UNIVERSITY OF CALIFORNIA
Los Angeles

Algorithms for Optimal Paths of One, Many, and an Infinite Number of Agents

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Alex Tong Lin

2020

ABSTRACT OF THE DISSERTATION

Algorithms for Optimal Paths of One, Many, and an Infinite Number of Agents

by

Alex Tong Lin
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2020
Professor Stanley J. Osher, Chair

In this dissertation, we provide efficient algorithms for modeling the behavior of a single agent, multiple agents, and a continuum of agents. For a single agent, we combine the modeling framework of optimal control with advances in optimization splitting in order to efficiently find optimal paths for problems in very high-dimensions, thus providing alleviation from the curse of dimensionality. For a multiple, but finite, number of agents, we take the framework of multi-agent reinforcement learning and utilize imitation learning in order to decentralize a centralized expert, thus obtaining optimal multi-agents that act in a decentralized fashion. For a continuum of agents, we take the framework of mean-field games and use two neural networks, which we train in an alternating scheme, in order to efficiently find optimal paths for high-dimensional and stochastic problems. These tools cover a wide variety of use-cases that can be immediately deployed for practical applications.

The dissertation of Alex Tong Lin is approved.

Christopher R. Anderson

Joseph M. Teran

Luminita A. Vese

Stanley J. Osher, Committee Chair

University of California, Los Angeles

2020

To my parents and my brothers.

Table of Contents

List of Figures

Acknowledgments

I'd like to thank my advisor Professor Stanley J. Osher for his immense generosity in helping me throughout my PhD studies. I am also enormously grateful to all my collaborators, especially: Guido Montúfar who has also been immensely generous, and took me under his wing as a second advisor, Wuchen Li for his generosity of ideas and his infinite ability to raise my confidence as a researcher, and Gary Hewer for his insightful ideas and looking out for me. I'd also like to thank the members of my committee for their kind suggestions. And finally, I'd like to thank my parents and my brothers.

Curriculum Vitae

2009 – 2012	B.S. in Mathematics, Highest Honors, University of California, Santa Barbara (UCSB).
2017	Masters, Mathematics, University of California, Los Angeles (UCLA).
2014 – Present	PhD Student, Mathematics, University of California (UCLA).

Publications

[1] Alex Tong Lin, Yat-Tin Chow, and Stanley J. Osher, "A Splitting Method for Overcoming the Curse of Dimensionality in Hamilton-Jacobi Equations Arising from Nonlinear Optimal Control and Differential Games with Applications to Trajectory Generation," (2018), journal Communications in Mathematical Sciences, **16**, (2018), arXiv:1803.01215 [oc] .

[2] Alex Tong Lin, Mark J. Debord, Katia Estabridis, Gary Hewer, Guido Montúfar, and Stanley J. Osher, "CESMA: Centralized Expert Supervises Multi-Agents," (2019), arXiv:1902.02311 [ma] .

[3] Alex Tong Lin, Samy W. Fung, Wuchen Li, Levon Nurbekyan, and Stanley J. Osher, "APAC-Net: Alternating the Population and Agent Control via Two Neural Networks to Solve High-Dimensional Stochastic Mean Field Games," (2020), arXiv:2002.10113 [lg] .

[4] Yonatan Dukler*, Wuchen Li*, Alex Tong Lin*, and Guido Montúfar* "Wasserstein of Wasserstein Loss for Learning Generative Models," (2019), International Conference of Machine Learning (ICML), **97**, (2019), PMLR 97:1716-1725 *Equal contribution .

[5] Alex Tong Lin, Yonatan Dukler, Wuchen Li, and Guido Montúfar "Wasserstein Diffusion Tikhonov Regularization," (2019), Neural Information Processing Systems (NeurIPS) Optimal Transport Workshop, (2019), arXiv:1909.0686 [lg] .

[6] Wuchen Li, Alex Tong Lin, and Guido Montúfar "Affine Natural Proximal," (2019), International Conference on Geometric Science of Information (GSI), (2019), Springer:978-3-030-26980-7

CHAPTER 0

Introduction

This dissertation provides efficient algorithms for optimal paths for one, many, and an infinite number of agents. We first examine the case of a singular agent by introducing an optimization splitting algorithm that efficiently solves optimal control problems, with the novelty of being *effective at both high-dimensional and space-time-dependent* cases. We then turn our attention to a multiple, but finite number of agents, i.e. multi-agents, by solving multi-agent reinforcement learning problems with CESMA (Centralized Expert Supervises Multi-Agents), which obtains decentralized multi-agent controllers from a centralized expert. Finally, we then examine the case of a continuum of agents by efficiently *solving high-dimensional and stochastic mean-field games* problems with an algorithm called APAC-Net (Alternating the Population and Control Neural Network), allowing us to model the behavior of a mass of interacting agents.

A single agent interacting with its environment can be effectively modeled with the theory of optimal control, where we seek the best control law that minimizes a given cost functional. In order to efficiently do this for high-dimensional and space-time-dependent problems, we take advances in optimization splitting and apply them to the general optimal control problem. The idea is to first discretize the problem in the time variable, and then turn the constrained minimization problem into an unconstrained saddle-point problem. Then we make use of the convex conjugate to conjure the Hamiltonian, and after renaming variables, we end up with an expression that is amenable to techniques from optimization splitting. In this way, instead of using grids – whose size grows exponentially with dimensions, and thus making computation intractable in higher dimensions – we calculate and adjust the trajectory curves of the control problem until an optimum is found. When considering more

than one agent, trajectories can get quite complicated as now each agent must now consider the actions of other agents.

For multiple agents, we view the problem in the Multi-agent Reinforcement Learning (MARL) setting, which is an extension of Reinforcement Learning (RL). Firstly, the goal of Reinforcement Learning is to find the best policy for an agent interacting with its environment that best maximizes a reward functional. The key difference between RL and Optimal Control is that in RL the agent does not have a model of the environment, and must use sample trajectories in order to find the best policy. MARL extends this framework to the multi-agent setting where now the goal is to find a policy for each agent that either maximizes an overall reward – such as in the cooperative setting – or finds an equilibrium. We introduce an algorithm called CESMA (Centralized Expert Supervises Multi-Agents) which first trains a centralized expert to solve the multi-agent problem, and then we use imitation learning to obtain decentralized multi-agent policies, the goal of MARL. This training procedure is effective for a finite number of agents, but when one wants to consider a tremendously large number of agents where infinite makes a great approximation, then we need to start considering densities of agents.

For a continuum of homogeneous agents, where dealing with a mass rather than individual particles is most appropriate, the field of Mean-Field Games (MFG) is perfectly suited as a framework. The MFG problem is a system of two PDEs – a Hamilton-Jacobi equation that acts as a value function for an individual agent, and a continuity equation that describes how the mass of agents move. In order to compute high-dimensional and stochastic MFGs, we avoid the use of grids, similar to our work in optimal control, and we reformulate the problem so that we are computing and adjusting characteristic curves. This is now done using two neural networks – one that computes the solution to the Hamilton-Jacobi PDE and another to compute the solution to the continuity equation – and they are trained in an adversarial fashion. From this, high-dimensional and stochastic MFG problems are now efficiently computable.

In this dissertation, we present efficient algorithms for modeling optimal behavior of one,

many, and an infinite number of agents. For the single agent case, we use advances in optimization to solve *high-dimensional and space-time-dependent* optimal control problems. For a multiple but finite number of agents, we use imitation learning and apply it to multi-agent reinforcement learning to obtain decentralized policies from a centralized expert. For an infinite number of agents, we solve *high-dimensional and stochastic* Mean-Field Games problems by training two neural networks in an adversarial fashion. These trio of algorithms are all novel, efficient, and easy-to-use for immediate application to real-world problems.

CHAPTER 1

One

Abstract: Recent observations have bridged splitting methods arising from optimization, to the Hopf and Lax formulas for Hamilton-Jacobi Equations. This has produced extremely fast algorithms in computing solutions of these PDEs. More recent observations were made in generalizing the Hopf and Lax formulas to state-and-time-dependent cases. In this article, we apply a new splitting method based on the Primal Dual Hybrid Gradient algorithm (a.k.a. Chambolle-Pock) to nonlinear optimal control and differential games problems, based on techniques from the derivation of the new Hopf and Lax formulas, which allow us to compute solutions at specified points directly, i.e. without the use of grids in space. This algorithm also allows us to create trajectories directly. Thus we are able to lift the curse of dimensionality a bit, and therefore compute solutions in much higher dimensions than before. And in our numerical experiments, we actually observe that our computations scale polynomially in time. Furthermore, this new algorithm is embarrassingly parallelizable. [83]

1.1 Introduction

Hamilton-Jacobi Equations (HJE) are crucial in solving and analyzing problems arising from optimal control, differential games, dynamical systems, calculus of variations, quantum mechanics, and the list goes on [42, 98].

Most methods to compute HJE use grids and finite-difference discretization. Some of these methods use ENO/WENO-type methods [100], and others use Dijkstra-type methods [32] such as fast marching [131] and fast sweeping [130]. But due to their use of grids, they suffer from the curse of dimensionality, i.e. they do not scale well with increases in dimension

in the space variable, i.e. they generally scale exponentially.

In past years, there has been an effort to mitigate the effects of dimensionality on computations of HJE. Some recent attempts to solve Hamilton-Jacobi equations use methods from low rank tensor representations [66], or methods based on alternating least squares [120], or methods by sparse grids [72], or methods using pseudospectral [111] and iterative methods [71]. There have also been attempts to mitigate the curse of dimensionality which have been motivated by reachability [5, 91]. In this work, we examine and advertize the effectiveness of splitting to solve Hamilton-Jacobi equations and to directly compute optimal trajectories.

We note that splitting for optimal control problems was used by [95] (2013), where they applied it to cost functionals with a quadratic and convex term. In terms of Hamilton-Jacobi equations, Kirchner et al. [74] (2018) have effectively applied PDHG [139, 38] (a.k.a. Chambolle-Pock [16]) to Hamilton-Jacobi equations arising from linear optimal control problems. They applied splitting to the Hopf formula to compute HJE for bounded input, high-dimensional linear control problems. Another main feature of their methods is they are able to directly generate optimal trajectories by making use of the the closed-form solution to linear ODEs. See also previous work by Kirchner et al. [73] where they apply the Hopf formula to a differential games problems, which resulted in complex “teaming" behavior even under linearized pursuit-evasion models.

In this current paper, we have worked in parallel with the above authors and have also applied splitting to Hamilton-Jacobi equations arising from nonlinear problems. Our volunteered method has some nice properties: (1) relatively quick computations of solutions in high dimensions (see 1.6.4, although one can easily extend to 100 dimensions for example, and also see 1.6.1.3 where we observe a linear relationship between computation time and dimension), especially when we include parallelization, the method is embarrassingly parallelizable [64], (2) the ability to *directly generate* optimal trajectories of the optimal control/differential games problems, (3) the ability to compute problems with non-linear ODE dynamics, (4) the ability to compute solutions for *nonconvex and nonsmooth* Hamiltonians and initial conditions, (5) the ease of parallelization of our algorithm to compute solutions

to HJE, i.e. each core can use the algorithm to compute the solution at a point, so given N cores we can compute solutions of the HJE at N points simultaneously, and (6) the ease of parallelization to directly compute trajectories, i.e. in our discretization of the time, we can parallelize by assigning each computational core a point in the time discretization.

Our work lies in using the techniques used to derive the Generalized Hopf and Lax formulas introduced by Y.T. Chow, J. Darbon, S. Osher, and W. Yin [23], which generalize to the state-and-time-dependent cases (note in the literature that the classical Lax formula is sometimes called the Hopf-Lax formula). See also previous work from the same authors, [19, 20], and also [27, 28] where they provide fast algorithms under convexity assumptions. To perform the optimization, we use a new splitting method that is based on the Primal Dual Hybrid Gradient (PDHG) method (a.k.a. Chambolle-Pock), which we *conjecture* to both converge to a local minimum for most well-behaved problems, and which we *conjecture* to also approximate the solution. To do this, we discretize the optimal control problem and the differential games problem in time, a technique inspired by [95] and [23]. This new splitting method has been experimentally seen (1.6) to be faster than the using coordinate-descent in most cases, which the authors in [23] use to compute the solutions.

As far as the authors know, the use of splitting as applied to minimax differential games problems, mainly on the state-and-time dependent equation (1.13) and (1.14), is new. In this case, we seem to be able to compute HJE with nonconvex Hamiltonians and nonconvex initial data (1.6.3.3), although they do have the structure of being convex-concave.

The paper is organized as follows:

- 1.2 Gives brief overviews of Hamilton-Jacobi Equations and its intimate connections to optimal control 1.2.1 and differential games 1.2.2.
- 1.3 Gives a brief overview of splitting methods from optimization, focusing on ADMM 1.3.1 and PDHG 1.3.2.
- 1.4 Presents the generalized Lax and Hopf formulas for optimal control and differential games that were conjectured by [23]. We also go through its discretization in 1.4.1,

which is the basis of our algorithm.

- 1.5 **Presents the main algorithms.**
- 1.6 Presents various computational examples.
- 1.7 Ends with a brief conclusion, and a discussion on future work.
- 1.9 Gives a more in-depth explanation on how to use the algorithms.

1.2 Hamilton-Jacobi Equations and Its Connection to Optimal Control and Differential Games

1.2.1 Hamilton-Jacobi Equations and Optimal Control

Most of our exposition on optimal control will follow [41], and also [40] (Chapter 10).

The goal of optimal control theory is to find a control policy that will drive a system while optimizing a criterion. Given an initial point $x \in \mathbb{R}^n$ and an initial time $t \in [0, T]$, where T is some fixed end-point time, the system will obey an ODE

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \boldsymbol{\alpha}(s), s), \quad t < s < T \\ \mathbf{x}(t) = x \end{cases}$$

where $\mathbf{f} : (\mathbb{R}^n \times A \times \mathbb{R}) \to \mathbb{R}$, where $A \subseteq \mathbb{R}^m$. We call $\mathbf{x}$ the *state*, and $\boldsymbol{\alpha}$ the *control*. And the functional we want to optimize is $J_{x,t} : \mathcal{A} \to \mathbb{R}$ where

$$J_{x,t}[\boldsymbol{\alpha}] := g(\mathbf{x}(T)) + \int_t^T L(\mathbf{x}(s), \boldsymbol{\alpha}(s), s)\, ds. \tag{1.1}$$

and where $\mathcal{A} := \{\boldsymbol{\alpha} : [t, T] \to A\}$ is some *admissable control* set, and $g : \mathbb{R}^n \to \mathbb{R}$ and $L : (\mathbb{R}^n \times A \times \mathbb{R}) \to \mathbb{R}$. We can either *minimize* the above functional, in which we call it a *cost*, or we can *maximize* it, in which we call it a *payoff*. For our exposition, we will choose to minimize $J_{x,t}[\cdot]$, so it will be a cost. Then we define the *value function*

$$\phi(x, t) = \min_{\alpha(\cdot) \in \mathcal{A}} J_{x,t}[\boldsymbol{\alpha}]. \tag{1.2}$$

Under some mild conditions on $\mathbf{f}$, g, and L, this value function will satisfy the *terminal-valued* Hamilton-Jacobi PDE (HJ PDE)

$$\begin{cases} \partial_t \phi(x,t) + H(x, \nabla_x \phi(x,t), t) &= 0, \qquad (x,t) \in \mathbb{R}^n \times (0,T) \\ \phi(x,T) &= g(x). \end{cases}$$

where $H(x,p,t) = \min_{a \in A} \{\langle \mathbf{f}(x,a,t), p\rangle + L(x,a,t)\}$.

To get an initial-valued PDE, can make a change of variables $t \to T - t$. Or equivalently we can reformulate the optimal control problem "backwards in time" so that we have

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \boldsymbol{\alpha}(s), s), \quad 0 < s < t \\ \mathbf{x}(t) = x \end{cases}$$

and

$$J_{x,t}[\boldsymbol{\alpha}] := g(\mathbf{x}(0)) + \int_0^t L(\mathbf{x}(s), \boldsymbol{\alpha}(s), s)\, ds.$$

Then our $\phi(x,t) = \min_{\alpha(\cdot) \in \mathcal{A}} J_{x,t}[\boldsymbol{\alpha}]$ will satisfy an *initial-valued* HJ PDE with $H(x,p,t) = \max_{a \in A} \{\langle \mathbf{f}(x,a,t), p\rangle - L(x,a,t)\}$. Note that if $\mathbf{f} = a$, then this form of the Hamiltonian expresses H as the convex conjugate [103] of L.

If we think from a physical perspective in which time moves forward, the first formulation feels more comfortable. If we come from the fields of PDE or mathematical optimization, the latter formulation will feel more comfortable.

So how does having a HJ PDE help us synthesize an optimal control? Using the first formulation as it feels more physically intuitive, we can heuristically argue that given an inital time $t \in (0,T]$ and a state $x \in \mathbb{R}^n$, we consider the optimal ODE

$$\begin{cases} \dot{\mathbf{x}}^*(s) = \mathbf{f}(\mathbf{x}^*(s), \boldsymbol{\alpha}^*(s), s), \quad t < s < T \\ \mathbf{x}^*(t) = x \end{cases}$$

where at each time $s \in (0,T)$, we choose the value of $\alpha^*(s)$ to be such that

$$\begin{aligned} &\langle \mathbf{f}(\mathbf{x}^*(s), \boldsymbol{\alpha}^*(s), s), \partial_x \phi(\mathbf{x}^*(s), s)\rangle + L(\mathbf{x}^*(s), \boldsymbol{\alpha}^*(s), s) \\ &\quad = \min_{a \in A} \{\langle \mathbf{f}(\mathbf{x}^*(s), a, s), \partial_x \phi(\mathbf{x}^*(s)\rangle + L(\mathbf{x}^*(s), a, s)\} \\ &\quad = H(\mathbf{x}^*(s), \partial_x \phi(\mathbf{x}^*(s), s), s) \end{aligned}$$

We call $\boldsymbol{\alpha}^*(\cdot)$ defined in this way as the *feedback control*, and this can be obtained from $\nabla_x \phi$ (see Section 10.3.3 of [40]). This is also related to Pontryagin' Maximum Principle (Chapter 4 of [41]).

Note that in the case $H(x,p,t) = H(p)$, then we have available the (classical) Hopf and Lax formulas which are expressions for the solutions $\phi(x,t)$ of the HJ PDE:

When the Hamiltonian $H(p)$ is convex and the initial date g is (uniformly) Lipschitz continuous, then we have the Lax formula:

$$\phi(x,t) = \min_{y \in \mathbb{R}^n} \left\{ g(y) + tH^* \left(\frac{x-y}{t} \right) \right\}$$

where $H^*(x) = \max_{v \in A} \{\langle v, x \rangle - H(v)\}$ is the convex conjugate of L.

And if the initial data g is (uniformly) Lipschitz continuous and convex, and H is continuous, then we have the Hopf formula,

$$\phi(x,t) = \sup_{y \in \mathbb{R}^n} \{-g^*(y) + \langle y, x \rangle - tH(y)\} = (g^*(y) + tH(y))^* \tag{1.3}$$

where g^* is the convex conjugate of g. Note the last equality implies the solution is convex in x. We note again that the argument minimum of the above expression is in-fact $\nabla_x \phi(x,t)$.

1.2.2 Hamilton-Jacobi Equations and Differential Games

Our exposition of differential games will follow [41] (Chapter 6), but also see [67, 138]. In the field of differential games, we restrict our exposition to *two-person, zero-sum differential games*. Let an initial point $x \in \mathbb{R}^n$ and an initial time $t \in [0,T]$ be given, where T is some fixed endpoint time. A two-person, zero-sum differential game will have the dynamics,

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s), \quad t < s < T \\ \mathbf{x}(t) = x \end{cases}$$

where $\mathbf{f} : (\mathbb{R}^n \times A \times B \times \mathbb{R}) \to \mathbb{R}$, and where $A \subseteq \mathbb{R}^m$ and $B \subseteq \mathbb{R}^\ell$. The control $\boldsymbol{\alpha}$ is the control for player I, and the control $\boldsymbol{\beta}$ is the control for player II. The functional will be $J_{x,t} : A(t) \times B(t) \to \mathbb{R}$ where

$$J_{x,t}[\boldsymbol{\alpha}, \boldsymbol{\beta}] := g(\mathbf{x}(T)) + \int_t^T L(\mathbf{x}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s)\, ds. \tag{1.4}$$

and where $A(t) := \{\boldsymbol{\alpha} : [t,T] \to A\}$ and $B(t) := \{\boldsymbol{\beta} : [t,T] \to B\}$ are *admissable control* sets, and $g : \mathbb{R}^n \to \mathbb{R}$ and $L : (\mathbb{R}^n \times A \times B \times \mathbb{R}) \to \mathbb{R}$.

In order to model that at each time, neither player has knowledge of the other's future moves, we use a concept of strategy that was used by Varaiya [133] as well as Elliot and Kalton [37]. This idea allows us to model that each player will select a control in response to all possible controls the opponent can select.

A *strategy* for player I is a mapping $\Phi : B(t) \to A(t)$ such that for all times $s \in [t,T]$

$$\tau \in [t,s], \quad \boldsymbol{\beta}(\tau) \equiv \hat{\boldsymbol{\beta}}(\tau) \qquad \text{implies} \qquad \Phi[\boldsymbol{\beta}](\tau) \equiv \Phi[\hat{\boldsymbol{\beta}}](\tau)$$

The $\Phi[\boldsymbol{\beta}]$ models player I's response to player II selecting $\boldsymbol{\beta}$. We similarly define a strategy $\Psi : A(t) \to B(t)$ for player II:

$$\tau \in [t,s], \quad \boldsymbol{\alpha}(\tau) \equiv \hat{\boldsymbol{\alpha}}(\tau) \qquad \text{implies} \qquad \Psi[\boldsymbol{\alpha}](\tau) \equiv \Psi[\hat{\boldsymbol{\alpha}}](\tau)$$

and $\Psi[\boldsymbol{\alpha}]$ models player II's response to player I selecting $\boldsymbol{\alpha}$.

Letting $\mathcal{A}(t)$ and $\mathcal{B}(t)$ be the set of strategies for player I and player II, respectively, then we define the *lower value function* as

$$\phi^-(x,t) = \inf_{\Psi[\cdot] \in \mathcal{B}(t)} \sup_{\boldsymbol{\alpha}(\cdot) \in A(t)} J_{x,t}[\boldsymbol{\alpha}, \Psi[\boldsymbol{\alpha}]] \tag{1.5}$$

and the *upper value function* as

$$\phi^+(x,t) = \sup_{\Phi[\cdot] \in \mathcal{A}(t)} \inf_{\boldsymbol{\beta}(\cdot) \in B(t)} J_{x,t}[\Phi[\boldsymbol{\beta}], \boldsymbol{\beta}]. \tag{1.6}$$

Note that we always have $\phi^-(x,t) \leqslant \phi^+(x,t)$ for all $x \in \mathbb{R}^n$ and $t \in [0,T]$. For a proof, see [42].

These value functions satisfy the *terminal-valued* HJ PDEs

$$\begin{cases} \partial_t \phi^-(x,t) + \max_{a \in A} \min_{b \in B} \{\langle \mathbf{f}(x,a,b,t), \nabla_x \phi^-(x,t) \rangle + L(x,a,b,t)\} = 0 \\ \phi^-(x,T) = g(x) \end{cases}$$

and

$$\begin{cases} \partial_t \phi^+(x,t) + \min_{b \in B} \max_{a \in A} \{\langle \mathbf{f}(x,a,b,t), \nabla_x \phi^+(x,t) \rangle + L(x,a,b,t)\} = 0 \\ \phi^+(x,T) = g(x) \end{cases}$$

where we have the *lower PDE Hamiltonian*

$$H^-(x,p,t) = \max_{a\in A}\min_{b\in B}\{\langle \mathbf{f}(x,a,b,t),p\rangle + L(x,a,b,t)\}$$

and the *upper PDE Hamiltonian*

$$H^+(x,p,t) = \min_{b\in B}\max_{a\in A}\{\langle \mathbf{f}(x,a,b,t),p\rangle + L(x,a,b,t)\}$$

In general, we have

$$\max_{a\in A}\min_{b\in B}\{\langle \mathbf{f}(x,a,b,t),p\rangle + L(x,a,b,t)\} \leqslant \min_{b\in B}\max_{a\in A}\{\langle \mathbf{f}(x,a,b,t),p\rangle + L(x,a,b,t)\}$$

and in most cases the inequality is strict, and thus the lower and upper value functions are different. But when the above is an equality, then the game is said to satisfy the *minimax conditions*, also called *Isaac's condition*, and we have $\phi^- = \phi^+$, and we say the game has *value*.

Our examples will focus on differential games which satisfy the minimax condition, and will thus have value.

In differential games, we usually run into nonconvex Hamiltonians, so it is the Hopf formula (1.3) that is used the most.

1.3 Splitting Algorithms from Optimization

Here we review a couple of splitting algorithms from optimization.

1.3.1 ADMM (Alternating Method of Multipliers)

ADMM [8], which is also known as Split-Bregman [54], is an optimization method to solve problems of the following form:

$$\begin{aligned} \min_{x,z\in X} \quad & f(x)+g(z) \\ \text{subject to} \quad & Ax+Bz=c \end{aligned}$$

where X is a finite-dimensional real vector space equipped with an inner product $\langle \cdot, \cdot \rangle$, and $f : X \to \mathbb{R}$ and $g : X \to \mathbb{R}$ are proper, convex, lower semicontinuous functions. We also have that A and B are continuous linear operators (e.g. matrices), with c a fixed element in X. Now we form the *augmented Lagrangian* of the above problem:

$$L_\rho(x, z, y) = f(x) + g(z) + \langle y, Ax + Bz - c \rangle + \frac{\rho}{2} \|Ax + Bz - c\|_2^2$$

Then we alternately minimize:

$$\begin{cases} x^{k+1} & = \arg\min_x L_\rho(x, z^k, y^k) \\ z^{k+1} & = \arg\min_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} & = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{cases}$$

where in the last step we update the dual variable. Note that the arg min expressions are frequently precisely the *proximal operator* [103] of a (not necessarily convex) function. The proximal operator is defined as: Given $f : \mathbb{R}^n \to \mathbb{R}$ a proper l.s.c. function, not necessarily convex, then,

$$(I + \lambda \partial f)^{-1}(v) := \arg\min_x \left\{ f(x) + \frac{1}{2\lambda} \|x - v\|_2^2 \right\} \tag{1.7}$$

The proximal of f with step-size λ is also denoted $\text{prox}_{\lambda f}(\cdot)$.

1.3.2 PDHG (Primal-Dual Hybrid Gradient)

The PDHG algorithm [139, 38], which also goes by the name Chambolle-Pock [16], attempts to solve problems of the form

$$\min_{x \in X} \quad f(Ax) + g(x)$$

where we make similar assumptions on X, f, g, and A as we did for ADMM. PDHG takes the Lagrangian dual formulation of the above problem and seeks to find a saddle point of the following problem:

$$\min_{x \in X} \max_{y \in Y} \langle Ax, y \rangle + g(x) - f^*(y)$$

where $f^*(y) = \sup_{x \in X} \{\langle x, y \rangle - f(y)\}$ is the *convex conjugate* of f. PDHG is also an alternating minimization technique that makes use of proximal operators. The updates are:

$$\begin{cases} y^{k+1} &= (I + \sigma \partial f^*)^{-1}(y^k + \sigma A \bar{x}^k) \\ x^{k+1} &= (I + \tau \partial g)^{-1}(x^k - \sigma A^* y^{k+1}) \\ \bar{x}^{k+1} &= x^{k+1} + \theta(x^{k+1} - x^k). \end{cases}$$

where $\sigma, \tau > 0$ are such that $\sigma\tau\|A\|^2 < 1$, and $\theta \in [0, 1]$, although $\theta = 1$ seems to work best in practice.

1.4 The Generalized Lax and Hopf formulas

A recent result by Y.T. Chow, J. Darbon. S. Osher, and W. Yin [23] gives a conjectured generalization to the Lax and Hopf formulas. Given a Hamilton-Jacobi Equation,

$$\begin{cases} \partial_t \phi + H(x, \nabla_x \phi(x,t), t) &= 0, \quad \text{in } \mathbb{R}^d \times (0, \infty), \\ \phi(x, 0) &= g(x). \end{cases}$$

we have that when $H(x, p, t)$ is smooth, and convex with respect to p, and possibly under some more mild conditions, we have

$$\phi(x,t) = \min_{v \in \mathbb{R}^d} \left\{ g(\mathbf{x}(0)) + \int_0^t \mathbf{p}(s) \cdot \nabla_p H(\mathbf{x}(s), \mathbf{p}(s), s) - H(\mathbf{x}(s), \mathbf{p}(s), s)\, ds \right\}$$

$$\text{where} \begin{cases} \dot{\mathbf{x}}(s) &= \nabla_p H(\mathbf{x}(s), \mathbf{p}(s)) \\ \dot{\mathbf{p}}(s) &= -\nabla_x H(\mathbf{x}(s), \mathbf{p}(s)) \\ \mathbf{x}(t) &= x \\ \mathbf{p}(t) &= v \end{cases}$$

where $\mathbf{x}$ and $\mathbf{p}$ are the characteristics of the PDE. The expression in the bottom braces are ODEs which $\mathbf{x}(\cdot)$ and $\mathbf{p}(\cdot)$ satisfy.

And when we move the convexity onto g, i.e. when $H(x, p, t)$ is smooth and g is convex,

then

$$\varphi(x,t) = \sup_{v\in\mathbb{R}^d}\left\{-g^\star(\mathbf{p}(0)) + x\cdot v + \int_0^t \mathbf{x}(s)\cdot\nabla_x H(\mathbf{x}(s),\mathbf{p}(s),s) - H(\mathbf{x}(s),\mathbf{p}(s),s)\,ds\right\}$$

$$\text{where}\begin{cases}\dot{\mathbf{x}}(s) &= \nabla_p H(\mathbf{x}(s),\mathbf{p}(s))\\ \dot{\mathbf{p}}(s) &= -\nabla_x H(\mathbf{x}(s),\mathbf{p}(s))\\ \mathbf{x}(t) &= x\\ \mathbf{p}(t) &= v\end{cases}$$

Chow, Darbon, Osher, and Yin used coordinate descent with multiple initial guesses to perform the optimization. They do this by first making an initial guess for $v \in \mathbb{R}^d$, then they compute the ODEs, and then compute the value of the objective, i.e. the first lines of the two formulas. Then they re-adjust one coordinate $v \in \mathbb{R}^d$ and repeat. Details can be found in their paper [23].

1.4.1 Discretizing the Generalized Lax and Hopf Formulas for Optimal Control

In order to derive the generalized Lax and Hopf formulas, we can first discretize the value function of the optimal control problem (1.1) and (1.2). Before we begin we note we are merely making formal calculations, much in the spirit of E. Hopf in his seminal paper where he derived the classical Hopf formula [65]. This is the procedure followed in [23]: We have the value function equals

$$\phi(x,t) = \min_{\mathbf{x}(\cdot),\mathbf{u}(\cdot)}\left\{g(\mathbf{x}(0)) + \int_0^t L(\mathbf{x}(s),\mathbf{u}(s),s)\,ds\right\}$$

where $\mathbf{x}(\cdot)$ and $\mathbf{u}(s)$ satisfy the ODE

$$\begin{cases}\dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s),\mathbf{u}(s),s), \quad 0 < s < t\\ \mathbf{x}(t) = x\end{cases}$$

Two notes: (1) we are formulating our optimal control problem "backwards in time" so that we end up with an *initial-valued* HJ PDE, and (2) here (x,t) are fixed points where we want to compute the HJE.

We discretize the time domain such that

$$0 < s_1 < s_2 < \cdots < s_N = t,$$

and we set $x_j = \mathbf{x}(s_j)$ and $u_j = \mathbf{u}(s_j)$. Note in our numerical examples, we make a uniform discretization of the time domain.

Now we use the backward Euler discretization of the ODE and set $x_N = x$ to obtain the optimization problem

$$\min_{\{x_j\}_{j=0}^N, \{u_j\}_{j=1}^N} \left\{ g(x_0) + \delta \sum_{j=1}^N L(x_j, u_j, s_j) \mid \{x_j - x_{j-1} = \delta f(x_j, u_j, s_j)\}_{j=1}^N, x_N = x \right\}$$

As usual in constrained optimization problems, we compute the Lagrangian function (i.e. Lagrange multipliers) to get:

$$g(x_0) + \delta \sum_{j=1}^N L(x_j, u_j, s_j) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f(x_j, u_j, s_j) \rangle + \langle p_N, x - x_N \rangle \tag{1.8}$$

Note that the constraint $x_N = x$ is trivially unneeded in the Lagrangian function. Then we minimize over $\{x_j\}_{j=0}^N$ and $\{u_j\}_{j=1}^N$, while maximizing over $\{p_j\}_{j=1}^N$ to obtain the expression,

$$\max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^{N-1}} \min_{\{u_j\}_{j=1}^N} \left\{ g(x_0) + \delta \sum_{j=1}^N L(x_j, u_j, s_j) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f(x_j, u_j, s_j) \rangle + \langle p_N, x - x_N \rangle \right\}.$$

After moving the minimum over $\{u_j\}_{j=1}^N$ inside, we get

$$\max_{\{p_j\}_{j=1}^N} \min_{\{x_k\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle + \langle p_N, x - x_N \rangle + \delta \sum_{j=1}^N \min_{u_j} \{ L(x_j, u_j, s_j) - \langle p_j, f(x_j, u_j, s_j) \rangle \} \right\}$$

$$= \max_{\{p_j\}_{j=1}^N} \min_{\{x_k\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle + \langle p_N, x - x_N \rangle - \delta \sum_{j=1}^N H(x_j, p_j, s_j) \right\}$$

After the above step, we have now been able to remove a numerical optimization in u by using the definition of the Hamiltonian. This considerably simplifies the problem, and reduces the dimensionality of the optimization.

We note that we need $p_N = 0$ in order for the maximization/minimization to not be infinite. And thus, we can remove the minimization with respect to x^N and we get

$$\phi(x,t) \approx \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^N H(x_j, p_j, s_j) \right\} \tag{1.9}$$

We can do a similar analysis using the forward Euler discretization to obtain,

$$\phi(x,t) \approx \max_{\{p_j\}_{j=0}^{N-1}} \min_{\{x_j\}_{j=0}^{N-1}} \left\{ g(x_0) + \sum_{j=0}^{N-1} \langle p_j, x_{j+1} - x_j \rangle - \delta \sum_{j=0}^{N-1} H(x_j, p_j, s_j) \right\} \tag{1.10}$$

but this latter expression has the disadvantage of coupling the g and H with respect to x_0, as they both depend on x_0. Although this could actually be an advantage as one may have H acting as a regularizer to g.

In order to obtain the discretized version of the generalized Hopf formula, we start with the Lax formula with backward Euler (1.9), and use the linear term $\langle p_1, x_0 \rangle$ and compute the convex conjugate; the calculation goes as follows:

$$\begin{aligned}
&\max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=0}^{N}} \left\{ g(x_0) + \sum_{j=1}^{N-1} \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\} \\
&= \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ \min_{x_0} \{ g(x_0) - \langle p_1, x_0 \rangle \} + \langle p_1, x_1 \rangle + \sum_{j=2}^{N-1} \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\} \\
&= \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ -g^*(p_1) + \langle p_1, x_1 \rangle + \sum_{j=2}^{N-1} \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\} \\
&= \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ -g^*(p_1) + \langle p_N, x \rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\}
\end{aligned}$$

where in the last equality, we performed a summation-by-parts and also used $x_N = x$. So we have the discretized version of the Hopf formula:

$$\phi(x,t) \approx \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ -g^*(p_1) + \langle p_N, x \rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\} \tag{1.11}$$

Note that it is a bit harder to perform the optimization when we approximate the ODE dynamics with forward Euler because we then must compute the convex conjugate of the sum $g(\cdot) + H(\cdot, p_0, s_0)$, which can be more complicated.

1.4.2 Discretizing the Generalized Lax and Hopf Formulas for Differential Games

Again following the procedure of [23], we have a conjectured generalization to the Lax and Hopf formulas for differential games, which we will discretize. Before we give the calculation

we qualify that, in the spirit of E. Hopf when he computed the Hopf formula in his seminal paper [65], these calculations are merely formal:

Given a two-person, zero-sum differential game with value (i.e., it satisfies the Isaacs conditions so that the minmax Hamiltonian and maxmin Hamilton are equal, see section 1.2.2), with given $x \in \mathbb{R}^{d_1}$ and $y \in \mathbb{R}^{d_2}$, and $t \in (0, \infty)$, and with dynamics

$$\begin{cases} \begin{pmatrix} \dot{\mathbf{x}}(s) \\ \dot{\mathbf{y}}(s)) \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \\ \mathbf{f}_2(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \end{pmatrix} & 0 < s < t \\ \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \end{cases}$$

we have that the value function satisfies

$$\phi(x, y, t) = \inf_{\boldsymbol{\alpha}(\cdot), \mathbf{x}(\cdot)} \sup_{\boldsymbol{\beta}(\cdot), \mathbf{y}(\cdot)} \left\{ g(\mathbf{x}(0), \mathbf{y}(0)) + \int_0^t L(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s)\, ds \right\} \tag{1.12}$$

Now, we discretize in time and approximate the ODE with backward Euler, and a formal computation gives us,

$$\approx \min_{\{\alpha_k\}_{k=1}^N \{x_k\}_{k=0}^N} \max_{\{\beta_k\}_{k=1}^N, \{y_k\}_{k=0}^N} \left\{ g(x_0, y_0) + \delta \sum_{k=1}^N L(x_k, y_k, \alpha_k, \beta_k, s_k) \right\}$$

$$\text{such that} \begin{cases} \begin{pmatrix} x_k - x_{k-1} \\ y_k - y_{k-1} \end{pmatrix} = \begin{pmatrix} f_1(x_k, y_k, \alpha_k, \beta_k, s_k) \\ f_2(x_k, y_k, \alpha_k, \beta_k, s_k) \end{pmatrix}, \quad k = 1, \dots, N \\ \begin{pmatrix} x_N \\ y_N \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \end{cases}$$

It is at this point we want to form the Lagrangian. The only trouble is that the concept of a "Lagrangian" for minimax problems (a.k.a. saddle-point problems) has not been well-examined. But in a paper by [35] (and also in [106]), the authors have a version of a Lagrangian for minimax problems, which we apply to our problem to get,

$$g(x_0, y_0) + \delta \sum_{j=1}^N L(x_j, y_j, \alpha_j, \beta_j, s_j) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f_1(x_j, y_j, \alpha_j, \beta_j, s_j) \rangle$$
$$+ \sum_{j=1}^N \langle -q_j, y_j - y_{j-1} - \delta f_2(x_j, y_j, \alpha_j, \beta_j, s_j) \rangle$$

Then we take the min max to obtain

$$\min_{\{\alpha_j\}_{j=1}^N,\{x_j\}_{j=0}^N}\max_{\{\beta_j\}_{j=1}^N,\{y_j\}_{j=0}^N}\left\{g(x_0,y_0)+\delta\sum_{j=1}^N L(x_j,y_j,\alpha_j,\beta_j,s_j)\right.$$
$$\left.+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}-\delta f_1(x_j,y_j,\alpha_j,\beta_j,s_j)\rangle+\sum_{j=1}^N\langle -q_j,y_j-y_{j-1}-\delta f_2(x_j,y_j,\alpha_j,\beta_j,s_j)\rangle\right\}$$
$$=\min_{\{x_j\}_{j=0}^N}\max_{\{y_j\}_{j=0}^N}\left\{g(x_0,y_0)+\delta\sum_{j=1}^N\min_{\alpha_j}\max_{\beta_j}\left\{L(x_j,y_j,\alpha_j,\beta_j,s_j)-\left\langle\begin{pmatrix}p_j\\-q_j\end{pmatrix},\begin{pmatrix}f_1(x_j,y_j,\alpha_j,\beta_j,s_j)\\f_2(x_j,y_j,\alpha_j,\beta_j,s_j)\end{pmatrix}\right\rangle\right\}\right.$$
$$\left.+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}\rangle+\sum_{j=1}^N\langle -q_j,y_j-y_{j-1}\rangle\right\}$$
$$=\min_{\{x_j\}_{j=0}^N}\max_{\{y_j\}_{j=0}^N}\left\{g(x_0,y_0)-\delta\sum_{j=1}^N H(x_j,y_j,p_j,-q_j,s_j)+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}\rangle+\sum_{j=1}^N\langle -q_j,y_j-y_{j-1}\rangle\right\}$$

Now we maximize over $\{p_j\}_{j=1}^N$ and minimize over $\{q_j\}_{j=1}^N$ to obtain,

$$\phi(x,y,t)\approx\min_{\{q_j\}_{j=1}^N}\max_{\{p_j\}_{j=1}^N}\min_{\{x_j\}_{j=0}^N}\max_{\{y_j\}_{j=0}^N}\left\{g(x_0,y_0)-\delta\sum_{j=1}^N H(x_j,y_j,p_j,-q_j,s_j)+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}\rangle\right.$$
$$\left.+\sum_{j=1}^N\langle -q_j,y_j-y_{j-1}\rangle\right\}$$

and after organizing a bit, we get,

$$\phi(x,y,t)\approx\min_{\{q_j\}_{j=1}^N}\max_{\{p_j\}_{j=1}^N}\min_{\{x_j\}_{j=0}^N}\max_{\{y_j\}_{j=0}^N}\left\{g(x_0,y_0)+\sum_{j=1}^N\left\langle\begin{pmatrix}p_j\\-q_j\end{pmatrix},\begin{pmatrix}x_j-x_{j-1}\\y_j-y_{j-1}\end{pmatrix}\right\rangle\right.$$
$$\left.-\delta\sum_{j=1}^N H(x_j,y_j,p_j,-q_j,s_j)\right\}\tag{1.13}$$

The authors in [35] apply their method to convex-concave differential games, which we also do in 1.6 (see also 1.6.3.3 where we have convex-concave initial conditions).

Note: If we can split $g(x,y)=e(x)+h(y)$, and if e is convex and h is concave, then we may take advantage of e^*, the convex conjugate of e, and h_*, the concave conjugate of h (the formula for the concave conjugate is the same as the convex-conjugate, but you change the sup to an inf) [108], in order to have an analogous Hopf formula:

$$\phi(x,y,t) \approx \min_{\{q_j\}_{j=1}^N} \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=1}^N} \max_{\{y_j\}_{j=1}^N} \left\{ -e^*(p_1) - h_*(-q_1) + \left\langle \begin{pmatrix} p_N \\ -q_N \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle \right.$$
$$\left. + \sum_{j=1}^{N-1} \left\langle \begin{pmatrix} p_j - p_{j+1} \\ -(q_j - q_{j+1}) \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\rangle - \delta \sum_{j=1}^{N} H(x_j, y_j, p_j, -q_j, s_j) \right\} \tag{1.14}$$

In some ways the function $e^*(p) + h_*(q)$ may perhaps be called the *convex-concave conjugate* for the convex-concave function $g(x,y)$.

Remark: The authors in [35] state that this "minimax Lagrangian," even in the simplest formulation given in their work, is new.

1.4.3 The advantage of the Hamiltonian for optimization

There is tremendous advantage in having a Hamiltonian. This is because if we want to instead perform optimization of the value function directly, we will be solving for the controls and this requires a constrained optimization technique.

The miraculous advantage of having a Hamiltonian for optimization purposes is it *encodes* information from both the running cost function L, as well as the dynamics $\dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \mathbf{u}(s), s)$. And thus we are now free to perform *unconstrained optimization*. But if we solve for the value function (1.1) and (1.2) directly, the we need to perform constrained optimization.

Another key additional advantage is that we lower the dimension of the numerical optimization by analytically minimizing over u, and conjuring the Hamiltonian.

1.5 The Main Algorithm: Splitting for Hamilton-Jacobi Equations

1.5.1 Splitting for HJE arising from Optimal Control

Before discussing the algorithms, we note that we do not yet have a proof of convergence nor approximation. This is currently a work-in-progress. But as shown in our numerical results 1.6, these algorithms seem to agree with classical methods used to solve Hamilton-Jacobi equations.

Taking the Lax formula with backward Euler (1.9) as an expository example, we can organize our problem to look similar to a primal-dual formulation which is attacked by splitting using PDHG. We stack variables and let

- $\tilde{x} = (x_0, x_1, \ldots, x_N)$, and similarly for $\tilde{p}$ and $\tilde{s}$,
- $\tilde{G}(\tilde{x}) = g(x_0)$,
- $\tilde{H}_\delta(\tilde{x}, \tilde{p}, \tilde{s}) = \delta \sum_{k=1}^{N} H(x_k, u_k, s_k)$,
- D be the difference matrix such that $\langle \tilde{p}, D\tilde{x} \rangle = \sum_{j=1}^{N-1} \langle p_j, x_j - x_{j-1} \rangle$

then our problem looks like:

$$\max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \tilde{G}(\tilde{x}) + \langle \tilde{p}, D\tilde{x} \rangle + \tilde{H}_\delta(\tilde{x}, \tilde{p}, \tilde{s}).$$

This looks similar to the problem that is attacked by PDHG, except for a couple of differences:

- PDHG solves a saddle point problem where the $\tilde{H}_\delta(\tilde{x}, \tilde{p}, \tilde{s})$ term does not depend on $\tilde{x}$ (nor $\tilde{s}$).
- In PDHG, the $\tilde{H}_\delta$ term is the convex conjugate of some function we want to minimize. But in our case, we have

  $$H(x, p, s) = \max_u \{\langle f(x, u, s), p \rangle - L(x, u, s)\}$$

  and $f(x, u, s)$ does not even have to be linear. So in some ways, H is a "generalized convex conjugate."

But we perform an alternating minimization technique similar to PDHG and we arrive at our main algorithm for optimal control:

For the Lax with backward Euler: Given an $x \in \mathbb{R}^d$ and some time $t \in (0, \infty)$, we set $\delta > 0$ small and let the time-grid size be $N = t/\delta + 1$ (we set $N = t/\delta$ for Hopf). Then we randomly initialize $\tilde{x} = (x_0, x_1, \ldots, x_N)$ but let $x_N = x$, and we randomly initialize $\tilde{p} = (p_0, p_1, \ldots, p_N)$ but let $p_0 \equiv 0$ as it is not minimized over, but used for computational accounting. And we let $\tilde{z} = \tilde{x}$. Then our algorithm follows the pattern of alternating optimization with quadratic penalty:

$$\begin{cases} \tilde{p}^{k+1} = \arg\max_{\tilde{p}} \left\{ \tilde{G}(\tilde{x}^k) - \tilde{H}_\delta(\tilde{x}^k, \tilde{p}, \tilde{s}) - \frac{1}{2\sigma}\|\tilde{p} - (\tilde{p}^k + D\tilde{z}^k)\|_2^2 \right\} \\ \tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ \tilde{G}(\tilde{x}) - \tilde{H}_\delta(\tilde{x}, \tilde{p}^{k+1}, \tilde{s}) + \frac{1}{2\tau}\|\tilde{x} - (\tilde{x}^k - D^T\tilde{p}^{k+1})\|_2^2 \right\} \\ \tilde{z}^{k+1} = \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k). \end{cases}$$

where $\sigma, \tau > 0$ are step-sizes with $\sigma\tau\|D\|^2 < 1$ and $\theta \in [0, 1]$ (as suggested in [16]). In our numerical experiments, $\theta = 1$ was frequently the best choice and also in practice, we would change the arg max into an arg min. So we have 1.

Algorithm 1 Splitting for HJE for Optimal Control, Lax with backward Euler

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

Initialize: $\delta > 0$ and set $N = t/\delta + 1$. And randomly initialize $\tilde{x}^0$ and $\tilde{p}^0$, but with $x_N^0 \equiv x_{\text{target}}$. And set $\tilde{z}^0 = \tilde{x}^0$. Also choose σ, τ such that $\sigma\tau\|D\|^2 < 1$ and $\theta \in [0, 1]$.

while tolerance criteria large **do**

$$\tilde{p}^{k+1} = \arg\min_{\tilde{p}} \left\{ -\tilde{G}(\tilde{x}^k) + \tilde{H}_\delta(\tilde{x}^k, \tilde{p}, \tilde{s}) + \frac{1}{2\sigma}\|\tilde{p} - (\tilde{p}^k + \sigma D\tilde{z}^k)\|_2^2 \right\}$$

$$\tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ \tilde{G}(\tilde{x}) - \tilde{H}_\delta(\tilde{x}, \tilde{p}^{k+1}, \tilde{s}) + \frac{1}{2\tau}\|\tilde{x} - (\tilde{x}^k - \tau D^T\tilde{p}^{k+1})\|_2^2 \right\}$$

$$\tilde{z}^{k+1} = \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k)$$

fval $= g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^N H(x_j, u_j, s_j)$

return fval

And a similar algorithm will be obtained when we use a forward Euler discretization 1.10 for the ODE dynamics. We can obtain better accuracy if we average the backward Euler and forward Euler approximations for the ODEs, which is reminiscent of the trapezoidal

approximation having better accuracy as it is the average of the forward and backward Euler.

We also have the Hopf formulation: Let,

- $\tilde{G}^*(\tilde{p}) = (g^*(p_1), 0, \ldots, 0)$, and
- let D be the difference matrix such that $\langle D\tilde{p}, \tilde{x}\rangle = \langle p_N, x\rangle + \sum_{k=1}^{N-1} \langle p_k - p_{k+1}, x_k\rangle$ (so this one differs from 1 as it acts on $\tilde{p}$)

then we have 2.

1.5.1.1 When to use the Hopf formula

We make the observation that the Lax formula is suitable (i.e. converges) when we have a convex Hamiltonian in p which is also bounded below in p (or satisfies a coercivity condition, see [40]; if we want a convex Hamiltonian that is not bounded in p, then we must use Hopf in this case.

Algorithm 2 Splitting for HJE for Optimal Control, Hopf (with backward Euler)

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

Initialize: $\delta > 0$ and set $N = t/\delta$. And randomly initialize $\tilde{x}^0$ and $\tilde{p}^0$, but with $x_N^0 \equiv x_{\text{target}}$. And set $\tilde{z}^0 = \tilde{x}^0$. Also choose σ, τ such that $\sigma\tau\|D\|^2 < 1$ and $\theta \in [0, 1]$.

while tolerance criteria large **do**

$$\tilde{p}^{k+1} = \arg\min_{\tilde{p}} \left\{ \tilde{G}^*(\tilde{p}^k) + \tilde{H}_\delta(\tilde{x}^k, \tilde{p}, \tilde{s}) + \tfrac{1}{2\sigma}\|\tilde{p} - (\tilde{p}^k + \sigma D^T \tilde{z}^k)\|_2^2 \right\}$$

$$\tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ -\tilde{G}^*(\tilde{p}) - \tilde{H}_\delta(\tilde{x}, \tilde{p}^{k+1}, \tilde{s}) + \tfrac{1}{2\tau}\|\tilde{x} - (\tilde{x}^k - \tau D\tilde{p}^{k+1})\|_2^2 \right\}$$

$$\tilde{z}^{k+1} = \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k)$$

$$\text{fval} = -g^*(p_1) + \langle p_N, x_{\text{target}}\rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j\rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j)$$

return fval

See 1.5.3 on how to perform the argmin/argmax in each iteration.

1.5.2 Splitting for HJE arising from Differential Games

For differential games, we use a similar algorithm to the optimal control case. We take the discretized version of the cost function (1.12) and perform an alternating minimization technique inspired by PDHG, but applied to minimax problems. Using the same notation as in 1 and 2, and the same D matrix as in 1, we have the algorithm for differential games in 3.

Algorithm 3 Splitting for HJE for Differential Games, Lax

Given: $(x_{\text{target}}, y_{\text{target}}) \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

Initialize: $\delta > 0$ and set $N = t/\delta + 1$. And randomly set $\tilde{x}^0$, $\tilde{y}^0$, $\tilde{p}^0$, and $\tilde{q}^0$, but with $x_N^0 \equiv x_{\text{target}}$ and $y_N^0 \equiv y_{\text{target}}$. And set $(\bar{\tilde{x}}^0, \bar{\tilde{y}}^0) = (\tilde{x}^0, \tilde{y}^0)$. Also choose σ, τ such that $\sigma\tau\|D\|^2 < 1$ and $\theta \in [0, 1]$.

while tolerance criteria large **do**

$$\tilde{p}^{k+1} = \arg\max_{\tilde{p}} \left\{ \tilde{G}(\tilde{x}^k, \tilde{y}^k) - \tilde{H}_\delta(\tilde{x}^k, \tilde{y}^k, \tilde{p}, -\tilde{q}^k, \tilde{s}^k) - \tfrac{1}{2\sigma}\|\tilde{p} - (\tilde{p}^k + \sigma D\bar{\tilde{x}}^k)\|_2^2 \right\}$$

$$\tilde{q}^{k+1} = \arg\min_{\tilde{q}} \left\{ \tilde{G}(\tilde{x}^k, \tilde{y}^k) - \tilde{H}_\delta(\tilde{x}^k, \tilde{y}^k, \tilde{p}^{k+1}, -\tilde{q}, \tilde{s}^k) + \tfrac{1}{2\sigma}\|\tilde{q} - (\tilde{q}^k + \sigma D\bar{\tilde{y}}^k)\|_2^2 \right\}$$

$$\tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ \tilde{G}(\tilde{x}, \tilde{y}^k) - \tilde{H}_\delta(\tilde{x}, \tilde{y}^k, \tilde{p}^{k+1}, -\tilde{q}^{k+1}, \tilde{s}^k) + \tfrac{1}{2\tau}\|\tilde{x} - (\tilde{x}^k - \tau D^T\tilde{p}^{k+1})\|_2^2 \right\}$$

$$\tilde{y}^{k+1} = \arg\max_{\tilde{y}} \left\{ \tilde{G}(\tilde{x}^{k+1}, \tilde{y}) - \tilde{H}_\delta(\tilde{x}^{k+1}, \tilde{y}, \tilde{p}^{k+1}, -\tilde{q}^{k+1}, \tilde{s}^k) - \tfrac{1}{2\tau}\|\tilde{y} - (\tilde{y}^k - \tau D^T\tilde{q}^{k+1})\|_2^2 \right\}$$

$$\begin{pmatrix} \bar{\tilde{x}}^{k+1} \\ \bar{\tilde{y}}^{k+1} \end{pmatrix} = \begin{pmatrix} \tilde{x}^{k+1} \\ \tilde{y}^{k+1} \end{pmatrix} + \theta \left(\begin{pmatrix} \tilde{x}^{k+1} \\ \tilde{y}^{k+1} \end{pmatrix} - \begin{pmatrix} \tilde{x}^k \\ \tilde{y}^k \end{pmatrix} \right).$$

$$\text{fval} = g(x_0, y_0) + \sum_{j=1}^N \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} x_j - x_{j-1} \\ y_j - y_{j-1} \end{pmatrix} \right\rangle - \delta \sum_{j=1}^N H(x_j, y_j, p_j, -q_j, s_j)$$

return fval

If we have $\tilde{G}(x, y) = \tilde{E}(x) + \tilde{H}(y)$ where E is convex and H is concave, then we may make use of convex-conjugates and concave-conjugates (see (1.14)) to obtain an analogous Hopf formula as in 2, but for differential games. Here D is the same difference matrix as in the Hopf case, 2. This is 4.

Algorithm 4 Splitting for HJE for Differential Games, Hopf (for separable convex-concave initial conditions)

Given: $(x_{\text{target}}, y_{\text{target}}) \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

Initialize: $\delta > 0$ and set $N = t/\delta$. And randomly set $\tilde{x}^0$, $\tilde{y}^0$, $\tilde{p}^0$, and $\tilde{q}^0$, but with $x_N^0 \equiv x_{\text{target}}$ and $y_N^0 \equiv y_{\text{target}}$. And set $(\bar{\tilde{x}}^0, \bar{\tilde{y}}^0) = (\tilde{x}^0, \tilde{y}^0)$. Also choose σ, τ such that $\sigma\tau\|D\|^2 < 1$ and $\theta \in [0, 1]$.

while tolerance criteria large **do**

$$\tilde{p}^{k+1} = \arg\max_{\tilde{p}} \left\{ -\tilde{E}^*(p_1) - \tilde{H}_*(-q_1) - \tilde{H}_\delta(\tilde{x}^k, \tilde{y}^k, \tilde{p}, -\tilde{q}^k, \tilde{s}^k) - \tfrac{1}{2\sigma}\|\tilde{p} - (\tilde{p}^k + \sigma D^T \bar{\tilde{x}}^k)\|_2^2 \right\}$$

$$\tilde{q}^{k+1} = \arg\min_{\tilde{q}} \left\{ -\tilde{E}^*(p_1) - \tilde{H}_*(-q_1) - \tilde{H}_\delta(\tilde{x}^k, \tilde{y}^k, \tilde{p}^{k+1}, -\tilde{q}, \tilde{s}^k) + \tfrac{1}{2\sigma}\|\tilde{q} - (\tilde{q}^k + \sigma D^T \bar{\tilde{y}}^k)\|_2^2 \right\}$$

$$\tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ -\tilde{E}^*(p_1) - \tilde{H}_*(-q_1) - \tilde{H}_\delta(\tilde{x}, \tilde{y}^k, \tilde{p}^{k+1}, -\tilde{q}^{k+1}, \tilde{s}^k) + \tfrac{1}{2\tau}\|\tilde{x} - (\tilde{x}^k - \tau D \tilde{p}^{k+1})\|_2^2 \right\}$$

$$\tilde{y}^{k+1} = \arg\max_{\tilde{y}} \left\{ -\tilde{E}^*(p_1) - \tilde{H}_*(-q_1) - \tilde{H}_\delta(\tilde{x}^{k+1}, \tilde{y}, \tilde{p}^{k+1}, -\tilde{q}^{k+1}, \tilde{s}^k) - \tfrac{1}{2\tau}\|\tilde{y} - (\tilde{y}^k - \tau D \tilde{q}^{k+1})\|_2^2 \right\}$$

$$\begin{pmatrix} \bar{\tilde{x}}^{k+1} \\ \bar{\tilde{y}}^{k+1} \end{pmatrix} = \begin{pmatrix} \tilde{x}^{k+1} \\ \tilde{y}^{k+1} \end{pmatrix} + \theta \left(\begin{pmatrix} \tilde{x}^{k+1} \\ \tilde{y}^{k+1} \end{pmatrix} - \begin{pmatrix} \tilde{x}^k \\ \tilde{y}^k \end{pmatrix} \right).$$

fval $= -e^*(p_1) - h_*(-q_1) + \left\langle \begin{pmatrix} p_N \\ -q_N \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle + \sum_{j=1}^{N-1} \left\langle \begin{pmatrix} p_j - p_{j+1} \\ -(q_j - q_{j+1}) \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\rangle -$
$\delta \sum_{j=1}^{N} H(x_j, y_j, p_j, -q_j, s_j)$

return fval

1.5.3 Remarks on how to perform the argmin/argmax in each iteration

In each iteration for the above algorithms, we have an optimization problem when updating $\tilde{x}^{k+1}$ ($\tilde{y}^{k+1}$) or $\tilde{p}^{k+1}$ ($\tilde{q}^{k+1}$). In some of our experiments, the optimization turned into a closed-form proximal expression (mainly when updating $\tilde{p}^{k+1}/\tilde{q}^{k+1}$, or we were able to make use of one step of gradient descent of the objective (mainly when updating $\tilde{x}^{k+1}$ or $\tilde{y}^{k+1}$ or when $\tilde{G} = g$ is involved). As an example, one way to update the Lax formula for optimal control (1) using a backward Euler discretization is

$$\begin{cases} \tilde{p}^{k+1} &= \text{prox}_{\sigma \tilde{H}_\delta(\bar{\tilde{x}}^k,\cdot)}(\tilde{p}^k + \sigma D\bar{\tilde{x}}^k) \\ \\ \tilde{x}^{k+1} &= \tilde{x}^k - \tau D^T\tilde{p}^{k+1} - \tau\nabla_{\tilde{x}}\tilde{G}(\tilde{x}^k) + \tau\nabla_{\tilde{x}}\tilde{H}_\delta(\tilde{x}^k, \tilde{p}^{k+1}, \tilde{s}) \\ \\ \bar{\tilde{x}}^{k+1} &= \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k). \end{cases}$$

and one way to update the Hopf formula for optimal control (2) is,

$$\begin{cases} \tilde{p}^{k+1} &= \text{prox}_{\sigma \tilde{H}_\delta(\bar{\tilde{x}}^k,\cdot)}(\tilde{p}^k + \sigma D\bar{\tilde{x}}^k - \sigma\nabla g^*(p_1)) \\ \\ \tilde{x}^{k+1} &= \tilde{x}^k - \tau D^T\tilde{p}^{k+1} + \tau\nabla_{\tilde{x}}\tilde{H}_\delta(\tilde{x}^k, \tilde{p}^{k+1}) \\ \\ \bar{\tilde{x}}^{k+1} &= \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k). \end{cases}$$

where we see the update for $\tilde{p}^{k+1}$ is a *proximal gradient* update [103] (Section 4.2).

There are many combinations we can use, but the intuition is to take a gradient step on the smooth part, and a proximal step on the non-smooth part. And if we have a sum of a smooth part and a non-smooth part, we can mix a gradient step with a proximal step (i.e. a proximal gradient step) as we have done above.

1.5.4 Computation of characteristic curves/optimal trajectories

A benefit from the new algorithm is that alongside computing solutions at each point, it also allows us to directly compute trajectories/characteristic curves of the Hamilton-Jacobi

equation at each point. In fact, our algorithm is a hybrid of the direct collocation method (a direct method) [135] and Pontryagin's Maximum Principle (an indirect method) [63].

We give some examples of characteristic curves/optimal trajectories in our numerical results (1.6).

1.5.5 The advantage of splitting over coordinate descent

The advantage of these methods over coordinate descent is not only its speed, but it also does not seem to require as many multiple initial guesses for nonconvex optimization. And in our numerical experiments in 1.6, we only used a single initial guess in all our examples. And for most examples in our experiments in 1.6, it only requires one guess.

It also the advantage that one can apply the method to *non-smooth* problems, as opposed to coordinate-descent, where one takes numerical gradients.

And practically, splitting is more straightforward to implement than coordinate descent, where we would require divided differences to numerically compute the gradients, and we also have available to us the multitude of splitting techniques from the optimization literature, such as ADMM and Douglas-Rachford splitting to name a few.

1.5.6 A remark on the connection between Hamilton-Jacobi equations and optimization, and the implications on future optimization methods

The relationships between Hamilton-Jacobi equations and optimization have been noted in the literature [109, 110]. More recently, there has been a connection between deep learning optimization and HJE [18]. More concretely, there is a straight-forward connection between Hamilton-Jacobi equations and the proximal operator (which can be interpreted as implicit gradient descent):

Given a function $f(\cdot)$, the proximal operator of f is

$$(I + \lambda \partial f)^{-1}(v) := \arg\min_x \left\{ f(x) + \frac{1}{2\lambda} \|x - v\|_2^2 \right\}$$

If we change the $\arg\min_x$ into a $\min_x$, then we get the familiar Lax (a.k.a. Hopf-Lax) formula for the Hamilton-Jacobi equation with $H(x) = \frac{1}{2}\|x\|_2^2$, and for $t = \lambda$. So in this way, the Generalized Lax and Hopf formulas can be viewed as a generalization of the proximal operator.

Also, the proximal operator, i.e. the $\arg\min_x$ operator, is featured heavily in our algorithms. In classical PDHG, the primal variable x and the dual variable p are decoupled. But for our algorithms, the coupling is in the form of a state-dependent Hamiltonian. This coupling of the primal and dual variables can have implications on future optimization methods, as we can then attempt various coupling functions, i.e. Hamiltonians, and examine their effectiveness in general optimization techniques.

We also reiterate that our algorithms have been able to perform nonconvex optimization without as many multiple initial guesses as in other algorithm such as coordinate descent. In fact, in all our examples in 1.6, we only used a single initial guess. So we feel a deeper theoretical examination of these algorithms will likely be beneficial to the theory and literature of nonconvex optimization methods.

1.6 Numerical Results

Here we present numerical examples using our algorithm. The computations were run on a laptop with an Intel i7-4900MQ 2.90Ghz×4 Haswell Core processor, of which only one core processor was utilized for computations. And the computations for the Eikonal equation and the Difference of Norms were computed in C++, while the (unnamed) Isaacs example and the Quadcopter were computed in MATLAB, version R2017b.

For initializations, we initialized $\tilde{x}^0 = (x_0^0, x_1^0, \ldots, x_N^0)$ to be such that each x_i^0 (for $i = 0, \ldots, N-1$) is a random point close to x_{target}, and we let $x_N^0 \equiv x_{\text{target}}$. In particular, for $\tilde{x}^0$ each coordinate, except for x_N^0, was randomly initialized so that $\|\tilde{x}^0 - (x_{\text{target}}, x_{\text{target}}, \ldots, x_{\text{target}})\|_\infty \leqslant 0.1$. We chose $\tilde{p}^0$ to be a random vector close to the origin so that $\|\tilde{p}^0 - 0\|_\infty \leqslant 0.1$.

We chose $\theta = 1$ in all cases.

The PDHG step-sizes σ and τ, and the time-step size δ all varied for each example.

The error tolerance for all optimal control examples were chosen such that primal and dual variables satisfy $\|x^{k+1} - x^k\|_2^2 < 10^{-8}$ and $\|p^{k+1} - p^k\|_2^2 < 10^{-8}$. The error tolerance for all the differential games example were also similarly $\|(x^{k+1}, y^{k+1}) - (x^k, y^k)\|_2^2 < 10^{-8}$ and $\|(p^{k+1}, q^{k+1}) - (p^k, q^k)\|_2^2 < 10^{-8}$, although we had to slightly modify our stopping criteria here.

For the difference of norms example, if the algorithm reached some max count, then we would examine the value function and stop the algorithm when the value of the value function for consecutive iterations reached a difference below some tolerance.

For the (unnamed) Isaacs example with fully convex initial conditions, we chose the error to be such that the relative error of the value function of consecutive iterations was less than 10^{-6}, i.e. $\left\| \frac{\text{fval}^{k+1} - \text{fval}^k}{\min(\|\text{fval}^{k+1}\|, 1)} \right\| < 10^{-6}$. This example turned out to be the harshest on our algorithm.

In all cases, when we derive the Hamiltonian, we are starting from an optimal control with a terminal condition and solving "backwards in time" (see 1.2.1) as this naturally gives us an initial-valued Hamilton-Jacobi PDE.

1.6.1 State-and-Time-Dependent Eikonal Equation (Optimal Control)

1.6.1.1 Brief background on Eikonal equations

The state-dependent Eikonal equations are HJE with Hamiltonians,

$$H^+(x, p) = c(x)\|p\|, \quad H^- = -c(x)\|p\|.$$

where $c(x) > 0$ and $\|\cdot\|$ is any norm; in our examples we take the Euclidean norm. We also test a state-and-time-dependent equation of Eikonal type, where $H(x, p, t) = c(x - t)\|p\|$.

They arise from optimal control problems that have dynamics

$$f(x, u) = c(x)u, \quad \text{with } \|u\| \leqslant 1 \text{ and } c(x) \neq 0 \text{ for all } x$$

and with cost-functional

$$J[u] = g(x(0)) + \int_0^t \mathcal{I}_{\leqslant 1}(u(s))\, ds$$

where $\mathcal{I}_{\leqslant 1}(\cdot)$ is the indicator function of the unit ball in $\mathbb{R}^n$, i.e. 0 for all points within the unit ball including the boundary, and $+\infty$ for all points outside.

This is a *nonlinear* optimal control example due to the presence of $c(x)$. Also, our algorithm is performing nonconvex optimization (due to the presence of $c(x)$, but also in our negative Eikonal equation example where we are performing minimization with a $-g(x)$ term where g is quadratic).

The Eikonal equation features heavily in the level set method [99, 98], which has made wide-ranging contributions in many fields.

Note the optimization in solving negative Eikonal equation can be obtained by examining the positive Eikonal equation. This is actually a general phenomenon of Hamiltonians that obey $H(x, -p, t) = H(x, p, t)$. This is because if ϕ solves a HJE with initial data g and Hamiltonian such that $H(x, -p, t) = H(x, p, t)$, then examining $-\phi$,

$$\left\{ \begin{array}{rl} (-\phi)_t + H(x, \nabla(-\phi), t) & = 0 \\ (-\phi)(x, 0) & = -g(x) \end{array} \right\} \quad \Leftrightarrow \quad \left\{ \begin{array}{rl} \phi_t - H(x, \nabla\phi, t) & = 0 \\ \phi(x, 0) & = g(x) \end{array} \right\}$$

so we see $-\phi$ solves the positive Eikonal equation with initial data $-g$ if and only if ϕ solve the negative Eikonal equation with initial data g. And note that both are viscosity solutions as this computation still holds when we compute the viscous version of the HJE [40] (Section 10.1).

1.6.1.2 Implementation details

Here we take

$$c(x) = 1 + 3\exp(-4\|x - (1, 1, 0, \ldots, 0)\|_2^2),$$

which is a positive bump function. The initial condition of our HJE PDE is

$$g(x) = -1/2 + (1/2)\left\langle A^{-1}x, x\right\rangle, \quad A = \mathrm{diag}(2.5^2, 1, 0.5^2, \ldots, 0.5^2).$$

We used the Lax version of our algorithm, Algorithm 1, for all cases in this section. For the $\tilde{x}^{k+1}$-update, we took one step of gradient descent. And for the $\tilde{p}^{k+1}$ update, we took the proximal of the ℓ_2-norm (a.k.a. the $shrink_2$ operator); in general the $shrink$ operator can be defined for any positively homogenous of degree 1 convex function ϕ. For the ℓ-1 norm, we have the $shrink_1$ operator (in $\mathbb{R}^n$) can be computed coordinate-wise and the i-th coordinate satisfies,

$$(\text{shrink}_1(x,\lambda)_i = \begin{cases} x_i - \lambda & \text{if } x_i > \lambda \\ 0 & \text{if } |x_i| \leqslant \lambda \\ x_i + \lambda & \text{if } x_i < -\lambda \end{cases}$$

and the $shrink_2$ operator also can be computed coordinate-wise and the i-th coordinate satisfies

$$(\text{shrink}_2(x,\lambda))_i = \begin{cases} \frac{x}{\|x\|_2}\max(\|x\|_2 - \lambda, 0) & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

For the negative Eikonal equation, since in our implementation we computed a positive Eikonal equation with initial data $-g$ and then took the negative, we were able to compute the proximal of the concave quadratic $-g$. We call this taking the *stretch* operator of g (see [19], Section 4.2.2).

For these Eikonal equations, we chose a time step-size of $\delta = 0.02$, and we computed in a $[-3,3]^2$ grid with a mesh size of 0.1 on each axis.

For the positive Eikonal equation (1.6.1.2), we chose a PDHG step-size that depended on the norm of $\nabla c(x)$. If $\|\nabla c(x)\|_2 > 0.001$, then we took $\sigma = 50$ and or else we took $\sigma = 0.5$. And we always took $\tau = 0.25/\sigma$ (the 0.25 comes from $\|D\|_2 = 2 - \epsilon$ for some small $\epsilon > 0$, and requiring $\sigma\tau < 1/\|D\|^2$). To compute the times $t = 0.1$, 0.2, 0.3, and 0.4, the computation time averaged to 4.39×10^{-4} seconds per point in C++.

For the negative Eikonal equation (1.6.1.2), we chose a PDHG step-size of $\sigma = 100$ and $\tau = 0.25/\sigma$. We picked $\theta = 1$ for all cases. For $t = 0.1$, 0.2, 0.3, 0.4, and 0.5, the computation time averaged to 0.0024 seconds per point in C++. We see that for the $t = 0.5$ curve, there

are kinks, which may be a result of the splitting finding sub/super solutions, rather than viscosity solutions [40] (Section 10.1). In this case, multiple initial guesses will alleviate this.

For the negative eikonal equation in 10 dimensions (1.6.1.2), we computed a 2-dimensional slice in $[-3,3] \times [-3,3] \times \{0\} \times \cdots \times \{0\}$. The time step-size was again $\delta = 0.02$ and we chose $\sigma = 100$, and $\tau = 0.25/\sigma$. For $t = 0.1$, 0.2, 0.3, and 0.4 (0.5 had no level sets) the computation time averaged to 0.004 seconds per point in C++.

We also computed a state-and-time-dependent Eikonal equation (1.6.1.2) where $H(x,p,t) = c(x - t(-1,1))\|p\|_2$. Here in our specific example, $c(x - t(-1,1))$ represents a bump function moving diagonally in the $(-1,1)$ direction as t increases. The time step-size were the same as in the positive eikonal case which is reasonably expected because we took the positive eikonal case and modified it. The computational time for $t = 0.1$, 0.2, 0.3, and 0.4 averaged to 5.012×10^{-4} seconds per point in C++.

In these examples, we achieved a speedup of about ten times over coordinate descent, and only one initial guess was used. In low dimensions, this problem can be solved with SQP (Sequential Quadratic Programming) on the value function, or using Lax-Friedrichs. We use these methods to check our accuracy and they agree to within 10^{-4} for each point (x,t) when using SQP.

We observe that the two different eikonal equations required vastly different step-sizes and it is worth examining how to choose step-sizes in a future work. We speculate the step-sizes may act as CFL conditions. This is a further point of study.

Comparing coordinate descent to our algorithm in MATLAB, we achieve about an 8-10 times speed-up.

The figures 1.6.1.2, 1.6.1.2, and 1.6.1.2, and 1.6.1.2 show the zero level sets of the HJE solution.

In 5, we give an explicit example of how we performed our algorithm on the negative eikonal equation.

Algorithm 5 Splitting for the negative Eikonal equation with convex initial data g (note: below we are actually solving the equivalent problem of the positive Eikonal equation with concave initial data $-g$)

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 >$ tol or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 >$ tol) and (count $<$ max_count) **do**

for $j = 1$ to N **do**

$p_j^{k+1} = \text{shrink}_2(p_j^k + \sigma(z_j^k - z_{j-1}^k), \sigma\delta c(x^k))$ (proximal)

for $j = 0$ **do**

$x_0^{k+1} = x_0^k - \tau(\underbrace{p_0^{k+1}}_{=0} - p_1^{k+1}) - \tau(\nabla(-g))(x_0^k)$ (gradient descent)

for $j = 1$ to $N - 1$ **do**

$x_j^{k+1} = x_j^k - \tau(p_j^{k+1} - p_{j-1}^{k+1}) - \tau(-\delta(\nabla c)(x_j^k)\|p_j^{k+1}\|_2)$ (gradient descent)

for $j = 0$ to N **do**

$z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$ (extrapolation step)

fval $= -g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^N c(x_j)\|p_j\|_2$

return fval

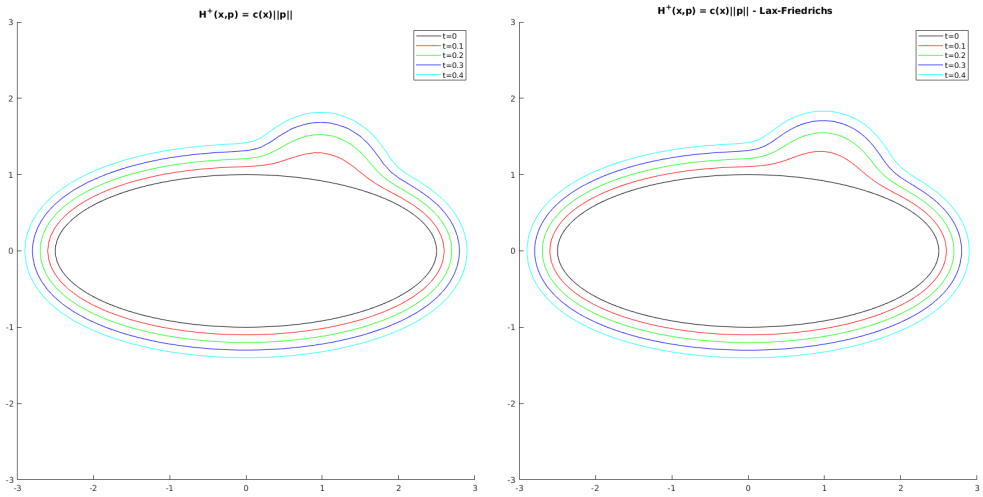


Figure 1.1: Eikonal equation with $H^+(x,p) = c(x)\|p\|_2$, in two spatial dimensions. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3, 0.4$. We observe that the zero level sets move outward as time increases. The left figure is computed using our new algorithm, while the right figure is computed using the conventional Lax-Friedrichs method.

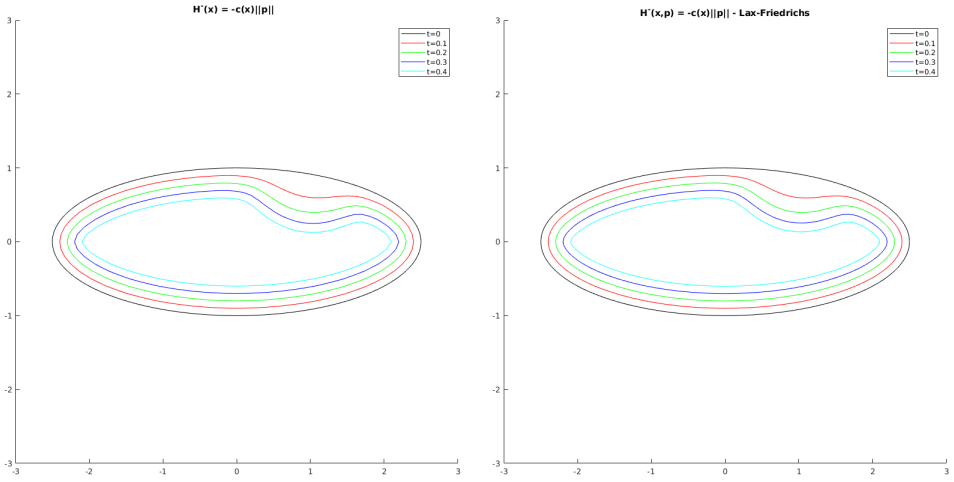


Figure 1.2: Eikonal equation with $H^-(x,p) = -c(x)\|p\|_2$, in two spatial dimensions. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3, 0.4$. We observe that the zero level sets move inward as time increases. Left is computed with our new algorithm, while the right is computed using the conventional Lax-Friedrichs method.

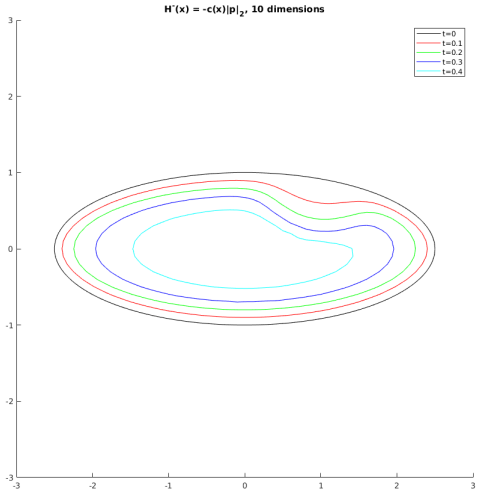


Figure 1.3: Eikonal equation with $H^-(x,p) = -c(x)\|p\|_2$, in ten spatial dimensions. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3, 0.4$. We observe that the zero level sets move inward as time increases.

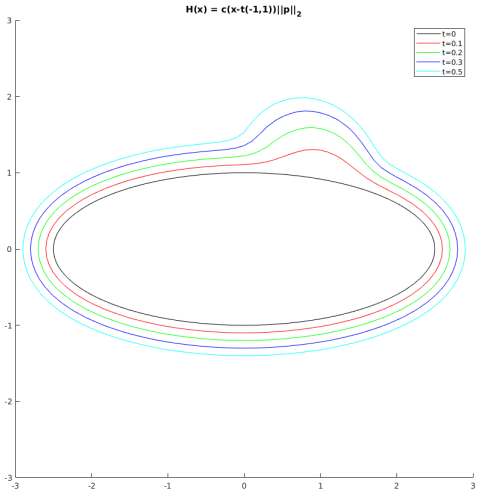


Figure 1.4: Eikonal equation with $H(x,p) = c(x - t(-1,1))\|p\|_2$. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3, 0.4$. We observe there is a similarity to the positive eikonal case, but the "bump" is more sheared to the left.

1.6.1.3 Dimensional scaling of the negative Eikonal equation

Here we examine how Algorithm 1 scales with dimension. We compute the negative Eikonal equation with the same speed c and initial data g as above, and with $\delta = 0.02$ and $\sigma = 100$ and $\tau = 0.25/\sigma$. We computed in a 2-dimensional slice $[-3, 3]^2 \times \{0\}^{d-2}$, and we computed from $d = 10$ to $d = 2000$ dimensions. We performed our analysis at time $t = 0.2$.

We chose this particular example as this is a nonlinear optimal control problem that requires us to optimize a nonconvex problem.

We used least-squares to fit both a linear function as well as a quadratic function. The coefficients were

$$\text{lin}(d) = (1.14 \times 10^{-4})d + 0.0021, \quad \text{quad}(d) = (-5.99 \times 10^{-9})d^2 + (1.27 \times 10^{-4})d - 0.00195$$

As we can see from the equations of the fit, and from 1.6.1.3, the quadratic fit has an extremely small quadratic coefficient. 1.6.1.3 shows the plot with the linear fit. This computation was done in C++.

We predict that for general problems, the scaling will be polynomial in time.

1.6.2 Difference of norms (Differential Games)

1.6.2.1 From differential games to HJE for the difference of norms

The state-dependent HJE for the difference of norms case arises from the following differential games problem: Given $x \in \mathbb{R}^{d_1}$ and $y \in \mathbb{R}^{d_2}$, and some $t > 0$, we have the following the

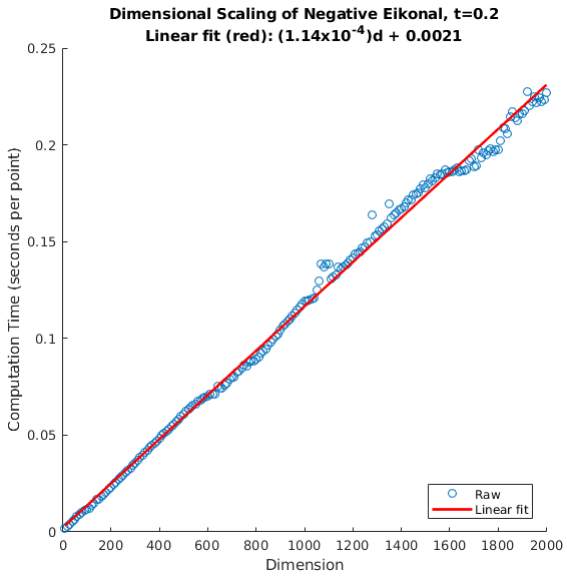


Figure 1.5: How our algorithm scales with dimension for the negative Eikonal equation at time $t = 0.2$. This is a nonlinear optimal control problem, and the optimization requires us to perform nonconvex optimization. The plot shows a linear fit.

dynamics

$$\begin{cases} \begin{pmatrix} \dot{\mathbf{x}}(s) \\ \dot{\mathbf{y}}(s) \end{pmatrix} = \begin{pmatrix} c_1(\mathbf{x}(s), \mathbf{y}(s))\, \boldsymbol{\alpha}(s) \\ c_2(\mathbf{x}(s), \mathbf{y}(s))\, \boldsymbol{\beta}(s) \end{pmatrix}, \quad 0 \leqslant s \leqslant t \\ \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \\ \\ \boldsymbol{\alpha}(s)\| \leqslant 1, \|\boldsymbol{\beta}(s)\| \leqslant 1, \text{ for all } s, \\ \\ \text{and } c_1, c_2 \text{ are positive functions.} \end{cases}$$

And the cost function is

$$J_{x,t}[\boldsymbol{\alpha}, \boldsymbol{\beta}] = g(\mathbf{x}(0), \mathbf{y}(0)) + \int_0^t \mathfrak{I}_{\leqslant 1}(\boldsymbol{\alpha}(s)) - \mathfrak{I}_{\leqslant 1}(\boldsymbol{\beta}(s))\, ds$$

where $\mathcal{I}_{\leqslant 1}(\cdot)$ is the indicator function the unit-ball, i.e. it equals 0 for points inside and on the unit-ball, and $+\infty$ for points outside. Our value function is then

$$\phi(x, y, t) = \inf_{\alpha, \|\alpha\| \leqslant 1} \sup_{\beta, \|\beta\| \leqslant 1} J_{x,t}[\boldsymbol{\alpha}, \boldsymbol{\beta}].$$

Then our Hamiltonian becomes,

$$\begin{aligned} H(x, y, p, q) &= \max_{\alpha} \min_{\beta} \left\{ \left\langle \begin{pmatrix} c_1(x, y)\alpha \\ c_2(x, y)\beta \end{pmatrix}, \begin{pmatrix} p \\ q \end{pmatrix} \right\rangle - (I_{\leqslant 1}(\alpha) - I_{\leqslant 1}(\beta)) \right\} \\ &= \max_{\alpha} \left\{ \langle c_1(x, y)\alpha, p_1 \rangle - I_{\leqslant 1}(\alpha) \right\} + \min_{\beta} \left\{ \langle c_2(x, y)\beta, q \rangle + I_{\leqslant 1}(\beta) \right\} \\ &= c_1(x, y)\|p\|_2 - c_2(x, y)\|q\|_2. \end{aligned}$$

In this case, we have nonlinear dynamics, aswell as nonconvex Hamiltonian.

1.6.2.2 Implementation details for the difference of norms

Here we take,

$$c_1(x) = 1 + 3\exp(-4\|x - (1, 1, 0, \ldots, 0)\|_2^2), \qquad c_2(x) = c_1(-x),$$

and the initial condition is

$$g(x) = -1/2 + (1/2) \left\langle A^{-1}x, x \right\rangle, \quad A = \operatorname{diag}(2.5^2, 1, 0.5^2, \ldots, 0.5^2).$$

which is the same initial condition as in our Eikonal equation example above 1.6.1.2.

For the 2-dimensional case, we used the Hamiltonian,

$$H(x_1, x_2, p_1, p_2) = c(x_1, x_2)\|p_1\|_2 - c(-x_1, -x_2)\|p_2\|_2$$

and for the 7-dimensional case, we used,

$$H(x_1, x_2, \ldots, x_7, p_1, p_{(2,\ldots,7)}) = c(x_1, \ldots, x_7)\|p_1\|_2 - c(-x_1, \ldots, -x_7)\|p_{(2,\ldots,7)}\|_2.$$

where $p_{(2,\ldots,7)} = (p_2, p_3, \ldots, p_7)$.

We compute our solutions in 2 and 7 dimensions. We compute the 2-dimensional case in a $[-3,3]^2$ grid, and we compute the 7-dimensional case in the two dimensional slice $[-3,3]^2 \times \{0\} \times \cdots \times \{0\}$. And we used 3.

For the $\tilde{p}^{k+1}$-update, we used the proximal of the ℓ_2-norm (a.k.a. the $shrink_2$ operator), and for the $\tilde{x}^{k+1}$-update, we used one step of gradient descent.

We took the time-step as $\delta = 0.02$, and we took the PDHG steps $\sigma = 50$, and $\tau = 0.25/\sigma$. The 0.25 comes from the fact that the PDHG algorithm requires $\sigma\tau\|D\|_2^2 < 1$, and $\|D\|_2 = 2 - \epsilon$, for some small $\epsilon > 0$.

The computation was done with a mesh size of $1/12 \approx 0.08333$ in each axis. For the 2-dimensional case, the computation averaged out to 0.0135 seconds per point in C++, and for the 7-dimensional case the computation averaged out to 0.01587 seconds per point in C++. If we compared the algorithms in MATLAB on the same computer, we achieved a 10-20 times speed-up compared to coordinate descent.

We note that at certain points, the trajectories would oscillate a little for larger times which may be due to the non-convexity and the non-unique optimal trajectories. So when a maximum count was reached, we would raise σ by 20, and we would also readjust $\tau = 0.25/\sigma$. We do not recommend choosing a high σ at all points, or else the algorithm would result in incorrect solutions. If the convergence was not fast enough, after some maximum count, we would switch our convergence criteria to the value function, as the error (between consecutive iterations) seemed to converge to zero. The procedure of raising the sigma is reminiscent of CFL conditions for finite-difference schemes, and we are examining how best to choose the PDHG step-sizes σ and τ. The best σ and τ to choose seem to be dependent on the point at which we are computing.

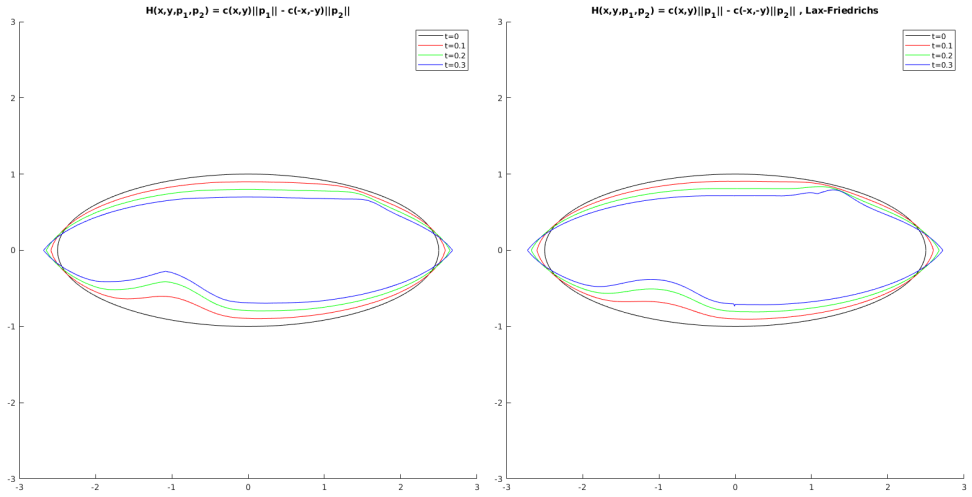


Figure 1.6: The difference of norms HJE in two spatial dimensions. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3$. We observe that the zero level sets move inward as time increases. Left is computed with our new algorithm, while the right is computed using the conventional Lax-Friedrichs method. Note there is an anomaly at the top-right of Lax-Friedrich computation. And there is also more of a corner in the bottom-left of the solution computed by the new method. This may be a result of the true solution, and which does not appear in the Lax-Friedrichs solution as it tends to smooth out solutions.

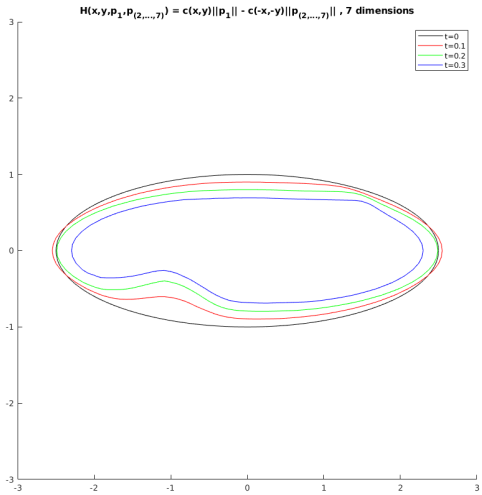


Figure 1.7: The difference of norms HJE in seven spatial dimensions. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3$.

1.6.3 An (unnamed) example from Isaacs (Differential Games)

1.6.3.1 From differential games to HJE for an unnamed example from Isaacs

We now examine a modified version of a differential game, obtained from [39] (Example 6.3), in which it is named "an unnamed example of Isaacs"; the original source is found in Isaac's seminal work [67] (Example 8.4.3). The dynamics are as follows:

$$\begin{cases} \dot{x}(s) &= 2\beta + \sin(\alpha) \\ \dot{y}(x) &= -c(x,y) + \cos(\alpha) \end{cases}$$

where $0 \leqslant \alpha \leqslant 2\pi$ and $-1 \leqslant \beta \leqslant 1$. These dynamics are nonlinear. We take the cost-functional as,

$$J[\alpha, \beta] = g(x(0), y(0)) + \int_0^t 1 \, ds$$

and the value function seeks to maximize with respect to $\alpha \in [0, 2\pi]$, and minimize with respect to $\beta \in [-1, 1]$. Then our Hamiltonian is,

$$\begin{aligned} H(x,y,p,q) &= \min_{\alpha,\alpha\in[0,2\pi]} \max_{\beta,\beta\in[-1,1]} \left\{ \left\langle \begin{pmatrix} 2\beta + \sin(\alpha) \\ -c(x,y) + \cos(\alpha) \end{pmatrix}, \begin{pmatrix} p \\ q \end{pmatrix} \right\rangle - 1 \right\} \\ &= \min_{\alpha,\alpha\in[0,2\pi]} \max_{\beta,\beta\in[-1,1]} \{2\beta p - c(x,y)q + p\sin(\alpha) + q\cos(\alpha) - 1\} \\ &= -c(x,y)q + 2|p| - \sqrt{p^2+q^2} - 1. \end{aligned}$$

This Hamiltonian is nonconvex. And the dynamics are nonlinear.

1.6.3.2 Implementation details for the (unnamed) example from Isaacs with fully convex initial conditions

We take

$$c(x) = 2(1 + 3\exp(-4\|x - (1,1,0,\ldots,0)\|_2^2)), \tag{1.15}$$

which is a positive bump function. The initial condition of our HJE PDE is

$$g(x) = -1/2 + (1/2)\left\langle A^{-1}x, x\right\rangle, \quad A = \operatorname{diag}(2.5^2, 1, 0.5^2, \ldots, 0.5^2).$$

This example is perhaps the harshest on our algorithm and turns out to be slower than coordinate descent, but in many ways this is not surprising. This is because this problem is highly nonconvex and the $g(x, y)$ that we use is a convex function – ideally we would like it to be a convex-concave function which would be suitable for saddle-point problems. Not only that, but our Hamiltonian is not bounded below with respect to q, and the Hopf formula requires this assumption.

Nevertheless, we show this example in order to advertise the generality of our algorithm. It might actually be surprising that our algorithm gives a solution that looks like the Lax-Friedrichs solution at all. We also not that we only used one initial guess, and in our experiments, using around 5 initial guesses smooths our the solution.

We use 3, but modified so we can utilize the convex portion of g (see the last paragraph of 1.5.2). We compute our solutions in a 2-dimensional $[-3, 3]^2$ grid. The $\tilde{p}^{k+1}$-update utilizes a combination of gradient descent for the q, and the $shrink_2$-operator for the p. The $\tilde{x}^{k+1}$-update uses one step of gradient descent.

We took the time-step as $\delta = 0.005$, and we took the PDHG steps as $\sigma = 20$, and $\tau = 0.25/\sigma$, where the 0.25 comes from the PDHG condition that $\sigma\tau\|D\|_2^2 < 1$, and $\|D\|_2 = 2 - \epsilon$ for some small $\epsilon > 0$.

As in the difference of norms example, we did have some points that were slower to converge. We alleviated this problem by rerunning the algorithm at the same point (without change σ nor τ) if we reached some maximum count, and sometimes took the solution if the maximum count was reached anyway.

The computational time on a $[-3, 3]^2$ grid, of mesh size $1/12 \approx 0.0833$ for each axis, averaged out to 0.412 seconds per point in MATLAB.

In this example, using 3 was slower than coordinate descent where it averaged to 0.133 seconds per point, and so we recommend using coordinate descent in this case.

1.6.3.2 gives the result of our algorithm.

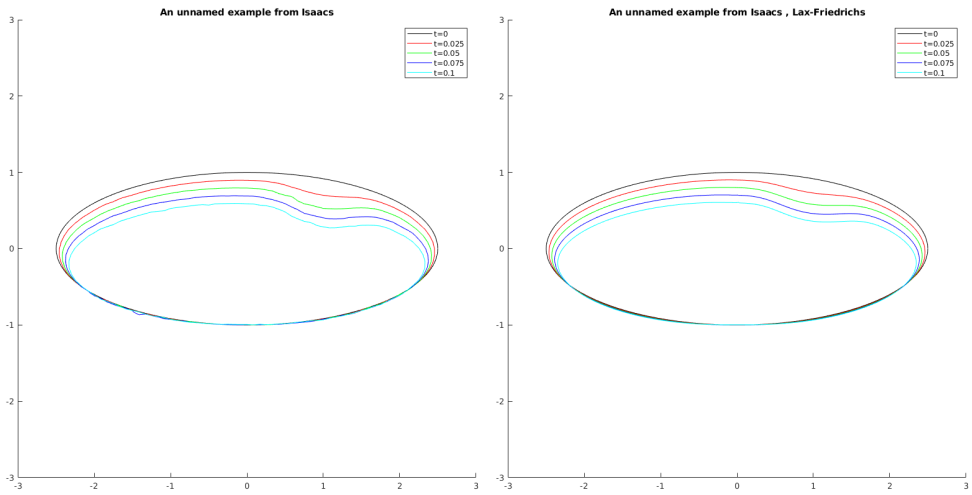


Figure 1.8: The zero level-sets for the HJE from an unnamed example of Isaacs. The times we computed were $t = 0.025$, 0.05, 0.075, and 0.1. The left figure is the result of our algorithm, while the right figure is the result of Lax-Friedrichs. Here we see this example is our harshest critic. But this is not surprising because the initial condition is a fully convex function, whereas we'd rather have it be convex-concave. And also the Hamiltonian is not bounded below with respect to q which as an assumption of the Hopf formula. Neverthless our algorithm is still able to achieve a result similar to Lax-Friedrichs, which might actually be the surprising part. We also note that we only used one initial guess here, but using multiple initial guess (around 5) smooths out the curves.

1.6.3.3 Implementation details for the (unnamed) example from Isaacs with convex-concave initial conditions

We take $c(x)$ to be the same as in the fully convex initial conditions (see 1.15). The initial condition of our HJE PDE is

$$g(x_1, x_2) = -1/2 + (1/2)(((2.5)x_1)^2 - (x_2)^2)$$

Here we have convex-concave initial conditions, and our algorithm works well. This is an example of a convex-concave game, in which [35] have also applied their method to linear differential games.

We use 4, and as in all other examples here, we only have one initial guess. We compute our solutions in a 2-dimensional $[-3, 3]^2$ grid. The $\tilde{p}^{k+1}$-update utilizes a combination of gradient descent for the q, and the $shrink_2$-operator the p. The $\tilde{x}^{k+1}$-update uses one step of gradient descent.

We took the time-step as $\delta = 0.005$, and we took the PDHG steps as $\sigma = 2$ for $t = 0.025$, 0.05, 0.075, and $\sigma = 10$ for $t = 0.1$. We always chose $\tau = 0.25/\sigma$, where the 0.25 comes from the PDHG condition that $\sigma\tau\|D\|_2^2 < 1$, and $\|D\|_2 = 2 - \epsilon$ for some small $\epsilon > 0$.

We computed this example in a $[-3, 3]^2$ grid with mesh size $1/12 \approx 0.0833$ for each axis. The computational time averaged to 0.125 seconds per point.

As in the difference of norms example, we did have some points that were slower to converge. We alleviated this problem by rerunning the algorithm at the same point (without change σ nor τ) if we reached some maximum count, and sometimes took the solution if the maximum count was reached anyway.

1.6.3.3 gives the result of our algorithm.

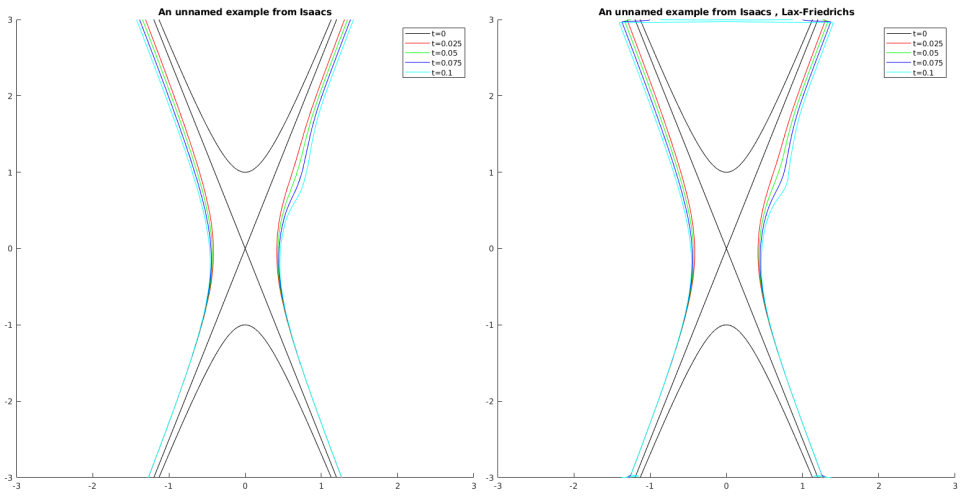


Figure 1.9: The zero level-sets for the HJE from an (unnamed) example of Isaacs with convex-concave initial conditions. The times we computed were $t = 0.025$, 0.05, 0.075, and 0.1. The left figure is the result of our algorithm, while the right figure is the result of Lax-Friedrichs.

1.6.4 Quadcopter (a.k.a. Quadrotor or Quad rotorcraft) (Optimal Control)

1.6.4.1 From optimal control to HJE for the quadcopter

A quadcopter is a multirotor helicopter that utilizes four rotors to propel itself across space. The dynamics of a quadcopter [52] are:

$$\begin{cases} \ddot{x} &= \frac{u}{m}\left(\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)\right) \\ \ddot{y} &= \frac{u}{m}\left(-\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\sin(\psi)\right) \\ \ddot{z} &= \frac{u}{m}\cos(\theta)\cos(\phi) - g \\ \ddot{\psi} &= \tilde{\tau}_{\psi} \\ \ddot{\theta} &= \tilde{\tau}_{\theta} \\ \ddot{\phi} &= \tilde{\tau}_{\phi} \end{cases}$$

where (x, y, z) is the position of the quadcopter in space, and (ψ, θ, ϕ) is the angular orientation of the quadcopter (a.k.a. Euler angles). The above second-order system turns into

the first-order system,

$$\begin{cases} \dot{x}_1 &= x_2 \\ \dot{y}_1 &= y_2 \\ \dot{z}_1 &= z_2 \\ \dot{\psi}_1 &= \psi_2 \\ \dot{\theta}_1 &= \theta_2 \\ \dot{\phi}_1 &= \phi_2 \\ \dot{x}_2 &= \frac{u}{m}\left(\sin(\phi_1)\sin(\psi_1) + \cos(\phi_1)\cos(\psi_1)\sin(\theta_1)\right) \\ \dot{y}_2 &= \frac{u}{m}\left(-\cos(\psi_1)\sin(\phi_1) + \cos(\phi_1)\sin(\theta_1)\sin(\psi_1)\right) \\ \dot{z}_2 &= \frac{u}{m}\cos(\theta_1)\cos(\phi_1) - g \\ \dot{\psi}_2 &= \tilde{\tau}_\psi \\ \dot{\theta}_2 &= \tilde{\tau}_\theta \\ \dot{\phi}_2 &= \tilde{\tau}_\phi \end{cases}$$

and so the right-side becomes our $\mathbf{f}(\mathbf{x}, \boldsymbol{\alpha})$. Here the controls are the variable u, $\tilde{\tau}_\psi$, $\tilde{\tau}_\theta$, $\tilde{\tau}_\phi$.

This is a 12-dimensional, nonlinear, optimal control problem.

Denoting $\mathbf{x} = (x_1, y_1, z_1, \psi_1, \theta_1, \phi_1, x_2, y_2, z_2, \psi_2, \theta_2, \phi_2)$, then our cost-functional is,

$$J[u, \tilde{\tau}_\psi, \tilde{\tau}_\theta, \tilde{\tau}_\phi] = g(\mathbf{x}(0)) + \int_0^t 2 + \|(u(s), \tilde{\tau}_\psi(s), \tilde{\tau}_\theta(s), \tilde{\tau}_\phi(s))\|_2^2 \, ds \tag{1.16}$$

where this cost functional was chosen to follow [120] and [66]. Therefore, our Hamiltonian

becomes,

$$H(\mathbf{x},\mathbf{p},t) = \max_{u,\tilde{\tau}_\psi,\tilde{\tau}_\theta,\tilde{\tau}_\phi} \left\{ \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} \psi_2 \\ \theta_2 \\ \phi_2 \end{bmatrix} \cdot \begin{bmatrix} p_4 \\ p_5 \\ p_6 \end{bmatrix} \right.$$
$$+\frac{u}{m} \begin{bmatrix} \sin(\phi_1)\sin(\psi_1) + \cos(\phi_1)\cos(\psi_1)\sin(\theta_1) \\ -\cos(\psi_1)\sin(\phi_1) + \cos(\phi_1)\sin(\theta_1)\sin(\psi_1) \\ \cos(\theta_1)\cos(\phi_1) \end{bmatrix} \cdot \begin{bmatrix} p_7 \\ p_8 \\ p_9 \end{bmatrix} - p_9 g + \begin{bmatrix} \tilde{\tau}_\psi \\ \tilde{\tau}_\theta \\ \tilde{\tau}_\phi \end{bmatrix} \cdot \begin{bmatrix} p_{10} \\ p_{11} \\ p_{12} \end{bmatrix}$$
$$\left. -2 - ||u||^2 - ||\tilde{\tau}_\psi||^2 - ||\tilde{\tau}_\theta||^2 - ||\tilde{\tau}_\phi||^2 \right\}$$

$$= \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} \psi_2 \\ \theta_2 \\ \phi_2 \end{bmatrix} \cdot \begin{bmatrix} p_4 \\ p_5 \\ p_6 \end{bmatrix}$$
$$+ \frac{1}{4m} \left\| \begin{bmatrix} \sin(\phi_1)\sin(\psi_1) + \cos(\phi_1)\cos(\psi_1)\sin(\theta_1) \\ -\cos(\psi_1)\sin(\phi_1) + \cos(\phi_1)\sin(\theta_1)\sin(\psi_1) \\ \cos(\theta_1)\cos(\phi_1) \end{bmatrix} \cdot \begin{bmatrix} p_7 \\ p_8 \\ p_9 \end{bmatrix} \right\|^2$$
$$- p_9 g + \frac{1}{4}\|p_{10}\|^2 + \frac{1}{4}\|p_{11}\|^2 + \frac{1}{4}\|p_{12}\|^2 - 2$$

1.6.4.2 Implementation details for the quadcopter

Here we have,

$$g(x) = -1/2 + (1/2)\left\langle A^{-1}x, x\right\rangle, \quad A = \mathrm{diag}(0.2, 1, 1, \ldots, 1).$$

In this case, we use the algorithm based on the generalized Hopf formula, 2.

We compute our solutions in a two dimensional slice of $\mathbb{R}^{12}$:

$$[-1,1] \times \{0\} \times \{0\} \times [-1,1] \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\},$$

i.e. we vary the x-coordinate, as well as the x-velocity-coordinate. Recall the order of the coordinates are: $\mathbf{x} = (x_1, y_1, z_1, \psi_1, \theta_1, \phi_1, x_2, y_2, z_2, \psi_2, \theta_2, \phi_2)$.

For both the $\tilde{p}^{k+1}$-update and the $\tilde{x}^{k+1}$-update, we used gradient descent, except for the update involving $g^*(p_1)$, where we did a proximal-gradient step, i.e. we performed a gradient descent, ignoring g^*, and then we fed the result into $\text{prox}_{\sigma(g^*)}(\cdot)$. This can be seen as a proximal-gradient step [103] (Section 4.2).

In this example, we chose as time-step size $\delta = 0.005$, and we chose times $t = 0.025, 0.05, 0.075$. For the PDHG step-sizes, we chose $\sigma = 5$, and $\tau = 0.25/\tau$, where as stated above, the 0.25 comes from the PDHG requirement $\sigma\tau\|D\|_2^2 < 1$, and $\|D\|_2 = 2 - \epsilon$, for some small $\epsilon > 0$.

The computation was done on a $[-1, 1]^2$ grid, with mesh size 0.01 in each axis. The computational time averaged to 0.0733 seconds per point in MATLAB.

In this case, not only are we able to compute level-sets of the HJE, but we are also able to take advantage of the characteristic curve/optimal trajectory generation that is freely offered by our algorithm.

To generate the curves/trajectories, we took a randomly-generated terminal point, which was exactly

$$x_{\text{target}} = (0.36, -0.62, -0.06, 0.23, 0.85, -0.66, 0.72, -0.45, 0.15, -0.75, 0.04, -0.83) \quad (1.17)$$

and we computed up to $t = 6$ seconds. We chose a time-step of $\delta = 0.05$, and we chose $\sigma = 11$ (and $\tau = 0.24/\sigma$ (as opposed to 0.25 as the latter will not converge). This took about 24s to compute in MATLAB. We verified our result by directly minimizing a discretized version of (1.16) (see also 1.4.1 on how we discretized), which is a direct collocation method [135]. We performed the optimization using a standard MATLAB minimization solver (fmincon with the SQP algorithm), and this agreed with our results. The solver took 133-347s to compute the trajectories, depending on the accuracy criteria, and we note that fmincon converges to our splitting result the longer we let the algorithm run. So in this case, 5-10+ times speedup. Computing trajectories at other points have found around an 8-10+ speedups.

1.6.4.2 gives the zero level-sets of the HJE, and 1.6.4.2 gives the result of computing the curves/trajectories. 1.6.4.2 plots the x, y, and z positions of the quadcopter as it moves through time.

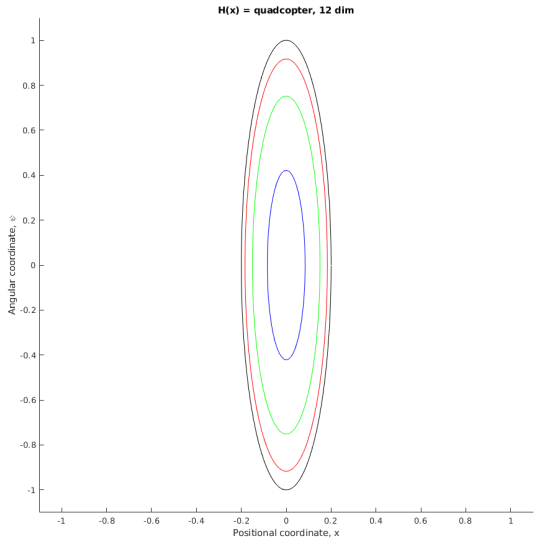


Figure 1.10: Here we compute the zero level-sets for the HJE arising from the quadcopter. The x-axis is the x_1-position of the quadcopter, and the y-axis is the angular position in the ψ_1 coordinate. The zero level-sets are computed for $t = 0.025$, 0.05, and 0.075. This is a 12-dimensional, nonlinear optimal control problem.

1.7 Discussion and Future Work

In this paper, we have presented a splitting method to compute solutions to general (i.e. convex and nonconvex) Hamilton-Jacobi equations which arise from general (i.e. linear and nonlinear) optimal control problems and general differential games problems.

Some nice properties of our algorithm include: (1) relatively fast computation of solutions in high-dimensions, especially when we parallelize the algorithm which is embarrassingly parallelizable [64] (2) it can generate optimal trajectories of the optimal control/differential games problems, (3) it can compute problems with non-linear ODEs, (4) it can compute solutions for nonconvex Hamiltonians, (5) and the algorithm is embarrassingly parallelizable, i.e. each core can use the algorithm to compute the solution at a point, so given N cores we can compute solutions of the HJE at N points simultaneously.

Splitting applied to optimal control problems has been used by [95] where they apply it to cost functionals having a quadratic and convex term. In terms of Hamilton-Jacobi equations,

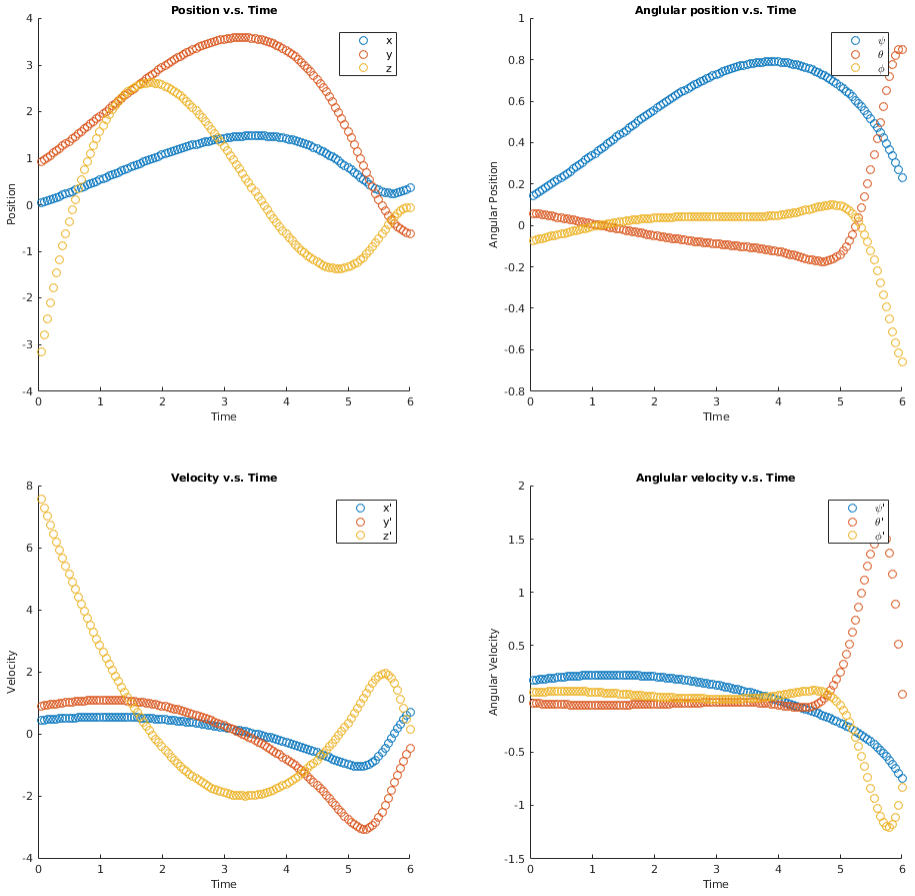


Figure 1.11: Here we compute the characteristic curves/optimal trajectories for the quadcopter. We are computing at the terminal point in (1.17) and we are computing at the terminal time $t = 6$ seconds. A plot of the trajectories computing using a different algorithm – SQP – looks identical.

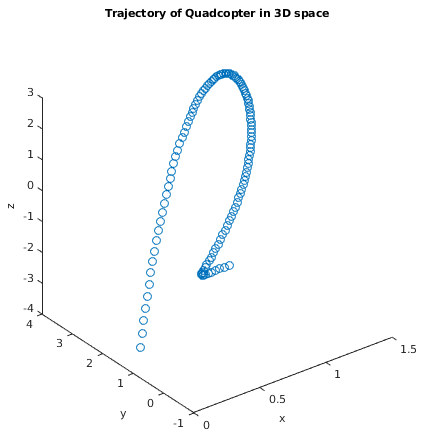


Figure 1.12: We plot the (x, y, z) coordinates of the quadcopter to give a plot of the trajectory of the quadcopter in 3D space.

the authors in Kirchner et al. [74] (2018) effectively applied it to Hamilton-Jacobi equations arising from linear optimal control problems by using the Hopf formula. They make use of the Hopf formula and the closed-form solution to linear ODEs to not only solve HJE, but to also directly compute optimal trajectories in high-dimensional systems. The authors of this current paper have been working in parallel and also applied splitting to HJE and trajectory generation for nonlinear optimal control problems and minimax differential games.

On a related note, see also previous work by Kirchner et al. [73] where they apply the Hopf formula to differential games and show that complex “teaming" behavior can arise, even with linearized pursuit-evasion models.

As far as we know, the idea to use splitting for differential games problem for the discretization in equation (1.13) and (1.14) is new. And we believe it is worth examining if this PDHG-inspired method to solve minimax/saddle-point problems may apply to more general minimax/saddle-point optimization problems.

The proof of convergence and the proof of approximation for our algorithm is still a work-in-progress. But it seems to be that for the examples in 1.6, we get relatively the correct answer, and our algorithm seems to scale linearly with dimension for even a nonlinear optimal control problem requiring nonconvex optimization (see 1.6.1.3).

We also believe that due to the deep connection between Hamilton-Jacobi equations and optimization methods (1.5.6), it is worthwhile to examine why our algorithm works. Not only that, but for our examples in 1.6, we were able to perform nonconvex optimization with only a single initial guess, whereas coordinate descent required multiple. And the authors also believe it is worth examining the algorithms 3 and 4 as the computation of minimax differential games problems using a splitting method seems new. In essence, it may be possible to generalize these algorithms to apply to general minimax/saddle-point problems with continuous constraints.

Some improvements to our algorithm for differential games problems (3 and 4) can be foreseen:

1. We have found speed-ups to our algorithm when we use acceleration methods [16, 17]

2. One may be able to devise a more sophisticated stopping criteria as that in Kirchner et al. [74], where they apply a step-size-dependent stopping criteria based off work by [53].

3. We would also like to utilize higher-order approximations for the ODE and integral when discretizing the value functions of the optimal control or differential games problems. We note that for the Lax discretizations (1 and 3), one can average the forward and backward Euler approximations to obtain higher accuracy, analogous to how the trapezoidal approximation is the average of the two.

4. And we believe we might be able to make use of having a closed-form solution, or an approximate solution, to computing the characteristic curves, i.e. closed form solutions to $\frac{d}{dt}\mathbf{x}(s) = H_p(\mathbf{x}(s), \mathbf{p}(s), s)$ and $\frac{d}{dt}\mathbf{p}(s) = -H_x(\mathbf{x}(s), \mathbf{p}(s), s)$, much as in [74], where they make use of having a closed-form solution to linear differential equations by utilizing the exponential operator.

5. There could be an advantage in combining the splitting method to pseudo-spectral methods [111].

6. In the algorithms for differential equations 3 and 4, we are solving a saddle-point problem using gradients. We might obtain faster convergence if we used a Hessian-inspired method, such as split form of BFGS.

1.8 Acknowledgements

We give our deepest thanks to Matthew R. Kirchner and Gary Hewer for their enlightening discussions and suggested edits during this work. They provided us with a wealth of information on the history of the subject and existing methods, as well as important problems. And they were generous in giving suggestions on future directions.

1.9 Appendix: A practical tutorial for implementation purposes

1.9.1 Optimal Control

Suppose we want to compute the Hamilton-Jacobi equations associated to the following optimal control problem:

$$\phi(x,t) = \min_{\mathbf{x}(\cdot),\mathbf{u}(\cdot)} \left\{ g(\mathbf{x}(0)) + \int_0^t L(\mathbf{x}(s),\mathbf{u}(s),s)\,ds \right\} \tag{1.18}$$

where $\mathbf{x}(\cdot)$ and $\mathbf{u}(s)$ satisfy the ODE

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s),\mathbf{u}(s),s), \quad 0 < s < t \\ \mathbf{x}(t) = x \end{cases} \tag{1.19}$$

Here $(x,t) \in \mathbb{R}^n \times [0,\infty)$ are fixed, and is the point that we want to compute the HJE solution φ.

Then we can use 1 or 2, which we describe in more detail below.

1.9.1.1 Practical tutorial for 1

If we want to use the discretized Lax formula (with a backward Euler discretization of the ODE dynamics), then:

1. Discretize the time domain:

$$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

2. Approximate (1.19) using backward Euler, and also discretize (1.18) to obtain (as in (1.9)),

$$\phi(x,t) \approx \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^N H(x_j, p_j, s_j) \right\}$$

where $x_j = \mathbf{x}(s_j)$, and $p_j = \mathbf{p}(x_j)$. Let us denote $x = x_{\text{target}}$ to clarify notation.

3. Initialize:

   (a) Choose $\delta > 0$, set $N = t/\delta$ (although note that since we are using the zero-th time-step, then we are updating $N+1$ points).

   (b) Randomly initialize $\tilde{x}^0 := (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$.

   (c) Randomly initialize $\tilde{p}^0 := (p_0^0, p_1^0, \ldots, p_{N-1}^0, p_N^0)$, but with $p_0^0 \equiv 0$, as we won't be updating p_0^0; it is only there for computational accounting.

   (d) Set $\tilde{z}^0 := (z_0^0, z_1^0, \ldots, z_N^0) = (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$.

   (e) Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).

   (f) Choose some tolerance tol > 0 small.

4. Set

$$D = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ -I & I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -I & I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \end{pmatrix}$$

where **0** is a (dim × dim) zero matrix, where dim is the size of the space variable, and I is the (dim × dim) identity matrix.

Note that in the algorithm, we can replace $(D\tilde{z}^k)_j = z_j^k - z_{j-1}^k$, and similarly with $(D^T\tilde{p}^k)_j = p_j^k - p_{j+1}^k$, so we can save time by not performing a full matrix multiplication.

5. Then perform the algorithm found in 6.

Algorithm 6 Practical tutorial for the Lax formula with backward Euler, for Optimal Control

---

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 >$ tol or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 >$ tol) and (count $<$ max_count) **do**

for $j = 1$ to N **do**

$p_j^{k+1} = \arg\min_p \left\{\delta H(x_j^k, p, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D\tilde{z}^k)_j)\|_2^2\right\}$

for $j = 0$ **do**

$x_0^{k+1} = \arg\min_x \left\{g(x) + \frac{1}{2\tau}\|x - (x_0^k - \tau(D^T\tilde{p}^k)_0)\|_2^2\right\}$ (note $p_0^k = 0$)

for $j = 1$ to $N - 1$ **do**

$x_j^{k+1} = \arg\min_x \left\{-\delta H(x, p_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D^T\tilde{p}^k)_j)\|_2^2\right\}$

for $j = 0$ to N **do**

$z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$

fval $= g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1}\rangle - \delta \sum_{j=1}^N H(x_j, p_j, s_j)$

return fval

1.9.1.2 Practical tutorial for 2

If we want to use the discretized Hopf formula, then:

1. Discretize the time domain:

$$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

2. Approximate (1.19) using backward Euler, and also discretize (1.18) to obtain (as in (1.11)),

$$\phi(x,t) \approx \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=1}^N} \left\{ -g^*(p_1) + \langle p_N, x\rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j\rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\}$$

where $x_j = \mathbf{x}(s_j)$, and $p_j = \mathbf{p}(x_j)$. Let us denote $x = x_{\text{target}}$ to clarify notation.

3. Initialize:

   (a) Choose $\delta > 0$, set $N = t/\delta$.

   (b) Randomly initialize $\tilde{x}^0 := (x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$.

   (c) Randomly initialize $\tilde{p}^0 := (p_1^0, \ldots, p_{N-1}^0, p_N^0)$.

   (d) Set $\tilde{z}^0 := (z_1^0, \ldots, z_N^0) = (x_1^0, \ldots, x_{N-1}^0, x_N^0)$.

   (e) Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).

   (f) Choose some tolerance tol > 0 small.

4. Set

$$D = \begin{pmatrix} I & -I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I & -I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \end{pmatrix}$$

where $\mathbf{0}$ is a (dim $\times$ dim) zero matrix, where dim is the size of the space variable, and I is the (dim $\times$ dim) identity matrix.

Note we can replace $(D^T \tilde{z^k})_j = z_j^k - z_{j-1}^k$ and $(D\tilde{p}^k)_j = p_j^k - p_{j+1}^k$, so we can save time by not performing a full matrix multiplication.

Also note that the D here is different than in 1 and 6.

5. Then perform the algorithm found in 7

Algorithm 7 Practical tutorial for the Hopf formula, for Optimal Control

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 >$ tol or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 >$ tol) and (count $<$ max_count) **do**

for $j = 1$ **do**

$p_j^{k+1} = \arg\min_p \left\{ g^*(p) + \delta H(x_1^k, p, s_1) + \frac{1}{2\sigma}\|p - (p_1^k + \sigma(D^T \tilde{z}^k)_1)\|_2^2 \right\}$

for $j = 2$ to N **do**

$p_j^{k+1} = \arg\min_p \left\{ \delta H(x_j^k, p, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D^T \tilde{z}^k)_j)\|_2^2 \right\}$

for $j = 1$ to $N - 1$ **do**

$x_j^{k+1} = \arg\min_x \left\{ -\delta H(x, p_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D\tilde{p}^k)_j)\|_2^2 \right\}$

for $j = 1$ to N **do**

$z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$

fval $= -g^*(p_1) + \langle p_N, x_{\text{target}} \rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j)$

return fval

1.9.2 Differential Games

Suppose we want to solve the differential games problem with the following dynamics,

$$\begin{cases} \begin{pmatrix} \dot{\mathbf{x}}(s) \\ \dot{\mathbf{y}}(s)) \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \\ \mathbf{f}_2(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \end{pmatrix} & 0 < s < t \\ \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \end{cases} \tag{1.20}$$

and with the following value function,

$$\phi(x,y,t) = \inf_{\boldsymbol{\alpha}(\cdot),\mathbf{x}(\cdot)} \sup_{\boldsymbol{\beta}(\cdot),\mathbf{y}(\cdot)} \left\{ g(\mathbf{x}(0),\mathbf{y}(0)) + \int_0^t L(\mathbf{x}(s),\mathbf{y}(s),\boldsymbol{\alpha}(s),\boldsymbol{\beta}(s),s)\, ds \right\} \tag{1.21}$$

Then we can discretize the above equation and use 3 and 4, which we describe in more detail below.

1.9.2.1 Practical tutorial for 3

If we want to use the discretized Lax formula for differential games (with a backward Euler discretization of the ODE dynamics) (1.13), then:

1. Discretize the time domain:

   $$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

   In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

2. Approximate (1.20) using backward Euler, and also discretize (1.21) to obtain (as in (1.13)),

$$\phi(x,y,t) \approx \min_{\{q_j\}_{j=1}^N} \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \max_{\{y_j\}_{j=0}^N} \left\{ g(x_0,y_0) + \sum_{j=1}^N \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} x_j - x_{j-1} \\ y_j - y_{j-1} \end{pmatrix} \right\rangle - \delta \sum_{j=1}^N H(x_j,y_j,p_j,-q_j,s_j) \right\}$$

   where $x_j = \mathbf{x}(s_j)$, and similarly for y_j, q_j, and p_j. Let us denote $x = x_{\text{target}}$ and $y = y_{\text{target}}$ to clarify notation.

3. Initialize:

   1. Choose $\delta > 0$, set $N = t/\delta$ (although note that since we are using the zero-th time-step, then we are updating $N+1$ points).
   2. Randomly initialize $\tilde{x}^0 := (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$, and similarly for $\tilde{y}^0$.
   3. Randomly initialize $\tilde{p}^0 := (p_0^0, p_1^0, \ldots, p_{N-1}^0, p_N^0)$, but with $p_0^0 \equiv 0$, as we won't be updating p_0^0; it is only there for computational accounting. Do a similary initialization for $\tilde{q}^0$.

4. Set $\tilde{z}^0 := (z_0^0, z_1^0, \ldots, z_N^0) = (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, and set $\tilde{w}^0 := (w_0^0, w_1^0, \ldots, w_N^0) = (y_0^0, y_1^0, \ldots, y_{N-1}^0, y_N^0)$.

5. Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0, 1]$ (we suggest $\theta = 1$).

6. Choose some tolerance tol > 0 small.

4. Set

$$D = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ -I & I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -I & I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \end{pmatrix}$$

where $\mathbf{0}$ is a (dim $\times$ dim) zero matrix, where dim is the size of the space variable, and I is the (dim $\times$ dim) identity matrix. Note this is a very sparse matrix and we take advantage of this.

Note that we can replace $D\tilde{z}^k = z_j^k - z_{j-1}^k$, and $D\tilde{w}^k = w_j^k - w_{j-1}^k$, and $D^T\tilde{p}^k = p_j^k - p_{j+1}^k$, and $D^T\tilde{q}^k = q_j^k - q_{j+1}^k$, so we don't have to perform the full matrix multiplication.

5. Then perform the algorithm found in 8.

1.9.2.2 Practical tutorial for 4

If we want to use the discretized Hopf formula for differential games (with a backward Euler discretization of the ODE dynamics) (1.14), then:

1. Discretize the time domain:

$$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

Algorithm 8 Practical tutorial for the Lax formula with backward Euler,for Differential Games

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 >$ tol or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 >$ tol or $\|\tilde{y}^{k+1} - \tilde{y}^k\|_2^2 >$ tol or $\|\tilde{q}^{k+1} - \tilde{q}^k\|_2^2 >$ tol) and (count $<$ max_count) **do**

for $j = 1$ to N **do**

$p_j^{k+1} = \arg\min_p \left\{\delta H(x_j^k, y_j^k, p, -q_j^k, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D\tilde{z}^k)_j)\|_2^2\right\}$

for $j = 1$ to N **do**

$q_j^{k+1} = \arg\min_q \left\{-\delta H(x_j^k, y_j^k, p_j^{k+1}, -q, s_j) + \frac{1}{2\sigma}\|q - (q_j^k + \sigma(D\tilde{w}^k)_j)\|_2^2\right\}$

for $j = 0$ **do**

$x_0^{k+1} = \arg\min_x \left\{g(x, y_0^k) + \frac{1}{2\tau}\|x - (x_0^k - \tau(D^T\tilde{p}^k)_0)\|_2^2\right\}$

for $j = 1$ to $N - 1$ **do**

$x_j^{k+1} = \arg\min_x \left\{-\delta H(x, y_j^k, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D^T\tilde{p}^k)_j)\|_2^2\right\}$

for $j = 0$ **do**

$y_0^{k+1} = \arg\min_y \left\{-g(x_0^{k+1}, y) + \frac{1}{2\tau}\|y - (y_0^k - \tau(D^T\tilde{q}^k)_0)\|_2^2\right\}$

for $j = 1$ to $N - 1$ **do**

$y_j^{k+1} = \arg\min_y \left\{-\delta H(x_j^{k+1}, y, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|y - (y_j^k - \tau(D^T\tilde{q}^k)_j)\|_2^2\right\}$

for $j = 0$ to N **do**

$z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$

$w_j^{k+1} = y_j^{k+1} + \theta(y_j^{k+1} - y_j^k)$

fval $= g(x_0, y_0) + \sum_{j=1}^N \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} x_j - x_{j-1} \\ y_j - y_{j-1} \end{pmatrix} \right\rangle - \delta \sum_{j=1}^N H(x_j, y_j, p_j, -q_j, s_j)$

return fval

2. Approximate (1.20) using backward Euler, and also discretize (1.21) to obtain (as in (1.14)),

$$\phi(x,y,t) \approx \min_{\{q_j\}_{j=1}^N} \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=1}^N} \max_{\{y_j\}_{j=1}^N} \left\{ -e^*(p_1) - h_*(-q_1) + \left\langle \begin{pmatrix} p_N \\ -q_N \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle \right.$$
$$\left. + \sum_{j=1}^{N-1} \left\langle \begin{pmatrix} p_j - p_{j+1} \\ -(q_j - q_{j+1}) \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\rangle - \delta \sum_{j=1}^{N} H(x_j, y_j, p_j, -q_j, s_j) \right\}$$

where $x_j = \mathbf{x}(s_j)$, and similarly for y_j, q_j, and p_j. Let us denote $x = x_{\text{target}}$ and $y = y_{\text{target}}$ to clarify notation.

3. Initialize:

   1. Choose $\delta > 0$, set $N = t/\delta$.

   2. Randomly initialize $\tilde{x}^0 := (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$, and similarly for $\tilde{y}^0$.

   3. Randomly initialize $\tilde{p}^0 := (p_0^0, p_1^0, \ldots, p_{N-1}^0, p_N^0)$, but with $p_0^0 \equiv 0$, as we won't be updating p_0^0; it is only there for computational accounting. Do the same initialization for $\tilde{q}^0$.

   4. Set $\tilde{z}^0 := (z_0^0, z_1^0, \ldots, z_N^0) = (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, and set $\tilde{w}^0 := (w_0^0, w_1^0, \ldots, w_N^0) = (y_0^0, y_1^0, \ldots, y_{N-1}^0, y_N^0)$.

   5. Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).

   6. Choose some tolerance tol > 0 small.

4. Set

$$D = \begin{pmatrix} I & -I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I & -I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \end{pmatrix}$$

where $\mathbf{0}$ is a (dim $\times$ dim) zero matrix, where dim is the size of the space variable, and I is the (dim $\times$ dim) identity matrix. Note this is a very sparse matrix and we take advantage of this.

Alose note that we can replace $D^T\tilde{z}^k = z_j^k - z_{j-1}^k$, and $D^T\tilde{w}^k = w_j^k - w_{j-1}^k$, and $D\tilde{p}^k = p_j^k - p_{j+1}^k$, and $D\tilde{q}^k = q_j^k - q_{j+1}^k$, so we don't have to perform the full matrix multiplication.

5. Then perform the algorithm found in 9.

Algorithm 9 Practical tutorial for the Lax formula with backward Euler, for Differential Games

Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0, \infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 >$ tol or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 >$ tol or $\|\tilde{y}^{k+1} - \tilde{y}^k\|_2^2 >$ tol or $\|\tilde{q}^{k+1} - \tilde{q}^k\|_2^2 >$ tol) and (count $<$ max_count) **do**

for $j = 1$ **do**

$p_1^{k+1} = \arg\min_p \left\{ e^*(p) + \delta H(x_1^k, y_1^k, p, -q_1^k, s_1) + \frac{1}{2\sigma}\|p - (p_1^k + \sigma(D^T \tilde{z}^k)_1)\|_2^2 \right\}$

for $j = 2$ to N **do**

$p_j^{k+1} = \arg\min_p \left\{ \delta H(x_j^k, y_j^k, p, -q_j^k, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D^T \tilde{z}^k)_j)\|_2^2 \right\}$

for $j = 1$ **do**

$q_1^{k+1} = \arg\min_q \left\{ -h_*(-q) - \delta H(x_1^k, y_1^k, p_1^{k+1}, -q, s_1) + \frac{1}{2\sigma}\|q - (q_1^k + \sigma(D^T \tilde{w}^k)_1)\|_2^2 \right\}$

for $j = 2$ to N **do**

$q_j^{k+1} = \arg\min_q \left\{ -\delta H(x_j^k, y_j^k, p_j^{k+1}, -q, s_j) + \frac{1}{2\sigma}\|q - (q_j^k + \sigma(D^T \tilde{w}^k)_j)\|_2^2 \right\}$

for $j = 1$ to $N - 1$ **do**

$x_j^{k+1} = \arg\min_x \left\{ -\delta H(x, y_j^k, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D\tilde{p}^k)_j)\|_2^2 \right\}$

for $j = 1$ to $N - 1$ **do**

$y_j^{k+1} = \arg\min_y \left\{ -\delta H(x_j^{k+1}, y, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|y - (y_j^k - \tau(D\tilde{q}^k)_j)\|_2^2 \right\}$

for $j = 1$ to N **do**

$z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$

$w_j^{k+1} = y_j^{k+1} + \theta(y_j^{k+1} - y_j^k)$

fval $= -e^*(p_1) - h_*(-q_1) + \left\langle \begin{pmatrix} p_N \\ -q_N \end{pmatrix}, \begin{pmatrix} x_{\text{target}} \\ y_{\text{target}} \end{pmatrix} \right\rangle + \sum_{j=1}^{N-1} \left\langle \begin{pmatrix} p_j - p_{j+1} \\ -(q_j - q_{j+1}) \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\rangle - \delta \sum_{j=1}^{N} H(x_j, y_j, p_j, -q_j, s_j)$

return fval

CHAPTER 2

Many

Abstract: We consider the reinforcement learning problem of training multiple agents in order to maximize a shared reward. In this multi-agent system, each agent seeks to maximize the reward while interacting with other agents, and they may or may not be able to communicate. Typically the agents do not have access to other agent policies and thus each agent is situated in a non-stationary and partially-observable environment. In order to obtain multi-agents that act in a decentralized manner, we introduce a novel algorithm under the framework of centralized learning, but decentralized execution. This training framework first obtains solutions to a multi-agent problem with a single centralized joint-space learner. This centralized expert is then used to guide imitation learning for independent decentralized multi-agents. This framework has the flexibility to use any reinforcement learning algorithm to obtain the expert as well as any imitation learning algorithm to obtain the decentralized agents. This is in contrast to other multi-agent learning algorithms that, for example, can require more specific structures. We present some theoretical error bounds for our method, and we show that one can obtain decentralized solutions to a multi-agent problem through imitation learning. [84]

2.1 Introduction

Reinforcement Learning (RL) is the problem of finding an action policy that maximizes reward for an agent embedded in an environment [125]. It has recently has seen an explosion in popularity due to its many achievements in various fields such as, robotics [80], industrial applications [43], game-playing [93, 119, 117], and the list continues. However, most of these

achievements have taken place in the single-agent realm, where one does not have to consider the dynamic environment provided by interacting agents that learn and affect one another.

This is the problem of Multi-agent Reinforcement Learning (MARL) where we seek to find the best action policy for each agent in order to maximize their reward. The settings may be cooperative, and thus they might have a shared reward, or the setting may be competitive, where one agent's gain is another's loss. Some examples of a multi-agent reinforcement learning problem are: decentralized coordination of vehicles to their respective destinations while avoiding collision, or the game of pursuit and evasion where the pursuer seeks to minimize the distance between itself and the evader while the evader seeks the opposite. Other examples of multi-agent tasks can be found in [101] and [87].

The key difference between MARL and single-agent RL (SARL) is that of interacting agents, which is why the achievements of SARL cannot be absentmindedly transferred to find success in MARL. Specifically, the state transition probabilities in a MARL setting are inherently non-stationary from the perspective of any individual agent. This is due to the fact that the other agents in the environment are also updating their policies, and so the Markov assumptions typically needed for SARL convergence are violated. This aspect of MARL gives rise to instability during training, where each agent is essentially trying to learn a moving target.

In this work, we present a novel method for MARL in the cooperative setting (with shared reward). Our method first trains a centralized expert with full observability, and then uses this expert as a supervisor for independently learning agents. There are a myriad of imitation/supervised learning algorithms, and in this work we focus on adapting DAgger (Dataset Aggregation) [113] to the multi-agent setting. After the imitation learning stage, the agents are able to successfully act in a decentralized manner. We call this algorithm Centralized Expert Supervises Multi-Agents (CESMA). CESMA adopts the framework of centralized training, but decentralized execution [76], the end goal of which is to obtain multi-agents that can act in a decentralized manner.

2.2 Related works

The most straight-forward way of adapting single-agent RL algorithms to the multi-agent setting is by having agents be independent learners. This was applied in [127], but this training method gives instability issues, as the environment is non-stationary from the perspective of each agent [90, 10, 26]. This non-stationarity was examined in [97], and stabilizing experience replay was studied in [48].

Another common approach to stabilizing the environment is to allow the multi-agents to communicate. In [122], they examine this using continuous communications so one may backpropagate to learn to communicate. And in [46], they give an in-depth study of communicating multi-agents, and also provide training methods for discrete communication. In [105], they decentralize a policy by examining what to communicate and by utilizing supervised learning, although they mathematically solve for a centralized policy and their assumptions require homogeneous communicating agents.

Others approach the non-stationarity issue by having the agents take turns updating their weights while freezing others for a time, although non-stationarity is still present [36]. Other attempts adapt Q-learning to the multi-agent setting: Distributed Q-Learning [79] updates Q-values only when they increase, and updates the policy only for actions that are not greedy with respect to the Q-values, and Hysteretic Q-Learning [89] provides a modification. Other approaches examine the use of parameter sharing [62] between agents, but this requires a degree of homogeneity of the agents. And in [129], their approach to non-stationarity was to input other agents' parameters into the Q function. Other approaches to stabilize the training of multi-agents are in [121], where the agents share information before selecting their actions.

From a more centralized view point, [96, 107, 123] derived a centralized Q-value function for MARL, and in [132], they train a centralized controller and then sequentially select actions for each agent. The issue of an exploding action space was examined in [128].

A few works that follow the framework of centralized training, but decentralized execu-

tion are: RLar (Reinforcement Learning as Rehearsal) [76], COMA (Counterfactual Multi-Agent), and also [116, 49] – where the idea of knowledge-reuse is examined. In [33], they examine decentralization of policies from an information-theoretic perspective. There is also MADDPG [87], where they train in a centralized-critics decentralized-actors framework; after training completes, the agents are separated from the critics and can execute in a fully distributed manner.

For surveys of MARL, see articles in [9, 102].

2.3 Background

In this section we briefly review the requisite material needed to define MARL problems. Additionally we summarize some of the standard approaches in general reinforcement learning and discuss their use in MARL.

Dec-POMDP: A formal framework for multi-agent systems is called a decentralized partially-observable Markov decision process (Dec-POMDP) [7]. A Dec-POMDP is a tuple $(I, \mathcal{S}, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, P, R)$ where I is the finite set of agents indexed 1 to M, S is the set of states, $\mathcal{A}_i$ is the set of actions for agent i, and thus $\prod_{i=1}^{M} \mathcal{A}_i$ is the joint action space, $\mathcal{O}_i$ is the observation space of agent i, and thus $\prod_{i=1}^{M} \mathcal{O}_i$ is the joint observation space, $P = P(\mathbf{s}', \mathbf{o}|\mathbf{s}, \mathbf{a})$ (where $\mathbf{o} = (o_1, \ldots, o_M)$ and similarly for the others) is the state-transition probability for the whole system, and $R : \mathcal{S} \times \prod_{i=1}^{M} \mathcal{A} \to \mathcal{R}$ is the reward. In the case when the joint observations $\mathbf{o}$ equals the world state of the system, then we call the system a decentralized Markov decision process (Dec-MDP).

DAgger: The Dataset Aggregation (DAgger) algorithm [113] is an iterative imitation learning algorithm that seeks to learn a policy from expert demonstration. The main idea is to allow the learning policy to navigate its way through the environment, and have it query the expert on states that it sees. It does this by starting with a policy $\hat{\pi}_2$ which learns from the dataset of expert trajectories $\mathcal{D}_1$ through supervised learning. Using $\hat{\pi}_2$, a new dataset is generated by rolling out the policy and having the expert provide supervision on

the decisions that the policy made. This new dataset is aggregated with the existing set into $\mathcal{D}_2 \supset \mathcal{D}_1$. This process is iterated, i.e. a new $\hat{\pi}_3$ is trained, another new dataset is obtained and aggregated into $\mathcal{D}_3 \supset \mathcal{D}_2$ and so on. Learning in this way has been shown to be more stable and have nicer convergence properties as learning utilizes trajectories seen from the learner's state distribution, as opposed to only the expert's state distribution.

Policy Gradients (PG): One approach to RL problems are policy gradient methods [126]: instead of directly learning state-action values, the parameters θ of the policy π_θ are adjusted to maximize the objective,

$$J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}\left[Q^\pi(s,a)\right],$$

where p^π is the state distribution from following policy π. The gradient of the above expression can be written as [126, 125]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[(\nabla_\theta \log \pi_\theta(s|a))Q^\pi(s,a)].$$

Many policy gradient methods seek to reduce the variance of the above gradient estimate, and thus study how one estimates $Q^\pi(s,a)$ above [115]. For example, if we let $Q^\pi(s,a)$ be the sample return $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$, then we get the REINFORCE algorithm [68]. Or one can choose to learn $Q^\pi(s,a)$ using temporal-difference learning [124, 125], and would obtain the Actor-Critic algorithms [125]. Other policy gradients algorithms are: DPG [118], DDPG [81], A2C and A3C [92], to name a few.

Policy Gradients have been applied to multi-agent problems; in particular the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [87] uses an actor-critic approach to MARL, and this is the main baseline we test our method against. Another policy gradient method is by [50] called Counterfactual Multi-Agent (COMA), who also uses an actor-critic approach.

2.4 Methods

In this section, we explain the motivation and method of our approach: Centralized Expert Supervises Multi-Agents (CESMA).

2.4.1 Treating a multi-agent problem as a single-agent problem

Intuitively, an optimal strategy of a multi-agent problem could be found by a centralized expert with full observability. This is because the centralized controller has the most information available about the environment, and therefore would not pay a high of cost of partial-observability that independent learners might. This is discussed more in Section 2.5.

To find this centralized expert, we treat a multi-agent problem as a single agent problem in the joint observation and action space of all agents. This is done by concatenating the observations of all agents into one observation vector for the centralized expert, and the expert learns outputs that represent the joint actions of the agents.

Our framework does not impose any other particular constraints on the expert. Any expert architecture that outputs an action that represents the joint-actions of all of the agents may be used. Due to that, we are free to use any standard RL algorithm for the expert such as DDPG, DQN, or potentially even analytically derived experts.

2.4.2 Curse of dimensionality and some reliefs

When training a centralized expert, both the observation space and action space can grow exponentially. For example, if we use a DQN for our centralized expert then the number of output nodes will typically grow exponentially with respect to the number of agents. This is due to each output needing to correspond to an element in the joint action space $\prod_{i=1}^{M} \mathcal{A}_i$.

One way to deal with the exponential growth in the joint action space is, rather than requiring the centralized expert to move all agents simultaneously, we can restrict it to moving only one agent at a time, while the others default to a “do nothing" action (assuming one is

available). Effectively this mean the growth in the action space is now linear with respect to the number of agents. We provide an experiment where we were able to decentralize such an expert in Appendix 2.8.4.

This problem has also been studied by QMIX [107] and VDNs (Value-Decomposition Networks) [123], where exponential scaling of the output space is solved by having separate Q values for each agent and then using the sum as a system Q. Due to the nature of the reduction technique, these approaches require their own theorems of convergence. Other techniques such as action branching [128] have been considered. An experiment where we decentralize QMIX/VDN-like centralized expert models (which grow linearly in the number of output nodes) can be found in Appendix 2.8.3.

In our experiments, we use DDPG (with Gumbel-Softmax action selection if the environment is discrete, as MADDPG does also) to avoid the exploding number of input nodes of the observation space, as well as exploding number of output nodes of the action space. Under this paradigm, the input and output nodes only grow linearly with the number of agents, as the output nodes of a neural network in DDPG is the chosen joint action, as opposed to a DQN, where the output nodes must enumerate all possible joint actions.

2.4.3 CESMA for multi-agents without communication

To perform imitation learning to decentralize the expert policy, we adapt DAgger to the multi-agent setting. There are many ways DAgger can be applied to multi-agents, but we implement a method that best allows the theoretical analysis from [113] to apply: Namely after training the expert, we do supervised learning on a single neural network with disconnected components, each corresponding to one of the agents.

In more detail, after training a centralized expert π^*, we initialize the M agents $\pi_1, \ldots, \pi_M$, and initialize the dataset of observation-label pairs $\mathcal{D}$. The agents then step through the environment, storing each observation $\mathbf{o} = (o_1, \ldots, o_M)$ (where o_i is agent i's observation) the multi-agents encounter, along with the expert action label $\mathbf{a}^* = \pi^*(\mathbf{o})$ (where $\mathbf{a}^* = (a_1^*, \ldots, a_M^*)$ and a_i^* is agent i's expert label action); so we store the pair $(\mathbf{o}, \mathbf{a}^*)$ in $\mathcal{D}$

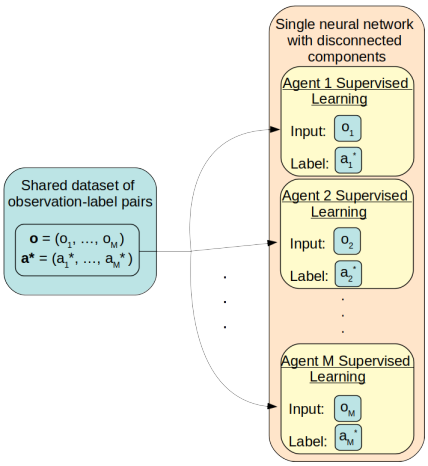


Figure 2.1: The centralized expert labels guide supervised learning for the multi-agents. The multi-agents make up the disconnected components of a single-agent learner.

at each timestep. After $\mathcal{D}$ has reached a sufficient size, at every kth time step (chosen by the practitioner; we used $k = 1$ in our experiments), we sample a batch from this dataset $\{(\mathbf{o}^{(\beta)}, \mathbf{a}^{*,(\beta)})\}_{\beta=1}^{B}$, and then distribute the data batch $\{(o_i^{(\beta)}, a_i^{*,(\beta)})\}_{\beta=1}^{B}$ to agent i, for supervised learning; we note the training can be done sequentially or parallel. Having a shared dataset of trajectories in this way allows us to view $(\pi_1, \ldots, \pi_M)$ as a single neural-network with disconnected components, and thus the error bounds from [113] directly apply, as discussed in Section 2.5. See Figure 2.1 for a diagram. Pseudo-code for our method is contained in Appendix 2.8.6. (In Appendix 2.8.2 we test whether giving each agent its own dataset would make a difference, and it did not seem so).

The aforementioned procedure is sufficient when the agents do not need to communicate, but when communication is involved we have to modify the above method.

2.4.4 CESMA for multi-agents with communication

The main insight for training an agent's communication action is that we can view a broadcasting agent and the receiving agent as one neural network connected via the communication nodes; then in this way we we can backpropagate the action loss of the receiving agent through to the broadcasting agent's weights.

In more detail, due to communication, the multi-agents now have two types of observations and actions.

We denote the physical actions (i.e. non-communication actions) as $\mathbf{a} = (a_1, \ldots, a_M)$ and the communication actions/**b**roadcasts as $\mathbf{b} = (b_1, \ldots, b_M)$. *For simplicity, let us assume that all agents can communicate with each other and each agent broadcasts the same thing to all other agents.* So we denote $c_i = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_M)$ as agent i's observation of the broadcast by other agents, and where b_j is agent j's broadcast to all other agents.

So for each agent i, we have $\pi_i(o_i, c_i) = (a_i, b_i)$. And we also denote $\pi_i(o_i, c_i)_{\text{action}} = a_i$, and $\pi_i(o_i, c_i)_{\text{comm}} = b_i$.

For training, as before we have a shared dataset of observations $\mathcal{D}$. But as the agents step through the environment, at each timestep we now store $((\mathbf{o}, \mathbf{c}), \hat{\mathbf{o}}, \hat{\mathbf{a}}^*)$, where $(\mathbf{o}, \mathbf{c})$ is the joint physical and communication observation of the previous timestep, $\hat{\mathbf{o}}$ is the physical observation at the current timestep, and $\hat{\mathbf{a}}^* = \pi^*(\hat{\mathbf{o}}) = (\hat{a}_1^*, \ldots, \hat{a}_M^*)$ is the expert action label; these are the necessary ingredients for training.

Then to train, we first obtain a sample from $\mathcal{D}$ (practically we perform batched training, but for simplicity we consider one sample), say $((\mathbf{o}, \mathbf{c}), \hat{\mathbf{o}}, \hat{\mathbf{a}}^*)$, and then we take the policies at the most-recent update $\pi_1^{\text{current}}, \ldots, \pi_M^{\text{current}}$ and form their broadcasts $b'_k = \pi_k^{\text{current}}(o_k, c_k)_{\text{comm}}$ for $k = 1, \ldots, M$. Then in principle, we want to minimize the loss function,

$$\min_{(\pi_1, \ldots, \pi_M)} \sum_{j=1}^{M} \ell(\hat{a}_j^*,\ \pi_j(\hat{o}_j, \hat{c}_j)_{\text{action}}),$$

where

$$\hat{c}_j = (b'_1, \ldots, b'_{j-1}, b'_{j+1}, \ldots, b'_M), \quad j = 1, \ldots, M.$$

In practice, we train each agent i separately by minimizing their *communication* loss and *action* loss which we describe below.

In order to train agent i's communication action, we make the insight that we can back-propagate the supervised learning loss of *other* agents through the communication nodes to agent i's parameters, precisely because the communication *output* of agent i becomes an observational *input* for the *other* agents.

Then to train the communication action of agent i, we sample $((\mathbf{o}, \mathbf{c}), \hat{\mathbf{o}}, \hat{\mathbf{a}}^*)$ from $\mathcal{D}$, and seek to minimize the communication loss function,

$$\min_{\pi_i} \sum_{j \neq i} \ell(\hat{a}_j^*,\ \pi_j(\hat{o}_j, \hat{c}'_j)_{\text{action}}), \qquad \text{(comm. loss for agent } i\text{)}$$

where

$$\hat{c}'_j = (b'_1, \ldots, \pi_i(o_i, c_i)_{\text{comm}}, \ldots, b'_{j-1}, b'_{j+1}, \ldots, b'_M),$$

where we assumed without loss of generality that $i < j$. And so because $\hat{c}'_j$ *depends on* π_i, then we can backpropagate agent j's supervised loss to agent i's parameters.

To train the physical action of agent i, we sample $((\mathbf{o}, \mathbf{c}), \hat{\mathbf{o}}, \hat{\mathbf{a}}^*)$ from $\mathcal{D}$ and want to minimize

$$\min_{\pi_i} \ell(\hat{a}_i^*,\ \pi_i(\hat{o}_i, \hat{c}'_i)_{\text{action}}), \qquad \text{(action loss for agent } i\text{)}$$

where $\hat{c}'_i = (b'_1, \ldots, b'_{i-1}, b'_{i+1}, \ldots, b'_M)$.

For a graphic overview, we give a diagram in Figure 2.2 for the backpropagation of the communication loss and the action loss, and provide pseudocode in Algorithm 11 in Appendix 2.8.6. In some sense, our method can be viewed as a hybrid of experience replay and supervised learning.

In this way, we have alleviated a bit the issue of sparse rewards for communication [47, Section 4]. Indeed, communication actions suffer from sparse rewards as a reward is only bestowed on the broadcasting agent when all the following align: it sends the right message, the receiving agent understands the message, and then acts accordingly. In our method with an expert supervisor, the correct action by the acting agent is clear.

2.5 Theoretical analysis

2.5.1 No-regret analysis, and guarantees

Although the proposed framework could handle a myriad of imitation learning algorithms, such as Forward Training [112], SMILe [112], SEARN [30], and more, we use DAgger in our

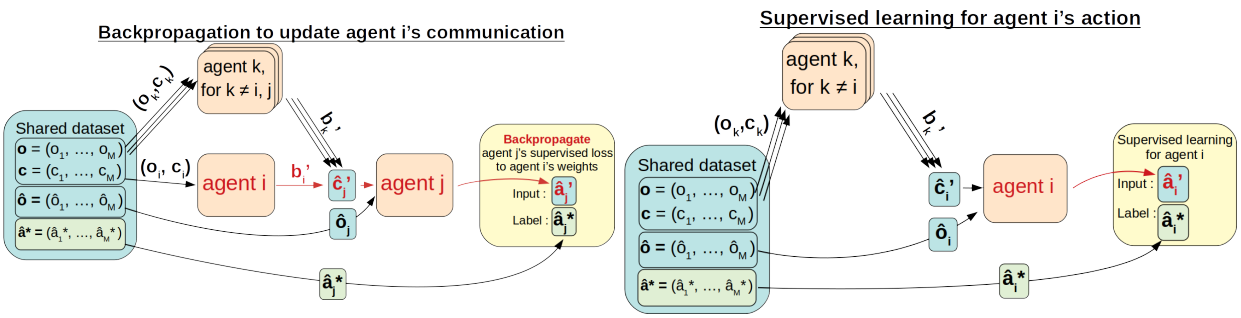


Figure 2.2: Decentralizing multi-agents that communicate. The top diagram shows how we update agent i's communication action by backpropagating the supervised loss of *other* agents. The red portions highlight the trail of backpropagation. The bottom diagram shows how we update the action of agent i.

experiments, and thus we follow its theoretical analysis, while providing multi-agent extensions. And since our method can be viewed as using a single-agent learner with disconnected components, we can directly apply the no-regret analysis from [113] to obtain theoretical insights.

Notationally, **(i)** we let ℓ be a surrogate loss of matching the expert policy π^* (e.g. the expected 0-1 loss at each state) and denote $r = r(s, a)$ the instantaneous reward which we assume to be bounded in $[0, 1]$, **(ii)** $(\pi_1^{(N)}, \ldots, \pi_M^{(N)})$ are the multi-agents after N updates of the policy using any supervised learning algorithm, and where each update is done after a T-step trajectory with T the task horizon, **(iii)** $d_{(\pi_1^{(N)},\ldots,\pi_M^{(N)})}$ is the average distribution of observations that come from following the multi-agent policy $(\pi_1^{(N)}, \ldots, \pi_M^{(N)})$ from a given initial distribution, **(iv)** $R(\pi_1^{(N)}, \ldots, \pi_M^{(N)})$ is the cumulative reward after an episode of the task, **(v)** and $U_t^{\pi'}(s, \pi)$ is the reward after t steps of executing π in only initial state s, and then following policy π' thereafter.

Then viewing the multi-agent policy as a joint single-agent policy we obtain the following *guarantee on the reward based on how well the multi-agents match the expert*, which is a direct rephrasing of [113, Theorem 3.2]:

Theorem 1. *If the number of policy updates N is $O(T \log^k(T))$ for sufficiently large $k \geqslant 0$,*

then there exists a joint multi-agent policy $(\hat{\pi}_1, \ldots, \hat{\pi}_M) \in \{(\pi_1^{(i)}, \ldots, \pi_M^{(i)})\}_{i=1}^N$ *such that*

$$R(\hat{\pi}_1, \ldots, \hat{\pi}_M) \geqslant R(\pi^*) - uT\epsilon_N - O(1),$$

where $u \geqslant 0$ *is such that* $U^{\pi^*}_{T-t+1}(s, \pi^*) - U^{\pi^*}_{T-t+1}(s, a) \leqslant u$ *for all actions* a *and* $t \in \{1, \ldots, T\}$, *and*

$$\epsilon_N = \min_{(\pi_1, \ldots, \pi_M)} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{o} \sim d_{(\pi_1^{(i)}, \ldots, \pi_M^{(i)})}} [\ell(\boldsymbol{o}, (\pi_1, \ldots, \pi_M)].$$

Here ϵ_N is best described as the true loss of the best learned policy in hindsight. The condition $U^{\pi^*}_{T-t+1}(s, \pi^*) - U^{\pi^*}_{T-t+1}(s, a) \leqslant u$ can best be described as saying the reward lost from not following the expert at initial state s, but following it after, is at most u.

2.5.2 The partial observability problem of decentralization, and its cost

In our setting of multi-agents, the centralized expert and the decentralized multi-agents have different structures of their policies, i.e. they are solving the problem in different policy spaces. The centralized expert observes the joint observations of all agents, and thus it is a function $\pi^* : \mathcal{O}_1 \times \cdots \times \mathcal{O}_M \to \mathcal{A}_1 \times \cdots \times \mathcal{A}_M$, and we can decompose π^* into

$$\pi^*(\mathbf{o}) = (\pi_1^*(\mathbf{o}), \ldots, \pi_M^*(\mathbf{o})),$$

where $\pi_i^* : \mathcal{O}_1 \times \cdots \times \mathcal{O}_M \to \mathcal{A}_i$. The goal of decentralization is to find multi-agent policies $\pi_1, \ldots, \pi_M$ such that

$$\pi^*(\mathbf{o}) = (\pi_1^*(\mathbf{o}), \ldots, \pi_M^*(\mathbf{o})) \overset{\text{want}}{=} (\pi_i(o_1), \ldots, \pi_M(o_M)).$$

Note that π_i^* is able to observe the joint observations while π_i is only able to observe its own local observation o_i. But from this constraint, this means we may encounter issues where

$$\pi_i^*(o_1, \ldots, o_{i-1}, o_i, o_{i+1}, \ldots, o_M) = a_i,$$
$$\text{but} \quad \pi_i^*(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, o_i, \tilde{o}_{i+1}, \ldots, \tilde{o}_M) = \tilde{a}_i,$$

so we want $\pi_i(o_i) = a_i$ or $\tilde{a}_i$, or even something else. Thus the multi-agent policy can act sub-optimally in certain situations, being unaware of the global state. This unfortunate

situation not only afflicts our algorithm, but any multi-agent training algorithm (and in general, any algorithm attempting to solve a POMDP). We call this the *partial observability problem of decentralization* (note partial observability affects any algorithm trying to solve a POMDP, but here we examine from the viewpoint of decentralization). More concretely, we can say there is a partial observability problem of decentralization if there exists observations $(o_1, \ldots, o_{i-1}, o_i, o_{i+1}, \ldots, o_M)$, and $(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, o_i, \tilde{o}_{i+1}, \ldots, \tilde{o}_M)$ such that

$$\begin{aligned} &\pi^*(o_1, \ldots, o_{i-1}, o_i, o_{i+1}, \ldots, o_M) \\ &\qquad \neq \pi^*(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, o_i, \tilde{o}_{i+1}, \ldots, \tilde{o}_M) \end{aligned}$$

Relating this to the no-regret analysis in Theorem 1, the partial observability problem means that under certain environments it may be impossible for the multi-agents to match the expert exactly; this manifests in a cost C_p where,

$$\begin{aligned} \epsilon_N &= \min_{(\pi_1, \ldots, \pi_M)} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\mathbf{o} \sim d_{(\pi_1^{(i)}, \ldots, \pi_M^{(i)})}} [\ell(\mathbf{o}, (\pi_1, \ldots, \pi_M)] \\ &\geqslant C_p, \quad \text{for all } N \geqslant 1, \end{aligned}$$

which implies from Theorem 1 that the best guarantee of the reward for the multi-agents is $R(\hat{\pi}_1, \ldots, \hat{\pi}_M) \geqslant R(\pi^*) - TC_p - O(1)$.

The main takeaway: In the original DAgger setting (i.e. the single-agent MDP setting), under reasonable assumptions on the distribution of states [see 113, Section 4.2], as $N \to \infty$ the cumulative reward of the learner can approximate the cumulative reward of the expert arbitrarily closely. Here when analyzing the multi-agent setting, we find that because $\epsilon_N \geqslant C_p$, then the no-regret analysis guarantees that after $O(T \log^k(T))$ updates we will find a multi-agent policy that obtains a cumulative reward that is within a cost of partial observability term of the expert. In relation to this, in Appendix 2.8.5, we perform experiments and analyse the supervised learning loss versus the reward obtained by the multi-agents.

2.5.3 The need for communication

Decentralization without communication is most effective when all multi-agents can observe the full joint observation. Then from the perspective of each agent the only non-stationarity is from other agents' policies (which is alleviated by decentralization).

But when each agent only has local observations, then to avoid the partial observability problem of decentralization, there is an incentive to communicate. Namely, we want for the multi-agent policy $(\pi_1, \ldots, \pi_M)$

$$\begin{aligned}\pi^*(\mathbf{o}) &= (\pi_1^*(\mathbf{o}), \ldots, \pi_M^*(\mathbf{o})) \\ &\overset{\text{want}}{=} (\pi_i(o_1, c_1), \ldots, \pi_M(o_M, c_M)),\end{aligned}$$

where c_i is the communication from either all or only some of the other agents, to agent i. Namely we view c_i as a function $c_i : \mathcal{O}_1 \times \cdots \times \mathcal{O}_{i-1} \times \mathcal{O}_{i+1} \times \cdots \times \mathcal{O}_M \to \mathcal{C}_i$ (where $\mathcal{C}_i$ is some communication action space). Then we have the following requirement for the communication protocol $\{c_i\}_{i=1}$ in order to fix the partial observability problem of decentralization,

Theorem 2. *If the multi-agent communication* $c_i : \mathcal{O}_1 \times \cdots \times \mathcal{O}_{i-1} \times \mathcal{O}_{i+1} \times \cdots \times \mathcal{O}_M \to \mathcal{C}_i$ *satisfies the condition*

$$\begin{aligned}&\pi^*(o_1, \ldots, o_{i-1}, o_i, o_{i+1}, \ldots, o_M) \\ &\quad \neq \pi^*(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, o_i, \tilde{o}_{i+1}, \ldots, \tilde{o}_M),\end{aligned}$$

implies that

$$\begin{aligned}&c_i(o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_M) \\ &\quad \neq c_i(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, \tilde{o}_{i+1}, \ldots, \tilde{o}_M)\end{aligned}$$

for all $i = 1, \ldots, M$*, then there is no partial observability cost of decentralization when the multi-agents use* $\{c_i\}_{i=1}^M$ *as their communication protocol, i.e. the multi-agents can match the expert perfectly on all observations.*

The theorem above says that a sufficient condition for the communication protocol $\{c_i\}_{i=1}$ is that from the perspective of, say, agent j, then c_j is able to provide information to agent

j about when the expert decides to output different actions for different global observations, even if the global observations share o_j as a local observation.

Paired with Theorem 1, this implies that under the correct communication protocal, the multi-agents can approximate the expert arbitrarily closely (and that we need $O(T \log^k(T))$ updates). Of course, in our experiments we learn this communication protocol.

2.6 Experiments

Our experiments are conducted in the Multi-Agent Particle Environment [94, 87] provided by OpenAI, which has basic simulated physics (e.g. Newton's law) and multiple multi-agent scenarios.

In order to conduct comparisons to MADDPG, we also use the DDPG algorithm with the Gumbel-Softmax [70, 88] action selection for discrete environments, as they do. For the single-agent centralized expert neural network, we always make sure the number of parameters/weights matches (or is lower) than that of MADDPG's. For the decentralized agents, we use the same number of parameters as the decentralized agents in MADDPG (i.e. the actor part). We always use the discount factor $\gamma = 0.9$, as that seemed to work best both for our centralized expert, and also MADDPG. Following their experimental procedure, we average our experiments over three runs, and plot the minimum and maximum reward envelopes. And for the decentralization, we trained three separate centralized experts, and used each of them to obtain three decentralized policies. Full details of our hyperparameters is in the appendix. And we always use two-hidden layer neural networks. Brief descriptions of each environment are provided, and fuller descriptions and some example pictures are placed in the appendix.

2.6.1 Cooperative Navigation

Here we examine the situation of N agents occupying N landmarks in a 2D plane, and the agents are either homogeneous or nonhomogenous. The (continuous) observations of each

agent are the relative positions of other agents, the relative positions of each landmark, and its own velocity. The agents do not have access to others' velocities so we have partial observability. The reward is based on how close each landmark has an agent near it, and the actions of each agent are discrete: up, down, left, right, and do nothing.

In Figure 2.3, we see that CESMA, when combining the number of samples in training the expert as well as decentralization, is able to achieve the same reward as MADDPG while utilizing fewer samples, i.e. CESMA is more sample efficient (the dashed red line is just a visual aid that extrapolates the reward for the decentralized curves, because we stop training once the reward sufficiently matches MADDPG). In Figure 2.4, we also noticed that the centralized expert is able to find a policy that achieves a higher reward than a *converged* MADDPG; and we were able to decentralize this expert to obtain decentralized multi-agent policies that achieved higher rewards than MADDPG. We provide further experiments in the appendix that tell the same story.

2.6.2 Cooperative Navigation with Communication

Here we adapt CESMA to a task that involves communication. In this scenario, the communication action taken by each agent at time step $t-1$ will appear as an observation to other agents at time step t. Although we require continuous communication to backprop, in practice we can use the softmax operator to provide the bridge between the discrete and continuous. And during decentralized execution, our agents are able to act with discrete communication inputs.

We examine two scenarios for CESMA that involve communication, and use the training scenario described in section 2.4.4. The first scenario called the "speaker and listener" environment has a speaker who broadcasts the correct goal landmark (in a "language" it must learn) out of a possible 3 choices, and the listener, who is blind to the correct goal landmark, must use this information to move there. Communication is a necessity in this environment. The second scenario is cooperative navigation with communication and here we have three agents whose observation space includes the goal landmark of the *other* agent(s), and not

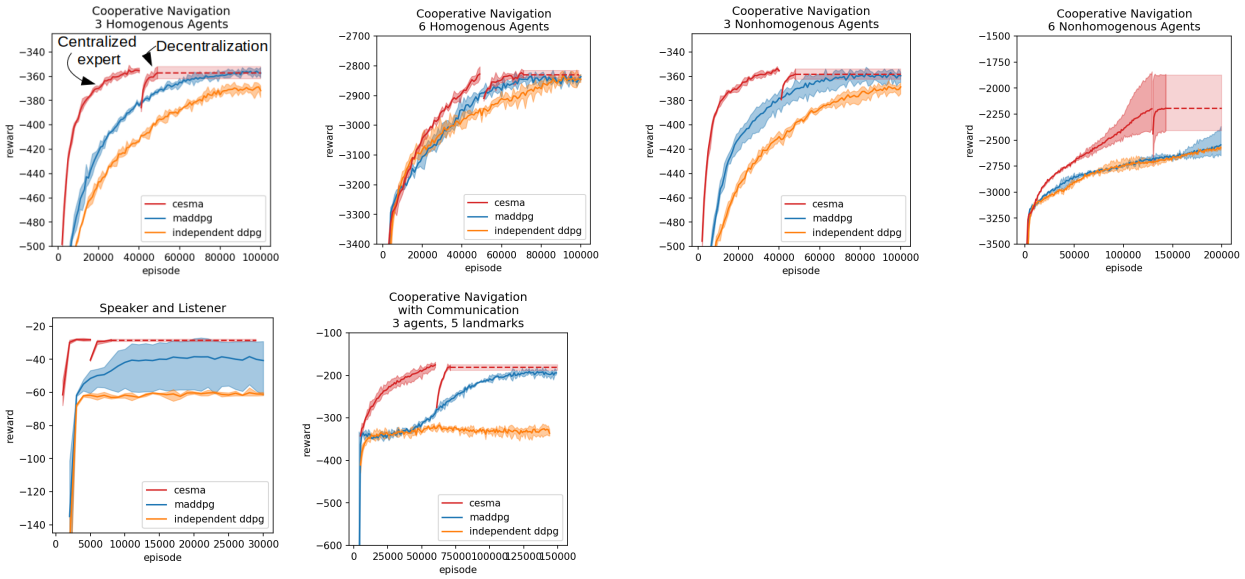


Figure 2.3: Reward curves for various multi-agent environments. We train the centralized expert until its reward matches or betters MADDPG's reward. Then we decentralize this expert until we achieve the same reward as the expert. The dashed red line is a visual aid which extrapolates the reward for the decentralized curves, because we stop training the multi-agents once the reward sufficiently matches the expert. We observe that CESMA is more sample-efficient than MADDPG.

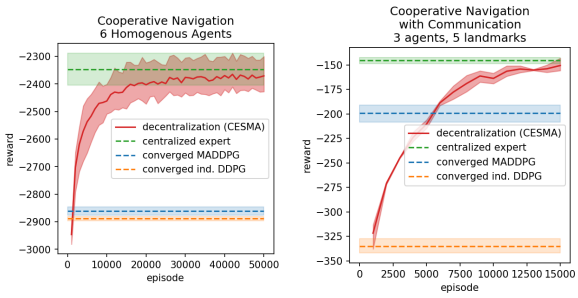


Figure 2.4: Reward curves for decentralization of a centralized expert policy that obtains a better reward than a *converged* MADDPG and independent DDPG. The horizontal dashed lines represent final rewards after convergence of the algorithms (i.e. no visible improvement of the reward after many episodes), and the red solid line represents decentralization of the centralized expert. This demonstrates that we are able to successfully decentralize expert policies that achieve better rewards than a *converged* MADDPG and independent DDPG. In other words, *CESMA is able to find better optimum that MADDPG and independent DDPG were not able to find.*

their own, and there are five possible goal landmarks.

We see in Figure 2.3 that we achieve a higher reward in a more sample efficient manner. For the speaker and listener environment, the centralized expert near-immediately converges, and same for the decentralization. And MADDPG has a much higher variance in its convergence. We also see in Figure 2.4 that the centralized expert was again able to find a policy that achieved a higher reward than a *converged* MADDPG, and we were able to successfully decentralize this to obtain a decentralized multi-agent policy achieving the same superior reward as the expert. We provide further experiments in the appendix that tell the same story.

2.7 Conclusion

We propose a MARL algorithm, called Centralized Expert Supervises Multiagents (CESMA), which takes the training paradigm of centralized training, but decentralized execution. The algorithm first trains a centralized expert policy, and then adapts DAgger to obtain decen-

tralized policies that execute in a decentralized fashion. We also formulated an approach that enables multi-agents to learn a communication protocol. Experiments in a variety of tasks show that CESMA can train successful decentralized multi-agent polices at a low sample complexity. Notably, the decentralization protocol often is able to achieve the same levels of cumulative reward as a centralized controller, which in our experiments often achieves higher rewards than the competing methods MADDPG and independent DDPG.

2.8 Appendix: More Experiments, Explanation of Environments and Hyperparameters, and Proofs of Theorems

2.8.1 Decentralizing expert policies that obtain higher rewards than MADDPG

2.8.1.1 Cooperative Navigation

For the cooperative navigation experiment, in Figure 2.5, we see in all cases the centralized expert is able to achieve a lower reward than MADDPG and DDPG. And futhermore we were able to decentralize the expert policy (which was chosen to be the one with highest reward) so as to reach this same superior reward. And we remark that our method seems to work better with more agents.

The six nonhomogeneous agents case works as a good experiment to see what happens when we stop the centralized expert before it truly converges. In this case, decentralization to achieve the same reward as the expert is quickest and occurs within the first 5,000 episodes. Intuitively, it makes sense that a suboptimal expert solution is faster to decentralize.

2.8.1.2 Cooperative Navigation with Communication

In the environments with communication, we see in figure 2.5 that both the centralized expert and the decentralized agents achieve a higher reward and in a more sample efficient manner. For the speaker and listener environment, the centralized expert near-immediately converges, and same for the decentralization process. And MADDPG has a much higher

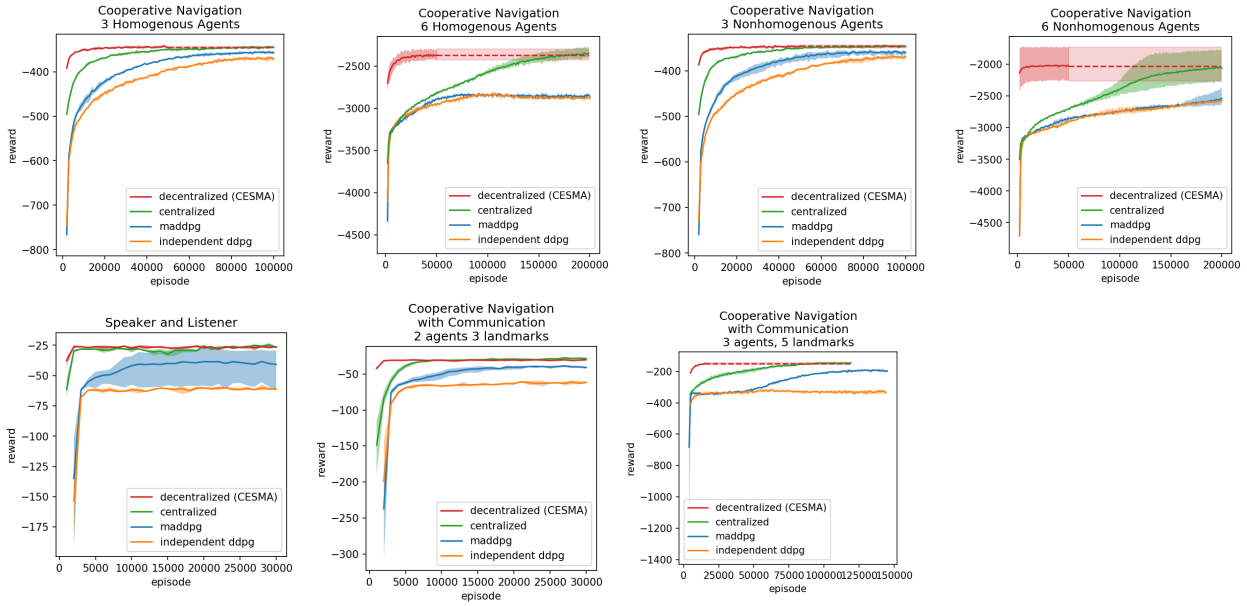
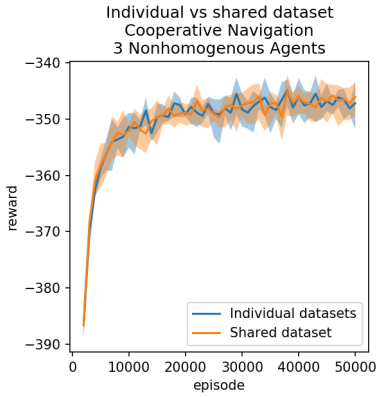


Figure 2.5: Reward curves for the various multi-agent environments. In these experiments, we test whether we can perform decentralization of centralized expert policies that achieve a superior reward to a converged MADDPG. Namely, we test whether we can decentralize to obtain the same superior reward as the centralized expert. The plots above show that we can and do in every experiment. The dashed red lines for the decentralized curves represent when we stop the decentralization procedure, as the reward sufficiently matches the expert. The envelopes of the learning curves denote the maximum and minimum. We in particular note that in some experiments, CESMA achieves a superior reward compared to a *converged* MADDPG.

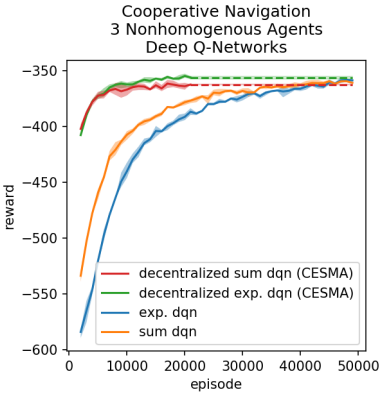
variance in its convergence. In the cooperative navigation with communication scenarios, the story is similar, that the centralized expert quickly converges, and the decentralization process is near immediate.

2.8.2 Experiment where each agent has its own dataset of trajectories



Here we describe an experiment where each agent has its own individual dataset of trajectories, versus a shared dataset. Namely, we plot the learning curves for decentralizing a policy in the two cases: (1) When each agent has its own dataset of trajectories, or (2) when there is a shared dataset of trajectories (which is the one we use in the experiments). We tested on the cooperative navigation environment with 3 nonhomogeneous agents. We hypothesized that the nonhomogeneity of the agents would have an effect on the shared reward, but this turned out not to be so. But it is interesting to note that in the main text, we found that the some agents had a bigger loss when doing supervised learning from the expert.

2.8.3 Experiment with DQNs



Here we examined decentralizing DQNs. We used the cross entropy loss for the supervised learning portion, and used the cooperative navigation environment with 3 nonhomogenous agents. The DQNs we used are: the exponential actions DQN, which is just a naive implementation of DQNs for the multi-agents, and a Centralized VDN where the system Q value is the sum of the individual agent Q values. We used a neural network with 200 hidden units, batch size 64, and for the exponential DQN, we used a learning rate and τ of 5×10^{-4}, and for the QMIX/Centralized VDN DQN we used a learning rate and τ of 10^{-3}. We also used a noisy action selection for exploration. We stopped training of the decentralization once the mulit-agents reached the same reward as the expert; the dashed lines are a visual-aid that extrapolates the reward.

2.8.4 One-at-a-Time-EXpert

2.8.5 Reward vs. loss, and slow and fast learners

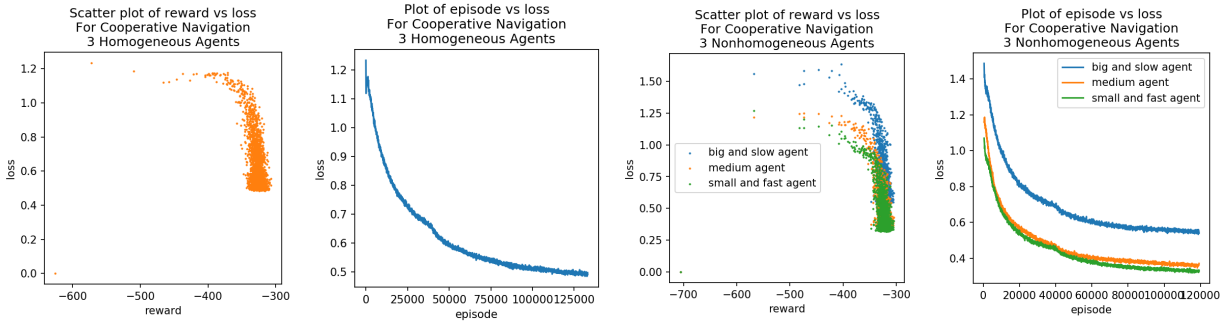


Figure 2.6: Reward vs. loss, and loss vs. episode.

In our experiments with cooperative navigation, when using the cross entropy loss, we did not find an illuminating correlation between the reward and the loss. We reran the experiments in a truer DDPG fashion by solving a continuous version of the environment, and used the mean-squared error for the supervised learning. We examined the loss in the cooperative navigation task with 3 agents, both homogeneous and nonhomogeneous agents. We plot the figures in Figure 2.6. We found that in these cases, the reward and loss were negatively correlated as expected, namely that we achieved a higher reward as the loss decreased. In

the nonhomogeneous case, we plot each individual agents' reward vs its loss and found that the big and slow agent had the biggest loss, followed by the medium agent, and the small and fast agent being the quickest learner. This example demonstrates that in nonhomogeneous settings, some agents may be slower to imitate the expert than others.

We also observe that there is a decrease in marginal reward vs loss – that is, at a certain point, one needs to obtain a much lower loss for a diminishing gains in reward.

2.8.6 Pseudo-algorithm of CESMA (without communication)

Algorithm 10 CESMA: Centralized Expert Supervises Multi-Agents

Require: A centralized policy π^* that sufficiently solves the environment.

Require: M agents $\pi_1, \dots \pi_M$, observation buffer $\mathcal{D}$ for multi-agent observations, batch size B

while $\pi_{\theta_1}, \dots, \pi_{\theta_M}$ not converged **do**
  Obtain observations $o_1, \dots, o_M$ from the environment
  Obtain agents' actions, $a_1 = \pi_1(o_1), \dots, a_M = \pi_M(o_M)$
  Obtain expert action labels $a_i^* = \pi^*(o_1, \dots, o_M)_i$, for $i = 1, \dots, M$
  Store the joint observation with expert action labels $((o_1, a_1^*), \dots, (o_M, a_M^*)$ in $\mathcal{D}$
  if $|\mathcal{D}|$ sufficiently large **then**
    Sample a batch of B multi-agent observations $\{((o_1^{(\beta)}, a_i^{*(\beta)}), \dots, (o_M^b, a_M^{*(\beta)}))\}_{\beta=1}^B$
    Perform supervised learning for π_i where the observation-label pairs $\{(o_i^{(\beta)}, a_i^{*(\beta)})\}_{\beta=1}^B$.

2.8.7 Pseudo-code of CESMA with communicating agents

We give a pseudocode in algorithm 11.

A diagram of the action loss and communication loss is given in figure 2.7 (action loss) and 2.8 (communication loss).

We remark that we also considered the case of a hybrid objective, where the actions are

learned by supervised learning from the expert, and the communication is learned similar to a standard RL algorithm (e.g. the Q-values are communication actions). Preliminary results showed this did not work well.

We review notation from the main text that appears in the pseudo-code of Algorithm 11: We denote the physical actions (i.e. non-communication actions) as $\mathbf{a} = (a_1, \ldots, a_M)$ and the communication actions as $\mathbf{b} = (b_1, \ldots, b_M)$. For simplicity, let us assume that all agents can communicate with each other, so we have $c_i = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_M)$.

So for each agent i, then we have $\pi_i(o_i, c_i) = (a_i, b_i)$. And we also denote $\pi_i(o_i, c_i)_{\text{action}} = a_i$, and $\pi_i(o_i, c_i)_{\text{comm.}} = b_i$. And we denote the communication action from agent i to agent j as $b_{i,j}$.

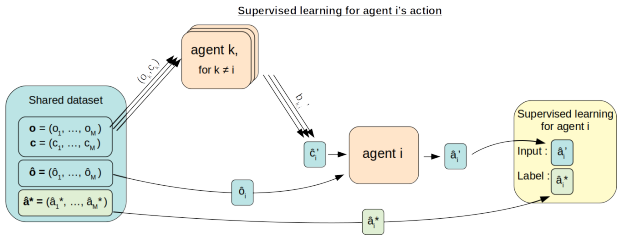


Figure 2.7: A diagram of the computation of the action loss for agent i. This diagram is a bigger version of the one found in the main text.

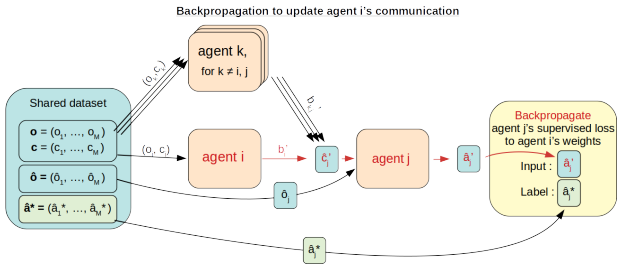


Figure 2.8: A diagram of the computation of the communication loss for agent i, derived from the supervised learning action loss of agent j. This diagram is a bigger version of the one found in the main text.

2.8.8 Environments used in the experiments

2.8.8.1 Cooperative navigation

The goal of this scenario is to have N agents occupy N landmarks in a 2D plane, and the agents are either homogeneous or heterogeneous. The environment consists of:

- Observations: The (continuous) observations of each agent are the relative positions of other agents, the relative positions of each landmark, and its own velocity. Agents do not have access to other's velocities, and thus each agent only *partially observes* the environment (aside from not knowing other agents' policies).

- Reward: At each timestep, if A_i is the ith agent, and L_j the jth landmark, then the reward r_t at time t is,

$$r_t = -\sum_{j=1}^{N} \min \{\|A_i - L_j\| : i = 1, \ldots, N\}$$

  This is a sum over each landmark of the minimum agent distance to the landmark. Agents also receive a reward of -1 at each timestep that there is a collision.

- Actions: Each agents' actions are discrete and consist of: up, down, left, right, and do nothing. These actions are acceleration vectors (except do nothing), which the environment will take and simulate the agents' movements using basic physics (i.e. Newton's law).

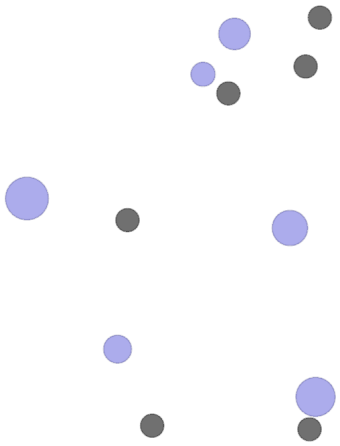


Figure 2.9: Example of cooperative navigation environment with 6 nonhomogeneous agents. The agents (blue) must decide how best to cover each landmark (grey).

2.8.8.2 Speaker listener

In this scenario, the goal is for the listener agent to reach a goal landmark, but it does not know which is the goal landmark. Thus it is reliant on the speaker agent to provide the correct goal landmark. The observation of the speaker is just the color of the goal landmark, while the observation of the listener is the relative positions of the landmark. The reward is the distance from the landmark.

- Observations: The observation of the speaker is the goal landmark. The observation of the listener is the communication from the speaker, as well as the relative positions of each goal landmark.
- Reward: The reward is merely the negative (squared) distance from the listener to the goal landmark.
- Actions: The actions of the speaker is just a communication, a 3-dimensional vector. The actions of the listener are the five actions: up, down, left, right, and do nothing.

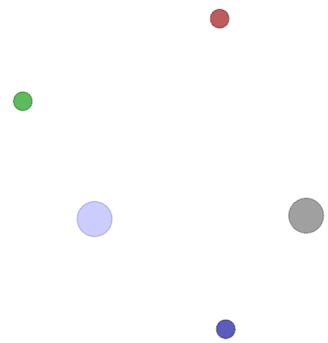


Figure 2.10: Example of the speaker and listener environment. The speaker (grey) must communicate to the agent which colored landmark to go towards (blue in this case).

2.8.8.3 Cooperative navigation with communication

In this particular scenario, we have one version with 2 agents and 3 landmarks, and another version with 3 agents and 5 landmarks. Each agent has a goal landmark that is only known by the other agents. Thus the each agent must communicate to the other agents its goal. The environment consists of:

- Observations: The observations of each agent consist of the agent's personal velocity, the relative position of each landmark, the goal landmark for the other agent (an 3-dimensional RGB color value), and a communication observation from the other agent.
- Reward: At each timestep, the reward is the sum of the distances between and agent and its goal landmark.
- Actions: This time, agents have a movement action and a communication action. The movement action consists of either not doing anything, or outputting an acceleration vector of magnitude one in the direction of up, down, left, or right; so do nothing, up, down, left right. The communication action is a one-hot vector; here we choose the communication action to be a 10-dimensional vector.

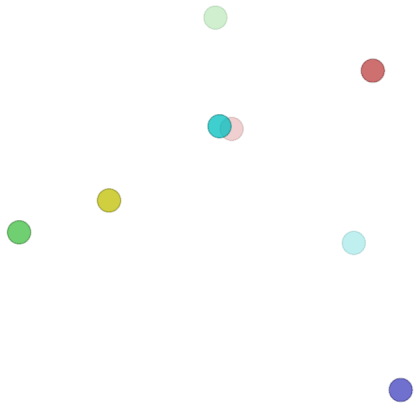


Figure 2.11: Example of cooperative navigation environment with communication. We have 3 agents and 5 landmarks. The lightly colored circles are agents and they must go towards their same-colored landmark.

2.8.9 Hyperparameters

- For all environments, we chose the discount factor γ to be 0.9 for all experiments, as that seemed to benefit both the centralized expert as well as MADDPG (and as well as independently trained DDPG). And we always used a two-hidden-layer neural network for all of MADDPG's actors and critics, as well as the centralized expert, and the decentralized agents. The training of MADDPG used the hyperparameters from the MADDPG paper [87], which we found to be quite optimal with the exception of having $\gamma = 0.9$ (instead of 0.95), as that improved MADDPG's performance. In the graphs, the reward is averaged every 1,000 episodes.

- For the cooperative navigation environments with 3 agents, for both homogeneous and nonhomogeneous: Our centralized expert neural network was a two-hidden-layer neural network with 225 units each (as that matched the number of parameters for MADDPG when choosing 128 as their number of hidden units for each of their 3 agents), and we used a batch size of 64. The learning rate was 0.001, and $\tau = 0.001$. We also clipped the gradient norms to 0.1. When decentralizing, each agent was a two-hidden-layer neural network with 128 units (as in MADDPG), where we trained with a batch size

of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 128 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.

- For the cooperative navigation with 6 agents, for both homogeneous and nonhomogeneous: Our centralized expert neural network was a two-hidden-layer neural network with 240 units each (as that matched the number of parameters for MADDPG when choosing 128 as their number of hidden units for each of their 3 agents' actor and critic), and we used a batch size of 32. The learning rate was 0.0001, and $\tau = 0.0001$. We also clipped the gradient norms to 0.1. When decentralizing, each agent was a two-hidden-layer neural network with 128 units (as in MADDPG), where we trained with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 128 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.

- For the speaker and listener environment: Our centralized expert neural network was a two-hidden-layer neural network with 64 units each (which gave a lower number of parameters than MADDPG when choosing 64 as their number of hidden units for each of their 2 agents' actor and critic), and we used a batch size of 32. The learning rate was 0.0001, and $\tau = 0.001$. When decentralizing, each agent was a two-hidden-layer neural network with 64 units (as in MADDPG), where we trained with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 64 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.

- For the cooperative navigation with communication environment: Our centralized expert neural network was a two-hidden-layer neural network with 95 units each (which matched the number of parameters as MADDPG when choosing 64 as their number of hidden units for each of their 2 agents' actor and critic), and we used a batch size of 32. The learning rate was 0.0001, and $\tau = 0.0001$. When decentralizing, each agent was a two-hidden-layer neural network with 64 units (as in MADDPG), where we trained

with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 64 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.

- We also run all the environments for 25 time steps.

2.8.10 Proofs of theorems

We prove Theorem 2 from the main text:

Theorem 2. *If the multi-agent communication* $c_i : \mathcal{O}_1 \times \cdots \times \mathcal{O}_{i-1} \times \mathcal{O}_{i+1} \times \cdots \times \mathcal{O}_M \to \mathcal{C}_i$ *satisfies the condition*

$$\pi^*(o_1, \ldots, o_{i-1}, o_i, o_{i+1}, \ldots, o_M) \neq \pi^*(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, o_i, \tilde{o}_{i+1}, \ldots, \tilde{o}_M)$$
$$\textit{implies that } c_i(o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_M) \neq c_i(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, \tilde{o}_{i+1}, \ldots, \tilde{o}_M)$$

for all $i = 1, \ldots, M$*, then there is no partial observability cost of decentralization when the multi-agents use* $\{c_i\}_{i=1}^M$ *as their communication protocol, i.e. the multi-agents can match the expert perfectly on all observations.*

Proof. By assumption, for an agent j with observations $\mathbf{o} = (o_1, \ldots, o_{j-1}, o_j, o_{j+1}, \ldots, o_M)$ and $\tilde{\mathbf{o}} = (\tilde{o}_1, \ldots, \tilde{o}_{j-1}, o_j \tilde{o}_{j+1}, \ldots, \tilde{o}_M)$ such that

$$\pi^*(\mathbf{o})_j = a_j, \quad \pi^*(\tilde{\mathbf{o}})_j = \tilde{a}_j, \quad \text{but} \quad a_j \neq \tilde{a}_j.$$

then denoting $\mathbf{o}_{-j}$ as the observation without o_j and similarly for $\tilde{\mathbf{o}}_{-j}$, then our assumption implies $c_j(\mathbf{o}_{-j}) \neq c_j(\tilde{\mathbf{o}}_{-j})$. Then clearly we can construct a policy where $\pi_j(o_j, c_j(\mathbf{o}_j)) \neq \pi_j(o_j, c_j(\tilde{\mathbf{o}}_j))$, because the inputs to π_j are different.

And so the multi-agents, using the communication protocol of $\{c_i\}_{i=1}^M$, can detect when an expert decides to change its action based on differences in the *global* observation (i.e. $\mathbf{o}$ and $\tilde{\mathbf{o}}$) even when the *local* observation (i.e. o_j) stays the same.

□

Algorithm 11 CESMA: Centralized Expert Supervises Multi-Agents (Communicating Agents)

Require: A centralized policy π^* that sufficiently solves the environment.

Require: M initial agents $\pi_1, \dots \pi_M$, observation buffer $\mathcal{D}$ for multi-agent observations, batch size B

Require: ℓ, the supervised learning loss

while $\pi_1, \dots, \pi_M$ not converged **do**

  Obtain the observations and communications $\{(o_i, c_i)\}_{i=1}^M$ from the environment.

  With these observations, obtain actions and step through the environment, to get new observations $\{\hat{o}_i\}_{i=1}^M$.

  Store the physical and communication observations together along with the expert label $(((o_1, c_1), \hat{o}_1, \hat{a}_1^*), \dots, ((o_M, c_M), \hat{o}_M, \hat{a}_M^*))$ in $\mathcal{D}$, where $\hat{a}_i^* = \pi^*(\hat{o}_1, \dots, \hat{o}_M)_i$.

  if $|\mathcal{D}|$ sufficiently large **then**

    Sample a batch of B multi-agent observations $\{((o_1^{(\beta)}, c_1^{(\beta)}), \hat{o}_1^{(\beta)}, \hat{a}_1^{*,(\beta)}), \dots, ((o_M^{(\beta)}, c_M^{(\beta)}), \hat{o}_M^{(\beta)}, \hat{a}_M^{*,(\beta)})\}_{\beta=1}^B$

    Obtain the up-to-date communication actions from each agent: $b_k^{(\beta)'} = \pi_k(o_k^{(\beta)}, c_k^{(\beta)})_{\text{comm}}$

    for each agent $i = 1$ to M **do**

      Communication loss:

      For each agent $j \neq i$, obtain the up-to-date communication $\hat{c}_j^{(\beta)}$, which contains agent i's communication action to agent j, so we can backprop to agent i's weights

      Obtain the communication loss,

$$\text{communication loss} = \frac{1}{B}\sum_{\beta=1}^{B}\frac{1}{M-1}\sum_{j=1,j\neq i}^{M}\ell(\pi^*(\hat{\mathbf{o}}^{(\beta)})_j,\ \ \pi_j(\hat{o}_j^{(\beta)},\ \hat{c}_j^{(\beta)'})_{\text{action}})$$

      where the subscript "action" denotes the physical action (and not the communication action), and where

$$\hat{c}_j^{(\beta)'} = (b_{1,j}^{(\beta)'}, \dots, \pi_i(o_i^{(\beta)}, c_i^{(\beta)}), \dots, b_{j-1,j}^{(\beta)'}, b_{j+1,j}^{(\beta)'}, \dots, b_{M,j}^{(\beta)'})$$

Action loss:

Obtain the action loss:

$$\text{action loss} = \frac{1}{B}\sum_{\beta=1}^{B} \ell(\pi^*(\hat{\mathbf{o}}^{(\beta)})_i,\ \ \pi_i(\hat{o}_i^{(\beta)},\ \hat{c}_i^{(\beta)}))_{\text{action}})$$

where the subscript "action" denotes the physical action (and not the communication action), and where,

$$\hat{c}_i^{(\beta)'} = (b_{1,i}^{(\beta)'}, \ldots, b_{i-1,i}^{(\beta)'}, b_{i+1,i}^{(\beta)'}, \ldots, b_{M,i}^{(\beta)'})$$

Update:

Update the weights of π_i where the total loss equals the action loss plus the communication loss, to obtain π_i^{new}.

Set $\pi_i \leftarrow \pi_i^{\text{new}}$, for $i = 1, \ldots, M$.

CHAPTER 3

Infinite

Abstract: We present APAC-Net, an alternating population and agent control neural network for solving stochastic mean field games (MFGs). Our algorithm is geared toward high-dimensional instances MFGs that are beyond reach with existing solution methods. We achieve this in two steps. First, we take advantage of the underlying variational primal-dual structure that MFGs exhibit and phrase it as a convex-concave saddle point problem. Second, we parameterize the value and density functions by two neural networks, respectively. By phrasing the problem in this manner, solving the MFG can be interpreted as a special case of training a generative adversarial network (GAN). We show the potential of our method on up to 100-dimensional MFG problems. [85]

3.1 Introduction

Mean field games (MFGs) are a class of problems that model large populations of interacting agents. They have been widely used in economics [1, 3, 59, 55], finance [45, 12, 15, 3], industrial engineering [31, 75, 56], and data science [136, 61, 13]. In mean field games, a continuum population of small rational agents play a non-cooperative differential game on a time horizon $[0, T]$. At the optimum, the agents reach a Nash equilibrium, where they can no longer unilaterally improve their objectives. Given the initial distribution of agents $\rho_0 \in \mathcal{P}(\mathbb{R}^n)$, where $\mathcal{P}(\mathbb{R}^n)$ is the space of all probability densities, the solution to MFGs are

obtained by solving the system of partial differential equations (PDEs),

$$\begin{aligned} &-\partial_t\phi - \nu\Delta\phi + H(x,\nabla\phi) = \mathcal{F}(x,\rho) && \text{(HJB)} \\ &\partial_t\rho - \nu\Delta\rho - \operatorname{div}(\rho\nabla_p H(x,\nabla\phi)) = 0 && \text{(FP)} \\ &\rho(x,0) = \rho_0, \quad \phi(x,T) = \mathcal{G}(x,\rho(\cdot,T)) \end{aligned} \tag{3.1}$$

which couples a Hamilton-Jacobi-Bellman (HJB) equation and a Fokker-Planck (FP) equation. Here, $\phi\colon \mathbb{R}^n \times [0,T] \to \mathbb{R}$ is the value function, $H\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is the Hamiltonian, $\rho(\cdot,t) \in \mathcal{P}(\mathbb{R}^n)$ is the distribution of agents at time t, $\mathcal{F}\colon \mathbb{R}^n \times \mathcal{P}\mathbb{R}^n \to \mathbb{R}$ denotes the interaction between the agents and the population, and $\mathcal{G}\colon \mathbb{R}^n \times \mathcal{P}(\mathbb{R}^n) \to \mathbb{R}$ is the terminal condition, which guides them to a final distribution. Under standard assumptions, i.e., convexity and coercivity of H and $\mathcal{G}$, the solution to (3.1) exists and is unique. See [77, 78, 21] for more details. This formulation can be viewed as a multi-agent reinforcement learning (RL) problem where there are infinitely many players [61, 114, 13], the key difference is that unlike in RL, the reward function and the dynamics (FP) are known. Although there is a plethora of fast solvers for the solution of (3.1) in two and three dimensions [2, 6, 24, 21, 22, 69], numerical methods for solving (3.1) in high dimensions are practically nonexistent due to the need for spatial discretization; this leads to the curse of dimensionality.

Our Contribution We present APAC-Net, an alternating population and agent control neural network approach for tractably solving high-dimensional MFGs in the stochastic case ($\nu > 0$). We phrase the MFG problem as a saddle-point problem [78, 6, 25] and parameterize the value function *and* the density function. This formulation draws a natural connection between MFGs and generative adversarial neural networks (GANs) [57], a powerful class of generative models that have shown remarkable success on various types of datasets [57, 4, 60, 86, 34, 14].

3.2 Variational Primal-Dual Formulation of Mean Field Games

We derive the mathematical formulation of MFGs for our framework; in particular, we arrive at a primal-dual convex-concave formulation tailored for our alternating networks approach. In [25], the authors observe that all MFG systems admit an infinite-dimensional two-player general-sum game formulation, and the potential MFGs are the ones that correspond to zero-sum games. An MFG system (3.1) is called potential, if there exist functionals $\mathcal{F}, \mathcal{G}$ such that $\delta_\rho \mathcal{F} = f(x, \rho)$, and $\delta_\rho \mathcal{G} = g(x, \rho)$ where

$$\langle \delta_\rho \mathcal{F}(\rho), \mu \rangle = \lim_{h \to 0} \frac{\mathcal{F}(\rho + h\mu) - \mathcal{F}(\rho)}{h}, \quad \langle \delta_\rho \mathcal{G}(\rho), \mu \rangle = \lim_{h \to 0} \frac{\mathcal{G}(\rho + h\mu) - \mathcal{G}(\rho)}{h}, \ \forall \ \mu. \tag{3.2}$$

A critical feature of potential MFGs is that the solution to (3.1) can be formulated as the solution to a convex-concave saddle point optimization problem. To this end, we begin by stating (3.1) as a variational problem [78, 6] akin to the Benamou-Brenier formulation for the Optimal Transport (OT) problem:

$$\begin{aligned} &\inf_{\rho, v} \int_0^T \left\{ \int_\Omega \rho(x,t) L(x, v(x,t)) dx \ + \mathcal{F}(\rho(\cdot, t)) \right\} dt + \mathcal{G}(\rho(\cdot, T)) \\ &\text{s.t. } \partial_t \rho - \nu \Delta \rho + \nabla \cdot (\rho v) = 0, \quad \rho(x, 0) = \rho_0(x), \end{aligned} \tag{3.3}$$

where $L \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is the Lagrangian function corresponding to the Legendre transform of the Hamiltonian H, $\mathcal{F}, \mathcal{G} \colon \mathbb{R}^n \times \mathcal{P}(\mathbb{R}^n) \to \mathbb{R}$ are mean field interaction terms, and $v \colon \mathbb{R}^n \times [0, T] \to \mathbb{R}^n$ is the velocity field. Next, setting ϕ as a Lagrange multiplier, we insert the PDE constraint into the objective to get

$$\begin{aligned} \sup_{\phi} \inf_{\rho(x,0) = \rho_0(x), v} &\int_0^T \left\{ \int_\Omega \rho(x,t) L(x, v(x,t)) dx \ + \mathcal{F}(\rho(\cdot, t)) \right\} dt + \mathcal{G}(\rho(\cdot, T)) \\ &- \int_0^T \int_\Omega \phi(x,t) \left(\partial_t \rho - \nu \Delta \rho + \nabla \cdot (\rho(x,t) v(x,t)) \right) dx \, dt. \end{aligned}$$

Finally, integrating by parts and minimizing with respect to v to obtain the Hamiltonian via $H(x, p) = \inf_v \{-p \cdot v + L(x, v)\}$, we obtain

$$\begin{aligned} \inf_{\rho(x,0) = \rho_0(x)} \sup_{\phi} &\int_0^T \left\{ \int_\Omega \left(\partial_t \phi + \nu \Delta \phi - H(x, \nabla \phi) \right) \rho(x,t) \, dx + \mathcal{F}(\rho(\cdot, t)) \right\} dt \\ &+ \int_\Omega \phi(x, 0) \rho_0(x) dx + \mathcal{G}(\rho(\cdot, T)) - \int_\Omega \phi(x, T) \rho(x, T) dx. \end{aligned} \tag{3.4}$$

This formula can also be obtained in the context of HJB equations in density spaces [24], or by integrating the HJB and the FP equations in (3.1) with respect to ρ and ϕ, respectively [25]. According to the interpretation in [25], Player 1 represents the *mean-field* or the *population as a whole* and their strategy is the population density ρ. Furthermore, Player 2 represents the *generic agent* and their strategy is the value function ϕ. The aim of Player 2 is to provide a strategy that yields the best response of a generic agent against the population. This interpretation is in accord with the intuition behind GANs. The formulation (3.4) is the cornerstone of our method.

3.3 Connections to GANs

Generative Adversarial Networks In generative adversarial networks (GANs) [57], we have a discriminator and generator, and the goal is to obtain a generator that is able to produce samples from a desired distribution. The generator does this by taking samples from a known distribution $\mathcal{N}$ and transforming them into samples from the desired distribution. Meanwhile, the purpose of the discriminator is to aid the optimization of the generator. Given a generator network G_θ and a discriminator network D_ω, the original GAN objective is to find an equilibrium to the minimax problem

$$\inf_{G_\theta} \sup_{D_\omega} \mathbb{E}_{x\sim\rho_0} \left[\log D_\omega(x)\right] + \mathbb{E}_{z\sim\mathcal{N}} \left[\log(1 - D_\omega(G_\theta(z)))\right].$$

Here, the discriminator acts as a classifier that attempts to distinguish real images from fake/generated images, and the goal of the generator is to produce samples that "fool" the discriminator.

Wasserstein GANs In Wasserstein GANs [4], the motivation is drawn from OT theory, where now the objective function is changed to the Wasserstein-1 (W1) distance in the Kantorovich-Rubenstein dual formulation

$$\inf_{G_\theta} \sup_{D_\omega} \mathbb{E}_{x\sim\rho_0} \left[D_\omega(x)\right] - \mathbb{E}_{z\sim\mathcal{N}} \left[D_\omega(G_\theta(z))\right], \quad \text{s.t.} \quad \|\nabla D\| \leqslant 1, \tag{3.5}$$

and the discriminator is required to be 1-Lipschitz. In this setting, the goal of the discriminator is to compute the W1 distance between the distribution of ρ_0 and $G_\theta(z)$. In practice, using the W1 distance helps prevent the generator from suffering "mode collapse," a situation where the generator produces samples from only one mode of the distribution ρ_0; for instance, if ρ_0 is the distribution of images of handwritten digits, then mode collapse entails producing only, say, the 0 digit. Originally, [4] used weight-clipping to enforce the Lipschitz condition of the discriminator network, but an improved method using a penalty on the gradient was used in [60].

GANs ↔ MFGs A Wasserstein GAN can be seen as a particular instance of a deterministic MFG [11, 6, 78]. Specifically, consider the MFG (3.4) in the following setting. Let $\nu = 0$, $\mathcal{G}$ be a hard constraint with target measure ρ_T (as in optimal transport), and let H be the Hamiltonian defined by

$$H(x,p) = \mathbb{1}_{\|p\| \leqslant 1} = \begin{cases} 0 & \|p\| \leqslant 1 \\ \infty & \text{otherwise} \end{cases}, \tag{3.6}$$

where we note that this Hamiltonian arises when the Lagrangian is given by $L(x,v) = \frac{\|v\|_2}{2}$. Then (3.4) reduces to,

$$\begin{aligned} \sup_{\phi} \int_\Omega \phi(x)\rho_0(x)\,dx - \int_\Omega \phi(x)\rho_T(x)\,dx \\ \text{s.t. } \|\nabla\phi(x)\| \leqslant 1, \end{aligned}$$

where we note that the optimization in ρ leads to $\partial_t\phi - H(x,\nabla\phi) = 0$. And since $H(p) = \mathbb{1}_{\|p\|\leqslant 1}$, we have that $\partial_t\phi = 0$, and $\phi(x,t) = \phi(x)$ for all t. We observe the above is precisely the Wasserstein-1 distance in the Kantorovich-Rubenstein duality [134].

3.4 APAC-Net

The training process for our MFG is similar to that of GANs. We initialize neural networks $N_\omega(x,t)$ and $N_\theta(x,t)$. We then let

$$\phi_\omega(x,t) = (1-t)N_\omega(x,t) + t\mathcal{G}(x), \quad G_\theta(x,t) = (1-t)z_b + tN_\theta(x,t), \tag{3.7}$$

where $z_b \sim \rho_0$ are samples drawn from the initial distribution. Thus, G_θ is the pushforward of ρ_0. One difference between our formulation and GANs is that ϕ_ω automatically encodes terminal condition by design.

Our strategy for training this GAN-like MFG consists of alternately training G_θ (the population), and ϕ_ω (the value function for an individual agent). Intuitively, this means we are *alternating the population and agent control neural networks* (APAC-Net) in order to find the equilibrium. Specifically, we train ϕ_ω by first sampling a batch $\{z_b\}_{b=1}^B$ from the given initial density ρ_0, and $\{t_b\}_{b=1}^B$ uniformly from $[0,1]$. Next, we compute the push-forward $x_b = G_\theta(z_b, t_b)$ for $b = 1, \ldots, B$. We then compute the loss,

$$\text{loss}_\phi = \frac{1}{B}\sum_{b=1}^{B} \phi_\omega(x_b, 0) + \frac{1}{B}\sum_{b=1}^{B} \partial_t\phi_\omega(x_b,t_b) + \nu\Delta\phi_\omega(x_b,t_b) - H(\nabla_x\phi_\omega(x_b,t_b))$$

where we can optionally add a regularization term $\lambda\frac{1}{B}\sum_{b=1}^{B} \|\partial_t\phi_\omega(x_b,t_b) + \nu\Delta\phi_\omega(x_b,t_b) - H(\nabla_x\phi_\omega(x_b,t_b)) + \mathcal{F}(x_b,t_b)\|^2$ to penalize deviations from the HJB equations [114]. This extra regularization term has also been found effective in, e.g., Wasserstein GANs [59], where the norm of the gradient (i.e., the HJB equations) is penalized. Finally, we backpropagate the loss to the weights of ϕ_w.

To train the generator, we again sample $\{z_b\}_{b=1}^B$ and $\{t_b\}_{b=1}^B$ as before, and compute

$$\text{loss}_\rho = \frac{1}{B}\sum_{b=1}^{B} \partial_t\phi_\omega(\rho_\theta(z_b),t_b) + \nu\Delta\phi_\omega(\rho_\theta(z_b),t_b) - H(\nabla_x\phi_\omega(\rho_\theta(z_b),t_b)) + \mathcal{F}(\rho_\theta(z_b),t_b).$$

We then backpropagate this loss with respect to the weights of G_θ. Our pseudocode is shown in Algorithm 12.

Algorithm 12 APAC-Net for Mean-Field Games

Require: ν diffusion parameter, $\mathcal{G}(x)$ terminal cost, H Hamiltonian, $\mathcal{F}$ interaction term.

Require: Initialize neural networks N_ω and N_θ, batch size B

Require: Set ϕ_ω and G_θ as in (3.7)

Require: k where we update G_θ every k epochs.

while not converged **do**

Train ϕ_ω:

Sample batch $\{(z_b, t_b)\}_{b=1}^B$ where $z_b \sim \rho_0$ and $t_b \sim \text{Unif}(0, T)$

$x_b \leftarrow G_\theta(z_b, t_b)$ for $b = 1, \ldots, B$.

$\ell_0 \leftarrow \frac{1}{B}\sum_{b=1}^B \phi_\omega(x_b, 0)$

$\ell_t \leftarrow \frac{1}{B}\sum_{b=1}^B \partial_t\phi_\omega(x_b, t_b) + \nu\Delta\phi_\omega(x_b, t_b) - H(\nabla_x\phi_\omega(x_b, t_b))$

$\ell_{\text{lam}} \leftarrow \lambda\frac{1}{B}\sum_{b=1}^B \|\partial_t\phi_\omega(x_b, t_b) + \nu\Delta\phi_\omega(x_b, t_b) - H(\nabla_x\phi_\omega(x_b, t_b)) + \mathcal{F}(x_b, t_b)\|^2$

Backpropagate the loss $\ell_{\text{total}} = \ell_0 + \ell_t + \ell_{\text{lam}}$ to ω weights.

Train ρ_θ **every** k **epochs**:

Sample batch $\{(z_b, t_b)\}_{b=1}^B$ where $z_b \sim \rho_0$ and $t_b \sim \text{Unif}(0, T)$

$\ell_t \leftarrow \frac{1}{B}\sum_{b=1}^B \partial_t\phi_\omega(\rho_\theta(z_b, t_b), t_b) + \nu\Delta\phi_\omega(\rho_\theta(z_b, t_b), t_b) - H(\nabla_x\phi_\omega(\rho_\theta(z_b, t_b), t_b)) + \mathcal{F}(\rho_\theta(z_b, t_b), t_b)$

Backpropagate the loss $\ell_{\text{total}} = \ell_0 + \ell_t$ to θ weights.

3.5 Related Works

High-dimensional MFGs and Optimal Control To the best of our knowledge, the first work to solve MFGs efficiently in high dimensions ($d = 100$) was done by [114]. Their work consisted of using Lagrangian coordinates and parameterizing the value function using a neural network. This combination allowed them to successfully avoid the need for spatial grids for MFG problems in the case where $\nu = 0$; that is, in the deterministic case where there is no diffusion. [114] the authors apply the Jacobi identity to estimate the population-density. This formula, however, is available only in the deterministic case. For problems

involving high-dimensional optimal control and differential games, spatial grids were also avoided [21, 22, 24, 82, 29].

Reinforcement Learning Our work bears connections with reinforcement learning (RL). When neither the Lagrangian L, nor the dynamics (constraint) in (3.3) are known, our formulation amounts to solving multi-agent RL problem with infinitely many identical agents. [61] propose a Q-Learning approach to solve these multi-agent RL problems. [13] study the convergence of policy gradient methods on mean field reinforcement learning (MFRL) problems, i.e., problems where the agents try instead to learn the control which is socially optimal for the entire population. [137] use an inverse reinforcement learning approach to learn the MFG model along with its reward function. [51] propose an actor-critic method for finding Nash equilibrium in linear-quadratic mean field games and establish linear convergence.

GAN-based approach A connection between MFGs and GANs is also made by [11]. However, APAC-Net differs from [11] in two fundamental ways. First, instead of choosing the value function to be the generator, we set the *density* function as the generator. This choice is motivated by the fact that the generator outputs samples from a desired distribution. It is also aligned with other generative modeling techniques arising in continuous normalizing flows [44, 58]. Second, rather than setting the generator/discriminator losses as the residual errors of (3.1), we follow the works of [25, 24, 6, 78] and utilize the underlying variational primal-dual structure of MFGs, see (3.4); this allows us to arrive at the Kantorovich-Rubenstein dual formulation of Wasserstein GANs [134].

3.6 Numerical Results

Experimental Setup We assume without loss of generality $T = 1$. In all experiments, our neural networks have three hidden layers, with 100 hidden units per layer. We use a Residual Neural Network (ResNet) for both networks, where the weight on the skip connection is 0.5. In the discriminator, we use the Tanh activation function, and in the generator, we use the

ReLU activation function. To train the network, we use ADAM with $\beta = (0.5, 0.9)$, learning rate 5×10^{-4} for the discriminator, and 1×10^{-4} for the generator, and batch size of 50.

We approximate the solution to a mean field game where the population must move from a starting point to an end point while avoiding obstacles. We thus choose the Hamiltonian to be

$$H(x, p, t) = c\|p\|_2 + \mathcal{F}(x, \rho(x, t))$$

where $\mathcal{F}(x, \rho(x, t))$ varies with the environment (either avoiding obstacles, or avoiding congestion, etc.). Furthermore, we choose as terminal cost

$$\mathcal{G}(\rho(\cdot, T)) = \int_\Omega \|x - x_T\|_2 \rho(x, T) dx$$

where the first term is the distance between the population and a target destination. For the obstacle and congestion problems, we only let the obstacles and congestion be contained within the first two dimensions, so that we are able to verify correctness with the two dimensional case. More experimental information, such as the epochs for each calculation, or various constants, will be left in the appendix. For all experiments, we chose the stopping criteria to be when the loss ℓ_{lam} in (12) plateaued.

Obstacles In this experiment, we compute the solution to a MFG where the agents are required to avoid obstacles. In this case, we let

$$\mathcal{F}(x = (x_1, x_2, \ldots, x_d)) = \mathcal{F}(x_1, x_2) = y = \begin{cases} y > 0 & \text{if } (x_1, x_2) \text{ is inside the obstacle, i.e. a collision} \\ 0 & \text{everywhere else.} \end{cases}$$

the complete analytic expression of the boundary is given in the appendix. Our initial density ρ_0 is a Gaussian centered at $(-2, -2, 0, \ldots, 0)$ with standard deviation $1/\sqrt{10} \approx 0.32$. We let the terminal function bet $\mathcal{G}(x) = \|(x_1, x_2) - (2, 2)\|_2$. The numerical results are shown in 3.1. Observe that the results are similar across dimensions, which means we have verified correctness.

Effect of Parameter ν We investigate the effect of the diffusion parameter ν on the behavior of the MFG solutions. In Fig. 3.2, we show the solutions for 2-dimensional MFGs

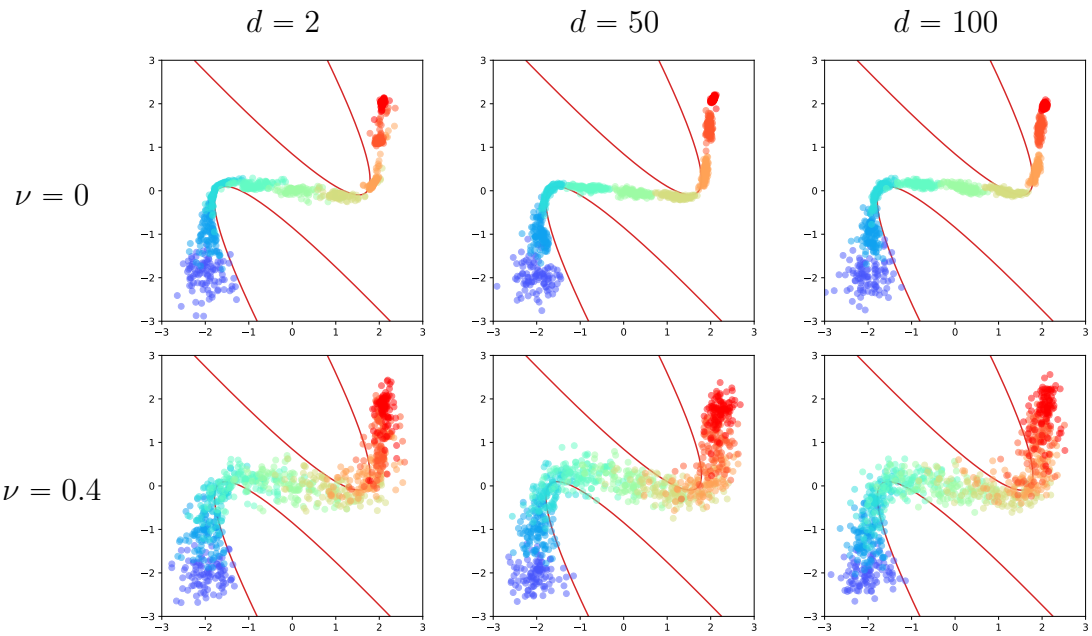


Figure 3.1: Computation of the obstacle problem in dimensions 2, 50, and 100 with stochastic parameter $\nu = 0$ and 0.4. For dimension 50 and 100, we plot the first two dimensions. The agents start at the blue points ($t = 0$) and end at the red points ($t = 1$). As can be seen, the results are similar across dimensions, which verifies correctness of the high-dimensional (50 and 100) computations.

using $\nu = 0, 0.2, 0.4$, and 0.6. The blue dots represent the initial starting points of the agents, the red dots represent the final-time positions, and the colors in between are intermediate time-points. As can be seen, as ν increases, the density of agents starts to widen, consistent with intuition from [104].

Congestion In this experiment, we now let the interaction term be a congestion, so that the agents are encouraged to spread out. We only let congestion be in the first two dimensions, so that

$$\mathcal{F}(\rho(x,t)) = \alpha_{\text{cong}} \int_0^T \int_0^T \frac{1}{\|(x_1,x_2)-(y_1,y_2)\|^2 + 1}\, d\rho(x,t)\, d\rho(y,t)\, dy$$

which is the (bounded) inverse average distance between pairs of agents. Here we let our initial density ρ_0 is a Gaussian centered at $(-2, 0, -2, \ldots, -2)$ with standard deviation

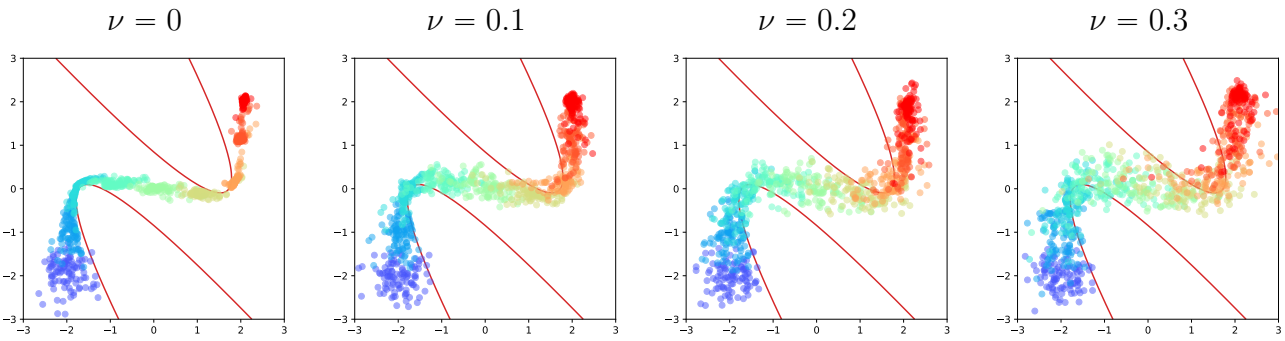


Figure 3.2: Comparison of 2D solutions for different values of ν.

$1/\sqrt{10} \approx 0.32$. We let the terminal function be $\mathcal{G}(x) = \|(x_1, x_2) - (2, 0)\|_2$.

3.7 Conclusion

We present APAC-Net, an alternating population-agent control neural network for solving high-dimensional stochastic mean field games. To this end, our algorithm avoids the use of spatial grids by parameterizing the controls, ϕ and ρ, using two neural networks, respectively. Our method is geared toward high-dimensional instances of these problems that are beyond reach with existing methods. Our method also has natural connections with Wasserstein GANs, where ρ acts as a generative network and ϕ acts as a discriminative network. Our experiments show that our method is effective in solving up to 100-dimensional MFGs. A future direction we intend to investigate is the theoretical guidelines on the design of network architectures.

Acknowledgments and Disclosure of Funding

The authors are supported by AFOSR MURI FA9550-18-1-0502, AFOSR Grant No. FA9550-18-1-0167, and ONR Grant No. N00014-18-1-2527.

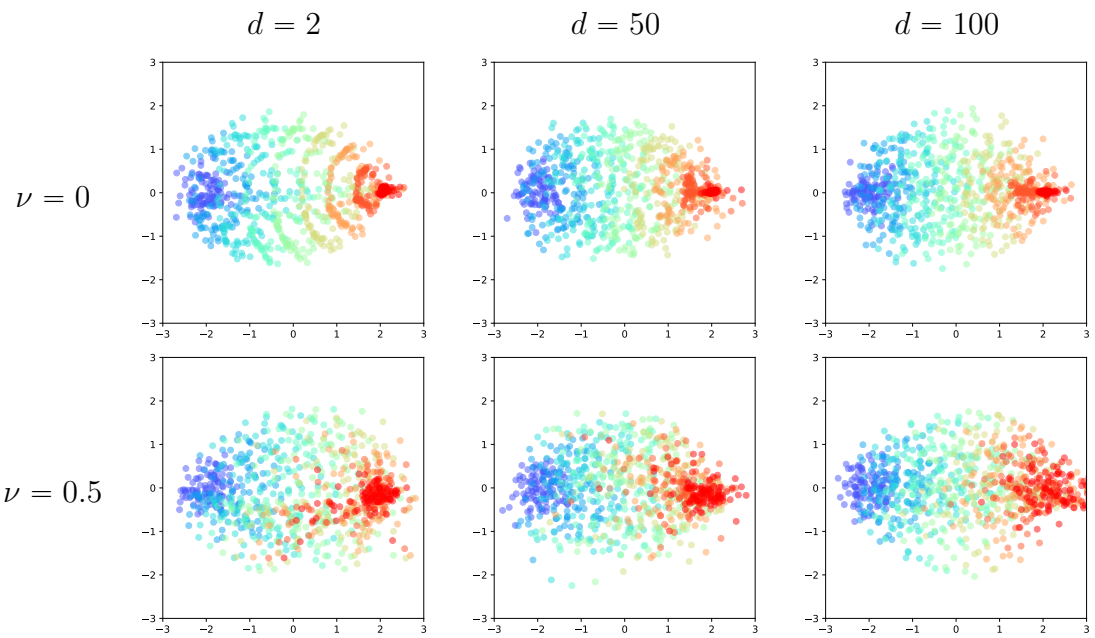


Figure 3.3: Computation of the congestion problem in dimensions 2, 50, and 100 with stochastic parameter $\nu = 0$ and 0.5. For dimension 50 and 100, we plot the first two dimensions. For the $\nu = 0$ case, we see in dimension 2 that the agents are more semi-circular, but this is still retained in a slightly more smeared fashion in dimensions 50 and 100. In the stochastic $\nu = 0.5$ case, we see the results are similar, verifying correctness of the computations for high-dimensions.

Broader Impact

Many applications involving a large population of agents such as swarm robotics, 5G networks, stock market, and spread disease modeling often require solving high-dimensional mean field games. Our work provides a way to solve these types of realistic problems since it overcomes the curse of dimensionality. Our work also bridges two recent and independent fields: mean field games and generative adversarial networks. Therefore, it sets the stage for the research and development of methods that exploit these connections for efficient training generative modeling and simulation of mean field games.

3.8 Appendix: Explanations of the environments and experimental setup

Obstacle In the obstacle problem, the obstacle penalty was calculated to be,

$$\mathcal{F}(x = (x_1, x_2, \ldots, x_d)) = \mathcal{F}(x_1, x_2) = (\max f_1(x_1, x_2), 0 + \max f_2(x_1, x_2), 0)\alpha_{\text{obst}}$$

with $\alpha_{\text{obst}} = 5$, and if we denote $v = (x_1, x_2)$, then letting $c_1 = (-2, 0.5)$ and $R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$ with $\theta = \pi/5$, and $Q = \left(\begin{smallmatrix} 5 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ and $b = (0, 2)$, then

$$f_1(x_1, x_2) = -\langle (v - c_1)R, Q(v - c_1)R \rangle - \langle b, (v - c_1)R \rangle - 1.$$

Similarly, letting $c_2 = (2, -0.5)$, then we let

$$f_2(x_1, x_2) = -\langle (v - c_2)R, Q(v - c_2)R \rangle + \langle b, (v - c_2)R \rangle - 1.$$

Our stopping criteria was when ℓ_{lam} in (12) plateaued. Therefore, the epochs we chose are: For dimensions 2 with $\nu = 0, 0.2, 0.4, 0.6$ we stopped at epoch 200k. For dimensions 50 and 100, as well as $nu = 0$ and 0.4, we stopped at epoch 300k.

Congestion Our stopping criteria was when ℓ_{lam} in (12) plateaued. The epochs we chose were 100k for the 2 dimensional cases, and 500k for the 50 and 100 dimensional cases.

Bibliography

[1] Yves Achdou, Francisco J. Buera, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. Partial differential equation models in macroeconomics. *Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.*, 372(2028):20130397, 19, 2014.

[2] Yves Achdou and Italo Capuzzo-Dolcetta. Mean field games: numerical methods. *SIAM Journal on Numerical Analysis*, 48(3):1136–1162, 2010.

[3] Yves Achdou, Jiequn Han, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. Income and wealth distribution in macroeconomics: A continuous-time approach. Working Paper 23732, National Bureau of Economic Research, August 2017.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[5] Somil Bansal, Mo Chen, Sylvia L. Herbert, and Claire J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. *CoRR*, abs/1709.07523, 2017.

[6] Jean-David Benamou, Guillaume Carlier, and Filippo Santambrogio. Variational mean field games. In *Active Particles, Volume 1*, pages 141–171. Springer, 2017.

[7] Daniel S Bernstein, Eric A Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized pomdps. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI)*, pages 52–57, 2005.

[8] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[9] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[10] Lucian Busoniu, Robert Babuska, and Bart De Schutter. Multi-agent reinforcement learning : An overview. 2010.

[11] Haoyang Cao, Xin Guo, and Mathieu Laurière. Connecting gans and mfgs. *arXiv preprint arXiv:2002.04112*, 2020.

[12] Pierre Cardaliaguet and Charles-Albert Lehalle. Mean field game of controls and an application to trade crowding. *Math. Financ. Econ.*, 12(3):335–363, 2018.

[13] René Carmona, Mathieu Laurière, and Zongjun Tan. Linear-quadratic mean-field reinforcement learning: convergence of policy gradient methods. *arXiv preprint arXiv:1910.04295*, 2019.

[14] Kenji Fukumizu Casey Chu, Kentaro Minami. Smoothness and stability in gans. *arXiv preprint arXiv:2002.04185*, 2020.

[15] Philippe Casgrain and Sebastian Jaimungal. Algorithmic trading in competitive markets with mean field games. *SIAM News*, 52(2), 2019.

[16] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, May 2011.

[17] Antonin Chambolle and Thomas Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1):253–287, Sep 2016.

[18] Pratik Chaudhari, Adam M. Oberman, Stanley Osher, Stefano Soatto, and Guillaume Carlier. Deep relaxation: partial differential equations for optimizing deep neural networks. *CoRR*, abs/1704.04932, 2017.

[19] Yat Tin Chow, Jérôme Darbon, Stanley Osher, and Wotao Yin. Algorithm for overcoming the curse of dimensionality for certain non-convex hamilton-jacobi equations, projections and differential games. *Annals of Mathematical Sciences and Applications*, 2016.

[20] Yat Tin Chow, Jérôme Darbon, Stanley Osher, and Wotao Yin. Algorithm for overcoming the curse of dimensionality for time-dependent non-convex hamilton–jacobi equations arising from optimal control and differential games problems. *Journal of Scientific Computing*, 73(2):617–643, Dec 2017.

[21] Yat Tin Chow, Jérôme Darbon, Stanley Osher, and Wotao Yin. Algorithm for overcoming the curse of dimensionality for time-dependent non-convex hamilton–jacobi equations arising from optimal control and differential games problems. *Journal of Scientific Computing*, 73(2-3):617–643, 2017.

[22] Yat Tin Chow, Jérôme Darbon, Stanley Osher, and Wotao Yin. Algorithm for overcoming the curse of dimensionality for certain non-convex hamilton–jacobi equations, projections and differential games. *Annals of Mathematical Sciences and Applications*, 3(2):369–403, 2018.

[23] Yat-Tin Chow, Jérôme Darbon, Stanley Osher, and Wotao Yin. Algorithm for overcoming the curse of dimensionality for state-dependent hamilton-jacobi equations. UCLA CAM 17-16, April 2017.

[24] Yat Tin Chow, Wuchen Li, Stanley Osher, and Wotao Yin. Algorithm for hamilton–jacobi equations in density space via a generalized hopf formula. *Journal of Scientific Computing*, 80(2):1195–1239, 2019.

[25] Marco Cirant and Levon Nurbekyan. The variational structure and time-periodic solutions for mean-field games systems. *Minimax Theory Appl.*, 3(2):227–260, 2018.

[26] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752, 1998.

[27] Jérôme Darbon. On convex finite-dimensional variational methods in imaging sciences and hamilton–jacobi equations. *SIAM Journal of Imaging Science*, 8(4):2268–2293, June 2015.

[28] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sceiences*, 3(1):19, September 2016.

[29] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sciences*, 3(1):19, 2016.

[30] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[31] A. De Paola, V. Trovato, D. Angeli, and G. Strbac. A mean field game approach for distributed control of thermostatic loads acting in simultaneous energy-frequency response markets. *IEEE Transactions on Smart Grid*, 10(6):5987–5999, Nov 2019.

[32] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[33] Roel Dobbe, David Fridovich-Keil, and Claire Tomlin. Fully decentralized policies for multi-agent systems: An information theoretic approach. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2941–2950. Curran Associates, Inc., 2017.

[34] Yonatan Dukler, Wuchen Li, Alex Tong Lin, and Guido Montúfar. Wasserstein of wasserstein loss for learning generative models. 2019.

[35] Pavel Dvurechensky, Yurii Nesterov, and Vladimir Spokoiny. Primal-dual methods for solving infinite-dimensional games. *Journal of Optimization Theory and Applications*, 166(1):23–51, Jul 2015.

[36] Maxim Egorov. Multi-agent deep reinforcement learning, 2016.

[37] Robert J. Elliott and Nigel J. Kalton. Values in differential games. *Bull. Amer. Math. Soc.*, 78(3):427–431, 05 1972.

[38] Ernie Esser, Xiaoqun Zhang, and Tony F. Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.

[39] Lawrence C. Evans. Envelopes and nonconvex hamilton–jacobi equations.

[40] Lawrence C. Evans. *Partial differential equations*. American Mathematical Society, Providence, R.I., 2010.

[41] Lawrence C. Evans. An introduction to mathematical optimal control theory version 0.2, 2017.

[42] Lawrence C. Evans and Panagiotis E. Souganidis. Differential games and representation formulas for solutions of hamilton-jacobi-isaacs equations. mar 1983.

[43] Richard Evans and Jim Gao. Deepmind ai reduces google data centre cooling bill by 40. 2017.

[44] Chris Finlay, Björn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode. *arXiv:2002.02798*, 2020.

[45] D. Firoozi and P. E. Caines. An optimal execution problem in finance targeting the market trading speed: An mfg formulation. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 7–14, Dec 2017.

[46] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[47] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In D. D. Lee,

M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2137–2145. Curran Associates, Inc., 2016.

[48] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.

[49] Jakob N. Foerster, Christian A. Schröder de Witt, Gregory Farquhar, Philip H. S. Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. *CoRR*, abs/1810.11702, 2018.

[50] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *CoRR*, abs/1705.08926, 2017.

[51] Zuyue Fu, Zhuoran Yang, Yongxin Chen, and Zhaoran Wang. Actor-critic provably finds nash equilibria of linear-quadratic mean-field games. In *International Conference on Learning Representations*, 2020.

[52] Luis Rodolfo García Carrillo, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard. *Modeling the Quad-Rotor Mini-Rotorcraft*, pages 23–34. Springer London, London, 2013.

[53] Tom Goldstein, Min Li, and Xiaoming Yuan. Adaptive primal-dual splitting methods for statistical learning and image processing. In *Advances in Neural Information Processing Systems*, pages 2089–2097, 2015.

[54] Tom Goldstein and Stanley Osher. The split bregman method for l1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[55] Diogo A Gomes, Levon Nurbekyan, and Edgard A Pimentel. *Economic models and mean-field games theory*. IMPA Mathematical Publications. Instituto Nacional de Matemática Pura e Aplicada (IMPA), Rio de Janeiro, 2015.

[56] Diogo A. Gomes and J. Saúde. A mean-field game approach to price formation in electricity markets. *Preprint*, 2018. arXiv:1807.07088 [math.AP].

[57] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[58] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations (ICLR)*, 2019.

[59] Olivier Guéant, Jean-Michel Lasry, and Pierre-Louis Lions. Mean field games and applications. In *Paris-Princeton lectures on mathematical finance 2010*, pages 205–266. Springer, 2011.

[60] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[61] Xin Guo, Anran Hu, Renyuan Xu, and Junzi Zhang. Learning mean-field games. In *Advances in Neural Information Processing Systems*, pages 4967–4977, 2019.

[62] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[63] Richard F. Hartl, Suresh P. Sethi, and Raymond G. Vickson. A survey of the maximum principles for optimal control problems with state constraints. *SIAM Review*, 37(2):181–218, 1995.

[64] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[65] EBERHARD HOPF. Generalized solutions of non-linear equations of first order. *Journal of Mathematics and Mechanics*, 14(6):951–973, 1965.

[66] M. B. Horowitz, A. Damle, and J. W. Burdick. Linear hamilton jacobi bellman equations in high dimensions. In *53rd IEEE Conference on Decision and Control*, pages 5880–5887, Dec 2014.

[67] Rufus P. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Dover Publications, 1999.

[68] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 09 1998.

[69] Matt Jacobs, Flavien Léger, Wuchen Li, and Stanley Osher. Solving large-scale optimization problems with a convergence rate independent of grid size. *SIAM Journal on Numerical Analysis*, 57(3):1100–1123, 2019.

[70] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[71] D. Kalise and K. Kunisch. Polynomial approximation of high-dimensional Hamilton-Jacobi-Bellman equations and applications to feedback control of semilinear parabolic PDEs. *ArXiv e-prints*, February 2017.

[72] Wei Kang and Lucas C. Wilcox. Mitigating the curse of dimensionality: sparse grid characteristics method for optimal feedback control and hjb equations. *Computational Optimization and Applications*, 68(2):289–315, Nov 2017.

[73] M. R. Kirchner, R. Mar, G. Hewer, J. Darbon, S. Osher, and Y. T. Chow. Time-optimal collaborative guidance using the generalized hopf formula. *IEEE Control Systems Letters*, 2(2):201–206, April 2018.

[74] Matthew Kirchner, Gary Hewer, Jerome Darbon, and Stanley Osher. A primal-dual

method for optimal control and trajectory generation in high-dimensional systems. UCLA CAM 18-04, January 2018.

[75] Arman C. Kizilkale, Rabih Salhab, and Roland P. Malhamé. An integral control formulation of mean field game based large scale coordination of loads in smart grids. *Automatica*, 100:312 – 322, 2019.

[76] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82 – 94, 2016.

[77] Jean-Michel Lasry and Pierre-Louis Lions. Jeux à champ moyen. ii–horizon fini et contrôle optimal. *Comptes Rendus Mathématique*, 343(10):679–684, 2006.

[78] Jean-Michel Lasry and Pierre-Louis Lions. Mean field games. *Jpn. J. Math.*, 2(1):229–260, 2007.

[79] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.

[80] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[81] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[82] Alex Tong Lin, Yat Tin Chow, and Stanley J Osher. A splitting method for overcoming the curse of dimensionality in hamilton–jacobi equations arising from nonlinear optimal control and differential games with applications to trajectory generation. *Communications in Mathematical Sciences*, 16(7), 2018.

[83] Alex Tong Lin, Yat Tin Chow, and Stanley J. Osher. A splitting method for overcoming the curse of dimensionality in hamilton–jacobi equations arising from nonlinear optimal control and differential games with applications to trajectory generation. *Communications in Mathematical Sciences*, 16(7), 1 2018.

[84] Alex Tong Lin, Mark J. Debord, Katia Estabridis, Gary A. Hewer, and Stanley J. Osher. CESMA: centralized expert supervises multi-agents. *CoRR*, abs/1902.02311, 2019.

[85] Alex Tong Lin, Samy Wu Fung, Wuchen Li, Levon Nurbekyan, and Stanley J. Osher. Apac-net: Alternating the population and agent control via two neural networks to solve high-dimensional stochastic mean field games, 2020.

[86] Alex Tong Lin, Wuchen Li, Stanley Osher, and Guido Montúfar. Wasserstein proximal of gans. 2018.

[87] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[88] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[89] L. Matignon, G. J. Laurent, and N. L. Fort-Piat. Hysteretic q-learning :an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69, Oct 2007.

[90] Laetitia Matignon, Guillaume j. Laurent, and Nadine Le fort piat. Review: Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *Knowl. Eng. Rev.*, 27(1):1–31, February 2012.

[91] Ian M. Mitchell, Mo Chen, and Meeko Oishi. Ensuring safety of nonlinear sampled data systems through reachability1. *IFAC Proceedings Volumes*, 45(9):108 – 114, 2012. 4th IFAC Conference on Analysis and Design of Hybrid Systems.

[92] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[93] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[94] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[95] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6):2432–2442, Nov 2013.

[96] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[97] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. *CoRR*, abs/1703.06182, 2017.

[98] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer–Verlag New York, Inc., 2003.

[99] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12 – 49, 1988.

[100] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for hamilton–jacobi equations. *SIAM Journal on Numerical Analysis*, 28(4):907–922, August 1991.

[101] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, Nov 2005.

[102] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[103] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[104] Christian Parkinson, David Arnold, Andrea L. Bertozzi, and Osher Stanley. A model for optimal human navigation with stochastic effects. *UCLA CAM preprint:19-50*, 2019.

[105] James Paulos, Steven W. Chen, Daigo Shishika, and Vijay Kumar. Decentralization of multiagent policies by learning what to communicate. 2018.

[106] Liqun Qi and Wenyu Sun. *An Iterative Method for the Minimax Problem*, pages 55–67. Springer US, Boston, MA, 1995.

[107] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. *CoRR*, abs/1803.11485, 2018.

[108] R. Rockafellar. *Conjugate Duality and Optimization*. Society for Industrial and Applied Mathematics, 1974.

[109] R. Tyrrell Rockafellar and Peter R. Wolenski. Convexity in hamilton–jacobi theory i: Dynamics and duality. *SIAM Journal on Control and Optimization*, 39(5):1323–1350, 2000.

[110] R. Tyrrell Rockafellar and Peter R. Wolenski. Convexity in hamilton–jacobi theory ii: Envelope representations. *SIAM Journal on Control and Optimization*, 39(5):1351–1372, 2000.

[111] I. Michael Ross and Fariba Fahroo. Legendre pseudospectral approximations of optimal control problems. In Wei Kang, Carlos Borges, and Mingqing Xiao, editors, *New Trends in Nonlinear Dynamics and Control and their Applications*, pages 327–342, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[112] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[113] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[114] Lars Ruthotto, Stanley Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *arXiv preprint arXiv:1912.01825*, 2019.

[115] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[116] Felipe Leno Da Silva, Matthew E. Taylor, and Anna Helena Reali Costa. Autonomously reusing knowledge in multiagent reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5487–5493. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[117] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc

Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[118] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395. JMLR.org, 2014.

[119] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[120] E. Stefansson and Y. P. Leong. Sequential alternating least squares for solving high dimensional linear hamilton-jacobi-bellman equation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3757–3764, Oct 2016.

[121] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[122] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016.

[123] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[124] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[125] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[126] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[127] Ming Tan. Readings in agents. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, chapter Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents, pages 487–494. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[128] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[129] Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems*, pages 871–878, 2004.

[130] Yen-Hsi Richard Tsai, Li-Tien Cheng, Stanley Osher, and Hong-Kai Zhao. Fast sweeping algorithms for a class of hamilton–jacobi equations. *SIAM Journal on Numerical Analysis*, 41(2):673–694, May 2003.

[131] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. In *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, volume 2, pages 1368–1373 vol.2, Dec 1994.

[132] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *CoRR*, abs/1609.02993, 2016.

[133] P. P. Varaiya. Differential games. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 3: Probability Theory*, pages 687–697, Berkeley, Calif., 1972. University of California Press.

[134] Cédric Villani. *Topics in Optimal Transportation*. American Mathematical Soc., 2003.

[135] Oskar von Stryk. *Numerical Solution of Optimal Control Problems by Direct Collocation*, pages 129–143. Birkhäuser Basel, Basel, 1993.

[136] E Weinan, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10, 2019.

[137] Jiachen Yang, Xiaojing Ye, Rakshit Trivedi, Huan Xu, and Hongyuan Zha. Deep mean field games for learning optimal behavior policy of large populations. In *International Conference on Learning Representations*, 2018.

[138] Jiongmin Yong. *Differential Games: A Concise Introduction*. World Scientific, 2014.

[139] Mingqiang Zhu and Tony Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. UCLA CAM, 2008.