the expected discounted cumulative reward from R_i given that the agent takes action a in state s and then continues with policy π. The update rules

$$Q_i(s, a) \leftarrow Q_i(s, a) + \alpha_i[R_i(s, a) + \gamma_i \max_{a' \in A(s)} Q_i(s', a') - Q_i(s, a)]$$

can be used to learn the optimal action-value function for each objective.

2.2 Normative Reasoning

Norms are rules describing how agents (human or artificial) should act. Reasoning over norms sets itself apart from classical logical reasoning by not solely assigning truth values to atomic propositions, but by also applying *deontic modalities* like obligation, permission, and prohibition to them. This introduces complex dynamics that classical logic cannot adequately encompass, like the inherent *violability* of norms. In normative reasoning, norms are often divided into *regulative* and *constitutive* norms [BV04].

2.2.1 Regulative norms

Regulative norms are used to describe what ideally should be the case. Formally,

Definition 2.2.1 *A regulative norm is of the form* $*(A|B)$ *where* $* \in \{\mathbf{O}, \mathbf{F}, \mathbf{P}\}$ *is a deontic modality.*

Regarding the deontic modalities, **O** refers to obligation, **F** refers to prohibition and **P** refers to strong permission. The expression $\mathbf{O}(A|B)$ can be read as "A is obligatory given that B holds true". Prohibition can be derived as an obligation of a negative statement, i.e. $\mathbf{F}(A|B) := \mathbf{O}(\neg A|B)$. Strong permission acts as an exception to an obligation or prohibition of the opposite, whereas weak permission just states the absence of a prohibition.

For our purposes we will only deal with *maintenance obligations* [Gov+07], where the target is obliged whenever the trigger holds. We leave out other notions of obligation that require the ability to reason with time, which is not available in our formalism.

2.2.2 Constitutive norms

In normative reasoning, constitutive norms elucidate what is actually the case. To be exact, they associate a more concrete proposition to a more abstract one. This is generally realized as "counts-as" rules:

Definition 2.2.2 *A constitutive norm is of the form* $\mathbf{C}(A, B|C)$*, with the intended meaning of "in context C, A counts as B".*

For example, a constitutive norm could be used in a normative system that forbids speeding to define speeding as "driving more than 10 km/h over the speed limit".

2.2.3 Normative systems

Normative systems are the formalism we will use to specify the desired behavior for a RL agent:

Definition 2.2.3 *A normative system is a triple $\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, > \rangle$ where $\mathcal{C}$ is a set of constitutive norms, $\mathcal{R}$ is a set of regulative norms and $>$ is a priority relation for resolving conflicting norms.*

Norms can conflict directly or indirectly. An example pair of directly conflicting norms would be $\mathbf{O}(A|B)$ and $\mathbf{F}(A|B) = \mathbf{O}(\neg A|B)$. Indirectly conflicting norms take the form $\mathbf{O}(A|B)$ and $\mathbf{O}(C|D)$, where B and D can both simultaneously hold in the environment we are reasoning about, but A and C cannot.

From a state-action pair $\bar{s} = (s, a)$ in a RL episode and a normative system $\mathcal{N}$ a logical theory $Th(\bar{s}, \mathcal{N})$ can be constructed for some logic of normative reasoning, e.g. defeasible deontic logic. This theory contains facts derivable from the state s, the fact that a is the action chosen in s and the norms from $\mathcal{N}$ to facilitate drawing conclusions. We can now define the violation of a normative system:

Definition 2.2.4 *Given a state-action pair $\bar{s} = (s, a)$ and a normative system $\mathcal{N}$, a violation of $\mathcal{N}$ is a formula φ such that $Th(\bar{s}, \mathcal{N}) \vdash_{\mathcal{L}} \mathbf{O}(\varphi) \wedge \neg\varphi$, where $Th(\bar{s}, \mathcal{N})$ is a logical theory for some logic of normative reasoning $\mathcal{L}$.*

We say $\bar{s}$ is a violating state-action pair for $\mathcal{N}$ if $|viol(\bar{s}, \mathcal{N})| > 0$ where $viol(\bar{s}, \mathcal{N})$ is the set of violations of $\mathcal{N}$ for $\bar{s}$.

2.2.4 Contrary-to-duty reasoning

Contrary-to-duty (CTD) obligations come into force when another obligation has been violated. A well-known example is Forrester's (or the Gentle Murder) paradox, which says [For84]:

1. You should not murder.
2. If you murder, you should murder gently.
3. You murder.

Here, the contrary-to-duty (obligation 2) is triggered by the violation of the primary obligation 1. We distinguish between *compensatory* CTDs and *sub-ideal* CTDs. If we interpret the above CTD scenario as compensatory, we are allowed to murder as much as we like, as long as we do it gently, because the compliance with the CTD obligation completely compensates for the violation of the primary obligation. However, in a sub-ideal CTD scenario murdering gently would still be condemned, but even more so if the CTD obligation was also violated.

Dealing with CTDs is a particularly characteristic challenge of normative reasoning. It is important that autonomous agents treat CTDs as sub-ideal, because otherwise the agent might violate norms more than necessary to maximise its other goals.

Norm	DDL rule
$\mathbf{C}(x, y\|z)$	$z, x \rightarrow_C y$
$\mathbf{F}(x\|y)$	$y \Rightarrow_O \neg x$
$\mathbf{P}(x\|z)$	$z \rightsquigarrow_O x$

Table 2.1: Three example norms and their equivalent rules in DDL. The defeater acts as a strong permission in context z, since it provides evidence for x, thereby preventing the conclusion of $\neg x$ from a defeasible rule and disabling any defeasible prohibitions of x.

2.3 Defeasible Deontic Logic

Deontic Logic is a branch of logic concerned with formalising normative reasoning and related notions. In this work, we will make use of a special kind of deontic logic, defeasible deontic logic (DDL) [Gov18]. It is defined formally in the following.

Definition 2.3.1 *Let AP be a set of propositional atoms. The base language of defeasible deontic logic (DDL) is a set of literals $\mathcal{L} = Lit \cup ModLit$ partitioned into plain literals $Lit = AP \cup \{\neg p \,|\, p \in AP\}$ and deontic literals $ModLit$, obtained by placing a plain literal in the scope of a deontic operator or a negated deontic operator from the set of deontic operators $Mod = \{O\}$.*

To model statements of normative reasoning, DDL uses the concept of rules:

Definition 2.3.2 *DDL rules take the form:*

$$r : A(r) \hookrightarrow_* C(r)$$

where r is the rule label, $A(r) = \{a_1, \ldots, a_n\} \subseteq 2^{\mathcal{L}}$ is the antecedent, $C(r)$ is the consequent, $* \in Mod$ for regulative rules, $* = C$ for constitutive rules and $\hookrightarrow_* \in \{\rightarrow_*, \Rightarrow_*, \rightsquigarrow_*\}$ (strict rules, defeasible rules and defeaters). For constitutive rules, the consequent $C(r)$ consists of plain literals $l \in Lit$. For regulative rules, $C(r) \subseteq ModLit$.

For strict rules, the consequent is invariably directly derived from the antecedent. With defeasible rules on the other hand, the consequent usually follows from the antecedent, unless there is evidence suggesting otherwise. This evidence may take the form of conflicting rules or defeaters, which do not serve to derive conclusions but rather act as barriers preventing a defeasible rule from arriving at a conclusion. Some example formalisations of norms in DDL are shown in table 2.1.

Analogously to the logical theories of normative reasoning $Th(\bar{s}, \mathcal{N})$ from last section we define defeasible theories for DDL:

Definition 2.3.3 *A defeasible theory is a tuple $\langle F, \mathcal{N}, > \rangle$, where F is a set of facts represented as plain literals, $\mathcal{N} = \langle R^O, R^C \rangle$ is a normative system expressed in DDL,*

R^O is a set of regulative rules, R^C is a set of constitutive rules and $>$ is a superiority relation over conflicting rules.

Conclusions from defeasible theories are given by the literals contained in them annotated with proof tags:

- A definitely provable literal is tagged with $+\Delta_*$ and either a fact, or derived only from strict rules and facts.
- A definitely refutable literal is tagged with $-\Delta_*$ and neither a fact nor derived from only strict rules and facts.
- A defeasibly provable literal is tagged with $+\partial_*$, not refuted by any facts or conflicting rules and is implied by some undefeated rule.
- A defeasibly refutable literal tagged with $-\partial_*$ is one where its complemented literal is defeasibly provable or an exhaustive search for a constructive proof for the literal fails.

For factual conclusions, $* := C$ and $* := O$ for deontic conclusions. In practise, we will compute a set of conclusions from a defeasible theory using a theorem prover for DDL, SPINdle [LG09].

CHAPTER 3

State of the art

This chapter gives a brief overview regarding related work in constrained reinforcement learning. One the one hand, this will help in gaining a picture of the current state of the art in this field. On the other hand the main contributions of this thesis heavily build on past work introduced in this chapter, thus its contents are vital for understanding the following chapters.

3.1 Safe Reinforcement Learning

The field of Safe RL is concerned with learning policies that obey safety constraints stating that certain undesirable states never occur. This problem can be seen as a special case of learning norm-compliant policies, as it is certainly possible to specify safety properties in a normative system, but it is unclear if the opposite holds true. It has been argued (e.g. in [WR19]) that developing norm-compliant agents is unnecessary and safe agents are sufficient. However, there are clear advantages to be gained from being able to develop autonomous agents that obey general normative systems. On the one hand, exhaustively specifying the relevant safety constraints in a complex environment will be a cumbersome and error-prone, if not outright impossible task. Having access to the expressive capabilities of normative systems including constitutive norms will reduce the risk of a crucial constraint missing from the safety specification. On the other hand, in complex environments it will be inevitable that the agent finds itself in an unforeseen state where no normative compliance is possible (normative deadlock). In the following we briefly introduce current safe RL frameworks based on Linear Temporal Logic (LTL) specifications and show that they lack the ability to enforce sub-ideal CTD norms in such situations.

[HAK20] converts a LTL formula expressing the safety properties into an automaton. The automaton is then used to modify the reward function of the MDP in which the RL agent is trained, such that the obtained policy maximizes the probability of obeying the

safety properties. *Shielding* [Als+18] [Jan+19] is another approach, which synthesizes a shield using a value function that gives the probability of the agent violating the safety specification for each state. This shield is able to provide safe actions to the agent or prevent it from taking unsafe actions.

These techniques of safe reinforcement learning are well established and efficient, but limited by what can be expressed in LTL. In [NBC22], the authors argue that LTL is not suited to express all the intricacies of normative reasoning. They prove that the construction of a LTL operator that directly represents the proposition $\mathbf{O}(A|B)$ is impossible. Furthermore, they demonstrate the limits of compliance specifications for normative systems, which are LTL formulas that evaluate to True on a sequence of RL states if and only if no norms in the normative systems are violated. These compliance specifications suffer from inaccuracies introduced by the necessity of identifying actions in LTL with the state transitions they induce. The LTL compliance specification cannot distinguish between two actions that share a state transition, even if one is prohibited and the other is permitted. Moreover, compliance specifications can only produce policies that fully comply with a normative system or interpret CTDs as compensatory. As explained in section 2.2.4, interpreting CTDs as compensatory is troublesome, as the agent might exploit this and violate more norms than necessary. Full compliance with the normative system can also be undesirable since it might render the agent unable to act when confronted with normative deadlock. We might also want the agent to intentionally violate norms in order to avoid further violations in the future.

3.2 The Normative Supervisor

The normative supervisor, as introduced in [Neu+21], is an external reasoning module designed to interact with a reinforcement learning agent. It primarily consists of a normative system, front- and back-end translators and a theorem prover for a logic of normative reasoning. For our purposes, the theorem prover and its associated logic are SPINdle and DDL, respectively. The front-end translator constantly processes new state-action pairs from the RL episode, converting them into a set of facts about the agent or the environment. These facts are combined with the output of the back-end translator - a set of DDL rules representing the normative system - and a set of non-concurrence rules into a defeasible theory. The non-concurrence rules are of the form

$$\bigcup_{a \in A(s)} \{\mathbf{C}(a, \neg a' \mid \top) \mid a' \in A(s) \setminus \{a\}\}$$

and express that only one action can be taken per time step. The conclusions drawn from a defeasible theory that was generated like this can be used in multiple ways.

The original application of the normative supervisor in [Neu+21] was online compliance checking (OCC) (see figure 3.1). Here the defeasible theory conclusions are converted into a set of actions that violate the fewest number of norms possible and can be returned to the agent, effectively filtering out non-compliant actions from the agent's repertoire. Another use case of the normative supervisor will be detailed in the following section.

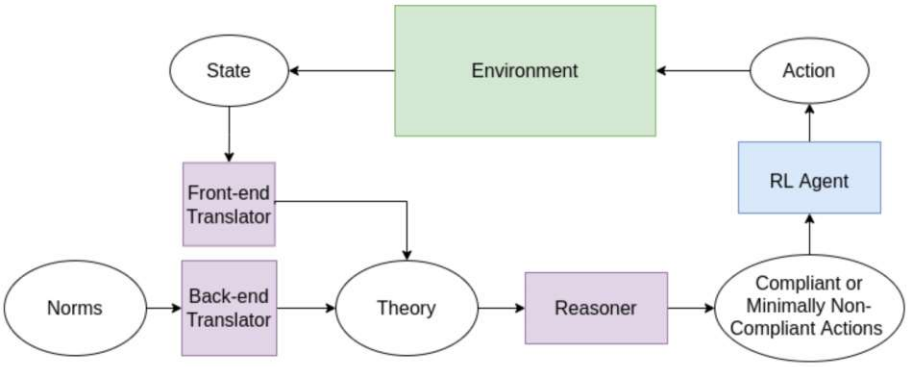

Figure 3.1: The process of online compliance checking [Neu23].

3.3 Norm-Guided Reinforcement Learning

Previous approaches to the problem of designing an ethical RL agent (e.g. [RLR21]) focused on rewarding or punishing the agent based on its actions, although they often lacked clear reasoning on when to give rewards and the magnitude of those rewards. Emery A. Neufeld addressed these challenges with the novel technique of Norm-Guided Reinforcement Learning (NGRL) [Neu22], using the normative supervisor and MORL to learn norm-compliant policies. In addition to the reward function $R_x(s,a)$ corresponding to an objective x of the agent, NGRL defines a second reward function called *non-compliance function* that punishes agents for violating a normative system $\mathcal{N}$:

Definition 3.3.1 *A non-compliance function for the normative system $\mathcal{N}$ is a function of the form:*

$$R_{\mathcal{N},p}(s,a) = \begin{cases} p & \text{if } Th(s,\mathcal{N}) \vdash +\partial_O \neg a \\ 0 & \text{otherwise} \end{cases}$$

where the penalty $p \in \mathbb{R}^-$ and $Th(s,\mathcal{N})$ is a defeasible theory built from $\mathcal{N}$ and a state s from a MDP.

In other words, $R_{\mathcal{N},p}(s,a)$ returns a fixed punishment if and only if action a violates $\mathcal{N}$ in state s.

NGRL seeks to learn an optimal-ethical policy over the MOMDP $\mathcal{M}_{\mathcal{N},p} = \langle S, A, (R_x, R_{\mathcal{N},p})^T, P \rangle$, where the term *optimal-ethical* is adapted from [RLR21] as:

Definition 3.3.2 *Let Π be the set of all policies over $\mathcal{M}_{\mathcal{N},p}$. Then a policy $\pi^* \in \Pi$ is ethical for $\mathcal{M}_{\mathcal{N},p}$ if and only if*

$$v^{\pi^*}_{\mathcal{N},p}(s) = \max_{\pi \in \Pi} v^{\pi}_{\mathcal{N},p}(s)$$

for all states s. Furthermore, let $\Pi_{\mathcal{N}}$ be the set of all ethical policies over $\mathcal{M}_{\mathcal{N},p}$. Then $\pi_* \in \Pi_{\mathcal{N}}$ is ethical-optimal for $\mathcal{M}_{\mathcal{N},p}$ if and only if

$$v_x^{\pi^*}(s) = \max_{\pi \in \Pi_{\mathcal{N}}} v_x^{\pi}(s)$$

for all states s.

Neufeld goes on to prove that NGRL learns ethical policies over $\mathcal{M}_{\mathcal{N},q}$ for any $q \in \mathbb{R}^-$, thus rendering the exact magnitude of the penalty in the non-compliance function irrelevant. In his experiments regarding the effectiveness of NGRL Neufeld employs two different MORL techniques for learning optimal solutions over MOMDPs that will be reiterated in the following.

Linear Scalarization

With this approach, the goal is to maximize the inner product $v_{\text{scalar}}(s) = \vec{w} \cdot \vec{v}(s)$ where $\vec{v}(s) = (v_x(s), v_{\mathcal{N},p}(s))^T$ is a vector of value functions associated with the reward functions R_x and $R_{\mathcal{N},p}$ and $\vec{w} \in \mathbb{R}_2^+$ is a weight vector. This is done by learning $\vec{Q}(s,a) = (Q_x(s,a), Q_{\mathcal{N},p}(s,a))^T$ and selecting actions using $Q_{\text{scalar}}(s,a) = \vec{w} \cdot \vec{Q}(s,a)$.

Thresholded Lexicographic Q-Learning

Thresholded Lexicographic Q-Learning (TLQL) is a technique particularly suitable for MORL problems where a single objective must be maximized whereas the other objectives only need to meet a certain threshold. Over $\mathcal{M}_{\mathcal{N},p}$ we want to maximize the reward from R_x but prioritize the reward from $R_{\mathcal{N},p}$. TLQ-Learning considers CQ-values instead of Q-values for action selection defined as:

$$CQ_i(s,a) = \min(Q_i(s,a), C_i)$$

However, $Q_{\mathcal{N},p}(s,a) \leq 0$ since $R_{\mathcal{N},p}(s,a) \leq 0$ for all state-action pairs. Therefore if we set the thresholds $C_x = +\infty$ and $C_{\mathcal{N},p} = 0$ then $CQ_i(s,a) = Q_i(s,a)$. If we then select actions lexicographically by only considering actions maximal for $Q_{\mathcal{N},p}$ and of those actions taking one with maximal Q_x value we can learn an ethical-optimal policy according to definition 3.3.2.

Neufeld tested NGRL in the environments described in section 4.2 using both tabular Q-learning and Q-learning with linear function approximation. He noted that the TLQL approach was incompatible with function approximation: Since the ethical objective is strictly prioritized the agent stayed still in its initial position as taking any other action carried a risk of violating the normative system.

3.3.1 NGRL with Violation Counting

NGRL with the non-compliance function from definition 3.3.1 fails to learn policies that respect sub-ideal CTD obligations. The reason is simple: The agent is assigned a fixed

penalty when it violates an obligation, no matter how many obligations are violated and how important the obligation is. As a remedy, Neufeld introduced NGRL with violation counting in his dissertation ([Neu23], Chapter 6.3). This variant of NGRL extends our MOMDP with an objective to minimize a violation counting function defined as:

Definition 3.3.3 *A violation counting function for a normative system $\mathcal{N}$ is a function of the form*

$$VC_{\mathcal{N}}(s,a) = |viol(\bar{s},\mathcal{N})|$$

Given a state-action pair $\bar{s} = (s,a)$ this function returns the number of violations of $\mathcal{N}$ associated with $\bar{s}$ according to definition 2.2.4.

NGRL with violation counting now selects actions based on the following extended TLQ procedure:

1. From the set of available actions in state s, select the ones with the highest $CQ_{\mathcal{N},p}$-values and add them to a set $norm(s)$.
2. From the actions in $norm(s)$ select the ones with the lowest $VC_{\mathcal{N}}$-values for state s and add them to a set $viol(s)$.
3. From the actions in $viol(s)$ select one with the highest CQ_x-value.

The objective associated with $R_{\mathcal{N},p}$ is prioritized over minimization of the violation counting function to preserve the ability of the agent to violate more norms than currently necessary in order to minimize violations in the long term.

The above procedure is able to produce policies that account for sub-ideal CTD obligations. However, as it relies on the TLQL technique, it too is incompatible with function approximation.

3.4 Lexicographic Multi-Objective Reinforcement Learning

In [Ska+22], the authors introduce algorithm schemes for solving lexicographic multi-objective reinforcement learning (LMORL) problems, where the m reward signals of a MOMDP are ordered with respect to some priority. The goal is to learn a policy that maximizes the later reward signals subject to the constraint that the earlier reward signals have already been maximized. Both value- and policy-based algorithms schemes are presented and proved to converge to lexicographically optimal policies in the limit.

For the value-based lexicographic algorithms the authors devise a lexicographic version of the ϵ-greedy algorithm for action selection (see algorithm 3.1). This algorithm takes a state s together with a sequence $Q_1, ..., Q_m$ of m state-action functions. With ϵ decaying

Algorithm 3.1: Lexicographic ϵ-greedy

Input: $Q_1, ..., Q_m$, state s, small $\epsilon > 0$, tolerance τ

return an action available in s uniformly at random with probability ϵ
else
  $\Delta \leftarrow A(s)$
  foreach $i \in \{1, ..., m\}$ **do**
    $x \leftarrow \max_{a' \in \Delta} Q_i(s, a')$
    $\Delta \leftarrow \{a \in \Delta \mid Q_i(s, a) \geq x - \tau\}$
  end
  return an action from Δ uniformly at random
end

over time the algorithm will in the limit select an action a such that a maximises Q_1 (with tolerance τ), and among all such actions, a also maximises Q_2 (with tolerance τ), and so on. The tolerance parameter $\tau \in \mathbb{R}_{>0}$ enables the agent to choose an action with a suboptimal Q_i-value as long as this value is close to the current maximum value. This grants the agent higher flexibility and ensures that later objectives are not completely disregarded in favor of prior objectives. τ can be a constant or a proportion, in which case τ is of the form $\sigma \cdot \max_{a' \in \Delta} Q_i(s, a')$ with $\sigma \in (0, 1)$.

Algorithm 3.2: Value-Based Lexicographic RL

Input: learning rate $\alpha \in (0, 1]$, small $\epsilon > 0$, discount $\gamma \in (0, 1)$

1 **foreach** *episode* **do**
2 Initialize s
3 **while** s *is not terminal* **do**
4 Choose a from s using lexicographic ϵ-greedy
5 Take action a, observe $R_i(s, a)$ and s'
6 **foreach** $i \in \{1, ..., m\}$ **do**
7 update Q_i
8 **end**
9 $s \leftarrow s'$
10 **end**
11 **end**

Algorithm 3.2 is the main algorithm for learning $Q_1, ..., Q_m$. The update rule on line 7 can be varied, but we will use the *lexicographic Q-Learning* update rule:

$$Q_i(s, a) \leftarrow (1 - \alpha) \cdot Q_i(s, a) + \alpha \cdot (R_i(s, a) + \gamma \max_{a' \in \Delta^\tau_{s,i}} Q_i(s', a'))$$

where $\Delta^\tau_{s,0} = A(s)$ and $\Delta^\tau_{s,i+1} := \{a \in \Delta^\tau_{s,i} \mid Q_i(s, a) \geq \max_{a' \in \Delta^\tau_{s,i}} Q_i(s, a') - \tau\}$.

The main difference to the regular Q-learning update (rule 2.1) is that the max-operator ranges only over actions that approximately lexicographically maximise all higher priority rewards. The Q-functions in this update rule can be represented by a table or approximated.

Skalse et al. also implement and experimentally evaluate some instantiations of their lexicographic algorithm schemes, demonstrating that the paradigm can be used to learn optimal policies under safety constraints.

CHAPTER 4

Methodology

In this chapter we suggest a new method of training norm-compliant agents that avoids the drawbacks of the approaches discussed before. Moreover, the setting for our experimental evaluation is detailed, the results of which are presented in the next chapter.

4.1 Norm-Guided Lexicographic Multi-Objective Reinforcement Learning

The previous chapter elucidated how regular NGRL falls short regarding sub-ideal CTD reasoning and that NGRL with violation counting is incompatible with function approximation. In this section we attempt to remedy these shortcomings by combining NGRL with LMORL, creating an algorithm able to use function approximation and learn norm-compliant policies that can deal with sub-ideal CTDs. To this end, we define a normative reward function of the form:

$$R_{\mathcal{N}}(s,a) = -|viol(\bar{s},\mathcal{N})|$$

where $\bar{s} = (s,a)$ and $\mathcal{N}$ is a normative system. Norm-Guided Lexicographic Multi-Objective RL (NGLMORL) now uses an algorithm for LMORL to learn a lexicographically optimal policy over the MOMDP $\mathcal{M}_{\mathcal{N}} = \langle S, A, (R_{\mathcal{N}}, R_x)^T, P\rangle$. Such a policy will seek to minimize the expected discounted cumulative penalties from $R_{\mathcal{N}}$ first, choosing actions that lead to the fewest number of violations in the long run. As a secondary priority, the algorithm maximizes the rewards from R_x. Using the tolerance parameter τ we can control how much leeway we want to give to the agent in action selection, letting it prioritize R_x in exceptional cases. For the experiments the reward $R_{\mathcal{N}}$ will be provided by the normative supervisor.

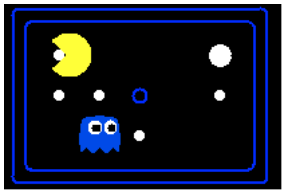

Figure 4.1: The miniature Pac-Man environment.

4.2 Case Studies

4.2.1 Environments

In accordance with previous studies on ethical/normative RL (e.g. [Noo+19], [Neu22]) we evaluate NGLMORL by employing it for the Atari video game *Pac-Man*. The environment of the Pac-Man agent is a maze-like gridworld, populated by "food pellets", "power pellets" and "ghosts". The goal of the agent is to win the game by eating all of the food pellets. Pac-Man is rewarded for increasing its game score, which can be done by eating a food pellet (10 points), winning (500 points) or eating a ghost (200 points). Ghosts can only be eaten when they are in a special "scared" state. Pac-Man can make all ghosts scared for 40 time steps when he eats a power pellet. If Pac-Man collides with a ghost that is not scared he loses the game and receives a penalty of -500 points. Additionally, there is a time penalty of -1 at every time step to encourage Pac-Man to finish the game quickly. At each time step, Pac-Man can choose to move north, south, east or west or to not move at all. Ghosts are unable to stop and continue in the same direction until they reach a dead end or an intersection. At dead ends they turn around and at intersections they can randomly choose to turn 90 degrees.

We use several different maze layouts according to the objective of the experiment. For most investigations, the *miniature Pac-Man* environment from [Neu22] will be used (see figure 4.1). This is a simplified version of the Pac-Man game on a 7×5 grid that significantly speeds up the training time. A single blue ghost and a power pellet are located in opposite corners. A total of 11 food pellets are contained in the maze, therefore the maximum attainable score without eating the ghost is 599.

The *regular Pac-Man* environment from [Noo+19], [Neu+21] and [NBC22] is a 20×11 grid containing 97 food pellets, a blue ghost, an orange ghost and 2 power pellets. The maximum attainable score in this environment is 2170 or 1370 when Pac-Man eats no ghosts.

We additionally use a 7×3 Pac-Man environment containing two ghosts, two power pellets and no food pellets to test the behaviour of the agent when it encounters normative deadlock. Winning the game is not possible in this environment. Instead, we monitor whether the agent chooses to eat any ghosts and if so, which of the two. We will refer to this environment as the *unfair Pac-Man* environment.

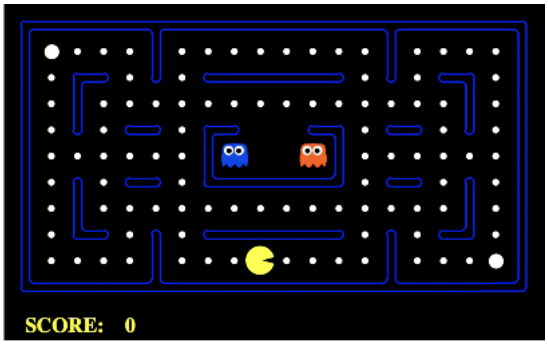

Figure 4.2: The regular Pac-Man environment.

Figure 4.3: A Pac-Man environment with normative deadlock.

4.2.2 Normative systems

In order to evaluate the normative compliance of NGLMORL agents we need suitable normative systems to which the agents should adhere. We employ here two normative systems from[Neu23] that regulate the ability of Pac-Man to eat ghosts. In the following the emphasis will be on the regulative norms from these systems, but they also contain constitutive norms to define important concepts for the regulative norms, such as being *next to* another environment object or eating a ghost.

Benevolent Pac-Man

This normative system contains a single regulative norm $\mathbf{O}(benevolent \mid \top)$ obliging Pac-Man to be benevolent. A defeasible rule ensures that any action counts as benevolent unless there is evidence to the contrary. Through a constitutive norm hierarchy of the form $\mathbf{C}(eat_{person}, \neg benevolent \mid \top), \mathbf{C}(eat_{ghost}, eat_{person} \mid \top)$ we define non-benevolence as “eating a person“ and “eating a person“ as eating a ghost.

Unfair benevolent Pac-Man

This is the normative system that will be used in tandem with the unfair Pac-Man environment. It is identical to the “benevolent Pac-Man“ normative system except that we add an additional CTD obligation $\mathbf{O}(eat_{orangeGhost} \mid \neg benevolent)$ specifying that eating an orange ghost is preferable to eating a blue ghost in case Pac-Man is not benevolent.

4.3 Experiments and Evaluation Criteria

This section documents the experiments used to evaluate NGLMORL agents. We will assess the agents based on the following criteria:

1. Normative compliance: To which degree is the agent able to comply with a normative system?
2. Game performance: Is the agent able to achieve above-average game scores and how often will it win the game?
3. Computational efficiency: How long does it take to train the agent?

For the miniature Pac-Man environment a tabular Q-learning NGLMORL agent (LTQ), a deep Q-network NGLMORL agent (LDQN), a regular tabular Q-learning agent (TQ) and a regular DQN were trained. The lexicographic agents were trained to obey the benevolent Pac-Man normative system, while the non-lexicographic agents were trained purely for winning the game, providing a suitable baseline for the evaluation. Implementation details such as hyperparameters and neural network architecture are documented in appendix A. Each of these agents were trained for 19000 episodes and tested over 1000. Furthermore, five copies of each agent were trained to obtain more robust and accurate results.

To judge the scaling of NGLMORL agents in larger environments, a LDQN agent was trained for 9000 episodes and evaluated over 1000 in the regular Pac-Man environment with the benevolent normative system. As a point of comparison a DQN agent was trained in the same manner for this environment. We refrain from training tabular agents in this environment as they are not suitable for environments of this size.

Finally, we trained a LTQ agent for 9000 episodes in the unfair Pac-Man environment with the unfair benevolent Pac-Man normative system to test the behaviour of the agent in a normative deadlock CTD scenario.

Tests were executed with a Intel i7-7700K CPU (4 cores, 4.20 GHz), a NVIDIA GeForce GTX 1080 GPU and 32 GB RAM on a Windows 10 PC. The code for reproducing the experiments is located at `https://github.com/bschiehl/NGLMORL`.

CHAPTER 5

Experimental Results

In this chapter we will present and discuss the results of the experimental evaluation of NGLMORL agents, according to the evaluation criteria laid down in section 4.3.

5.1 Miniature Pac-Man Experiments

In the following the results of the experiments in the miniature Pac-Man environment will be presented separately for each agent. See table 5.1 for a summary of the results.

Agent	Avg win rate (%)	Avg score	Avg ghosts eaten	Avg evaluation time (s)
TQ	81.84	525.70	586.2	214.05
DQN	94.50	734.62	993.2	2391.60
LTQ	95.46	556.00	37.4	971.02
LDQN	94.98	545.49	1.2	28071.00

Table 5.1: Summary of the experimental results in the miniature Pac-Man environment.

5.1.1 Tabular Q-Learning

On average, the tabular Q-learning agent won 81.84% of its games in the miniature Pac-Man environment. As can be seen in table 5.2, the minimum win rate from all 5 tabular Q-learning agents is 68%, the maximum win rate is 90% and the standard deviation is 7.75%. The average game score achieved by the tabular Q-learning agent is 525.70, the minimum is 436.29, the maximum is 593.82 and the standard deviation is 54.43. However, the agent with the highest win rate did not have the highest game score. This is because the highest win rate agent (agent 4) only ate 380 ghosts, while the agent with the highest game score (agent 1) had a comparable win rate (85%) but learned to eat a lot more ghosts (771).

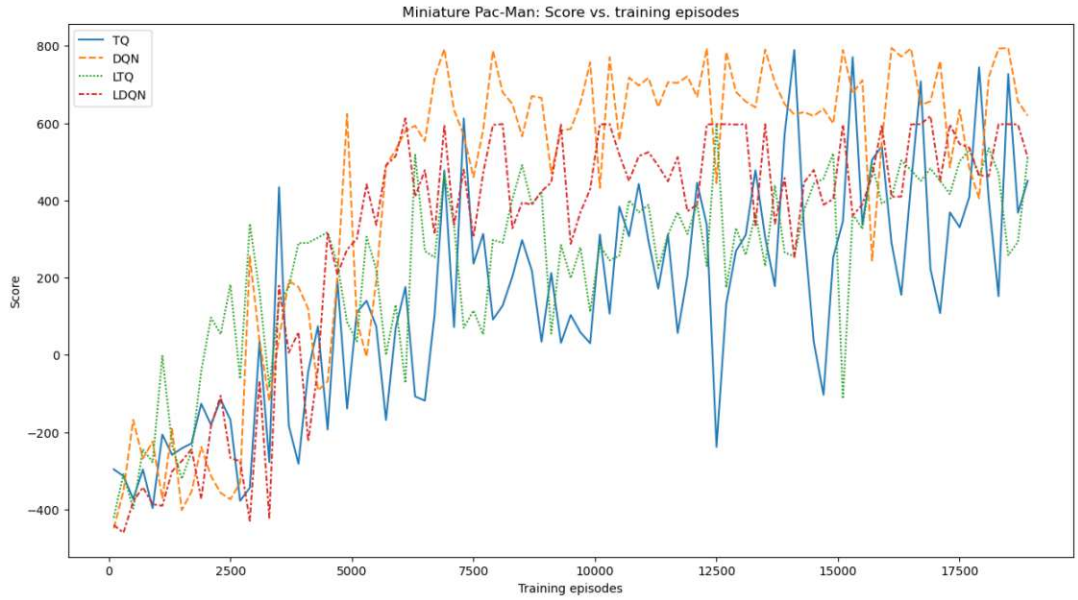

Figure 5.1: Average game score of highest scoring agents in steps of 200 episodes during training in the miniature Pac-Man environment versus the number of training episodes.

Remember that the goal for regular RL agents in the Pac-Man environment is to maximize the game score, therefore the optimal policy should be able to both win most games and also eat a lot of ghosts. Despite the small environment, this seems to be a challenging task for the tabular agent, as evidenced by the fact that agent 3 had the lowest win rate and game score but also ate the most ghosts (846). The average amount of ghosts eaten by all tabular Q-learning agents is 586.2, the minimum is 290 and the standard deviation is 216.90, so this varied quite a lot between the agents. Some agents focused on ghosts to increase their score, others focused on winning the game but few managed to achieve both consistently.

Figure 5.1 shows that even agent 1 is very unstable during training as the average game score quickly oscillates by hundreds of points multiple times.

In the experiments from [Neu22], the tabular Q-learning agent was trained for 9000 games and achieved a win rate of 68.5%, an average game score of 441.71 and ate 851 ghosts. These numbers are very comparable to agent 3 but can be improved by additional

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg game score	593.82	519.90	436.29	568.98	509.28
Win rate (%)	84.9	87.0	68.0	90.0	79.3
Ghosts eaten	771	290	846	380	644

Table 5.2: Average game scores, win rates and number of ghosts eaten of the five tabular Q-learning agents in the miniature Pac-Man environment.

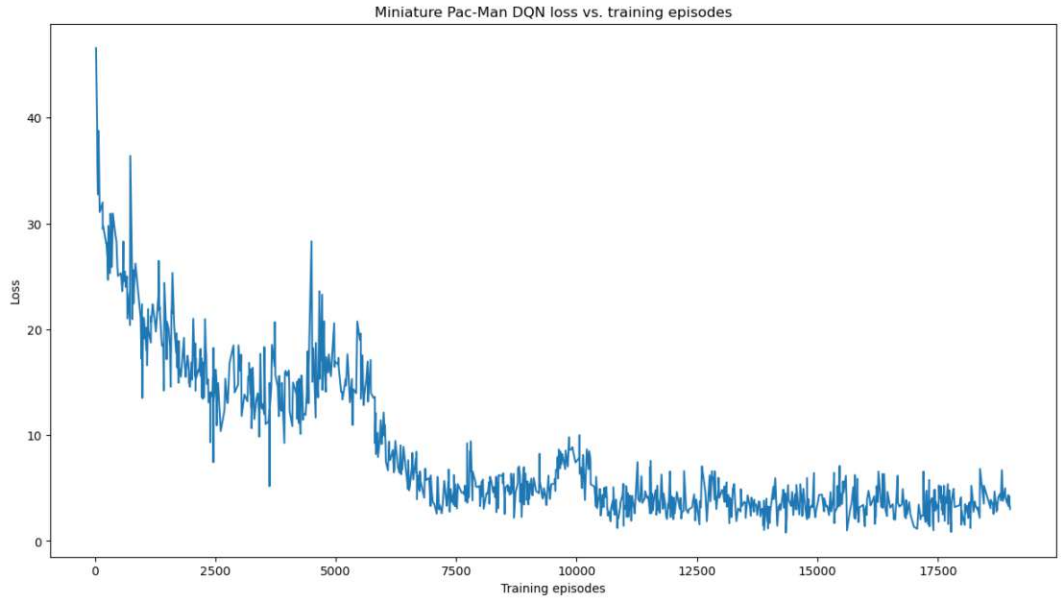

Figure 5.2: Value of the highest scoring DQN agent loss function versus number of training episodes.

training as is apparent from the other agents. Nevertheless, we note that none of the tabular agents was able to learn a policy that is close to optimality.

5.1.2 Deep Q-Network

The results for the DQN agent in the miniature Pac-Man environment are shown in table 5.3. Its minimum and maximum average scores are 684.29 and 787.05, respectively. On average, the achieved game score is 734.62, with a standard deviation of 38.59. The DQN agent won 94.5% of its test games on average, with a standard deviation of 3.5%. DQN agent 4 had the lowest win rate (89.6%), while DQN agent 1 had the highest (99.3%). The agent learned to almost always eat the ghost for extra points, with agents 1,2 and 3 eating the ghost in every single test game. Agents 4 and 5 ate the ghost in 983/1000 test games. Most importantly, the DQN agent improved on the tabular agents by learning to eat the ghost while still winning the game.

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg game score	787.05	760.01	744.73	684.29	697.00
Win rate (%)	99.3	96.9	95.3	89.6	91.4
Ghosts eaten	1000	1000	1000	983	983

Table 5.3: Average game scores, win rates and number of ghosts eaten of the five DQN agents in the miniature Pac-Man environment.

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg game score	567.43	545.13	563.90	582.64	521.03
Win rate (%)	96.8	94.3	96.4	98.5	91.3
Ghosts eaten	17	52	30	8	80

Table 5.4: Avg game scores, win rates and number of ghosts eaten of the five lexicographic tabular Q-learning agents in the miniature Pac-Man environment.

Figure 5.2 shows a lineplot of DQN agent 1's loss function value during training. We can see it decreasing almost monotonically, with only a slight upwards trend around 5000 training episodes. After about 7500 training games it reaches a plateau, not significantly decreasing further. A similar observation can be made in figure 5.1. Here we see that the average game score of the DQN agent reaches its highest point after about 7500 training episodes and then oscillates around the same values. We can conclude that at the 7500 episode mark this agent already found a nearly optimal policy. However, other agents like DQN 4 might need the full 19000 training episodes or even more to get to this point.

From figure 5.1 we see that the DQN consistently scores higher than the tabular Q-learning agent during training - a fact that is reflected in the test results as well - and that the DQN's game scores do not oscillate as heavily. The advantage of tabular Q-learning lies in its computational efficiency, as training the DQN for 19000 training episodes took about 11 times longer than training the simple tabular agent.

5.1.3 Lexicographic Tabular Q-Learning

Table 5.4 contains the results of the experiments involving the lexicographic tabular Q-learning (LTQ) agents. On average, the LTQ agents achieved a game score of 556.00 with a standard deviation of 21.18. The average win rate of the LTQ agent is 95.46% with a standard deviation of 2.5%. In the 1000 test games the LTQ agent ate 37.4 ghosts on average (25.9 standard deviation). Agent 4 achieved the highest average game score (582.64), highest win rate (98.5%) and fewest ghosts eaten (8) while agent 5 achieved the lowest average game score (521.03), lowest win rate (91.3%) and most ghosts eaten (80).

Since the regular tabular Q-learning agents only have the goal of maximizing game score we might have expected them to achieve a higher win rate than the LTQ agents, but actually the opposite is the case. The normative penalty for eating ghosts made the LTQ agent try to avoid ghosts altogether, making it more likely to win the game. This increased win rate is also the reason why the LTQ agent achieved a higher average game score than the regular Q-learning agents, despite eating way fewer ghosts. Although the amount of ghosts eaten is significantly reduced, the LTQ agent still violated norms in up to 8% of its test games, which might be unacceptable for some applications.

Interestingly, for the LTQ agent the average game score and the amount of ghosts eaten are negatively correlated with a Pearson correlation coefficient of -0.4. When the agent ate a ghost, it only achieved a mean score of 151.26. Otherwise, it reached a mean score

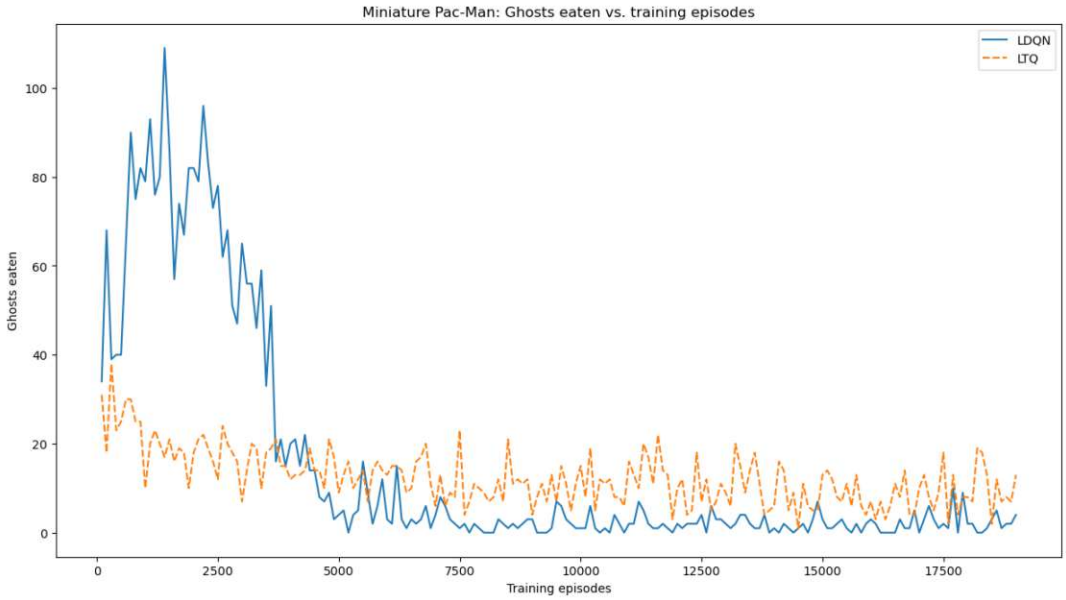

Figure 5.3: Number of norm violations returned by the normative supervisor in steps of 100 during training of the highest scoring lexicographic agents versus number of training episodes.

of 571.75. This result is counter-intuitive because the game rewards a bonus of +200 points for eating a ghost, but it can be explained by the fact that the agent tends to lose the game shortly after violating the norm. In fact, the LTQ agent only won 38% of the games in which it ate a ghost. The reason could be that the agent found itself in an unexpected situation which it did not sufficiently encounter in its training games and therefore chose suboptimal actions.

Figure 5.3 shows the number of normative violations incurred during the training of LTQ agent 4 versus the number of training episodes. The number starts out low and decreases only slightly over the course of training. The agent quickly learns that eating ghosts leads to penalties, so it never tries to increase its game score by eating more ghosts. In figure 5.1 we can see that the average game score of the LTQ agent still increases with more training games, therefore the agent is learning to win the game without eating more ghosts. The average game score during training of the LTQ agent does not oscillate quite as much as that of the tabular Q-learning agent, but we still observe some instabilities resulting from the tabular Q-function representation.

Training and testing the LTQ agent took about 4.5 times longer than the evaluation of the tabular Q-learning agent. This is mainly because of the costly calls to the normative supervisor to get the normative penalty after every action of the agent.

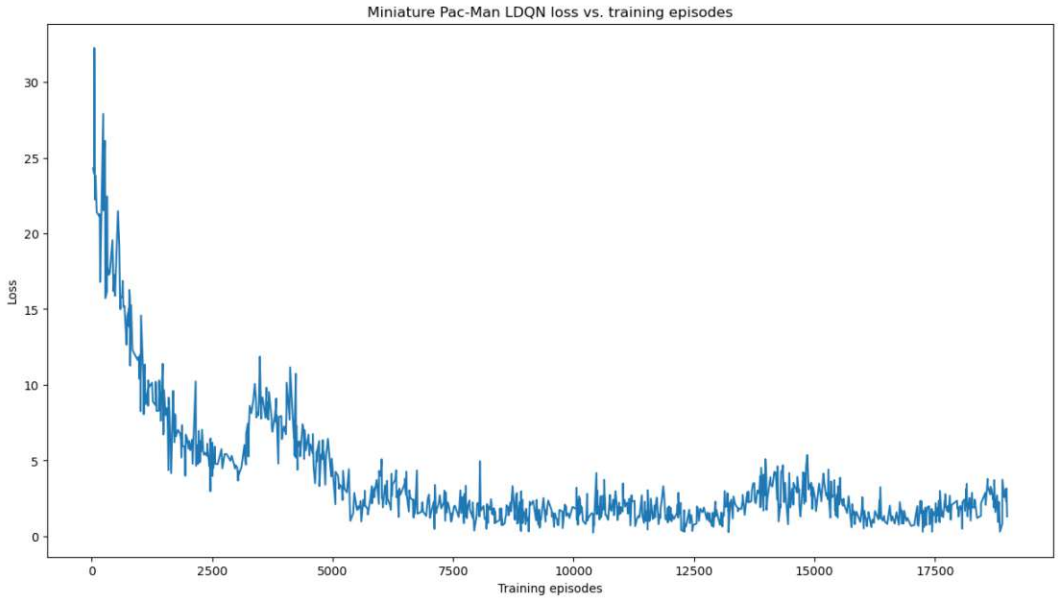

Figure 5.4: Value of the highest scoring LDQN agent loss function versus number of training episodes.

5.1.4 Lexicographic Deep Q-Network

In table 5.5 the experimental results for the five LDQN agents are displayed. The clear standout is agent 1, which won all of the test games, achieved an average game score of 597.05 and did not eat a single ghost. Agent 2 performed considerably worse, as it only achieved an average game score of 467.27 and won 87.4% of its games, but it was still perfectly compliant with the normative system. The overall average game score of the LDQN agent is 545.49 and the mean win rate is 95%, with standard deviations of 49.78 and 4.8%, respectively. The only agent that violated the normative system is agent 5, as it ate 6 ghosts. In terms of the non-normative objective it also performed below average, being second-to-last regarding average game score and win rate.

Clearly, the LDQN is able to outperform the LTQ agent in learning a lexicographically optimal policy. The average game score of agent 1 is only 2 points below the maximum possible score without eating ghosts. However, the game performance of the LDQN agent is a bit more inconsistent. Still, when it comes to normative compliance the LDQN agent is unmatched. Over the course of 5 test executions the LDQN agent only ate 6 ghosts, while the LTQ ate 187. We can conclude that the LDQN was able to learn a norm-compliant policy without sacrificing much of the excellent game performance of the regular DQN agent.

Figure 5.4 shows the value of the loss function of agent 1 during training. Similar to figure 5.2 the curve decreases monotonically except for a local maximum at about 4000 training episodes. At around 6000 training episodes the curve reaches an approximate

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg game score	597.05	467.27	574.12	581.84	507.14
Win rate (%)	100.0	87.4	97.7	98.5	91.3
Ghosts eaten	0	0	0	0	6

Table 5.5: Average game scores, win rates and number of ghosts eaten of the five lexicographic DQN agents in the miniature Pac-Man environment.

equilibrium. This once again coincides with the highest point of the LDQN agent's training score in figure 5.1, after which the score does not significantly decrease anymore.

The plot of the number of norm violations by LDQN agent 1 over the course of training in figure 5.3 is very distinct from what we observed with the LTQ agent. The value starts out on a comparable level to that of the LTQ agent but then quickly rises, peaking at about 2000 training episodes. The curve then falls steeply before reaching the level of the LTQ agent again at around 3750 training episodes. After roughly 5000 training episodes the agent has learned to avoid violating norms and it violates even fewer norms than the LTQ agent. The sharp increase in violations at the beginning of training is likely related to the concrete implementation of the agent, especially the dynamic tolerance parameter (details can be found in appendix A).

Computationally, training the LDQN agent is very expensive, because it involves both training a neural network and numerous calls to the normative supervisor. This is why evaluating the LDQN agent took about 12 times longer than evaluating the DQN agent and 132 times longer than evaluating the tabular Q-learning agent.

5.1.5 Discussion and Comparison to NGRL

This section will briefly review and discuss the results of the lexicographic agents in the miniature Pac-Man environment. Moreover, we will compare the different agents to the state of the art in normative RL: Norm-guided RL agents. An overview is given by table 5.6. Since the agents from [Neu22] were only trained for 9000 episodes we show the test results of the lexicographic agents after the same amount of training games to enable a

Agent	Win rate (%)	Avg score	Ghosts eaten
NGRL	82.00	433.74	142
TQ (OCC)	46.60	25.22	0
NGRL (OCC)	86.30	448.23	1
LTQ	82.9	441.147	118
LDQN	98.7	583.54	0

Table 5.6: Summary of the experimental results involving lexicographic and NGRL agents after training for 9000 games in the miniature Pac-Man environment. The first three rows are taken from [Neu22].

fair comparison. We use the evaluation criteria from section 4.3 as a framework for our comparison:

- Normative compliance: The clear winner in terms of normative compliance is the LDQN agent. Without online compliance checking (OCC) the NGRL agents ate 142 ghosts in 1000 test games [Neu22]. With OCC, this number was reduced to 1 for NGRL agents[1] and 0 for tabular Q-learning. The LTQ agent ate 118 ghosts in its test games. While this is a slight improvement over NGRL without OCC, we may want to completely eliminate norm violating behavior if possible. The LDQN agent was able to accomplish this without the help of OCC. For especially safety-critical applications, we may even want to deploy a LDQN agent with OCC, effectively guaranteeing that the agent will violate the minimum number of norms possible.
- Game performance: The win rate and average game score of the LTQ agent were slightly better compared to NGRL without OCC and slightly worse compared to NGRL with OCC, but overall these agents performed on a similar level. The tabular Q-learning agent with OCC performed way worse than all the other agents. This is to be expected, as the optimal policy of the MDP it was trained in differs greatly from the optimal policy of the MDP it is deployed in. The best agent in terms of win rate and average game score is once again the LDQN agent, as it won almost all of its test games.
- Computational efficiency: During training, the computationally most efficient agent is the tabular Q-learning agent with OCC, as OCC only comes into effect during execution time. The NGRL and LTQ agents take longer to train due to the requests to the normative supervisor. However, as we saw in table 5.1, training a neural network leads to even larger computational demand. In the above experiments, the time needed for training and evaluating the LDQN agent was about 30 times longer than the time needed for the LTQ agent. With that being said, at execution time the LDQN agent is more efficient than the methods involving OCC as the LDQN agent is able to operate without further calls to the normative supervisor.

To summarize, for the miniature Pac-Man environment the LDQN agent is the best choice if normative compliance has priority. The LTQ agent can nevertheless be a suitable alternative if training a LDQN agent is too expensive and rare norm violations can be tolerated.

[1] This single consumed ghost results from an edge case in the Pac-Man environment described in figure 2(c) from [Neu+21]. In this situation the agent collides with a ghost and a power pellet in the same time step, consuming both instantly. This is not properly recognized as a norm violation by the normative supervisor.

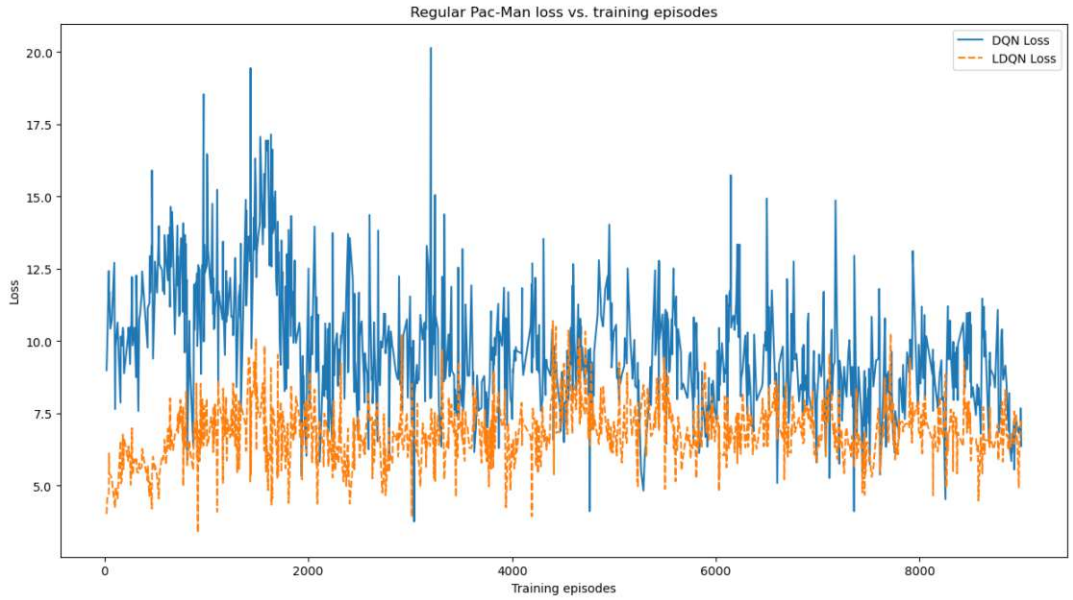

Figure 5.5: Value of the DQN and LDQN agent loss function in the regular Pac-Man environment versus number of training episodes.

5.2 Regular Pac-Man Experiments

In the following the experimental results from the regular Pac-Man environment will be discussed. A summary is shown in table 5.7. Note that in this environment, the agent can eat a maximum of 4 ghosts, as there are 2 power pellets and the agent can eat 2 ghosts after consuming a power pellet.

The DQN agent did not do well in this complex environment. About 17% of the DQN agent's test games were won. The average score was 437.33 and the agent ate 1080 ghosts in total. As can be seen in figure 5.5 the value of the DQN agent's loss function varies greatly over the course of training and no clear decrease is visible, suggesting that the agent is not able to learn a policy that consistently increases reward. However, it is worth noting that despite the low win rate, the DQN agent was able to maintain a positive average game score. The agent achieved a mean game score of 193.79 in games that it lost. Taking into consideration that the agent ate about 1 ghost per game, it must have scored about 500 points by eating food pellets to offset the -500 score penalty for losing.

Agent	Win rate (%)	Avg score	Blue g. e.	Orange g. e.	Evaluation time (s)
DQN	17.1%	437.33	535	545	6929.88
LDQN	3.0%	300.49	600	588	88358.80

Table 5.7: Summary of the experimental results in the regular Pac-Man environment ("ghosts eaten" is abbreviated as "g. e.").

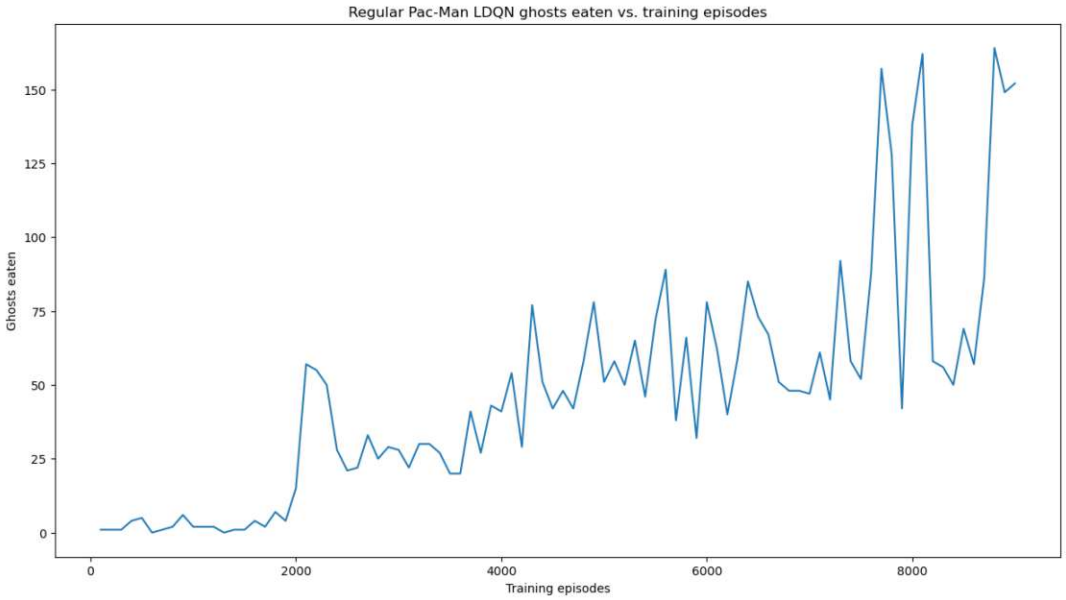

Figure 5.6: Number of norm violations returned by the normative supervisor in steps of 100 during training of the LDQN agent in the regular Pac-Man environment versus the number of training episodes.

Since eating a food pellet grants the agent 9 points (with the -1 score penalty per time step) about 55 food pellets must have been consumed by the agent on average. Hence, the agent was mostly able to survive for an extended amount of time, but rarely able to successfully close out the game.

In other experiments where we trained a DQN agent in the regular Pac-Man environment for 54000 games the agent was able to win 68.2% of its test games with an average game score of 1194.30, but even more training games did not significantly improve performance. Therefore, while the performance can be improved with more training, no amount of training games is able to elevate the DQN agent's performance in the regular Pac-Man environment to the level that we have come to expect from it in the miniature Pac-Man environment.

Unfortunately, experiments with the LDQN agent did not yield better results in this environment. The LDQN won 30 of its 1000 test games and achieved an average game score of 300.49. Similarly to the DQN agent the value of the LDQN loss function shows no clear trend of converging in figure 5.5. Nevertheless, the LDQN agent also achieved high scores despite its low win rate, as it scored 261.66 points on average in its losses.

The LDQN agent also failed in its normative objective by eating more total ghosts than the DQN agent (1188). In figure 5.6 we see that the number of ghosts eaten by the LDQN agent increased over time instead of decreasing. After about 7000 training episodes the number begins to heavily oscillate, as if the agent was undecided whether it should obey

the normative system or increase its game score.

The reasons for these failures could be manifold. The universal approximation theorems for neural networks tell us that for any function, a sufficiently deep or wide neural network can be constructed to approximate this function to an arbitrary degree. Therefore, it is certain that a combination of network architecture and hyperparameter configuration exists that can better approximate a lexicographically optimal policy. However, any specific neural network architecture will be unable to learn all functions and we were unable to explore a large number of alternative configurations due to the extensive computational resources required.

5.3 Unfair Pac-Man Experiments

The final set of experiments involves observing how the lexicographic agent acts in a state of normative deadlock. Recall the layout of the unfair Pac-Man environment: A blue and a red ghost are stationed at the far ends of a small 5×1 corridor. There are two power pellets, one to the east of the blue ghost and one to the west of the orange ghost. Pac-Man's starting position is between the power pellets.

When Pac-Man eats a food pellet and there are no additional food pellets left, it wins the game. Since there are no food pellets in the unfair Pac-Man environment, winning the game is not possible. Therefore we disregard game performance in our evaluation. Instead, we focus on whether the agent chooses to eat ghosts and if so, which ones.

If Pac-Man has access to its full set of actions (including stopping and moving in any direction) there are two types of game outcomes:

1. If the agent chooses to move either east or west as its first action, it consumes a power pellet. Furthermore, a ghost will move into Pac-Man at the following time step, ensuring that Pac-Man eats the ghost. The consumed ghost will subsequently reappear in its starting position and the other ghost will have moved to the remaining power pellet. Thus, Pac-Man will be stuck between ghosts and lose the game after eating a ghost.

2. If the agent chooses to stop as its first action (either explicitly or by moving against a wall) then the ghosts will advance onto the power pellets. With no way to escape and no access to power pellets the agent will then lose the game without eating a ghost.

We conducted two sets of experiments with a LTQ agent and the unfair Pac-Man normative system, where eating ghosts is a norm violation but consuming the orange ghost is less bad than eating the blue ghost. In the first set, Pac-Man could take any of its five actions. This resulted in the agent choosing to stay in place in every test game, leading to the second of the outcomes described above. In the second set, Pac-Man was

forced to pick between moving east or west. It chose to move east in all test games, thereby eating the orange ghost.

These results provide evidence that the lexicographic agents are able to obey sub-ideal contrary-to-duty obligations. If the agent interpreted $\mathbf{O}(eat_{orangeGhost} \mid \neg benevolent)$ as a compensatory CTD obligation then it would have eaten the orange ghost in the first set of experiments to increase its score. Instead, the agent chose to forgo the score increase to not violate any norms. Moreover, when we forced the agent to violate a norm, it chose to violate only one norm instead of two.

CHAPTER 6

Conclusion

In summary, we initially covered the basics of value-based RL, multi-objective RL, normative reasoning and deontic logic. We then examined existing approaches for constrained RL, noting the advantages of normative RL over safe RL. The novel technique of norm-guided RL was presented and we saw that regular NGRL cannot teach agents to obey CTD obligations. NGRL with violation counting was designed to alleviate this shortcoming but instead it is incompatible with function approximation, an indispensable part of modern deep RL algorithms. As a solution we proposed norm-guided lexicographic multi-objective RL (NGLMORL), combining norm-guided RL with lexicographic multi-objective RL. With this method, agents learn normative compliance by learning a lexicographically optimal policy that first tries to minimize normative penalties given to the agent by a normative supervisor and maximizes non-normative rewards as a second priority. We tested the capabilities of these agents in several layouts of the Pac-Man arcade game. In a miniature version of the game the agents did very well, winning more games than their non-lexicographic counterparts and mostly complying with the normative system. With regards to NGRL, the lexicographic tabular Q-learning agent scored comparably, while the lexicographic DQN vastly outperformed it in terms of game score and normative compliance. However, in a larger environment the agents were unable to learn a good policy. Finally, we showed experimentally that NGLMORL agents obey sub-ideal CTD obligations in states of normative deadlock.

The experiments have revealed the following limitations of NGLMORL:

- Hyperparameters and neural network architecture needs to be tuned in every new environment. We saw that without a proper configuration the agents may fail to learn a good policy and comply with the normative system. Moreover, hyperparameter optimization is a lengthy and computationally costly process.
- NGLMORL inherits the scaling issues of NGRL because it must still call a theorem prover after action selection during training. For widespread application of the

technique it is desirable to develop more efficient ways of calculating the normative penalty. However, these issues do not affect the agent during operation.

- The agent must be retrained from scratch whenever we want it to follow a different normative system. Since ethical, legal and social norms change over time a seamless way to integrate normative system change into the agent's policy is crucial.
- During operation, NGLMORL agents suffer from transparency issues because all information that the agent receives about the normative system is collapsed into opaque value functions. Without online compliance checking by the normative supervisor we lose the ability to record events that constitute a norm violation and to generate an explanation for why the violation occurred.

This leads us to the following possibilities for future research:

- By finding a suitable hyperparameter configuration in the regular Pac-Man and other environments we can deepen our understanding of how each individual hyperparameter affects the behavior of the agent, facilitating the deployment of the technique in more complex environments.
- In this work we have applied NGLMORL only with value-based RL algorithms, but much of the recent success in deep RL rests on the shoulders of policy-based RL. Exploring the challenges of using NGLMORL with a policy-based lexicographic algorithm would be an interesting research direction.
- The DDL theorem prover SPINdle used by the normative supervisor sufficed for our purposes but could be replaced by a more efficient solution, e.g. answer set programming (ASP). Even better would be a theorem prover able to reason with the temporal dimension of norms, i.e. achievement obligations (which require a proposition to *eventually* be fulfilled) and norms with deadlines. However, that would require a computationally feasible temporal deontic logic as a prerequisite.
- When designing the normative reward function $R_{\mathcal{N}}$ of NGLMORL we assumed that each violated norm in $\mathcal{N}$ contributes the same amount to the overall normative penalty. However, in actuality the violation of some norms may be punished more severely than the violation of others. Therefore the normative supervisor should be augmented with the ability to assign weights to norms depending on their significance.

APPENDIX A

Technical details

This section documents the hyperparameters and neural network architecture used for the experiments described in section 4.3.

A.1 Tabular agents

The tabular Q-learning agent was trained with a learning rate $\alpha = 0.2$, a constant $\epsilon = 0.05$ and a discount factor of $\gamma = 0.8$.

The lexicographic tabular Q-learning agent (LTQ) was trained with the same learning rate and ϵ-value, but had a discount factor of $\gamma = 0.99$. Additionally, it used a proportional lexicographic tolerance parameter $\tau = 0.001$.

A.2 Deep agents

The neural network approximating the Q-function for the DQN and LDQN agents consists of two 2D convolutional layers and two fully connected linear layers. The first convolutional layer has 6 input channels, 32 output channels and a 3×3 kernel with stride 1. The second convolutional layer has 32 input and 64 output channels. In the miniature Pac-Man environment, the second convolutional layer uses a 2×2 kernel with stride 1. When training in the regular Pac-Man environment, the second convolutional layer uses a 3×3 kernel with stride 1. The first linear layer scales the output of the second convolutional layer down to 256 hidden units and the second linear layer scales it down to $4m$ output units where m is the number of reward functions in the MOMDP we are training in. The deep agents do not have the ability to choose the "Stop" action, they have to choose a direction to move at every time step.

The input to the neural network is a tensor of dimension $(6 \times h \times w)$ where h and w are the height and width of the environment, respectively. This tensor contains 6 binary

matrices encoding the positions of walls, the agent, ghosts, scared ghosts, food and power pellets.

The loss function used by the deep agents is the smooth L1 loss. For two vectors $\vec{x}, \vec{y}$ of dimension n, the smooth L1 loss is defined as $L(\vec{x}, \vec{y}) = \{l_1, ..., l_n\}^T$ with

$$l_i = \begin{cases} \frac{(x_i - y_i)^2}{2} & \text{if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5 & \text{otherwise} \end{cases}$$

for $1 \leq i \leq n$. The AdamW algorithm [LH19] is used for optimization.

The deep agents use an exponentially decaying ϵ-value. It is calculated as

$$\epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) * e^{\frac{-t}{d}}$$

where t is the current time step, d is a constant controlling the decay rate, ϵ_{start} is the starting value and ϵ_{end} is the value of ϵ as $t \to \infty$. In miniature Pac-Man, $\epsilon_{start} = 0.9$, $\epsilon_{end} = 0.05$ and $d = 2x$ where x is the number of training episodes. In regular Pac-Man the values of ϵ_{start} and ϵ_{end} are the same, but $d = 10x$.

The tolerance parameter τ for LDQN is also variable throughout training. It is calculated as

$$\tau_{start} + (\tau_{end} - \tau_{start}) * \frac{i}{x}$$

where i is the number of the current episode, x is the total number of training episodes, τ_{start} is the starting value and τ_{end} is the value at the end of training. In contrast to the LTQ agent, the LDQN agent uses the tolerance as a constant instead of a proportion. In the experiments, the values of τ_{start} and τ_{end} are different depending on whether they are used with the normative reward function $R_{\mathcal{N}}$ or the environment reward R_x. For $R_{\mathcal{N}}$, $\tau_{start} = 0.1$ and $\tau_{end} = 1$. For R_x, $\tau_{start} = 0.5$ and $\tau_{end} = 0.1$.

The other hyperparameters for DQN and LDQN in the miniature Pac-Man environment are as follows: A learning rate $\alpha = 0.001$, a discount factor $\gamma = 0.99$, a target net update rate $\tau' = 0.005$, a replay buffer size $N = 10000$ and a minibatch size $n = 64$. The values for the regular Pac-Man environment are the same, except $\alpha = 0.0001$.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Als+18] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. „Safe Reinforcement Learning via Shielding“. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 29, 2018). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v32i1.11797. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11797 (visited on 06/05/2023).

[BV04] Guido Boella and Leendert Van der Torre. „Regulative and Constitutive Norms in Normative Multiagent Systems“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*. AAAI Press, 2004, pp. 255–266.

[For84] James William Forrester. „Gentle Murder, or the Adverbial Samaritan“. In: *Journal of Philosophy* 81.4 (1984). Publisher: Journal of Philosophy Inc, pp. 193–197. DOI: 10.2307/2026120.

[Gov+07] Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. „Characterising Deadlines in Temporal Modal Defeasible Logic“. In: *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*. Ed. by Mehmet A. Orgun and John Thornton. Vol. 4830. Lecture Notes in Computer Science. Springer, 2007, pp. 486–496. DOI: 10.1007/978-3-540-76928-6_50. URL: https://doi.org/10.1007/978-3-540-76928-6%5C_50.

[Gov18] Guido Governatori. „Practical Normative Reasoning with Defeasible Deontic Logic“. In: *Reasoning Web International Summer School*. Springer, Sept. 21, 2018, pp. 1–25.

[HAK20] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. „Cautious Reinforcement Learning with Logical Constraints“. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 13, 2020, pp. 483–491. ISBN: 978-1-4503-7518-4. (Visited on 04/06/2023).

[int19] Independent high-level expert group on artificial intelligence. *Ethics Guidelines for trustworthy AI*. European commission, Apr. 8, 2019. URL: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai.

[Jan+19] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alexandru C. Serban, and Roderick Bloem. *Safe Reinforcement Learning via Probabilistic Shields*. Nov. 25, 2019. DOI: 10.48550/arXiv.1807.06096. arXiv: 1807.06096[cs]. URL: http://arxiv.org/abs/1807.06096 (visited on 04/06/2023).

[LG09] Ho-Pun Lam and Guido Governatori. „The Making of SPINdle". In: *Rule Interchange and Applications*. Ed. by Guido Governatori, John Hall, and Adrian Paschke. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 315–322. ISBN: 978-3-642-04985-9. DOI: 10.1007/978-3-642-04985-9_29.

[LH19] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. Jan. 4, 2019. DOI: 10.48550/arXiv.1711.05101. arXiv: 1711.05101[cs,math]. URL: http://arxiv.org/abs/1711.05101 (visited on 11/10/2023).

[Mni+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. „Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015). Number: 7540 Publisher: Nature Publishing Group, pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236 (visited on 10/02/2023).

[NBC22] Emery A. Neufeld, Ezio Bartocci, and Agata Ciabattoni. „On Normative Reinforcement Learning via Safe Reinforcement Learning". In: *PRIMA 2022: Principles and Practice of Multi-Agent Systems: 24th International Conference, Valencia, Spain, November 16–18, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, Nov. 16, 2022, pp. 72–89. ISBN: 978-3-031-21202-4. DOI: 10.1007/978-3-031-21203-1_5. URL: https://doi.org/10.1007/978-3-031-21203-1_5 (visited on 04/06/2023).

[Neu+21] Emery Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. „A Normative Supervisor for Reinforcement Learning Agents". In: *Automated Deduction – CADE 28*. Ed. by André Platzer and Geoff Sutcliffe. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 565–576. ISBN: 978-3-030-79876-5. DOI: 10.1007/978-3-030-79876-5_32.

[Neu22] Emery Neufeld. „Reinforcement Learning Guided by Provable Normative Compliance“. In: *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*. 2022, pp. 444–453. DOI: 10.5220/0010835600003116. arXiv: 2203.16275[cs]. URL: http://arxiv.org/abs/2203.16275 (visited on 05/21/2023).

[Neu23] Emeric Alexander Neufeld. „Norm Compliance for Reinforcement Learning Agents“. Accepted: 2023-06-02T09:28:28Z. Thesis. Technische Universität Wien, 2023. DOI: 10.34726/hss.2023.112881. URL: https://repositum.tuwien.at/handle/20.500.12708/177391 (visited on 06/05/2023).

[Noo+19] Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R. Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. „Teaching AI Agents Ethical Values Using Reinforcement Learning and Policy Orchestration“. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 6377–6381. DOI: 10.24963/ijcai.2019/891. URL: https://doi.org/10.24963/ijcai.2019/891.

[Ope+19] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. *Dota 2 with Large Scale Deep Reinforcement Learning*. Dec. 13, 2019. DOI: 10.48550/arXiv.1912.06680. arXiv: 1912.06680[cs, stat]. URL: http://arxiv.org/abs/1912.06680 (visited on 11/22/2023).

[Ope23] OpenAI. *GPT-4 Technical Report*. Mar. 27, 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774[cs]. URL: http://arxiv.org/abs/2303.08774 (visited on 11/22/2023).

[RLR21] Manel Rodriguez-Soto, Maite Lopez-Sanchez, and Juan A. Rodriguez Aguilar. „Multi-Objective Reinforcement Learning for Designing Ethical Environments“. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 545–551. DOI: 10.24963/ijcai.2021/76. URL: https://doi.org/10.24963/ijcai.2021/76.

[Sal+17] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. „Deep Reinforcement Learning framework for Autonomous Driving“. In: *Electronic Imaging* 29.19 (Jan. 29, 2017), pp. 70–76. ISSN: 2470-1173. DOI: 10.2352/ISSN.2470-1173.2017.19.AVM-023. arXiv: 1704.

02532[cs,stat]. URL: http://arxiv.org/abs/1704.02532 (visited on 11/22/2023).

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. 526 pp. ISBN: 978-0-262-03924-6.

[Sil+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. Dec. 5, 2017. DOI: 10.48550/arXiv.1712.01815. arXiv: 1712.01815[cs]. URL: http://arxiv.org/abs/1712.01815 (visited on 11/22/2023).

[Ska+22] Joar Skalse, Lewis Hammond, Charlie Griffin, and Alessandro Abate. „Lexicographic Multi-Objective Reinforcement Learning“. In: Thirty-First International Joint Conference on Artificial Intelligence. Vol. 4. ISSN: 1045-0823. July 16, 2022, pp. 3430–3436. DOI: 10.24963/ijcai.2022/476. URL: https://www.ijcai.org/proceedings/2022/476 (visited on 05/24/2023).

[SMW15] Richard S. Sutton, A. Rupam Mahmood, and Martha White. *An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning*. Apr. 20, 2015. DOI: 10.48550/arXiv.1503.04269. arXiv: 1503.04269[cs]. URL: http://arxiv.org/abs/1503.04269 (visited on 10/02/2023).

[TV97] J.N. Tsitsiklis and B. Van Roy. „An analysis of temporal-difference learning with function approximation“. In: *IEEE Transactions on Automatic Control* 42.5 (May 1997). Conference Name: IEEE Transactions on Automatic Control, pp. 674–690. ISSN: 1558-2523. DOI: 10.1109/9.580874. URL: https://ieeexplore.ieee.org/document/580874 (visited on 10/02/2023).

[Wat89] Christopher John Cornish Hellaby Watkins. „Learning from Delayed Rewards“. PhD thesis. Cambridge, UK: King's College, May 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[WR19] Aimee Wynsberghe and Scott Robbins. „Critiquing the Reasons for Making Artificial Moral Agents“. In: *Science and Engineering Ethics* 25 (June 2019). DOI: 10.1007/s11948-018-0030-8.