Master thesis on Intelligent Interactive Systems

Universitat Pompeu Fabra

Policy gradient methods on a multi-agent game with Kullback-Leibler costs

Borja Requena Pozo

Supervisor: Vicenç Gómez

Co-Supervisor: Martí Sanchez Fibla

July 2017

Master thesis on Intelligent Interactive Systems

Universitat Pompeu Fabra

Policy gradient methods on a multi-agent game with Kullback-Leibler costs

Borja Requena Pozo

Supervisor: Vicenç Gómez

Co-Supervisor: Martí Sanchez Fibla

July 2017

Contents

Dedication

I'd like to dedicate this work to my brother, who has gone through a rather tough year, but we have both just finished a major stage of our lives.

Acknowledgement

I would like to express my gratitude to my thesis supervisors for being so open to work propositions and help me find a tailored thesis that fit my interests.

I would also like to thank all people who have had to bear with me working during what were supposed to be summer holidays and lead with my workaholic tendencies.

Special thanks to Bernat, with whom I have been working for quite some time already and it is always a pleasure to discuss about our respective projects. I hope we keep working together in the future!

Abstract

In nature we find all kinds of multi-agent systems sustained upon cooperative behaviours. In this work, we study multi-agent systems by means of the Stag-Hunt game, which presents a conflict between mutual benefit and personal risk. In particular, we consider the probabilistic inference approach for reinforcement learning on a grid-based variant of this game. We analyze the behavior of two different policy gradient algorithms in the presence of function approximation: the standard REINFORCE algorithm and the Cross-Entropy (CE) method, which differ on the functional form of the loss. However, even though both REINFORCE and CE share the same global optimal solution, we have found that REINFORCE behaves too greedily compared with CE. In agreement with previous results based on probabilistic graphical models, we obtain two different qualitative optimal solutions (risk- and payoff-dominant) as a function of a temperature parameter, whose transition is better observed using the CE method. We also analyze the difference between using or not path-cost, in addition to the end-cost. It is known that adding path-cost makes the problem harder using an explicit probabilistic graphical model, since it increases its tree-width. Nevertheless, we observe the opposite effect for policy gradient methods, for which path-cost enhances the performance of the resulting controls in all circumstances. This is explained because the samples used by policy gradients are generally more informed with path-cost. Finally, we also consider a distributed version of the algorithm, with partial observability and feature sharing between the agents. In this setting, we show the feasibility of generalizing to larger grids using training data from smaller grids.

Keywords: Reinforcement learning; Policy gradient; Kullback-Leibler control; Stag-Hunt game; Multi-agent systems

Chapter 1

Introduction

With this thesis we aim to obtain better understanding of multi-agent systems. In nature, we find all kinds of different societies constructed upon the interaction of individuals and sustained on the emergence of cooperative behaviours.

Cooperation is the process of working together for a mutually beneficial end. Thing that allows the achievement of greater deeds or goods that could never be obtained by the actions of an individual. We find all kinds of examples around us and in ourselves resulting into phenomena such as *coevolution*, symbiotic assemblies, the association into flocks, schools or swarms, etc.

For example, an alga and a fungus work together as a lichen to survive in environments that would be lethal for both species separately. Killer whales display all kinds of smart cooperative hunting strategies to safely catch otherwise impossible preys [1]. Fire ants, apart from foraging, nest construction and food cultivation, are known for the capability of cooperating to construct self-assemblages such as ladders, chains, walls and rafts that allow them to survive potentially lethal floods for any individual [2].

In the case of human society, cooperation has lead to the emergence of social conventions and roles. Conventions are self-sustaining, in the sense that we will continue to conform to them as long as we expect others to, and, also, arbitrary, in the

sense that, at least, one alternative regularity exists and would be equally acceptable as long as everyone coordinated on it [3]. This is the case of traffic regulations, currencies, the way we address to other people, etc [4, 5, 6, 7].

The study of multi-agent systems is usually approached by the use of games that serve as a model for different decision-making situations, such as the *Prisoner's dilemma* [8, 9] or the *Battle of the Exes*, which is a variation of the old *Battle of the Sexes* [10, 11]. Although other approaches are also used, like the case of a multi-agent system managing a common resource-pool [12], which allows to see how differences between individuals play a key role in the emergence of inequality, roles and exclusion. In our case, however, we focus on the *Stag-Hunt* game [13].

The *Stag-Hunt* game is a cooperative game in which a set of hunters can choose between hunting a hare or a stag. Hunters can catch a hare by themselves and receive a small reward R_h, but it requires, at least, the coordination of two hunters in order to get a stag, which provides them with a much larger reward R_s. Hence, if only one hunter decides to chase the stag, it will not be able to get it, thus receiving a null reward.

In this thesis, we use the framework of optimal control as probabilistic graphical model inference [14, 15, 16, 17], for which finding the optimal strategy corresponds to computing the posterior probability over agent trajectories that follow the optimally controlled dynamics. We consider the formulation introduced in [14], which known as Kullback-Leibler (KL-) control. KL-control problems, also known as linearly-solvable MDPs [18], enjoy several computational advantages, such as a linear Bellman equation [19, 20], compositionality of optimal control laws [21, 22], or fast learning rates [23]. The continuous-time version of these problems is known as Path Integral control [20], and has been used for multi-agent systems in [24, 25, 26].

We follow the version of the game introduced in [14], which is played in a grid with a fixed position for the preys: stags and hares, and an initial position for the hunters, who can freely roam around it for a fixed amount of time T. Then, the result of the game is computed depending on the hunter positions at the end time.

	Stag	**Hare**
Stag	3,3	0,1
Hare	1,0	1,1

Table 1: Payoff matrix of a 2-player *Stag-Hunt* game with $R_s = 3$ and $R_h = 1$.

The nature of the problem presents a conflict between personal risk and mutual benefit, provided that successful coordination to hunt the stag awards great benefits for the implicated players, but a synchronization failure can lead to null profit for all of them. This allows for the study of cooperation within social structures [27] and collaborative behavoiour in multi-agent systems [28] and can also be used to understand cooperativity directly in humans [29, 30].

For the case of a 2-player game, the payoff matrix, represented in Table 1, shows that the game has two *Nash equilibria* corresponding to both players going either for stags (*payoff* equilibrium) or hares (*risk-dominant* equilibrium). In game theory, *Nash equilibrium* is a solution for which no single player has anything to gain by changing *only* their own strategy [31].

Therefore, there are two qualitatively different stable equilibria that can both be recovered as optimal solutions as a temperature parameter λ varies [14]. Then, for low temperature ($\downarrow \lambda$), the optimal solution is to coordinate hunters in pairs to get all the stags. In contrast, for high temperature ($\uparrow \lambda$), the optimal solution becomes hunting hares individually.

Such problem is formulated in [14] as a stochastic control problem for which the computation of the global optimal strategy is equivalent to a Kullback-Leibler (KL) minimization problem. This optimization is addressed using variational methods for approximate inference in probabilistic graphical models (PGMs).

This reinforcement learning (RL) formulation as inference is often referred to as *maximum entropy RL*, *maximum entropy control* or *KL-control*, due to the fact that minimizing the KL-divergence (control cost) corresponds to maximizing both the expected reward *and* the expected conditional entropy, in contrast to the standard control objective that only cares about reward. The temperature parameter λ serves

as natural mechanism to interpolate between entropy maximization and standard simple cost optimization, provided that, as $\lambda \to 0$, the optimal solution approaches the standard RL optimal control solution [32].

Nevertheless, performing exact inference can be too costly for complex instances and approximate inference methods are needed to obtain approximate optimal controls [33, 34, 35]. The approach in [14] makes explicit use of the graphical model structure and takes advantage of certain conditional independences in the agents dynamics. However, it requires to *unfold* (ground) the entire sequence of state variables before performing inference. Although this grounding benefits from the explicit conditional independencies in the graphical structure, it still has some limitations for large-scale problems, or for problems with path-cost (a state cost term at each time-step). Such cost terms increase the tree-width of the PGM significantly, and thus the complexity of the inference task. In addition, this centralized approach makes it difficult for modeling a distributed system in which agents have partial observations of the environment.

Contrary to performing approximate inference on a PGM, as done in [36, 14], here we consider policy gradient methods, which have proven effective for controlling complex systems [37, 38]. In contrast to PGMs, policy gradients iteratively use trajectory samples and do not need to ground the state variables prior to the inference. In addition, they can naturally make use of path-cost. This rises the question of whether a sample-based method is able to recover the optimal solution transition observed in the *Stag-Hunt* game [14] and what are the implications of considering path-cost.

As a result, we obtain two methods consisting on a greedy cost optimization method and the Cross-Entropy method, which we want to compare in the different frameworks presented along the thesis.

Cooperation appears naturally as optimal solution on a centralized control with full state observability [14]. However, these kind of implementations are not viable in most real-world applications. Thereby, we also implement KL-control for

individual agents with partial observability who share the same policy in order to see whether cooperative behaviours can emerge in more realistic scenario. Besides, the distributed approach allows for better scalability in the sense of increasing the number of agents it can handle.

Finally, the partial observability of the distributed control might provide strategies that are less dependent of the specific game scenario. This has the potential to learn individual strategies in one instance of the *Stag-Hunt* game that can be applied to solve other different instances of the same game with different amount and distribution of preys and hunters. Thus, we wonder whether this approach can generalize.

Up to this point, we have laid down the research questions that we aim to answer with this thesis. Let us gather them up for the sake of clarity:

- Can we recover the sharp transition between the two regimes of the *Stag-Hunt* game with a sample-based method?
- What is the effect of formulating the problem with and without path cost?
- What are the differences between a greedy cost optimization method (such as REINFORCE) and the Cross-Entropy method?
- Do we obtain cooperation with individual agents?
- Is there a way to scale and generalize the control?

The rest of the thesis is structured as follows:

In the following chapter (2) we first introduce KL-control problems explaining what they are and how to solve them, followed by a derivation of the methods in question.

The next chapter (3) presents an in-depth thorough explanation of the implementation into the problem and the procedures behind the results, which are shown in the following chapter afterwards (4) and serve as empirical proof to answer the research questions recently introduced.

The last chapter (5) contains the discussion derived from the results in the previous chapter trying to answer to the different questions. Finally, summarizes it all up concisely presenting the obtained conclusions and possible future work.

Chapter 2

Kullback-Leibler control and policy gradient

We first introduce the class of KL-control problems and then present two alternative ways for finding a parametrized policy based on sample trajectories.

2.1 KL-control problems

We consider a Markov decision process with discrete state variable x and $\tau = x_{1:T}$ a trajectory through state space from initial time 1 to end-time T. We will assume that our controller can fully determine the transition probabilities $p(x'|x)$ as long as they are consistent with the so-called passive dynamics $q(x'|x)$, that characterizes the behavior of the system in the absence of controls [1]. The probability distributions of a trajectory τ that follows the passive and the controlled dynamics are, respectively:

$$q(\tau|x_0) = \prod_{t=0}^{T-1} q(x_{t+1}|x_t), \qquad p(\tau|x_0) = \prod_{t=0}^{T-1} p(x_{t+1}|x_t). \tag{2.1}$$

We now define $R(\tau) = \sum_{t=1}^{T} R(x_t)$ to be an arbitrary state-dependent cost incurred when following trajectory τ.

[1] This means that $p(x'|x) = 0$ whenever $q(x'|x) = 0$.

The KL-control problem is to find the probability distribution $p(\tau|x_0)$ that minimizes the total expected cost

$$\mathcal{C}(p|x_0) = \sum_{\tau} p(\tau|x_0) \left(\lambda \log \frac{p(\tau|x_0)}{q(\tau|x_0)} + R(\tau) \right) \tag{2.2}$$

$$= \lambda D_{\mathrm{KL}}(p \parallel q) + \langle R(\tau) \rangle_p. \tag{2.3}$$

The $D_{\mathrm{KL}}(p \parallel q)$ term represents a *control* cost, with larger costs for transitions that deviate more from the passive dynamics. The second term represents the expected *state* cost, with respect to (w.r.t.) the controlled probability p. The parameter λ controls the relative strength between both cost terms.

The optimal solution for p is found by minimizing (2.2) w.r.t. p. The solution and the optimal cost are given by [37]

$$p^*(\tau|x_0) = \frac{1}{Z(x_0)} q(\tau|x_0) e^{-\frac{1}{\lambda} R(\tau)}, \tag{2.4}$$

$$\mathcal{C}^*(p|x_0) = \mathcal{C}(p^*|x_0) = -\lambda \log Z(x_0). \tag{2.5}$$

The optimal control for the first step, $p^*(x_1|x_0)$, is given by the marginal distribution

$$p^*(x_1|x_0) = \sum_{x_{2:T}} p^*(\tau|x_0), \tag{2.6}$$

and can be computed exactly by backwards message passing or approximately using sample trajectories from the uncontrolled dynamics $q(\tau|x_0)$.

In practice, for high dimensional state spaces, a parametrized controller is used $p_\theta(\cdot)$ to represent the controlled process. If p_θ is sufficiently flexible, e.g., a neural network, one can approximate p^* with arbitrary accuracy, i.e., $p_{\theta^*} \approx p^*$.

Two possible ways to find p_θ are described next. Both methods are iterative and require the gradients $\nabla_\theta \log p_\theta(\tau|x_0)$.

2.2 Policy gradient methods

We describe two ways for learning the parameters θ of the controlled process p_θ, both based on gradient descent.

One possible way to learn p_θ is to directly minimize the expected cost (2.2) using stochastic gradient descent, as it is done by the REINFORCE algorithm [39]. Define $V_\theta(\tau|x_0) := \left(\lambda \log \frac{p_\theta(\tau|x_0)}{q(\tau|x_0)} + R(\tau)\right)$ as the total cost of a random trajectory τ starting at state x_0.

$$\begin{aligned}
\nabla_\theta \mathcal{C}(p_\theta|x_0) &= \nabla_\theta \left\langle V_\theta(\tau|x_0)\right\rangle_{p_\theta} && (2.7)\\
&= \nabla_\theta \sum_\tau p_\theta(\tau|x_0) V_\theta(\tau|x_0) && (2.8)\\
&= \sum_\tau \left(\nabla_\theta p_\theta(\tau|x_0)\right) V_\theta(\tau|x_0) + \sum_\tau p_\theta(\tau|x_0)\nabla_\theta \lambda \log p_\theta(\tau|x_0) && (2.9)\\
&= \sum_\tau \left(p_\theta(\tau|x_0)\nabla_\theta \log p_\theta(\tau|x_0)\right) V_\theta(\tau|x_0) + \lambda \nabla_\theta \sum_\tau p_\theta(\tau|x_0) && (2.10)\\
&= \sum_\tau \left(p_\theta(\tau|x_0)\nabla_\theta \log p_\theta(\tau|x_0)\right) V_\theta(\tau|x_0) && (2.11)\\
&= \left\langle V_\theta(\tau|x_0)\nabla_\theta \log p_\theta(\tau|x_0)\right\rangle_{p_\theta} && (2.12)
\end{aligned}$$

This algorithm starts with an initial guess for the parameters $\theta^{(0)}$ and iterates the following steps until convergence

1. Sample a batch of N trajectories $\{\tau^{(i)}\}, i = 1, \ldots, N$ from p_θ.
2. Compute their costs $V_\theta^{(i)} = \lambda \log \frac{p_\theta(\tau^{(i)}|x_0)}{q(\tau^{(i)}|x_0)} + R(\tau^{(i)})$.
3. Normalize $\omega_\theta^{(i)} = \frac{1}{Z} V_\theta^{(i)}, Z = \sum_i V_\theta^{(i)}$.
4. Compute $\nabla_\theta \mathcal{C}(p_\theta|x_0) = \sum_i \omega_\theta^{(i)} \nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$.
5. Update $\theta^{(i+1)} \leftarrow \theta^{(i)} - \alpha \nabla_\theta \mathcal{C}(p_\theta|x_0)$, for learning rate α.

Another possible way to learn p_θ is through the Cross-Entropy (CE) method, which does not directly optimize the expected cost, but uses the explicit solution (2.4), as

in [37]. The CE method for KL-control was introduced in [40] and minimizes the following objective [2]

$$D_{\mathrm{KL}}(p^* \parallel p_\theta) = \sum_\tau p^*(\tau|x_0) \log \frac{p^*(\tau|x_0)}{p_\theta(\tau|x_0)} = \left\langle \log \frac{p^*(\tau|x_0)}{p_\theta(\tau|x_0)} \right\rangle_{p^*} . \tag{2.13}$$

The gradient of (2.13) is $-\left\langle \nabla_\theta \log p_\theta(\tau|x_0) \right\rangle_{p^*}$. One can obtain estimates from p^* by sampling from p_θ and using importance sampling corrections

$$p^*(\tau|x_0) = p_\theta(\tau|x_0)\omega_\theta(\tau|x_0) \tag{2.14}$$

$$\nabla_\theta D_{\mathrm{KL}}(p^* \parallel p_\theta) = -\left\langle \omega_\theta(\tau|x_0) \nabla_\theta \log p_\theta(\tau|x_0) \right\rangle_{p_\theta} , \tag{2.15}$$

with correction weights

$$\omega_\theta^{(i)}(\tau|x_0) \propto \frac{q(\tau^{(i)}|x_0)}{p_\theta(\tau^{(i)}|x_0)} e^{-\frac{1}{\lambda}R(\tau^{(i)})} = e^{-\frac{1}{\lambda}V_\theta^{(i)}} \tag{2.16}$$

Here we start with an initial guess for the parameters $\theta^{(0)}$ and iterate the following steps until convergence

1. Sample a batch of N trajectories $\{\tau^{(i)}\}, i = 1, \ldots, N$ from p_θ.
2. Compute normalized importance weights $\omega_\theta^{(i)} = \frac{1}{Z} e^{-\frac{1}{\lambda}V_\theta^{(i)}}, Z = \sum_i e^{-\frac{1}{\lambda}V_\theta^{(i)}}$.
3. Compute $\nabla_\theta D_{\mathrm{KL}}(p^* \parallel p_\theta) = -\sum_i \omega_\theta^{(i)} \nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$.
4. Update $\theta^{(i+1)} \leftarrow \theta^{(i)} - \alpha \nabla_\theta D_{\mathrm{KL}}(p^* \parallel p_\theta)$, for learning rate α.

Despite optimizing a different objective, the solution p_{θ^*} found by both greedy optimization and Cross-Entropy method is the same if a global minimum is reached and the parametrization p_{θ^*} can express the optimal distribution p^*. Note that the only difference is that the sample trajectories are exponentially weighted using Cross-Entropy. The preferred method depends on the application.

[2] The greedy cost optimization actually corresponds to optimize the reversed KL divergence, i.e., $D_{\mathrm{KL}}(p_\theta \parallel p^*)$.

The methods are also known as policy gradient on the I-projection and M-projection respectively.

Chapter 3

Application: The KL-stag-hunt problem

In this work, we consider two major formulations of the problem: a centralized and a distributed version. In the centralized setting, the policy acts on the joint state that considers all M agents $p_\theta(\mathbf{x})$, where x_i is the state of agent i, with $i = 1, \ldots, M$. In contrast, in the distributed setting we have M shared policies that act on each individual agent $p_\theta(x_i)$.

For both approaches, we apply the two learning algorithms derived in the previous chapter corresponding to policy gradients on the I-Projection and M-Projection. However, the implementation is not straightforward. In this chapter, we present the missing elements required to make the algorithms practical in our particular scenario.

The Stag-Hunt game, as presented in [14], is played for a finite time T and, at each time-step, the hunters perform an action. Then, the payoff (or state-cost) of the game is computed according to the positions of the hunters at final time T after the last action, as introduced in Chapter 1.

In this work, we consider two variants of the game. In the first one (finite-horizon formulation), the state cost is applied only at end-time, i.e., $R(x_t) = 0, \forall t < T$, as

in [14]. Whereas, in the second one, the state cost is applied at each time-step, thus considering a state path cost.

Initially, we analyze the scenario in which the position of the preys is fixed. Thus, a slight change in their configuration would lead to a completely different problem.

A prey is considered hunted when it shares position with, at least, the minimum amount of hunters (1 for hares, 2 for stags) required for hunting it. Besides, there is no competition for the preys, so every hunter on a hunted prey will receive its reward. Stags provide a reward $R_s = 3$ larger than the hares $R_h = 1$. Nevertheless, the state cost $R(x)$ counts the rewards as negative contributions, i.e., hunting a prey will reduce the cost function.

Depending on the state, each individual agent can have up to five available actions corresponding to staying still and moving one step in the four directions of the grid: up, down, left, right. However, the number of applicable actions of the agent may be reduced down to four or three in case that it is found at an edge of the grid or a corner. The probability of taking each of the actions is represented by a soft-max policy $p_\theta(x'|x)$ defined over state-transition features

$$p_\theta(x'|x) = \frac{e^{\theta^\top \phi(x',x)}}{\sum_{x'} e^{\theta^\top \phi(x',x)}}, \tag{3.1}$$

where θ are the policy parameters and $\phi(x', x)$ are features that encode both current and next state.

The feature representation of the states is different depending on the approach, centralized or distributed, so it is explained in their respective sections 3.1.1 & 3.2.1.

As described previously, the policy is learned by policy gradient on both the I-Projection and M-Projection. In order to do so, let us take the equations (2.12) and (2.15) corresponding to the gradient of the function to be optimized for each of the methods respectively.

We will empirically estimate the expectations over the distribution $p_\theta(\tau|x_0)$ by sampling a batch of N trajectories $\tau^{(i)}, i = 1, \ldots, N$ drawn from the policy p_θ. Starting

with the I-projection case, we obtain

$$\nabla_\theta \mathcal{C}(p_\theta|x_0) = \lambda \nabla_\theta D_{\mathrm{KL}}(p_\theta \parallel p^*) \tag{3.2}$$

$$= \langle V_\theta(\tau|x_0) \nabla_\theta \log p_\theta(\tau|x_0) \rangle_{p_\theta} \tag{3.3}$$

$$\approx \frac{1}{Z} \sum_{i=1}^{N} V_\theta(\tau^{(i)}|x_0) \nabla_\theta \log p_\theta(\tau^{(i)}|x_0), \; Z = \sum_{i=1}^{N} V_\theta(\tau^{(i)}|x_0) \tag{3.4}$$

Analogously, for the M-projection case, we can write

$$\nabla_\theta D_{\mathrm{KL}}(p^* \parallel p_\theta) = -\langle \omega_\theta(\tau|x_0) \nabla_\theta \log p_\theta(\tau|x_0) \rangle_{p_\theta} \tag{3.5}$$

$$\approx -\frac{1}{Z} \sum_{i=1}^{N} \omega_\theta(\tau^{(i)}|x_0) \nabla_\theta \log p_\theta(\tau^{(i)}|x_0), \; Z = \sum_{i=1}^{N} \omega_\theta(\tau^{(i)}|x_0) \tag{3.6}$$

For the case of a soft-max policy, the gradient $\nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$ that appears in both algorithms, also known as the *score function*, becomes

$$\nabla_\theta \log p_\theta(\tau|x_0) = \nabla_\theta \log \prod_{t=0}^{T-1} \frac{e^{\theta^\top \phi(x_{t+1}, x_t)}}{\sum_{x_{t+1}} e^{\theta^\top \phi(x_{t+1}, x_t)}} \tag{3.7}$$

$$= \nabla_\theta \left[\sum_{t=0}^{T-1} \theta^\top \phi(x_{t+1}, x_t) - \log \sum_{x_{t+1}} e^{\theta^\top \phi(x_{t+1}, x_t)} \right] \tag{3.8}$$

$$= \sum_{t=0}^{T-1} \phi(x_{t+1}, x_t) - \frac{\nabla_\theta \sum_{x_{t+1}} e^{\theta^\top \phi(x_{t+1}, x_t)}}{\sum_{x_{t+1}} e^{\theta^\top \phi(x_{t+1}, x_t)}} \tag{3.9}$$

$$= \sum_{t=0}^{T-1} \phi(x_{t+1}, x_t) - \frac{\sum_{x_{t+1}} e^{\theta^\top \phi(x_{t+1}, x_t)} \phi(x_{t+1}, x_t)}{\sum_{x_{t+1}} e^{\theta^\top \phi(x_{t+1}, x_t)}} \tag{3.10}$$

$$= \sum_{t=0}^{T-1} \phi(x_{t+1}, x_t) - \langle \phi(x_{t+1}, x_t) \rangle_{p_\theta(x'|x)} \,, \tag{3.11}$$

which corresponds to the sum of the differences between the observed features and the expected features under policy p_θ along the trajectory.

Depending on the method, this gradient is multiplied by the corresponding weights, which are proportional to $V_\theta^{(i)}(\tau^{(i)}|x_0)$ for the I-projection (3.4) or $\omega_\theta^{(i)}(\tau^{(i)}|x_0)$ for the M-projection (3.6).

More precisely, for a given sample trajectory $\tau^{(i)} = x_{1:T}^{(i)}$, the return used in the greedy cost optimization (I-projection) is computed as

$$V_\theta^{(i)}(\tau^{(i)}|x_0) = \lambda \log \frac{p_\theta(\tau^{(i)}|x_0)}{q(\tau^{(i)}|x_0)} + R(\tau^{(i)}) \tag{3.12}$$

$$= \lambda \left[\log \prod_{t=0}^{T-1} p_\theta(x_{t+1}^{(i)}|x_t^{(i)}) - \log \prod_{t=0}^{T-1} q(x_{t+1}^{(i)}|x_t^{(i)}) \right] + R(\tau^{(i)}) \tag{3.13}$$

$$= \lambda \left[\sum_{t=0}^{T-1} \log p_\theta(x_{t+1}^{(i)}|x_t^{(i)}) - \log q(x_{t+1}^{(i)}|x_t^{(i)}) \right] + \sum_{t=1}^{T} R(x_t^{(i)}) \tag{3.14}$$

$$= \sum_{t=0}^{T-1} \lambda \log \frac{p_\theta(x_{t+1}^{(i)}|x_t^{(i)})}{q(x_{t+1}^{(i)}|x_t^{(i)})} + R(x_{t+1}^{(i)}), \tag{3.15}$$

then, the weight used in the Cross-Entropy (M-projection) is

$$\omega_\theta^{(i)}(\tau^{(i)}|x_0) = \exp\left(-\frac{1}{\lambda} V_\theta^{(i)}(\tau^{(i)}|x_0) \right) \tag{3.16}$$

and both need to be normalized before multiplying with the score $\nabla_\theta \log p_\theta$.

In the following sections there are explained the centralized and distributed approaches in depth detailing the state feature representation $\phi(x', x)$ and all the aspects of the procedure to compute θ with the help of pseudo-codes, which, in general, follow the following structure:

initialize θ

for learning iterations

  sample trajectories following $p_\theta \ \rightarrow V_\theta, \phi, \langle\phi\rangle$

  compute cost function gradient

  update θ with gradient descent

  check convergence $||\theta_k - \theta_{k-1}||$

3.1 Centralized control

The centralized controller specifies, at each time-step, a transition probability between two fully observable joint states. This naive approach is clearly not scalable to large systems, since the number of possible next states grows exponentially with the number of agents. However, we use it as a starting point to analyze the effect of the proposed algorithms.

For this case, we have chosen to solve the optimal control for a Stag-Hunt game of two hunters with four hares, one at each corner, and a stag at the center of the grid. It is a 5×5 grid and the hunters play for a total of $T = 4$ time-steps ($\tau = x_{1:4}$). Figure 1 shows the initial configuration of the game (x_0).

Provided that the rewards are $R_s = 3$ and $R_h = 1$ for stag and hares respectively, the possible outcomes are:

Reward 3: if both hunters land on the stag.

Reward 2: if both hunters land on hares.

Reward 1: if only one hunter lands on a hare.

Reward 0: if no hunter lands on any prey.

3.1.1 State representation

The full state contains the position of all hunters and preys within the grid. Since the position of the preys is fixed, given a state x_t at a time t, there are A^M possible future states x_{t+1}, where A is the amount of possible actions per agent (typically 5) and M is the amount of hunters. Thus, the state space scales exponentially with the number of players, thing that prevents this kind of control of being applied to situations in which several agents need to be considered.

We use a simpler representation of the state in which the identity of each agent is ignored.

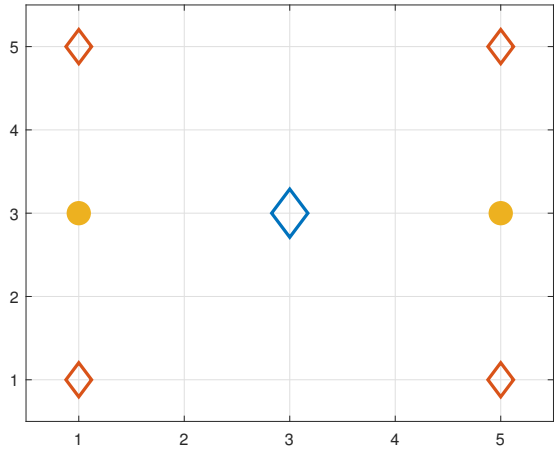


Figure 1: Stag-Hunt initial configuration. The hunters are represented by yellow dots, the hares are represented by red diamonds and the stag is represented with a blue diamond.

As mentioned previously, in Equation (3.1), the feature vector $\phi(x', x)$ encodes the transition between the current and the next state. We have defined it as the concatenation of the individual feature representation of the next state x' with the individual feature representation of the state x, denoted as

$$\phi(x', x) = \text{concat}\left(\phi(x'), \phi(x)\right) \tag{3.17}$$

With this, let us introduce the way we do the feature representation of a state $\phi(x)$. The feature vector of a state contains, on the one hand, boolean indicators of the preys that are considered hunted and, on the other hand, the contributions of the hunters to a set of continuous radial basis functions (RBFs) that span the entire grid.

Notice that the positions of the preys are not explicitly represented in the feature vector of the state because they are fixed. This way, the control learns where to put the hunters in order to receive the rewards as it always plays and trains in the same scenario (fixed x_0). Nonetheless, without the indicator features, the problem would be illposed, as we need to be able to differentiate exactly when the preys are hunted or not.

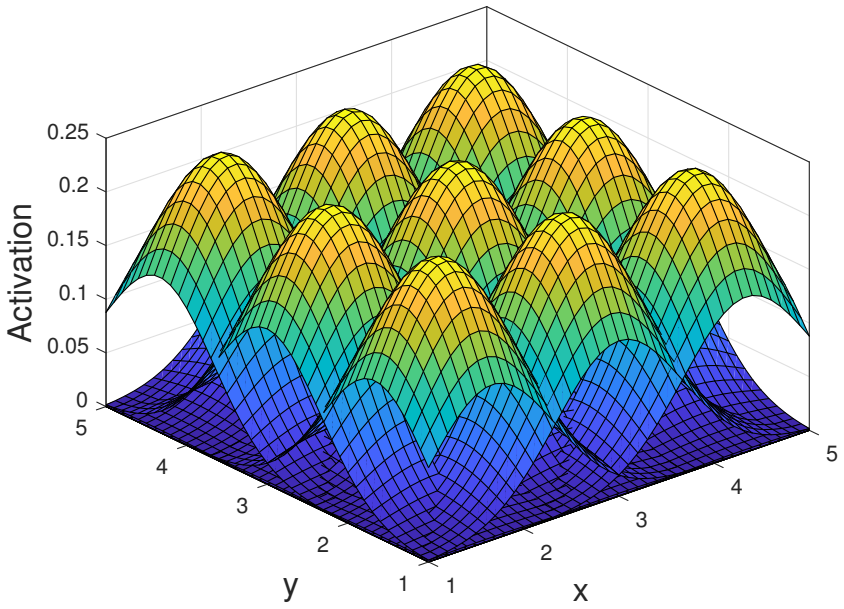


Figure 2: Radial basis functions for a 5×5 grid.

The number of radial basis functions and their parameters (centers and widths) are determined by the size of the grid. We choose $n = \max(\text{round}(\sqrt{d}), 3)$ equally spaced functions per dimension of length d separated by $dx = \frac{d-1}{n}$ along it. The standard deviation along the dimension is $\sigma = dx\frac{1+o}{6}$, where o is the overlap parameter. Figure 2 shows the resulting functions for the case that we are considering.

When computing the contributions of the hunters to the RBFs, instead of considering their separate contributions to the functions, we sum the activations of all hunters in each of the individual functions. This way, we have as many features as RBFs and the players are anonymized, provided that their identity is completely irrelevant.

The set of state features is the concatenation of the indicator features together with the basis functions. Hence, in the case that we are solving, Figure 1, the feature vector $\phi(x)$ that describes a state has length 14 corresponding to:

- 5 boolean indicators corresponding to the 4 hares and the stag.
- The contributions to the 9 RBFs.

Which means that the feature vector encoding the state transition $\phi(x', x)$ has length

28 and, therefore, so does θ. Notice that, with this state representation, the length of the feature vector does not depend on the amount of players.

With all the core aspects defined: x_0, $T = 4$, $R_s = 3$, $R_h = 1$, $\phi(x)$, $\phi(x', x)$ and $p_\theta(x', x)$, we can proceed to develop on the learning algorithm implementations.

3.1.2 Policy gradient on I-Projection

In order to better understand the algorithm, let us start by introducing a pseudo-code as reference that will be explained step by step afterwards.

initialize $\theta \sim \mathcal{N}(0, 2^2)$
define baseline $b = -\lambda \log(A^M)$
for its
  for rolls
    reset initial state x_0
    for $t = 0 : T - 1$
      $p_\theta \rightarrow x_{t+1},\ p_\theta(x_{t+1}|x_t),\ \phi(x_{t+1}, x_t),\ \langle\phi(x_{t+1}, x_t)\rangle$
      save return $V_\theta = \lambda \log \frac{p_\theta(x_{t+1}, x_t)}{q(x_{t+1}|x_t)} + R(x_{t+1}) + b$
      save score $\nabla_\theta \log p_\theta = \phi(x_{t+1}, x_t) - \langle\phi(x_{t+1}, x_t)\rangle$

  trajectory returns $V_\theta(\tau) = \text{sum}(V_\theta, t)$
  trajectory scores $\nabla_\theta \log p_\theta(\tau) = \text{sum}(\nabla_\theta \log p_\theta, t)$
  normalization factor $Z = \text{sum}(V_\theta(\tau))$
  $\nabla_\theta \mathcal{C}(p_\theta|x_0) = \text{sum}(\nabla_\theta \log p_\theta(\tau) V_\theta(\tau))/Z$
  $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{C}$

  check convergence $||\theta_k - \theta_{k-1}|| = ||\alpha \nabla_\theta \mathcal{C}|| < \delta$

The code computes the gradient of the cost function to be minimized following Equation (3.4) to update the parameter θ via gradient descent according to a learning rate $\alpha > 0$.

We start by randomly initializing the parameter vector $\theta \sim \mathcal{N}(0, 2^2)$. Then, we define a state-independent baseline function $b(x)$ to be added as state cost with the goal to ease the learning process avoiding possible numerical instabilities and providing better sample weights.

This function, overestimates the first term of the return V_θ corresponding to $\lambda \log \frac{p_\theta}{q}$ at each time step and subtracts it. The term is maximum for $p_\theta = 1$ and $q = 1/A^M$, being A the maximum possible number of individual actions and M the number of hunters ($A = 5,\ M = 2$ for our case). Hence, $b = -\lambda \log(A^M)$.

The baseline state cost b plays a key role in the normalization of the returns by Z. The possibly mixed sign of Z terms $V_\theta(\tau^{(i)}|x_0)$, positive due to the KL-divergence and negative due to the state-cost, can lead to numerical instabilities, e.g., $Z \simeq 0$. The baseline ensures that all of them share sign.

Apart from that, the trajectories $\tau^{(i)}$ that obtain the best rewards are the ones with smallest return $V_\theta(\tau^{(i)}|x_0)$. When normalizing them, the resulting weights $\omega_\theta^{(i)} = \frac{1}{Z} V_\theta(\tau^{(i)}|x_0)$ of such trajectories are the smallest amongst them all, thus having the smallest contributions to the gradient, while we would like them to be the most relevant.

Adding a baseline function that overestimates the positive term of the return and subtracts it forces it to be negative. This means that the trajectories that better optimize the cost have the largest returns (in absolute value), thus increasing their weight on the gradient, which eases the learning process. Also, notice that, since $V_\theta(\tau^{(i)}) < 0, \forall \theta, i$, then $Z < 0$ which means that the sign of the gradient is changed. This can be solved by, simply, performing gradient ascent (changing the sign again) or redefining $Z = \sum_i |V_\theta(\tau^{(i)})| > 0$.

At the beginning of each learning iteration, we start by sampling the trajectories, which consists on performing several rollouts *rolls* of the game of length T going from x_0 to x_T following the policy p_θ.

At the beginning of each rollout we reset our state to be the initial state x_0. Then, at

each time-step t, the policy p_θ provides the probability distribution over the possible future states x_{t+1}, from which we take one at random with probability $p_\theta(x_{t+1}|x_t)$ and transition feature vector $\phi(x_{t+1}, x_t)$. Besides, we compute the expected feature vector as

$$\langle \phi(x_{t+1}, x_t) \rangle_{p_\theta} = \sum_{x_{t+1}} p_\theta(x_{t+1}|x_t)\phi(x_{t+1}, x_t) \tag{3.18}$$

With this, we compute the terms of the return and the score functions corresponding to the time t of the trajectory that is being sampled as

$$V_\theta(x_{t+1}, x_t|x_0) = \lambda \log \frac{p_\theta(x_{t+1}|x_t)}{q(x_{t+1}|x_t)} + R(x_{t+1}) + b \tag{3.19}$$

$$\nabla_\theta \log p_\theta(x_{t+1}, x_t|x_0) = \phi(x_{t+1}, x_t) - \langle \phi(x_{t+1}, x_t) \rangle \tag{3.20}$$

Bear in mind that the state cost $R(x)$ accounts the rewards R_s, R_h as negative. Furthermore, as we consider path-cost and end-cost, for the case of the end-cost $R(x_t) = 0, \forall t < T$, i.e., we only compute the state cost at the last time-step $R(x_T)$.

With the sampled trajectories, we proceed to compute the gradient of the cost function as stated in Equation (3.4) and using the explicit expressions (3.11) and (3.15) to compute the trajectory score and return respectively. In order to do so, we simply need to sum over time the results saved in each sampled trajectory (3.19), (3.20) to obtain $V_\theta(\tau^{(i)}|x_0)$ and $\nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$.

After computing the normalization factor $Z = \sum_i V_\theta(\tau^{(i)}|x_0)$, we, finally, proceed with the calculation of the cost function gradient $\nabla_\theta \mathcal{C}(p_\theta|x_0)$. This is achieved by multiplying each trajectory score $\nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$ with their respective normalized weight $\frac{1}{Z} V_\theta(\tau^{(i)}|x_0)$ and summing the resulting vectors of all trajectories. However, to reduce the computational cost, we can work with the unnormalized returns and normalize the final result as Z is a common factor (3.4).

Finally, we update θ by *summing* the gradient multiplied by a learning rate α. We sum the gradient because, despite minimizing the cost $\mathcal{C}(p_\theta|x_0)$, given that $Z < 0$, we need to change the sign of the gradient to account for it. As mentioned previously,

this could also be done by defining Z as the sum of absolute values of the return and maintaining the sign of the gradient.

As an additional step, we keep track of the difference between the new and previous θ to see how the algorithm converges. We consider that the algorithm has converged when the module of the difference between the new and the previous parameter θ is smaller than a threshold δ and the algorithm halts. However, if this condition is never fulfilled, the algorithm halts upon reaching the limit of learning iterations.

3.1.3 Policy gradient on M-Projection

In order to better understand the algorithm, allow us to start by introducing a pseudo-code that will be thoroughly explained afterwards, like in the previous section 3.1.2.

initialize θ randomly

for its

  for rolls

    reset initial state x_0

    for $t = 0 : T - 1$

      $p_\theta \rightarrow x_{t+1},\ p_\theta(x_{t+1}|x_t),\ \phi(x_{t+1}, x_t),\ \langle\phi(x_{t+1}, x_t)\rangle$

      save return $V_\theta = \lambda \log \frac{p_\theta(x_{t+1}|x_t)}{q(x_{t+1}|x_t)} + R(x_{t+1})$

      save score $\nabla_\theta \log p_\theta = \phi(x_{t+1}, x_t) - \langle\phi(x_{t+1}, x_t)\rangle$

  trajectory returns $V_\theta(\tau) = \text{sum}(V_\theta, t)$

  trajectory scores $\nabla_\theta \log p_\theta(\tau) = \text{sum}(\nabla_\theta \log p_\theta, t)$

  trajectory weights $\omega_\theta(\tau) = \exp(-V_\theta(\tau)/\lambda)$

  normalization factor $Z = \text{sum}(\omega_\theta(\tau))$

  $\nabla_\theta D_{\text{KL}}(p^* \parallel p_\theta) = -\text{sum}(\nabla_\theta \log p_\theta(\tau)\omega_\theta(\tau))/Z$

  $\theta \leftarrow \theta - \alpha\nabla_\theta D_{\text{KL}}(p^* \parallel p_\theta)$

  check convergence $||\theta_k - \theta_{k-1}|| = ||\alpha\nabla_\theta D_{\text{KL}}(p^* \parallel p_\theta)|| < \delta$

The general procedure is very similar to the previously described for the case of the I-projection with some adaptations to compute the gradient of the cost function (2.13) by following Equation (3.6) in order to updated the parameter θ via gradient descent according to a learning rate $\alpha > 0$.

We start by initializing θ randomly following a normal random distribution with $\mu = 0,\ \sigma = 2$, as before.

In this algorithm, there is no need for a baseline function, provided that the returns $V_\theta(\tau^{(i)}|x_0)$ are exponentiated (3.16) and, thus, the weights $\omega_\theta(\tau^{(i)}|x_0)$ to normalize are positive definite. Besides, the trajectories that best optimize the cost, naturally have the largest contributions to the gradient.

At the beginning of each learning iteration, we start by sampling the trajectories performing several rollouts, *rolls*, of the game of length T, from x_0 to x_T. The sampling procedure is exactly equal to that of the I-projection: starting from x_0, following policy p_θ, at each time-step t, obtain the next state x_{t+1} with probability $p_\theta(x_{t+1}|x_t)$ and transition features $\phi(x_{t+1}, x_t)$. Compute the expected feature vector $\langle\phi(x_{t+1}, x_t)\rangle_{p_\theta}$ following (3.18) and the return and score terms as described in Equations (3.19) and (3.20) with $b = 0$.

In order to compute the trajectory scores and returns, we simply sum over time, respectively, the terms (3.19) and (3.20) saved for each of the sampled trajectories to obtain $\nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$ and $V_\theta(\tau^{(i)}|x_0)$, as described in Equations (3.11) and (3.15).

Then, we proceed to compute the trajectory weights $\omega_\theta(\tau^{(i)}|x_0)$ as in (3.16) and the normalization factor $Z = \sum_i \omega_\theta(\tau^{(i)}|x_0)$, with which we can finally proceed to compute $\nabla_\theta D_{\mathrm{KL}}(p^* \parallel p_\theta)$. This is achieved by multiplying each trajectory score $\nabla_\theta \log p_\theta(\tau^{(i)}|x_0)$ by its respective weight $\omega_\theta(\tau^{(i)}|x_0)$, summing the resulting vectors of all trajectories and multiplying by $-\frac{1}{Z}$, just as stated in Equation (3.6).

Finally, we update θ by subtracting the gradient multiplied by the learning rate α.

As an additional step, we keep track of the convergence as described for the case of the I-projection.

3.2 Distributed control

The distributed control specifies, at each time-step, a transition probability between two individual relative states for every agent, who have partial observability of the joint state and move independently according to a common policy. This approach is scalable to larger systems with both larger grids and several agents as it only scales polynomially with the number of players, apart from providing an excellent framework for distributed computing.

Initially, we have chosen to solve the optimal control for the same Stag-Hunt game introduced in the centralized control (section 3.1), whose initial state x_0 is represented in Figure 1, in order to compare the results between both approaches. As a reminder, it is a game of two hunters with four hares, one at each corner, and a stag at the center of a 5×5 grid, where the hunters play for a total of $T = 4$ time-steps ($\tau = x_{1:4}$).

Provided that the rewards are $R_s = 3$ and $R_h = 1$ for stag and hares respectively, the possible individual outcomes are:

Reward 3: if both hunters land on the stag.

Reward 1: if the hunter lands on a hare.

Reward 0: if the hunter does not land on any prey.

3.2.1 State representation

Agents are completely independent and can only see within a rectangle of side $2r+1$ centered on them, where r is the vision radius an can be different between directions. Nevertheless, in this specific case, we have taken $r = 2$, meaning that the hunters can see the whole grid when located at its center. Figure 3 shows the vision field of the agents in the initial state $\mathbf{x}_0$.

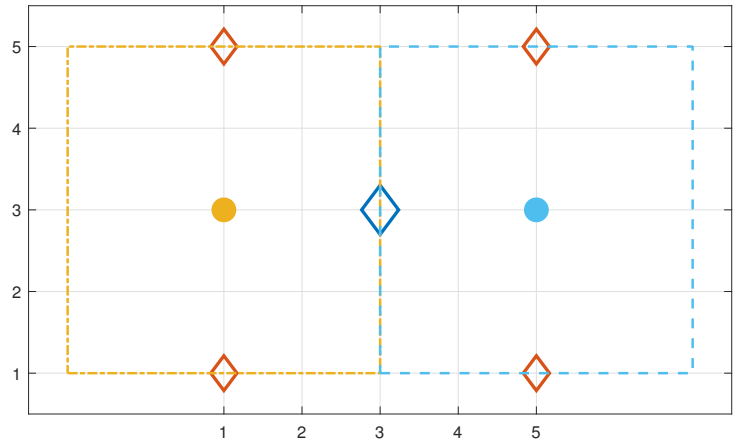


Figure 3: Agent vision range in the initial configuration of the Stag-Hunt game. The hunters are represented by coloured dots, the hares are represented by red diamonds and the stag is represented with a blue diamond. The vision field limit of each hunter shares the colour of its respective dot.

Hence, hunters perceive a state x relative to them, which corresponds to the representation of a part of the whole joint state $\mathbf{x}$. Therefore, in x, the positions of the preys and the other hunters changes depending on the agent position.

Since the control is individually applied to every agent, given a relative state x_t at time-step t, there are A possible future relative states x_{t+1}, being A the amount of possible actions an agent can do (typically 5). Thus, the complexity of the problem grows polynomially with the number of agents as, at each time-step, there need to be considered $M \times A$ future relative states, where M is the amount of players, despite having A^M possible future joint states $\mathbf{x}_{t+1}$, thing that allows this kind of control to be applied to situations with several agents.

We use a simpler representation of the relative state in which the identity of the other agents is ignored and their discrete relative positions are encoded using continuous activation of basis functions. However, although the positions of the preys are fixed in the grid, their relative position with respect to the agents changes as they move. Besides, unlike the hunters, the identity of the preys (stag, hare) is relevant for the problem, thus requiring a set of features that encode both their relative position and nature.

As in previous cases, the policy p_θ is defined over the feature vector $\phi(x', x)$ that encodes the transition between current and next relative states. Following the same philosophy as in the centralized approach, we define it as the concatenation of the individual feature representation of the next relative state x' with the current one x, as expressed in Equation (3.17).

Now, allow us to introduce the way we do the feature representation of a relative state $\phi(x)$. The feature vector contains a vector that represents the position and the nature of the preys together with the contributions of the other hunters to a set of RBFs that encode their discrete relative positions.

The way of choosing the RBFs and computing the hunter contributions to them is the same as explained in section 3.1.1. The difference here is that the RBFs are not defined w.r.t. the grid dimensions, but w.r.t. the range of vision of the agent. This can be understood as that functions move together with the agent.

Provided that we have taken $r = 2$, the vision range or window of the hunters correspond to a 5×5 grid, which means that the set of RBFs corresponding to each of them is the same as the one represented in Figure 2.

At the same time, the preys and their relative positions w.r.t. the hunter are encoded in a vector that represents all possible positions within the vision window. Each component of the vector contains either 2, 1 or 0 depending on whether there is, respectively, a stag, a hare or no prey in the position that it represents.

The set of relative state features is the concatenation of the prey vector together with the basis functions. Hence, for our case, the feature vector $\phi(x)$ has length 34 corresponding to:

- The prey representation in the 25 positions within the vision window.
- The contributions to the 9 RBFs.

Thereby, the feature vector encoding the relative state transition $\phi(x', x)$ has length

64 and so does θ. Notice that, with this approach, the number of features scales quadratically with the vision range r.

3.2.2 Policy gradient algorithms

In this section we cover both policy gradient algorithms focusing on the adaption to the distributed problem, as the computational details are thoroughly explained in sections 3.1.2 and 3.1.3 for the I-projection and M-projection respectively.

Despite being individual agents, given that the policy is shared among them, they all also share their experience to update the parameter vector θ.

Let us introduce a pseudo-code omitting the mathematical expressions, as it is not scope.

initialize θ randomly
for its
  for rolls
    initial joint state $\mathbf{x}_0$
    for $t = 0 : T - 1$
      for hunters
        $p_\theta \rightarrow x_{t+1},\ p_\theta(x_{t+1}|x_t),\ \nabla_\theta \log p_\theta$
      joint action $\mathbf{x}_t \rightarrow \mathbf{x}_{t+1}$
      for hunters
        individual return V_θ

  for hunters
    trajectory coeffs. $V_\theta(\tau),\ \nabla_\theta \log p_\theta(\tau)$ and (CE only:) $\omega_\theta(\tau)$
    normalization Z and gradient $\nabla_\theta D_{\mathrm{KL}}$
    $\theta \leftarrow \theta - \alpha \nabla_\theta D_{\mathrm{KL}}$

  check convergence $||\theta_k - \theta_{k-1}|| < \delta$

The main subtlety to take into account is that, despite all agents taking the actions individually, it is their joint action that changes the joint state $\mathbf{x}$.

The joint action is the set of *all* individual actions taken by the players at a time-step t. Hence, as the hunter policies need to be computed sequentially in our straight approach, there cannot be any modification of the joint state $\mathbf{x}$ before they all have it. Hence, there cannot be computed anything that depends on the joint state until they have all acted. More precisely, we cannot compute the individual state-costs $R(x_{t+1})$ and, therefore, the returns $V_\theta(x_{t+1}, x_t|x_0)$.

This can be illustrated with a very simple example: imagine two hunters next to a stag that want to move towards it at the current time-step. Ideally, both actions would be executed simultaneously, however, we have to first compute the policy for one hunter and then for the other. Hence, if one hunter moved towards the stag before the other, he would find himself alone, thus not receiving any reward. Then, when the second one moved in, it would receive a reward as they would both be on the stag. Therefore, we need to compute the state-cost after the joint action is executed.

In order to achieve this, at every time-step t, we first go over all hunters and get their next relative states x_{t+1} with probability $p_\theta(x_{t+1}|x_t)$, according to the shared policy p_θ. With it, there can be computed $\phi(x_{t+1}, x_t)$ and $\langle\phi(x_{t+1}, x_t)\rangle_{p_\theta}$ to obtain the individual score following Equations (3.18) and (3.20), as described in previous sections, provided that these only depend on the policy and the action taken by the individual agent.

With all the individual actions $x_t^{(m)} \to x_{t+1}^{(m)},\ m = 1, \ldots, M$, with M begin the total number of players, we execute the joint action of all agents to update the joint state $\mathbf{x}_t \to \mathbf{x}_{t+1}$.

Once we have this, we can proceed to compute the individual state costs $R(x_{t+1})$ for all hunters and, therefore, the individual return terms $V_\theta(x_{t+1}, x_t|x_0)$.

Once the trajectories are sampled, we proceed to individually apply the learning

algorithms described in sections 3.1.2 or 3.1.3 to all the hunters updating the same shared parameter θ and, therefore, sharing their experience.

Chapter 4

Results

In this chapter, we present the results obtained with the methods introduced in Chapter 3. In order to make it clearer, this chapter follows a structure that is very similar to the previous one. With these results, we strive to answer the research questions that were introduced in Chapter 1 and summarized in page 5.

With this goal, we have implemented the REINFORCE (I-projection) and CE (M-projection) algorithms (Chapter 2) following the centralized and the distributed approaches considering end-cost and path-cost (Chapter 3).

Provided that we are interested in the analysis of the algorithms as a function of the temperature parameter λ, as done in [14], we have trained 100 polices p_θ for $\lambda = 0.1, 0.2, \ldots, 2$ considering all the combinations of approach, algorithm and state cost. Thereby, combining centralized and distributed, I and M projections and end and path costs. This provides a total of $8,000$ policies p_θ trained under the same conditions: 100 learning iterations and $2,000$ game rollouts ($T = 4$) per iteration. Then, they are evaluated considering both their most probable trajectory (MAP trajectory), as in [14], and 1000 runs of the Stag-Hunt game following the policy p_θ.

The most probable trajectory is obtained by taking, at each time-step t, the most probable next state x_{t+1} according to the policy $p_\theta(x_{t+1}|x_t)$. On the other hand, the stochastic evaluation over 1000 games is done by randomly selecting the next state

x_{t+1} according to the learned policy $p_\theta(x_{t+1}|x_t)$.

We have measured the performance of the learned policy by means of the mean probability with which actions are taken during the game, the reward obtained at end-time, the mean reward obtained along the trajectory all averaged over $1,000$ game runs and the maximum reward obtained along the maximum probable trajectory.

The mean action probability provides an idea of how deterministic the policy is, provided that lower probabilities indicate more stochastic policies, as the probability mass is spread over more next states. The end reward allows us to evaluate the performance of the control in a finite-horizon scenario, while the mean reward accounts for an infinite-horizon scenario. Finally, the MAP trajectory reward provides information about whether the control leans towards a risk-dominant or a payoff-dominant nature [14].

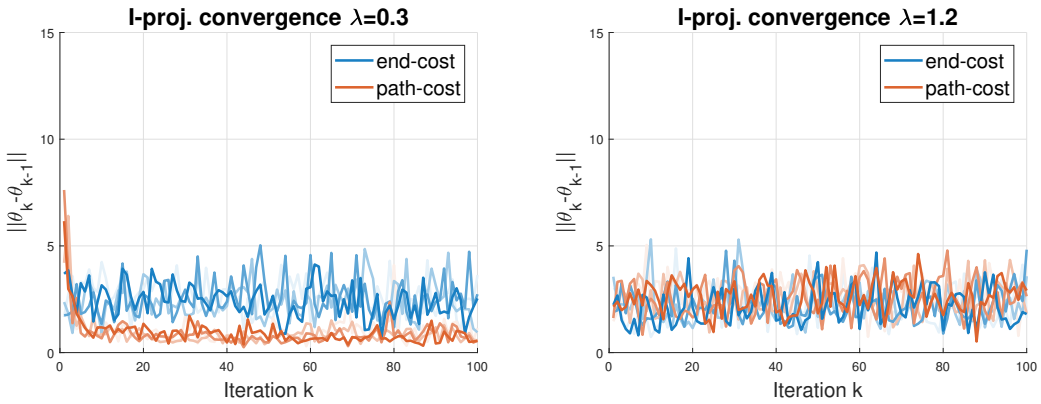
In this last case, we take the maximum state reward obtained along the trajectory and not the mean nor the end, provided that we want label the control. The stochasticity of the policy, specially for high values of λ, makes more difficult to keep the agents at the same position. This way, we keep the most probable best possible state cost of the control.

With this, in order to evaluate every policy, we take the average results over its 1000 game runs. The result for every value of λ is taken out of the 100 policy evaluations of those trained under the determined conditions, i.e., centralized or distributed, REINFORCE or Cross-Entropy, end-cost or path-cost and λ.

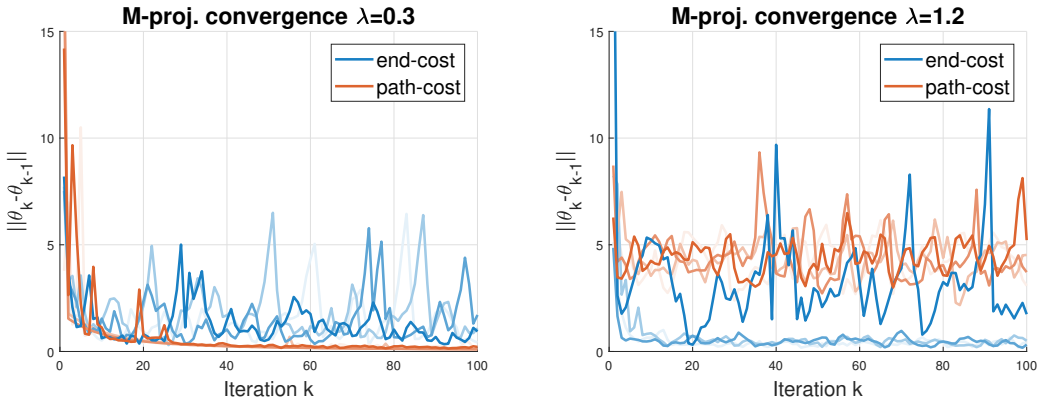
4.1 Centralized control

In this section, we present the results obtained with the centralized approach with both the policy gradient on the I-projection (REINFORCE) and on the M-projection (Cross-Entropy) algorithms.

Before analyzing the resulting controls, let us first focus on the convergence of the



(a) Convergence curves of the parameter vector θ with REINFORCE.

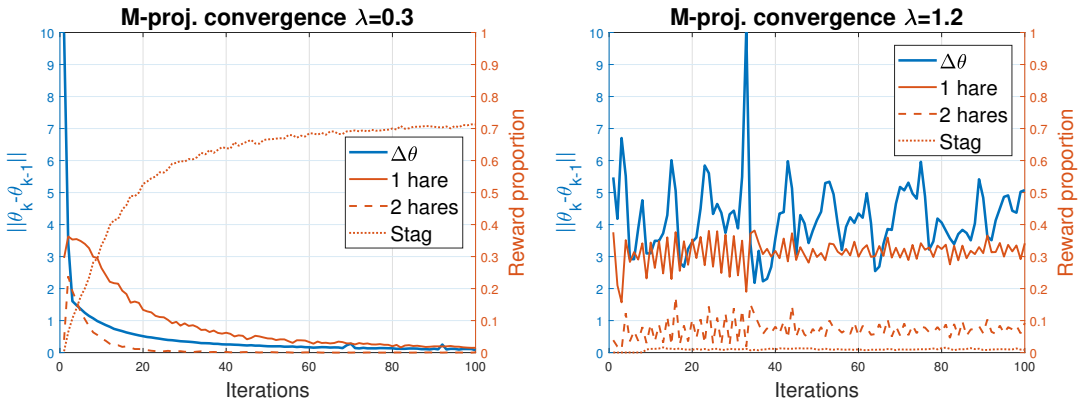


(b) Convergence curves of the parameter vector θ with Cross-Entropy.

Figure 4: Sample convergence curves of the parameter vector θ during learning process with REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost and path-cost for $\lambda = 0.3$ (left) and $\lambda = 1.2$ (right).

algorithms in order to see the way they reach them. Figure 4 shows a few sample convergence curves of both algorithms considering end-cost and path-cost for two values of λ, where it can be seen that the M-projection provides a spikier learning curve than the I-projection, which means that it is more sensitive to the relevant events than it is to the rest.

The addition of path-cost catalyzes the learning process for low λ but it does not seem to have any relevant effect for high λ, when the learning is noisier, which is characterized by larger fluctuations. Do not be mislead by Figure 4b (right) in which it seems that the addition of path-cost results into divergence due to having higher θ variations. Bear in mind that the θ vector output with path-cost has larger module and, therefore, its fluctuations are larger in presence of noise. In addition, with



(a) Convergence for $\lambda = 0.3$.

(b) Convergence for $\lambda = 1.2$.

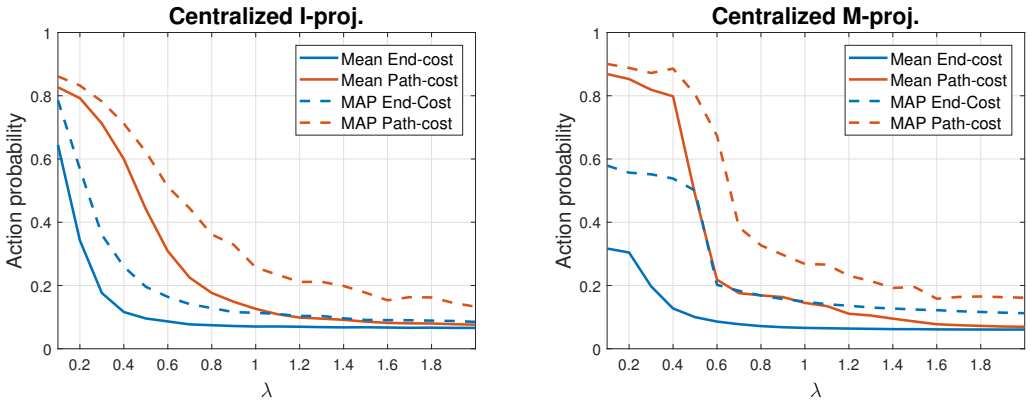
Figure 5: Representative sample convergence curves of the Cross-Entropy algorithm with path-cost for $\lambda = 0.3$ (a) and $\lambda = 1.2$ (b). The reward proportion represents the amount of actions that result into hunting a prey out of the total of actions taken during the game rollouts of the learning iteration.

end-cost, the algorithm barely finds any prey to learn from, resulting into rather low θ variations.

In some cases, specially for large values of λ, the algorithms reach the maximum number of iterations and fail to converge according to our criterion. However, if we look at the obtained policies after the maximum number of iterations, Figure 5, we observe that the obtained policies, while very stochastic, are robust and systematically reproduce the same proportion of rollouts that reach the different targets across iterations. Possible ways to enforce convergence in the whole range of λ could be decreasing the learning rate during learning or adapting the convergence threshold as a function of λ, which is not considered in this work.

Notice that, in our specific case, it is impossible to obtain any state-reward in the first action out of $T = 4$, thus, Figure 5a shows how the algorithm approaches the limit in which the control brings the agents onto the stag in the second action and keeps them there for the rest of the game (stag proportion of 0.75) obtaining the maximum possible game score.

Moving on to the resulting controls, Figure 6 shows the mean probability with which actions are taken during the game for both algorithms, the policy gradient on the I-projection, Figure 6a, and on the M-projection, Figure 6b. There can be seen



(a) Mean probability with REINFORCE (I-proj).

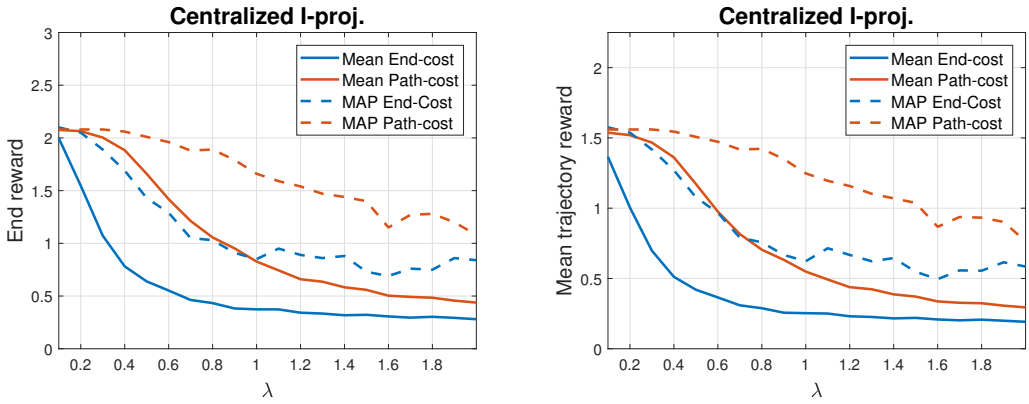
(b) Mean probability with Cross-Entropy (M-proj).

Figure 6: Mean probability along a game with which actions are taken as a function of λ obtained with the control learned by the centralized REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost and path-cost.

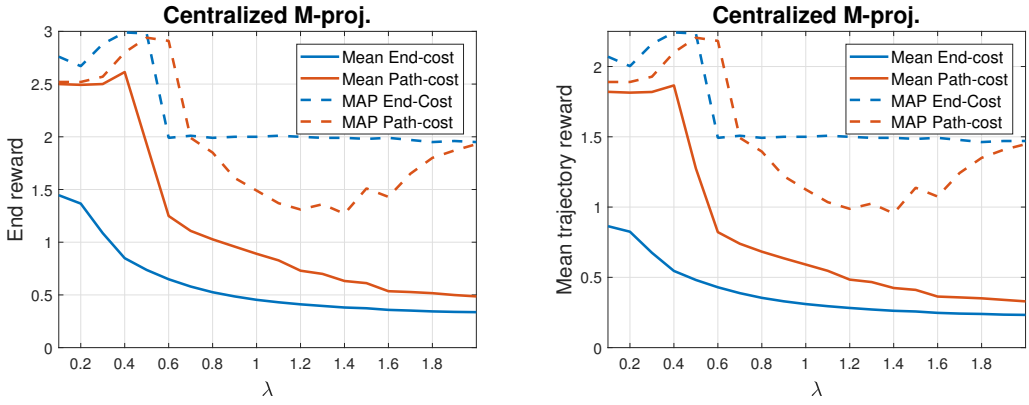
how the policies become more stochastic as a function of λ and, in both cases, the consideration of path-cost results into a more deterministic policy, specially for low values of λ. The biggest effect is observed in the M-projection, which is considerably less deterministic than the I-projection for end-cost. Bear in mind that both methods optimize the same theoretical objective and the only difference should be the way they achieve the optimal solution.

For high λ, policies tend to the limit of a uniform random distribution, as the maximum probability taken (along MAP trajectory) approaches the average and the average tends to the limit of $1/A$, with A being the number of total joint possible actions (maximum 25). However, the consideration of path-cost keeps the probability peaks higher with growing λ, as the difference between the average probability and its maximum is higher than when considering end-cost. This reading cannot be applied for low λ, provided that the policy is so deterministic that the average tends to the probability peak.

Figure 7 contains the control performance in terms of game scores. The reward of the MAP trajectories is, actually, the mean of the results shown in Figure 8. We can see how the path-cost consideration not only increases the game rewards obtained along the trajectories, but also the rewards at end-time, thus providing better results



(a) Mean end reward and mean average reward with REINFORCE.

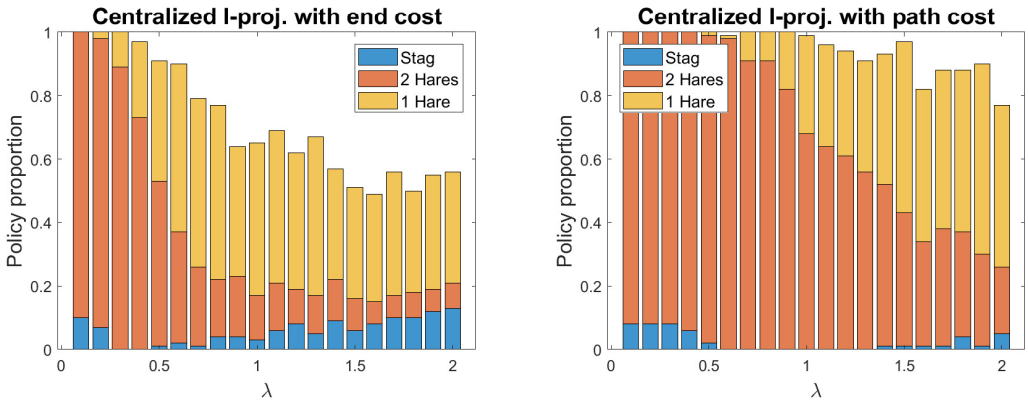


(b) Mean end reward and mean average reward with Cross-Entropy.

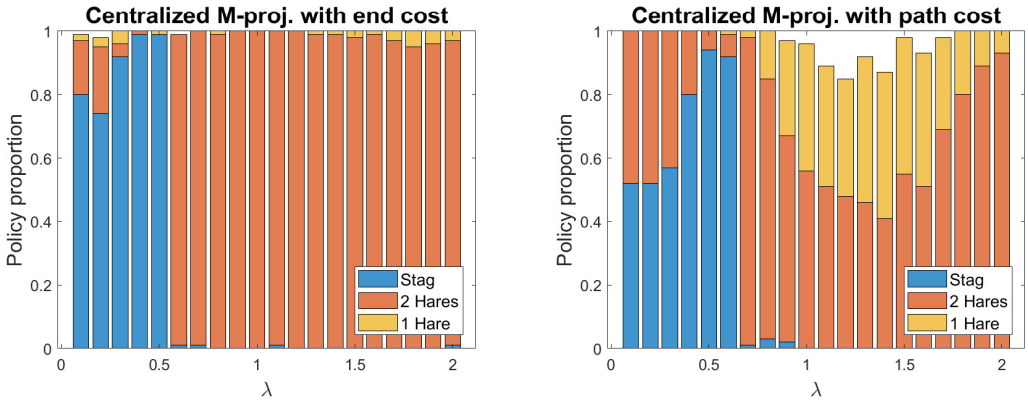
Figure 7: Mean end reward (left) and mean average reward along a game (right) as a function of λ obtained with the control learned by using the centralized REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost and path-cost.

in both a finite-horizon and an infinite-horizon scenarios. This could be due to the fact that path-cost provides more useful information during the sampling.

We observe a remarkable difference in the effect of the path-cost consideration between algorithms, REINFORCE and CE. For the REINFORCE, the resulting game scores slightly increase with path-cost, whereas for the Cross-Entropy method they represent a major increase on average. This is due to the fact that path-cost drastically increases the deterministic character of the policy in the M-projection and, therefore, being able to obtain preys more consistently for low λ. Besides, the CE method is capable of coordinating to get the stag, increasing the gap even further and outperforming the REINFORCE algorithm, which only gets hares as shown in Figure 8.



(a) Policy distribution considering end-cost and path-cost with REINFORCE.



(b) Policy distribution considering end-cost and path-cost with Cross-Entropy

Figure 8: Proportion of policies focusing on the different possible game objectives as a function of λ labeled according to the best game outcome along the MAP trajectory. Policies are learned by using the centralized REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost (left) and path-cost (right).

Regarding the risk and payoff dominant regimes [14], we compute the proportion of policies that led to different outcomes following the MAP trajectory in order to understand the qualitative differences between the obtained policies. This is shown in Figure 8, which presents significant differences between algorithms. Figure 8a clearly shows how the I-projection mainly gets hares (risk dominant) for all λ and the introduction of path-cost helps its convergence making it more robust, specially for high values of λ.

In contrast, for the M-projection in Figure 8b, there can be observed a sharp differentiation between the two regimes as a function of λ: for low λ values, stags predominate (payoff dominant) and for high λ hares predominate (risk dominant). Furthermore, the consideration of path-cost seems to disturb the convergence of the

CE algorithm. However, in Figure 7b, there can be seen how, despite having worse MAP trajectories, on average, the game outcomes are better.

Notice that there is a shift in the λ that marks the regime transition when changing from end-cost to path-cost (Figure 8b) due to the fact that the difference between going for hares and stags in path-cost is larger than in end-cost, thus prolonging the payoff dominant regime.

From these results we can conclude that the CE algorithm is able to obtain the optimal solution, while the REINFORCE generally falls in a local minimum. Besides, the consideration of path-cost enhances the overall performance helping on the convergence and allowing the controls to become more deterministic.

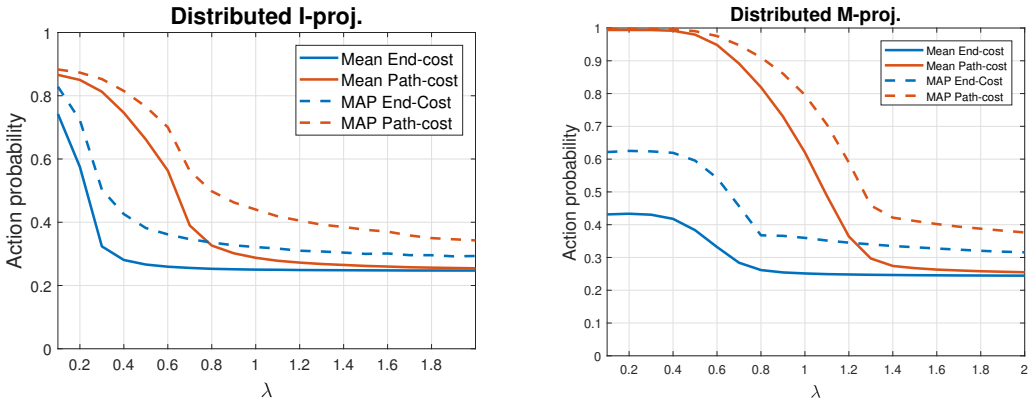
4.2 Distributed control

In this section, we present the results obtained with the distributed approach with both the policy gradient on the I and the M projections.

Figure 9 shows the mean probability with which actions are taken during the game for both algorithms: REINFORCE, Figure 9a, and Cross-Entropy, Figure 9b. There can be observed how, with end-cost, the M-projection provides a less deterministic control as happened with the centralized approach in the previous section 4.1. The policies turn more stochastic with increasing λ tending to a randomly uniform distribution. Bear in mind that now the policy considers only five actions at maximum.

The addition of path-cost provides more deterministic policies for both cases. However, the effect is more pronounced in the case of the CE algorithm, for which the resulting policies are strongly deterministic. Besides, it keeps the probability peaks higher in the distribution as the policy tends to a uniform distribution for high λ.

Figure 10 shows the control performance in terms of individual game scores. Again, path-cost not only increases the performance for the infinite-horizon scenario, but also for the finite-horizon problem. For end-cost only, the performance of both algorithms is pretty similar despite going for different preys, as it can be seen in



(a) Mean probability with REINFORCE. (b) Mean probability with Cross-Entropy.

Figure 9: Mean probability along a game with which actions are taken as a function of λ obtained with the control learned by the distributed REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost and path-cost.

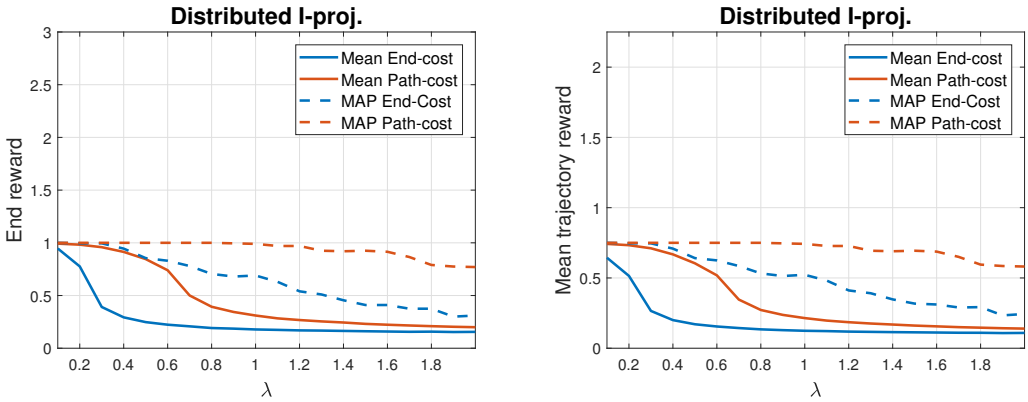
Figure 11, due to CE being significantly more stochastic, Figure 9.

Nevertheless, the consideration of path-cost has a massive effect on the CE algorithm attaining the maximum reachable game scores for low λ. The agents reach the stag in the second movement (out of $T = 4$) and stay on it until the end of the game (mean 2.25). The MAP trajectory clearly represents this behaviour until there is a sharp drop in the output indicating the change of regime from payoff dominant to risk dominant. The difference for the REINFORCE algorithm is not so pronounced provided that it gets to the limit of the maximum hare reward (mean 0.75) instead of stag.

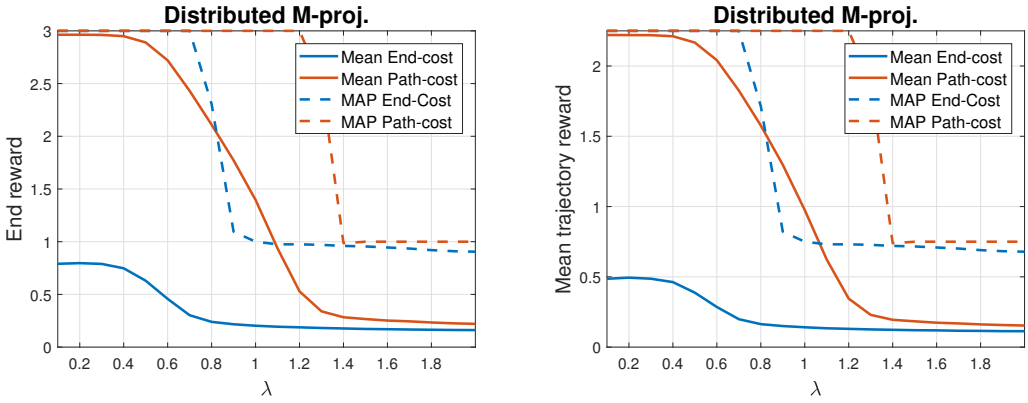
Figure 11 shows a substantial difference between algorithms. Like in the centralized approach, the solution obtained by REINFORCE is characterized by the risk-dominant regime in which agents are distributed among hares, while CE provides the sharp differentiation between the two regimes, risk and payoff dominant.

For the case of the I-projection, shown in Figure 8a, the addition of path-cost increases the performance of the algorithm as it becomes more robust and the agents can get both to the hares more consistently.

For the case of the M-projection, represented in Figure 8b, the consideration of path cost, in contrast to the centralized approach, seems to enhance the convergence of

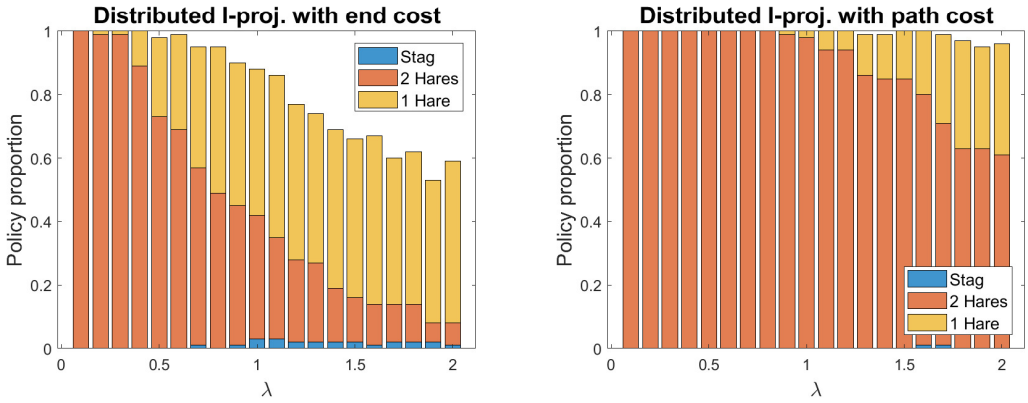


(a) Mean end reward and mean average reward with REINFORCE

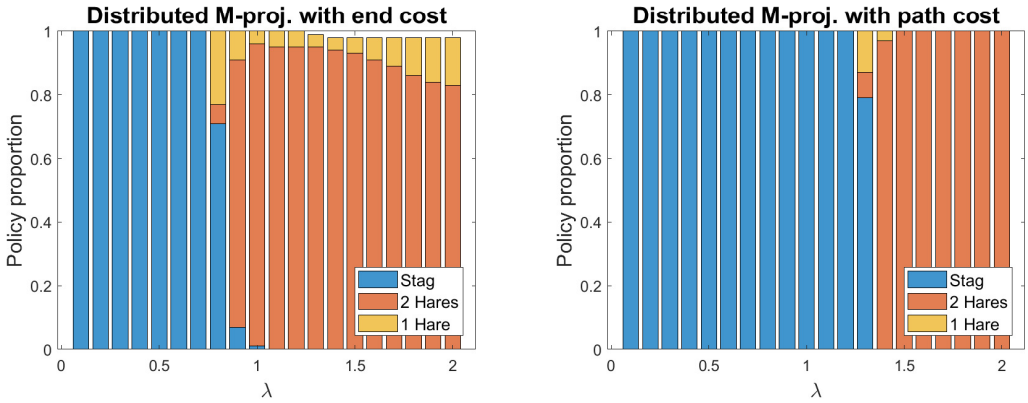


(b) Mean end reward and mean average reward with Cross-Entropy

Figure 10: Mean end reward (left) and mean average reward along a game (right) as a function of λ obtained with the control learned by using the distributed REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost and path-cost.



(a) Policy distribution considering end-cost and path-cost with REINFORCE.



(b) Policy distribution considering end-cost and path-cost with Cross-Entropy

Figure 11: Proportion of policies focusing on the different possible game objectives as a function of λ labeled according to the best game outcome along the MAP trajectory. Policies are learned by using the distributed REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost (left) and path-cost (right).

the algorithm, specially for high λ values, in the risk dominant regime.

Notice that the shift in the λ that marks the regime transition when going from end-cost to path-cost is higher now, as the hunters, unlike in the centralized case, cannot perceive the reward of going both to hares $2 \times R_h$ as they are completely independent. Therefore, the differences between hares and stags are higher, which extends the payoff dominant regime.

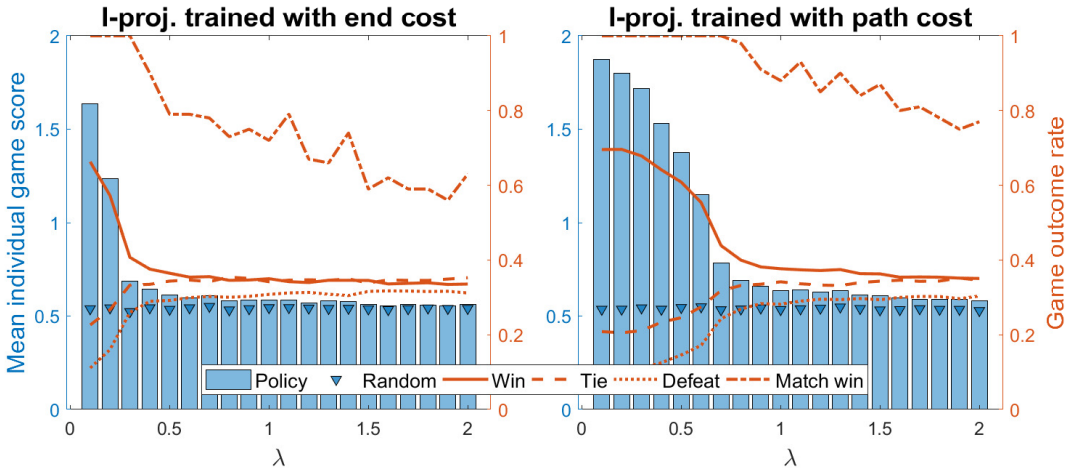
From these results we can conclude that the distributed control is more sensitive to the addition of path-cost than the centralized control. Furthermore, the regime transition can be obtained with individual agents better than with the centralized version.

4.2.1 Control generalization

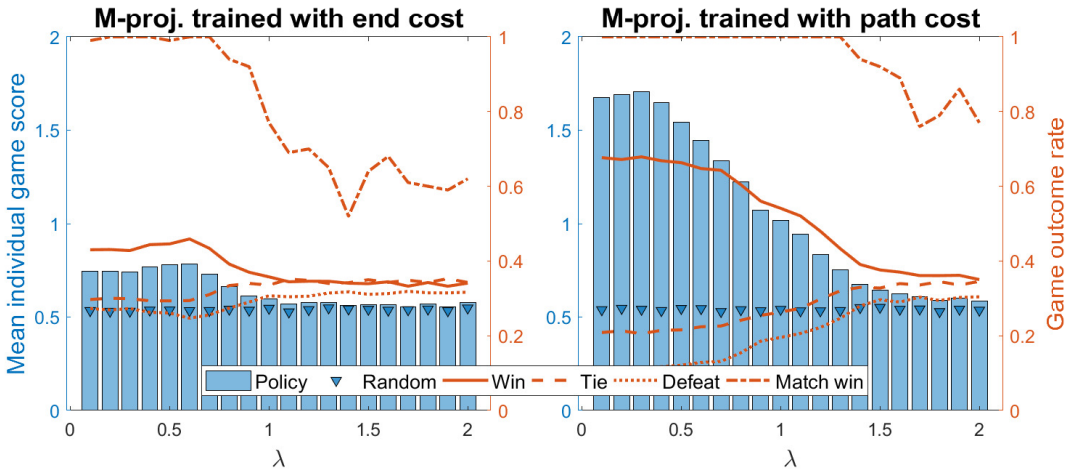
Provided that the agents have partial observability of the whole state and that, thereby, the policy is defined over features that are relative to the agent and not to the problem itself, we want to test its behaviour when facing a different initial configuration $\mathbf{x}_0$ from the one that has been used to learn θ. In other words, we want to see whether a control trained in one specific scenario can generalize to other different ones.

In order to evaluate the generalization of the policies, we set up a Stag-Hunt tournament that confronts two teams: the learned policies and a random policy. Each learned policy is compared to a random policy in a match of $G = 100$ random 1-shot Stag-Hunt games and contribute to the team with their match score. The tournament is won by the team with the highest score. This allows us to see from which value of λ it becomes worth to use a learned policy from a different game in terms of state-costs.

The random stag hunt games are generated according to a set of arbitrary conditions. The grid dimensions $d_x,\ d_y$ are independent and taken randomly ranging from 4 to 10. The minimum number of players H is 2 and can get up to 8 with a probability proportional to the total number of positions within the grid $N = d_x \times d_y$. More



(a) Generalization with REINFORCE.



(b) Generalization with Cross-Entropy

Figure 12: Mean individual game reward obtained by the learned policies and random policies together with mean game outcome rate and match win rate for the learned policies with the distributed REINFORCE (a) and Cross-Entropy (b) algorithms considering end-cost (left) and path-cost(right).

precisely, with probability $p = N/150$, we add randomly from 1 to 3 additional players and the process is repeated with probability $p/2$. The number of hares is taken as $h = \text{round}(1.75H)$ and there is, at least, one stag every two players $s = \text{round}(H/2)$. Then, the hunters, hares and stags are randomly distributed over the grid avoiding the superposition of hares and stags.

Figure 12 shows the results obtained from the evaluation of the policies learned in this section 4.2 against uniform policies that serve as baseline reference. Bear in mind that the random walk is the optimal solution for $\lambda \to \infty$.

Similarly to Figure 10, we observe that policies obtain better **state costs** for low λ, when they are more deterministic, of the order of three times better than the random. As the policies approach a uniform distribution with increasing λ, the game results tend towards the random baseline. This is also captured in the evolution of game win, tie and defeat rates as they approach $\frac{1}{3}$ with λ.

Nevertheless, despite what might be low game win rates, the match win rate is kept rather high. This means that, even though the learned policies do not surpass the random walk in all scenarios, almost all the policies provide better results on the long run (over 100 games).

The way the game scores in Figure 12 compare to the results shown in Figure 10 is by multiplying the mean trajectory rewards (right) of the latter by $T = 4$. As expected, the control cannot effectively achieve the performance that it had in the training scenario obtaining game outcomes near 3 and 9 on average, for the I and M projections respectively and low λ, corresponding on standing on a hare and a stag for three out of the four turns. Both algorithms present similar results for the path-cost (Figure 12 (right)), but the CE, despite having a slightly lower outcome peak, keeps the top performance for higher λ. The mean individual game outcomes between 1.5 and 2 mean that each hunter nearly gets 2 hares per game on average.

Notice that there is not a big loss of performance in the generalization for the REINFORCE algorithm, as it focuses on hares and has only dropped from 3 to 2 on average output, which can be interpreted as that hunters need two actions to reach a hare instead of one. In contrast, the CE algorithm has dropped from 9 to 2 (Figures 10b (right) and 12b (right)), provided that it is no longer able to efficiently obtain all the game stags.

With the obtained results, we can conclude that it is generally better to use the learned algorithms to obtain better state-costs for any λ, specially, for low values. Moreover, the M-projection with path-cost has the best performance as a function of λ.

Chapter 5

Discussion

In this chapter we analyze all the results obtained in Chapter 4 in the context of the whole work in order to withdraw the conclusions presented at the end of the chapter, right before we present the future work.

5.1 Discussion

Let us start by comparing the general features and performance of the REINFORCE and the Cross-Entropy algorithms.

The main difference is the capability to find the cooperative solution of the stag, which is optimal for low λ. While the CE algorithm is rather successful in this task, the REINFORCE algorithm is too greedy to converge towards the stag and falls into the local minimum of the hare solution. Notice that both algorithms optimize the same theoretical objective and the only difference should be the way the achieve the optimal solution.

At the beginning of the learning process, the algorithms start with a random parameter θ and start exploring by doing several rollouts of the game. Combinatorially, the probability of the agents landing on hares is orders of magnitude larger than the probability of the agents landing *together* on the stag, and, even more, depending on θ, it can happen that, on the first set of games, the hunters do not get any stag.

Under these conditions, a greedy cost-minimization algorithm like the REINFORCE is driven directly to the hare solution, Figures 8a and 11a.

In contrast, the Cross-Entropy algorithm is less subject to the initial conditions and is able to find the stag solution in most situations due to its exploratory character. Furthermore, it is capable of recovering the sharp transition between risk-dominant and payoff-dominant regimes as a function of λ introduced in [14]. Besides, it does not only obtain the regime transition in the centralized control, but also in the distributed approach (Figures 8b and 11b), therefore, proving that it can be obtained with a sample-based algorithm and that cooperation can appear with independent agents.

Moving on to the effect of the path-cost consideration, it increases the performance in *all* situations, Figures 7, 10 and 12, specially on the M-projection. The Cross-Entropy algorithm, even though it finds the optimal solution, only with end-cost, it is very stochastic and its game output suffers from it. Then, with the addition of path-cost, it becomes strongly deterministic (for low λ) remarkably boosting the results, Figures 6b and 9b. Moreover, path-cost shifts the transition to the risk-dominant regime to higher λ, providing better rewards overall, Figures 8b and 11b.

This extreme performance enhancement is not observed in the REINFORCE algorithm, provided that, with end-cost, the resulting control is naturally more deterministic than with the CE algorithm, thus having less margin to improve in this field, Figures 6 and 9. Besides, it mainly goes for hares and path-cost does not help it go for stags, which provides lesser rewards.

Adding path costs, therefore, makes the problem less *sparse* in the cost definition and this has a positive effect regarding sample efficiency, since there is more information obtained per sample. We remark that path cost is a problem feature that generally has a negative impact in the graphical model approach, provided that it quickly increases the tree-width of the resulting graphical model [14]. In contrast, as we have shown in this work, it is beneficial for the sampling-based methods.

For high λ, the resulting policies p_θ approach the limit of a uniform distribution

over future states, making the difference between algorithms barely noticeable. The reason behind this behaviour lies on the cost itself. The trajectory cost or return $V_\theta(\tau|x_0)$ penalizes the transitions that deviate from the passive dynamics q weighted by λ. Hence, for low λ, this term is irrelevant compared to the state-costs, but as λ grows, the penalization for deviating from q dominates, thus forcing the policy p_θ to tend to q, which is a uniform distribution over possible transitions [37]. Because of this, we have mainly focused on the output for the lower range of λ and pay interest to those algorithms that can maintain high outputs for increasing values of λ, which is what is obtained with the addition of path-cost.

Finally, our distributed approach has proven to be able to perform decently on game scenarios that differ from the training game in all aspects: map size and number and position of hunters and preys, Figure 12. We observe that hunters obtain, on average, a couple of hares per run for both algorithms. Therefore, the difference in the REINFORCE performance w.r.t. the training scenario is rather low, provided that it already provided the hare solution. Its decrease can be understood as that hunters need one more action to reach the preys, which is something we can expect when dealing with maps that can get up to twice as big as the training one.

In contrast, the major difference is found for the CE algorithm, which was capable of efficiently hunting the stag in the training scenario obtaining the maximum possible game score and now, instead, obtains an average of two hares per run when facing different scenarios no longer being able to obtain all the stags within the map. In fact, there needs to be taken into account that, given that the game time T is not increased, for some scenarios, it can be impossible to reach the stags. Nevertheless, its performance is in pair with the REINFORCE algorithm as it is still able to go for the hares triplicating the random outcomes. Even more, given that it remains deterministic for higher λ, it is able to keep a high performance for a wider λ range.

As λ increases and the policies approach the uniform distribution, the mean results tend to the random results as well as the game win rate approaches $\frac{1}{3}$ together with the tie and defeat rates. However, the match win rate prevails high. As a reminder, a match is a set of $G = 100$ games in random scenarios, which means that, even though

the policies do not outperform the random walk in all scenarios, they are better on the long run. This is due to the fact that the learned policies, despite tending towards a uniform distribution, still have moderate probability peaks (Figure 9) that guide them to the preys. Thereby, they are capable of obtaining rewards more consistently, which also means that they have less null game runs (games without reward).

With this, we can say that the distributed control generalizes to completely new scenarios at the cost of a sub-optimal solution (mainly gets hares) and can be scaled to larger and more complex problems. However, this sub-optimal strategy is what already was obtained with REINFORCE. This is a pretty interesting result, suggesting that arbitrarily large systems can be reduced to the simplest representations of themselves for training and, then, apply the resulting control back into to the original problem at the cost of full optimality.

5.2 Conclusions

The main conclusions that can be withdrawn from this work are the following

- A sample-based method can recover the sharp transition between payoff-dominant and risk-dominant regimes.
- Cooperation naturally emerges between independent agents as long as it is the optimal solution to the problem.
- A greedy cost optimization algorithm, like REINFORCE, is unable to accomplish the previous two points as it is too greedy and consistently gets stuck into sub-optimal solutions.
- The Cross-Entropy method is more robust to initial conditions and consistently finds the optimal solution.
- The addition of path-cost increases overall performance w.r.t. end-cost for all the approaches and algorithms considered.

- A distributed control with features relative to the agents is able to generalize to completely new, larger and more complex scenarios.

5.3 Future work

There are several aspects that have not been covered within this work and some more that have opened to keep working on.

It would be interesting to do a bit of feature interpretation, consisting on the analysis of the policy parameters θ weights in order to see what are the most relevant features ϕ and understand the way the control takes the actions.

We would like to find a way to better scale the centralized control to systems including several agents as it supposes a strong limitation for real applications. Besides, an interesting experiment would be to try to reproduce the results obtained in this simple scenario with two agents training in larger and more complex ones and see whether we are capable of hunting the stags.

Another aspect to consider is doing a better analysis of the distributed control generalization in order to see whether cooperation is preserved or whether the stags mislead the hunters that are found alone as they are used to having a partner during training, which would be related to the feature analysis. Furthermore, there should be done some testing to find training methods that allow for better control generalization, being it training with a smart initial state, train in different initial states, etc. Apart from an analysis of overfitting vs generalization training in different game configurations as a function of the agent vision range r.

Finally, the distributed control could be brought a step forward and have it learn individual policies for each agent or simplify the state representation by, for example, losing resolution on the prey position by applying an encoding method similar to the one tracking the hunter positions with RBFs to avoid scaling as $\sim 2r^2$.

List of Figures

List of Tables

Bibliography

[1] Pitman, R. L. & Durban, J. W. Cooperative hunting behavior, prey selectivity and prey handling by pack ice killer whales (orcinus orca), type b, in antarctic peninsula waters. *Marine Mammal Science* **28**, 16–36 (2012).

[2] Mlot, N. J., Tovey, C. A. & Hu, D. L. Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences* **108**, 7669–7673 (2011).

[3] Lewis, D. *Convention: A philosophical study* (John Wiley & Sons, 2008).

[4] Clark, H. H. & Wilkes-Gibbs, D. Referring as a collaborative process. *Intentions in communication* 463–493 (1990).

[5] Garrod, S. & Anderson, A. Saying what you mean in dialogue: A study in conceptual and semantic co-ordination. *Cognition* **27**, 181–218 (1987).

[6] Caldwell, C. A. & Smith, K. Cultural evolution and perpetuation of arbitrary communicative conventions in experimental microsocieties. *PloS one* **7**, e43807 (2012).

[7] Centola, D. & Baronchelli, A. The spontaneous emergence of conventions: An experimental study of cultural evolution. *Proceedings of the National Academy of Sciences* **112**, 1989–1994 (2015).

[8] Helbing, D., Schönhof, M., Stark, H.-U. & Hołyst, J. A. How individuals learn to take turns: Emergence of alternating cooperation in a congestion game and the prisoner's dilemma. *Advances in Complex Systems* **8**, 87–116 (2005).

[9] Friedman, D. & Oprea, R. A continuous dilemma. *American Economic Review* **102**, 337–63 (2012).

[10] Hawkins, R. X. & Goldstone, R. L. The formation of social conventions in real-time environments. *PloS one* **11**, e0151670 (2016).

[11] Freire, I. T., Moulin-Frier, C., Sanchez-Fibla, M., Arsiwalla, X. D. & Verschure, P. Modeling the formation of social conventions in multi-agent populations. *arXiv preprint arXiv:1802.06108* (2018).

[12] Perolat, J. *et al.* A multi-agent reinforcement learning model of common-pool resource appropriation. In *Advances in Neural Information Processing Systems*, 3643–3652 (2017).

[13] Skyrms, B. *The stag hunt and the evolution of social structure* (Cambridge University Press, 2004).

[14] Kappen, H. J., Gómez, V. & Opper, M. Optimal control as a graphical model inference problem. *Machine learning* **87**, 159–182 (2012).

[15] Ziebart, B. D. Modeling purposeful adaptive behavior with the principle of maximum causal entropy (2010). PhD thesis.

[16] Toussaint, M. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, 1049–1056 (ACM, 2009).

[17] Todorov, E. General duality between optimal control and estimation. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 4286–4292 (IEEE, 2008).

[18] Todorov, E. Efficient computation of optimal actions. *Proceedings of the national academy of sciences* **106**, 11478–11483 (2009).

[19] Todorov, E. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, 1369–1376 (2007).

[20] Kappen, H. J. Linear theory for control of nonlinear stochastic systems. *Physical review letters* **95**, 200201 (2005).

[21] Jonsson, A. & Gómez, V. Hierarchical linearly-solvable markov decision problems. In *26th International Conference on Automated Planning and Scheduling*, ICAPS'16, 193–201 (AAAI Press, 2016).

[22] Todorov, E. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, 1856–1864 (2009).

[23] Neu, G. & Gómez, V. Fast rates for online learning in Linearly Solvable Markov Decision Processes. In *Proceedings of the 2017 Conference on Learning Theory*, vol. 65 of *Proceedings of Machine Learning Research*, 1567–1588 (PMLR, 2017).

[24] Williams, G., Aldrich, A. & Theodorou, E. A. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics* **40**, 344–357 (2017).

[25] Gómez, V., Thijssen, S., Symington, A. C., Hailes, S. & Kappen, H. J. Real-time stochastic optimal control for multi-agent quadrotor systems. In *26th International Conference on Automated Planning and Scheduling* (2016).

[26] Van Den Broek, B., Wiegerinck, W. & Kappen, B. Graphical model inference in optimal control of stochastic multi-agent systems. *Journal of Artificial Intelligence Research* **32**, 95–122 (2008).

[27] Skyrms, B. *Evolution of the social contract* (Cambridge University Press, 2014).

[28] Yoshida, W., Dolan, R. J. & Friston, K. J. Game theory of mind. *PLoS computational biology* **4**, e1000254 (2008).

[29] Yoshida, W., Seymour, B., Friston, K. J. & Dolan, R. J. Neural mechanisms of belief inference during cooperative games. *Journal of Neuroscience* **30**, 10744–10751 (2010).

[30] Yoshida, W. *et al.* Cooperation and heterogeneity of the autistic mind. *Journal of Neuroscience* **30**, 8815–8818 (2010).

[31] Osborne, M. J. & Rubinstein, A. *A course in game theory* (MIT press, 1994).

[32] Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909* (2018).

[33] Lauritzen, S. L. & Spiegelhalter, D. J. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)* 157–224 (1988).

[34] Jordan, M. I. *Learning in graphical models*, vol. 89 (Springer Science & Business Media, 1998).

[35] Murphy, K. P., Weiss, Y. & Jordan, M. I. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 467–475 (Morgan Kaufmann Publishers Inc., 1999).

[36] Matsubara, T., Gómez, V. & Kappen, H. J. Latent Kullback Leibler control for continuous-state systems using probabilistic graphical models. *30th Conference on Uncertainty in Artificial Intelligence (UAI)* (2014).

[37] Thalmeier, D., Gómez, V. & Kappen, H. J. Action selection in growing state spaces: control of network structure growth. *Journal of Physics A: Mathematical and Theoretical* **50**, 034006 (2017).

[38] Gómez, V., Kappen, H. J., Peters, J. & Neumann, G. Policy search for path integral control. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **8724 LNAI**, 482–497 (2014).

[39] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* **8**, 229–256 (1992).

[40] Kappen, H. J. & Ruiz, H. C. Adaptive importance sampling for control and inference. *Journal of Statistical Physics* **162**, 1244–1266 (2016).