DRAMA at the PettingZoo: Dynamically Restricted Action Spaces for Multi-Agent Reinforcement Learning Frameworks

Michael Oesterle, Tim Grams, Christian Bartelt
Institute for Enterprise Systems (InES), University of Mannheim
michael.oesterle@uni-mannheim.de, tim.nico.grams@uni-mannheim.de, christian.bartelt@uni-mannheim.de

Abstract

The Agent Environment Cycle *(AEC) of PettingZoo has been a major paradigm shift in the implementation of Multi-Agent Reinforcement Learning (MARL) frameworks, providing a unified and concise interface for any kind of multi-agent environment. Based on this model, we propose DRAMA, a principled approach for dynamic action space restrictions. DRAMA can be used to add statically computed physical constraints as well as a self-learning multi-agent governance: It generalizes the idea of action masking to continuous action spaces and self-learning restrictions, while being fully compatible with the AEC implementation of PettingZoo—and, by transitivity, with most major MARL frameworks. In this paper, we provide the theoretical background of restricted multi-agent systems, present an extension of PettingZoo via wrapper classes, and show the potential of our approach for various use cases. By treating dynamic restrictions as an additional player of a multi-agent system, our approach offers novel capabilities and flexibility in handling multi-agent environments and thus serves as a valuable tool for researchers and practitioners in the field.*

Keywords: Multi-Agent Reinforcement Learning, OpenAI Gym, PettingZoo, Multi-Agent Systems, Action Space Restriction

1. Introduction

Since its first release in 2017, OpenAI's *Gym* environment specification (Brockman et al., 2016) has become the standard for Reinforcement Learning (RL) (Sutton & Barto, 2018) environments represented as Markov Decision Processes (MDPs) (Bellman, 1957) or, more generally, as Partially Observable Markov Decision Processes (POMDPs) (Åström, 1965). Gym's minimalistic design offers enough freedom and flexibility to allow users to create and train RL agents in their own environments. Consequently, popular RL frameworks like Keras RL (Plappert, 2016), Tensorforce (Kuhnle et al., 2017), Coach (Caspi et al., 2017), Acme (Hoffman et al., 2020), Stable Baselines (Raffin et al., 2021), and CleanRL (Huang et al., 2022) adopt Gym environments as their default environment class.

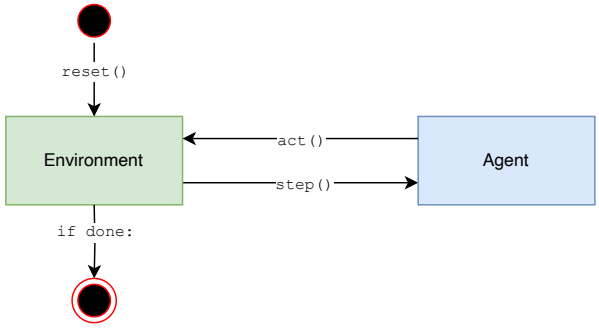


Figure 1. Agent-Environment Loop for a single-agent setting (as implemented in Gym).

However, Gym is designed solely for single-agent learning, employing a loop between the agent `act()` and environment `step()` until the episode is `done` (see Figure 1). To enable multi-agent settings, approaches based on (Partially Observable) Stochastic Games (Shapley, 1953) have been proposed, but, as Terry et al., 2021 have pointed out, implementing them in code faces several unsolved challenges. To overcome these limitations, they introduce the *Agent Environment Cycle* (AEC) model and the corresponding PettingZoo library (Terry et al., 2021), which has gained widespread adoption and works seamlessly with RL frameworks such as The Autonomous Learning Library (Nota, 2020), AI-Traineree (Laszuk, 2020), PyMARL (Samvelyan et al., 2019), RLlib (Liang et al., 2018), Stable Baselines (Hill et al., 2018; Raffin et al., 2021), CleanRL (through SuperSuit) (Huang et al., 2022; Terry et al., 2020), and Tianshou (Weng et al., 2022).

In Gym and PettingZoo, agents typically have access to the same set of actions throughout an episode, either as a discrete set or a box-shaped continuous space. Recent versions of PettingZoo seem to allow

URI: https://hdl.handle.net/10125/107324
978-0-9981331-7-1

changing observation and action spaces at run-time, but instructions are inconsistent[1], and compatibility issues arise with RL algorithms that expect invariant input-output shapes (which is the case for most common deep learning algorithms).

A commonly used solution for dynamic action spaces is *invalid action masking* (Huang & Ontañón, 2022; Vinyals et al., 2017). However, this method, which involves providing a Boolean vector of valid and invalid actions as part of the observation, is limited to discrete spaces and can be inefficient. For instance, in Dota 2, where the action space comprises $1,837,080$ actions (Berner et al., 2019), the masking approach becomes burdensome with respect to storage of observation batches.

Dynamic restrictions of the action spaces, as imposed in many real-world scenarios by physical, legal, or other constraints (Boutilier et al., 2018; Chandak et al., 2020; Mandel et al., 2017), can therefore not be represented by existing RL frameworks in a principled way. To address this limitation, we propose an extension with the following components:

1. The action space, referred to as the *base space*, remains static.
2. Agents receive a *restriction* as part of their observation, representing an arbitrary subset of the base space.
3. Restrictions, represented by `gym.spaces`, efficiently capture arbitrary sets of valid actions.
4. The internal representation of valid actions in a restriction is opaque, while compatibility with RL models is ensured through fixed-length flattening.
5. Restrictions can be defined by the environment or provided by a *restrictor* agent which produces restrictions as actions.
6. A restrictor agent can be treated like any other agent and may be an RL agent or a static function.
7. The interplay between the environment, restrictor, and agents is managed by a *restriction wrapper*.

The theoretical background, formal model and reference implementation of DRAMA are discussed in Sections 3 and 4, preceded by a recap of related work in Section 2. Additionally, we present three use cases in Section 5 before concluding the paper.

[1]The documentation contains an example with the comment "If your spaces change over time, remove this line (disable caching)", but also says "This space should never change for a particular agent ID". The docstring of `AECEnv.action_space()` even states that the function "MUST return the same value for the same agent name".

The reference implementation of DRAMA, along with documentation and examples, has been published at https://github.com/michoest/drama-wrapper.

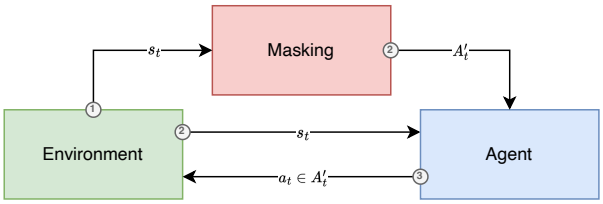
2. Related Work

Existing RL libraries typically have limitations in supporting dynamic observation and action spaces due to the aggregation of trajectories into batches, which require homogeneous tensors (as mentioned in the documentations of Tianshou and RLlib). Padding is often the only method used to handle heterogeneous data. Furthermore, most state-of-the-art RL algorithms rely on fixed neural architectures that can only process flat input and output arrays of constant size (notably, algorithms without function approximators, like tabular Q-learning, do not have this restriction). Although pre-processing techniques can be employed to flatten complex spaces, they also require data of fixed shape.

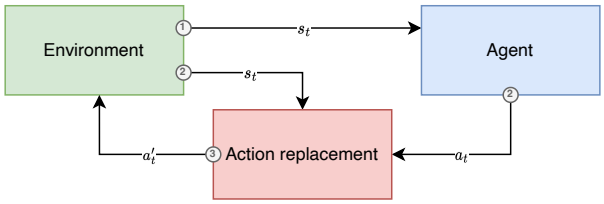
However, RL environments often possess complex space structures, ranging from simple discrete and continuous spaces (Brockman et al., 2016) to mixed discrete-continuous variants (Neunert et al., 2020) and parametric spaces (Fan et al., 2019; Hausknecht & Stone, 2016). To reconcile fixed input and output for RL agents and changing action spaces for RL environments, several solutions have been proposed. These solutions can be categorized as *masking*, where the agent is informed of valid actions and selects from this set, or *replacement*, where an invalid action chosen by the agent is replaced with a valid one (following some replacement strategy), as discussed by Krasowski et al., 2023 and illustrated in Figure 2. Note that we only consider the *environment perspective* here; of course, an agent can also mask or replace actions internally before outputting an action to the environment.

Currently, the most commonly used masking approach for discrete action spaces is invalid action masking, which employs a Boolean masking vector to provide the mask from the environment (Huang & Ontañón, 2022; Vinyals et al., 2017). There is, to the best of our knowledge, no analogous method for infinite or continuous spaces, such that continuous environments need to be discretized for masking (Sinclair et al., 2020; Uther & Veloso, 1998). As for replacement approaches, various alternatives have been proposed, including random replacement and projection (see Krasowski et al., 2023), and penalization (Dietterich, 2000). However, penalty-based RL methods have been shown not to scale well for a large number of invalid actions (Huang & Ontañón, 2022).

Numerous methods exist for handling irregular action spaces *within* an agent's action policy.



(a) Masking: The subset of valid actions is given to an agent before it chooses its action.



(b) Replacement: Invalid actions are replaced by valid ones after an agent has chosen its action.

Figure 2. Intervention points for action space restrictions (cf. Krasowski et al., 2023).

Actor-critic methods (Konda & Tsitsiklis, 1999) can internally penalize the choice of invalid actions while ensuring that the final selected actions are valid. Dulac-Arnold et al., 2016 propose embedding discrete action spaces into continuous spaces using nearest-neighbor methods. Conversely, Y. Tang and Agrawal, 2020 suggest discretizing continuous spaces for masking purposes. Zahavy et al., 2018 train an Action Elimination Network (AEN) to reduce the set of feasible actions, and Kanervisto et al., 2020 enhance learning through action space shaping. Discarding invalid actions and re-sampling is another straightforward method that can be implemented either within an agent's action policy or as a feature of the environment (by setting $\delta(s, a^*) = s$ for any invalid action a^*). However, this method scales poorly when the ratio of invalid actions is high.

The concept of restricting action spaces as a means of governance in multi-agent systems has been explored by Pernpeintner et al., 2021 using search-based optimization; other techniques are RL over a small discrete action space (Oesterle et al., 2022), and tree search over continuous spaces (Oesterle & Sharon, 2023). This dynamic action space shaping complements the commonly used approach of reward shaping, as defined by Normative Systems (Andrighetto et al., 2013; Chopra et al., 2018) or the game-theoretically grounded Vickrey-Clarke-Groves (VCG) mechanisms (Nisan & Ronen, 2004).

In our proposed DRAMA approach, we offer the option to train the restrictor as part of the RL training process. A similar technique, albeit not embedded in a standard framework, is used by Reinforcement Mechanism Design (Cai et al., 2018; P. Tang, 2017).

3. Theoretical Framework

3.1. Background

Reinforcement Learning. The basic underlying mathematical model of any Reinforcement Learning framework is an MDP, formally defined as a tuple

$$(\mathcal{S}, A, r, \delta) \ ,$$

where $\mathcal{S}$ is the set of environmental states, A is the agent's action space, $r : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$ is the agent's reward function, and $\delta : \mathcal{S} \times A \to \Delta_{\mathcal{S}}$ is the (stochastic) transition function[2].

The (stochastic) action policy of the agent is given as a function $\pi : \mathcal{S} \to \Delta_A$. Using this notation, the interaction between agent and environment can be succinctly described by the evolution formula

$$s_{t+1} = \delta(s_t, \pi^{(t)}(s_t)) \ , \tag{1}$$

where the output of the stochastic functions is sampled according to the respective distribution[3].

The learning behaviour of the agent, i.e., the fact that π can change over time in order to maximize the agent's cumulative reward, is not described by the MDP.

In case of partial observability (when the agent cannot see the entire state of the environment), a set $\mathcal{O}$ of possible observations and an observation function $\sigma : \mathcal{S} \to \mathcal{O}$ are added to Equation (1):

$$s_{t+1} = \delta(s_t, \pi(\sigma(s_t))) \ .$$

Multi-Agent Reinforcement Learning. The model for a multi-agent RL framework is a straightforward extension of an MDP, called a *Stochastic Game* (SG). It is a tuple

$$(I, \mathcal{S}, \boldsymbol{A}, \boldsymbol{r}, \delta) \ ,$$

where I is the set of agents, $\mathcal{S}$ is the set of environmental states, $\boldsymbol{A} = (A_i)_{i \in I}$ are the agents' action spaces[4], $r_i : \mathcal{S} \times \boldsymbol{A} \times \mathcal{S} \to \mathbb{R}$ is agent i's reward function, and $\delta : \mathcal{S} \times \boldsymbol{A} \to \Delta_{\mathcal{S}}$ is the transition function.

[2] Δ_X denotes the set of probability distributions over a (finite or infinite) set X.

[3] The use of the time step as a superscript indicates that a variable or function evolves with t.

[4] By convention, vectors and sets of variables are written in bold face.

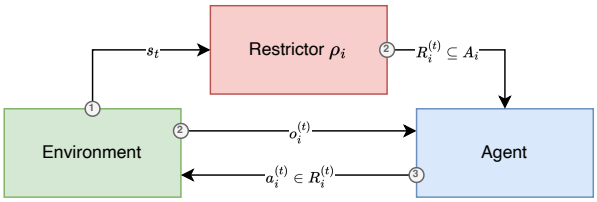


Figure 3. Agent-Restrictor-Environment loop of DRAMA.

As above, each agent has an action policy $\pi_i : \mathcal{S} \to \Delta_A$ defining its behaviour; the evolution of the system is therefore described by

$$s_{t+1} = \delta(s_t, \boldsymbol{\pi}^{(t)}(s_t)) \ .$$

Accounting for partial observations (*Partially Observable Stochastic Game*, or POSG), the extension

$$s_{t+1} = \delta(s_t, \boldsymbol{\pi}^{(t)}(\boldsymbol{\sigma}(s_t))) \tag{2}$$

is straight-forward.

3.2. DRAMA model

As outlined in Section 1, we keep the action spaces $\boldsymbol{A}$ static and introduce action space restrictions $R_i^{(t)} \subseteq A_i$, allowing for the A_i to be dynamically constrained. In the POSG context, we represent this with an additional functional entity (the *restrictor*) which is now part of the agent-environment loop (see Figure 3).

The formal model of DRAMA is a tuple

$$(I, \mathcal{S}, \boldsymbol{\mathcal{O}}, \boldsymbol{\sigma}, \boldsymbol{A}, \boldsymbol{\rho}, \boldsymbol{r}, \delta) \ ,$$

where $\boldsymbol{\mathcal{O}} = (\mathcal{O}_i)_{i \in I}$ are the observation sets, $\sigma_i : \mathcal{S} \to \mathcal{O}_i$ are the agents' observation functions, and $\boldsymbol{\rho} = (\rho_i)_{i \in I}$ with $\rho_i : \mathcal{S} \to 2^{A_i}$ are the restriction functions that are applied to each agent, respectively. Accordingly, the agent policies now take a restriction as an additional argument, i.e., $\pi_i : \mathcal{S} \times 2^{A_i} \to \Delta_A$[5] with the requirement that $\mathrm{supp}(\pi_i(s, R)) \subseteq R$[6]. This requirement ensures that actions that are forbidden by the restriction R are taken with probability zero.

The evolution function in this model is derived from Equation (2) as

$$s_{t+1} = \delta(s_t, \boldsymbol{\pi}^{(t)}(\boldsymbol{\sigma}(s_t), \boldsymbol{\rho}^{(t)}(s_t))) \ .$$

[5]We denote with $2^S := \{S' : S' \subseteq S\}$ the power set of an arbitrary set S, both finite and infinite.

[6]For a function $f : X \to \mathbb{R}$, $\mathrm{supp}(f) := \{x \in X : f(x) \neq 0\}$ denotes the *support* of f.

4. Implementation

In the standard AEC, three entities are of importance: `gym.Spaces` are passed back and forth between an `AECEnv` and one or more `Agents`, as shown in the minimal execution loop:

```
env.reset()
for agent in env.agent_iter():
    observation, _⋆ = env.last()
    action = agents[agent].act(observation)
    env.step(action)
```

DRAMA directly builds upon this setup, using the exact same loop. We define three more classes with their respective base classes, each corresponding to one of the above entities:

`Restriction(Space)` represents any subset of an action space.

`Restrictor(Agent)` is an agent whose actions are `Restrictions`.

`RestrictionWrapper(AECEnv)` manages the succession of agent and restrictor actions, as well as the enhancement of agent observations with the respective restrictions.

This reflects one of the fundamental design decisions of DRAMA: The agent policies $\boldsymbol{\pi}$ and the restriction policies $\boldsymbol{\rho}$ are defined in the same way, such that both kinds of policies can be learned within the training process as implemented by PettingZoo-compatible MARL frameworks. Hence, any restriction needs to be a valid `gym.Space` which can be batched for training and evaluation workflows, and restriction policies are queried (and potentially trained) like an agent policy in the AEC. Moreover, DRAMA is designed to be extensible by sub-classing any of its components (e.g., to define more complex restrictions), even though the reference implementation contains the necessary classes for a range of applications.

4.1. Restriction

Discrete Restriction. For discrete spaces `gym.Discrete(n, start=s)` with the action set $\{s, s+1, ..., s+n-1\}$, restrictions have traditionally been implemented as action masks. These masks are Boolean vectors of length n, where each entry indicates whether an action is allowed (`True`) or forbidden (`False`). While this approach is suitable for small n, it becomes inefficient when n is large and only a small fraction of actions is allowed at each step. To address this, we introduce two implementations of `DiscreteRestriction`:

`DiscreteSetRestriction` stores a set of allowed actions, while `DiscreteVectorRestriction` follows the conventional vector representation, but as a subclass of `gym.Space`.

Continuous Restriction. We provide two restriction classes for one-dimensional continuous spaces: The `IntervalUnionRestriction` class represents a union of closed allowed intervals, and `BucketSpaceRestriction` represents a Boolean vector of equally sized allowed and forbidden buckets. For multi-dimensional spaces, these classes can be combined as long as the dimensions are independent, such as in `gym.Box` spaces. For more complex dependencies between dimensions (e.g., the "circle restriction" $A = [0, 1]^2$, $R = \{a \in A : ||a||_2 \leq 1\}$), we offer the generic `PredicateRestriction` class, which, however, is not flattenable.

4.2. Restrictor

The `Restrictor` class is designed as a regular agent but with `Restrictions` as actions. Consequently, the AEC can handle DRAMA natively without any special considerations for restrictors.

Observation Space. The observation structure of a restrictor is not predefined but can include any information available in the environment, such as the identifier of the next agent or the agents' latest rewards. In particular, it is not necessarily linked to the observation functions $\boldsymbol{\sigma}$ of the agents. Optionally, a custom preprocessing function can be applied before calling `Restrictor.act()` (see Section 4.3).

Action Space. Usually, the action space of a restrictor comprises all possible restrictions of a given agent action space A (for a more flexible mapping, see "Action Post-Processing"). To represent this space, which is mathematically equivalent to the power set of A, we provide the base class `RestrictorActionSpace`. Initialized with the base space A, it enables the restrictor to generate any `Restriction` compatible with A.

Reward Function. The restrictor's reward function can be constructed using any information available in the environment. By default, we use the *social welfare* $r = \sum_{i \in I} r_i$, which sums over all agent rewards, but user-defined functions are supported as an optional parameter when defining the restrictor.

4.3. Restriction Wrapper

The `RestrictionWrapper` is responsible for managing the interaction between the environment, agents, and restrictor(s). Prior to querying an agent, the wrapper requests a restriction from the corresponding restrictor and then passes this restriction to the agent.

Agent-Restrictor Mapping. In the simplest case of DRAMA, a single restrictor is utilized for all agents. However, multiple restrictors can be defined to accommodate, for instance, agents with different action spaces. By establishing a mapping between the set of agents and the set of restrictors, the wrapper can obtain the appropriate restrictions for each agent from the corresponding restrictor.

Observation Pre-Processing. The default observation for a restrictor is set as `env.state()`. Optionally, a pre-processing function can be specified for each restrictor, which the wrapper applies in analogy to the agent observation functions $\boldsymbol{\sigma}$.

Action Post-Processing. In situations where a restrictor functions as a learning agent, its restriction space might not align with the action space of the other agents. For instance, if the restrictor selects from a predetermined set of restrictions, it is advisable to define its action space as `Discrete`, and subsequently map the chosen action to a corresponding restriction. To accommodate these scenarios, a restrictor-generated restriction can undergo post-processing before being provided to the respective agent. This allows for seamless integration and compatibility between the restrictor's actions and the agent's expected inputs.

Agent Observations. The observation received from the environment is passed to the agents as part of a two-key dictionary: {`"observation": ..., "restriction": ...`}. The keys of this dictionary can be customized. For example, using `DiscreteVectorRestriction` in conjunction with the restriction key `action_mask` ensures seamless compatibility with Tianshou's built-in agents.

By default, the wrapper flattens all observation and action spaces, including restrictions, into fixed-shape `gym.Box` spaces (with possible padding or overflow) to ensure compatibility with existing libraries. To anticipate the emergence of algorithms in the future that can natively handle a wider range of spaces (i.e., any class adhering to the `gym.Space` specification), we offer three more options: (a) flattening into variable-shape `gym.Sequence` spaces, (b) applying a custom flattening function, and (c) using the original spaces without flattening.

Restriction Violations. The consequences of violating a restriction can be arbitrary and may be individual per agent. By default, an invalid

action causes the wrapper to throw a custom `RestrictionViolationException`. An invalid action can also be replaced by sampling uniformly from the allowed set or projected to the nearest action, and custom methods can be added by specifying a function for each restrictor.

5. Use Cases

In this section, we provide several illustrations of how DRAMA can be applied to a variety of multi-agent scenarios. It is important to note that these use cases are intentionally designed to be simple. The primary emphasis is placed on exploring the interaction and learning dynamics between the agents and restrictor(s) facilitated by the restriction wrapper.

5.1. Learning optimal restrictions in a continuous-action game

To demonstrate learned restrictions in a continuous action space, let's consider the Parameterized Cournot Game as defined in Oesterle and Sharon, 2023. In this game, two players choose their production quantities $\boldsymbol{q} = (q_1, q_2) \in \mathbb{R}^2$ for a good, with the price of the good defined as $p(\boldsymbol{q}) = \max(p_{max} - q_1 - q_2, 0)$, where $p_{max} > 0$. Both players have a constant production cost of $c \geq 0$ per unit. The players' utilities, representing their profits, are given by $u_i(\boldsymbol{q}) = q_i \cdot (p(\boldsymbol{q}) - c)$. Importantly, the equilibrium strategy in this game is not socially optimal, meaning that restricting the action space can potentially increase social welfare.

We aim to learn an optimal restriction by observing the actions of the agents. The restrictor waits until the agents' strategies converge and then formulates an optimal restriction based on its estimation of the game's parameters, namely p_{max} and c. In response to the restriction, the agents adjust their actions, resulting in a stable outcome that differs from the original equilibrium. The social welfare, as depicted in Figure 4, exhibits a noticeable increase at the point where the restrictor comes into action. While this learning behavior may be simplistic, it effectively demonstrates the dynamic interaction between the restrictor and the agents.

5.2. Training RL agents with dynamic restrictions in an obstacle avoidance scenario

Using DRAMA, we train an RL agent to navigate a dynamic environment with complex action spaces. In this scenario, restrictions are not necessarily tied to the agent's observation but serve as an additional source

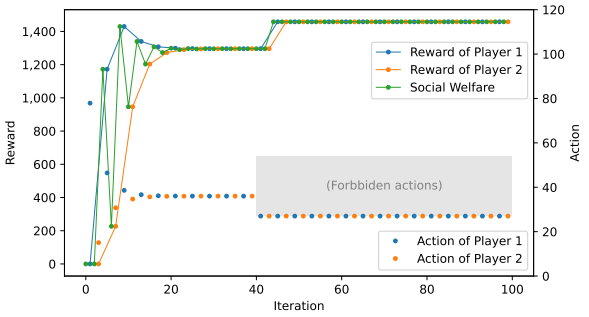
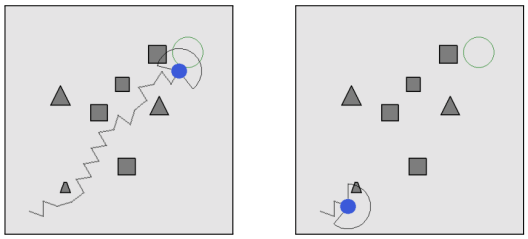


Figure 4. Reward and action curves for the Parameterized Cournot Game with a learning restrictor. At iteration 40, when the restriction is defined, the reward of both agents undergoes an abrupt improvement, showing that the restriction is boosting social welfare. This plot explicitly shows the alternation between restrictor and agent actions.



(a) Projection **(b) Unrestricted**

Figure 5. Examplary agent trajectories for the navigation task. While projection allows surrounding obstacles by bringing actions into the feasible set, an unrestricted agent is stuck with repeated forbidden movements.

of information. Consider a navigation task where an agent aims to reach a goal on a two-dimensional map. The environment can contain temporary obstacles, such as other agents or objects, that may not be directly sensed by the agent. An external entity can therefore suggest restrictions on the agent's action space to avoid collisions, such that the agent must select actions that maximize the expected return over varying subsets of the action space.

The environment, as shown in Figure 5, is a 15x15 field where the agent A (blue circle) has a location $l_t^A \in \mathbb{R}^2$, perspective $p_t^A \in [0, 360]$, and starting position $p_0^A = (2, 2)$. At each time step t, the agent observes l_t^A and p_t^A, as well as the distance and angle to the goal $g = (12, 12)$. It then chooses an angle $a_t^A \in [-110, 110]$ to determine the subsequent step's direction (with a step length of 1). The goal is reached when the distance is ≤ 1. Seven obstacles of various shapes, defined by their location and radius, are randomly generated at each episode's start and are not observed by the agent.

To handle the dynamic restrictions, we employ the `IntervalUnionRestriction` class to represent the union of open intervals that correspond to actions

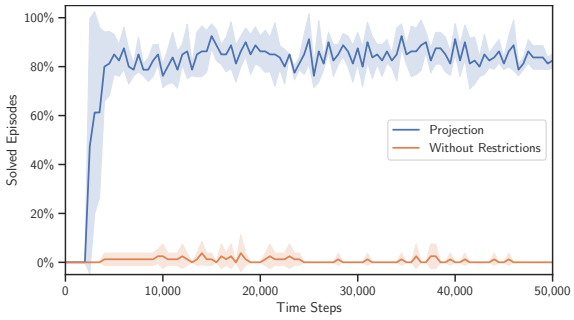


Figure 6. Mean and standard deviation for the fraction of solved evaluation episodes, measured every 500 steps.

leading to collisions. The valid action space is also computed based on these intervals. Since the number of intervals can vary, the boundaries cannot be treated as static model inputs. We train a *Twin Delayed DDPG* (TD3) algorithm (Fujimoto et al., 2018) to find the shortest path to the goal. We compare two cases: First, the agent learns without knowledge of restrictions and may collide with obstacles. Second, we provide dynamic restrictions to the agent using DRAMA, allowing the agent to choose feasible actions that are closest to its preferred actions.

The experiment demonstrates that DRAMA improves learning in scenarios with dynamic action spaces. The average fraction of evaluation environments where the agent succeeds significantly increases when handling restrictions, as depicted in Figure 6 for multiple model runs. With projection, most obstacles can be smoothly navigated, while unrestricted agents frequently encounter obstacles, as can be seen from the trajectories in Figures 5a and 5b. We note, however, that the average number of steps remains relatively high compared to the shortest path, even when restrictions are used and the success rate is high. This highlights the need for agents to make more informed decisions when dealing with variable action spaces. Notably, Grams, 2023 have recently explored RL architectures for dynamic restrictions and conducted experiments in this environment.

5.3. Training an RL restrictor for a discrete action space

Consider a traffic network where agents $i \in I$ are tasked with selecting a shortest route from their starting points s_i to their respective destinations d_i. The travel time along each road segment is influenced by its utilization, i.e., the relative number of agents using it. This is described by a latency model $l_e(u) = a + bu^c$, where individual parameters (a, b, c) are assigned to each edge (see Maerivoet & Moor, 2005).

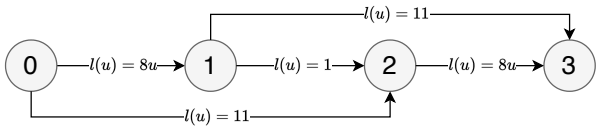


Figure 7. Traffic network with dynamic restrictions. The restrictor learns which roads (edges) to close in order to maximize throughput, measured as the negative mean of all agents' travel times.

In this context, a phenomenon known as the *Braess Paradox* (Braess, 1968) has been observed, where closing roads in the network can actually lead to an *increase in overall throughput* under specific conditions. Motivated by this paradox, we train a restrictor to determine the optimal configuration of open and closed roads. This restrictor operates within a network of self-interested agents who aim to minimize their individual travel times. The restrictor serves as a *governance* mechanism with the objective of minimizing the total travel time.

For the sake of simplicity, we utilize the graph network depicted in Figure 7. All simple paths in this network are enumerated and used as the discrete action space for the agents. At each step, each agent i selects the shortest route from s_i to d_i based on the current edge latencies. Without any governance in place, all agents traveling from $s = 0$ to $t = 3$ naturally choose the route $0 \to 1 \to 2 \to 3$, resulting in an average travel time of ≈ 17 (as shown by the red line in Figure 8).

To improve traffic flow, we introduce a governance mechanism that can selectively close individual roads (i.e., remove edges). The action space for the governance is represented by `MultiBinary(5)` (although, in our implementation, we ensure that there is always at least one open path available for agents to use). While the agents use a fixed strategy to determine the shortest routes given the restrictions, the governance, acting as an RL agent, learns the optimal set of restrictions by observing the agents and the environment. In our approach, we use an off-the-shelf DQN algorithm (Mnih et al., 2013), where the current edge latencies serve as the observation.

In our setting, the governance learns to close the edge $1 \to 2$ (as illustrated in Figure 9), leading the agents to distribute themselves across the routes $0 \to 1 \to 3$ and $0 \to 2 \to 3$, with each route having an approximate utilization of 50%. As a result, the average travel time decreases to ≈ 15 (indicated by the green line in Figure 8), which indeed represents the optimal configuration for the given network structure.

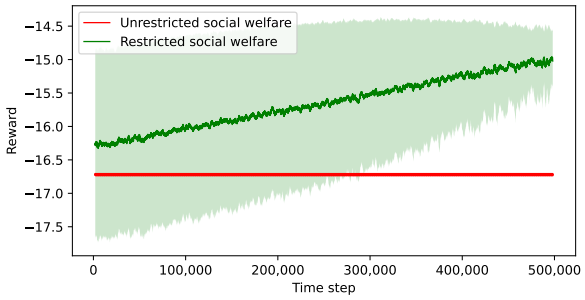


Figure 8. Social welfare (sliding average over 5,000 steps, mean and standard deviation over 5 runs) of traffic network during training (green), compared to unrestricted traffic (red).

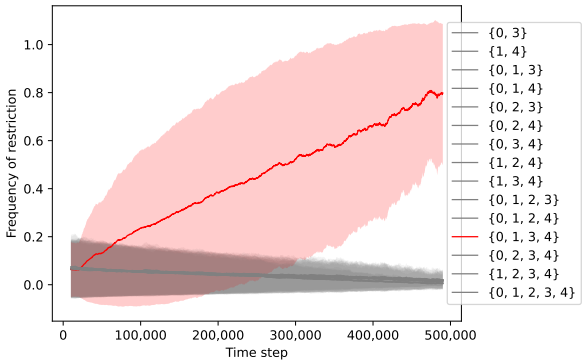


Figure 9. Frequency of the restrictions chosen by the restrictor during training (sliding average over 1,000 steps, mean and standard deviation over 5 runs). The dominant restriction $\{0, 1, 3, 4\}$ corresponds to closing the edge $(1, 2)$ in the network.

6. Conclusion

In this paper, we have extended the Agent Environment Cycle for MARL with a component which has thus far been handled in a limited, "hacky" way: complex dynamic action space restrictions.

Many practical scenarios involve nuanced action constraints, such as physical, legal, or safety considerations, which require intelligent agents to navigate through a complex decision-making space while adhering to restrictions. By explicitly integrating and modeling dynamic action space restrictions, we provide a more realistic and comprehensive framework for the development of intelligent agents (and self-learning restrictors!) capable of effectively operating within the confines of real-world constraints.

With this work, we also want to encourage the research and development of restriction-aware RL agents (and restriction classes) to pave the way for practical applications in domains where compliance with explicit rules and regulations is paramount.

Finally, we would like to propose an idea for governance based on human language: The governance of human communities primarily relies on the formulation and implementation of laws conveyed through natural language. These laws encompass explicit and implicit guidelines delineating permissible actions and behavioral constraints (i.e., action space restrictions), as well as information about penalties (i.e., reward shaping definitions) for breaking the restrictions. It therefore appears plausible that LLMs possess the capacity to learn and optimize text-based rules in alignment with the behavioral tendencies exhibited by the (human and/or artificial) agents subject to these rules. The recently published *ChatArena* library (Wu et al., 2023) offers a testbed for such environments, while Park et al., 2023 explore language-based *Generative Agents* which can exhibit emergent social behaviors, albeit without governance. Combining these approaches with DRAMA could bridge the gap between RL and language models, thereby facilitating the automated generation of intricate rule sets guided by specific objectives. Ultimately, this holds the potential to revolutionize the process of law creation for human and artificial communities.

References

Andrighetto, G., Governatori, G., Noriega, P., & van der Torre, L. (Eds.). (2013). *Normative Multi-Agent Systems*. Dagstuhl Follow-Ups.

Åström, K. (1965). Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, *10*(1).

Bellman, R. (1957). A markovian decision process. *Indiana Univ. Math. J.*, *6*.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., … Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. https://arxiv.org/abs/1912.06680

Boutilier, C., Cohen, A., Hassidim, A., Mansour, Y., Meshi, O., Mladenov, M., & Schuurmans, D. (2018). Planning and learning with stochastic action sets. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*.

Braess, D. (1968). Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, *12*(1).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Cai, Q., Filos-Ratsikas, A., Tang, P., & Zhang, Y. (2018). Reinforcement mechanism design for E-commerce. *Proceedings of the 2018 World Wide Web Conference*.

Caspi, I., Leibovich, G., Novik, G., & Endrawis, S. (2017). Reinforcement learning coach. https://doi.org/10.5281/zenodo.1134899

Chandak, Y., Theocharous, G., Nota, C., & Thomas, P. (2020). Lifelong learning with a changing action set. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*.

Chopra, A., van der Torre, L., & Verhagen, H. (Eds.). (2018). *Handbook of Normative Multiagent Systems*. College Publications.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, *13*(1).

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., & Coppin, B. (2016). Deep reinforcement learning in large discrete action spaces.

Fan, Z., Su, R., Zhang, W., & Yu, Y. (2019). Hybrid actor-critic reinforcement learning in parameterized action space. *International Joint Conference on Artificial Intelligence*.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*.

Grams, T. (2023). Dynamic interval restrictions on action spaces in deep reinforcement learning for obstacle avoidance. *Master's Thesis*. https://arxiv.org/abs/2306.08008

Hausknecht, M., & Stone, P. (2016). Deep reinforcement learning in parameterized action space. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., & Wu, Y. (2018). Stable baselines.

Hoffman, M. W., Shahriari, B., Aslanides, J., Barth-Maron, G., Momchev, N., Sinopalnikov, D., Stańczyk, P., Ramos, S., Raichuk, A., Vincent, D., Hussenot, L., Dadashi, R., Dulac-Arnold, G., Orsini, M., Jacq, A., Ferret, J., Vieillard, N., Ghasemipour, S. K. S., Girgin, S., … de Freitas, N. (2020). Acme: A research framework for distributed reinforcement learning. https://arxiv.org/abs/2006.00979

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & Araújo, J. G. (2022). CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, *23*(274).

Huang, S., & Ontañón, S. (2022). A closer look at invalid action masking in policy gradient algorithms. *The International FLAIRS Conference Proceedings*, *35*.

Kanervisto, A., Scheller, C., & Hautamäki, V. (2020). *Action Space Shaping in Deep Reinforcement Learning*. IEEE.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. In S. Solla, T. Leen, & K. Müller (Eds.), *Advances in neural information processing systems*. MIT Press.

Krasowski, H., Thumm, J., Müller, M., Schäfer, L., Wang, X., & Althoff, M. (2023). Provably safe reinforcement learning: A theoretical and experimental comparison.

Kuhnle, A., Schaarschmidt, M., & Fricke, K. (2017). *Tensorforce: A tensorflow library for applied reinforcement learning*. https://github.com/tensorforce/tensorforce

Laszuk, D. (2020). AI traineree: Reinforcement learning toolset.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M. I., & Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning. In J. G. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning, ICML 2018*. PMLR.

Maerivoet, S., & Moor, B. D. (2005). Transportation planning and traffic flow models.

Mandel, T., Liu, Y.-E., Brunskill, E., & Popović, Z. (2017). Where to add actions in human-in-the-loop reinforcement learning. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, J. T., Hafner, R., Romano, F., Buchli, J., Heess, N., & Riedmiller, M. A. (2020). Continuous-discrete reinforcement learning for hybrid control

in robotics. *CoRR*, *abs/2001.00449*. http : //arxiv.org/abs/2001.00449

Nisan, N., & Ronen, A. (2004). Computationally Feasible VCG Mechanisms. *Journal of Artificial Intelligence Research*, *29*.

Nota, C. (2020). The autonomous learning library.

Oesterle, M., Bartelt, C., Lüdtke, S., & Stuckenschmidt, H. (2022). Self-learning governance of black-box multi-agent systems. In N. Ajmeri, A. Morris-Martin, & B. T. R. Savarimuthu (Eds.), *Coordination, organizations, institutions, norms, and ethics for governance of multi-agent systems XV - international workshop, COINE 2022, revised selected papers*. Springer.

Oesterle, M., & Sharon, G. (2023). Socially optimal non-discriminatory restrictions for continuous-action games. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(10), 11638–11646. https : //doi.org/10.1609/aaai.v37i10.26375

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior.

Pernpeintner, M., Bartelt, C., & Stuckenschmidt, H. (2021). Governing black-box agents in competitive multi-agent systems. In A. Rosenfeld & N. Talmon (Eds.), *Multi-agent systems - 18th european conference, EUMAS 2021, revised selected papers*. Springer.

Plappert, M. (2016). *keras-rl*. https://github.com/keras-rl/keras-rl

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268).

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C.-M., Torr, P. H. S., Foerster, J., & Whiteson, S. (2019). The StarCraft Multi-Agent Challenge. *CoRR*, *abs/1902.04043*.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the National Academy of Sciences*, *39*.

Sinclair, S., Wang, T., Jain, G., Banerjee, S., & Yu, C. (2020). Adaptive discretization for model-based reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. A Bradford Book.

Tang, P. (2017). Reinforcement mechanism design. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*.

Tang, Y., & Agrawal, S. (2020). Discretizing continuous action space for on-policy optimization.

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., et al. (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, *34*.

Terry, J., Black, B., & Hari, A. (2020). Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*.

Uther, W. T. B., & Veloso, M. M. (1998). Tree based discretization for continuous state space reinforcement learning. *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., … Tsing, R. (2017). StarCraft II: A new challenge for reinforcement learning. https://arxiv.org/abs/1708.04782

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, Y., Su, H., & Zhu, J. (2022). Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, *23*(267).

Wu, Y., Jiang, Z., Khan, A., Fu, Y., Ruis, L., Grefenstette, E., & Rocktäschel, T. (2023). *Chatarena: Multi-agent language game environments for large language models*. GitHub.

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., & Mannor, S. (2018). Learn what not to learn: Action elimination with deep reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems*.