Title

Geometric and Topological methods in Reinforcement Learning and Biological Image Segmentation

Author

Wu, Yue

Publication Date

2024

Peer reviewed|Thesis/dissertation

Geometric and Topological methods in
Reinforcement Learning and Biological Image Segmentation

By

Yue Wu
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Jesús A. De Loera, Chair

Xin Liu

Shiqian Ma

Committee in Charge

2024

Contents

Abstract

Geometry plays a crucial role in machine learning. We study the geometric properties of machine learning problems and use that information to develop new algorithms that are accurate and efficient. We present three works that lie at the intersection of machine learning and geometry, and we hope to promote more research on geometry-inspired machine learning methods.

We first introduce *geometric policy iteration* (GPI), a new dynamic programming approach for finite Markov decision process. Recently discovered polyhedral structures of the value function for finite discounted Markov decision processes (MDP) shed light on understanding the success of reinforcement learning. We investigate the value function polytope in greater detail and characterize the polytope boundary using a hyperplane arrangement. We further show that the value space is a union of finitely many cells of the same hyperplane arrangement, and relate it to the polytope of the classical linear programming formulation for MDPs. Inspired by these geometric properties, we propose GPI to solve discounted MDPs. GPI updates the policy of a single state by switching to an action that is mapped to the boundary of the value function polytope, followed by an immediate update of the value function. This new update rule aims at a faster value improvement without compromising computational efficiency. Moreover, our algorithm allows asynchronous updates of state values which is more flexible and advantageous compared to traditional policy iteration when the state set is large. We prove that the complexity of GPI achieves the best known bound $\mathcal{O}\left(\frac{|\mathcal{A}|}{1-\gamma}\log\frac{1}{1-\gamma}\right)$ of policy iteration and empirically demonstrate the strength of GPI on MDPs of various sizes.

In the second project, we consider the *integer feasibility problem*, the challenge of deciding whether a system of linear equations and inequalities has a solution with integer values. This is a famous NP-complete problem with applications in many areas of Mathematics and Computer Science. We show that the integer feasibility problem can be transformed into a 3-D tensor game which we call the *Feasibility Game* (FG). To win the game the player must find a path between the initial state and a final terminal winning state, if one exists. Finding such a winning state is equivalent to solving the integer feasibility problem. The key algebraic ingredient is a Gröbner basis of the toric ideal for the underlying axial transportation polyhedron. The Gröbner basis can be seen as a set of connecting moves (actions) of the game. We then propose a novel RL approach that trains an agent to predict moves in continuous space to cope with the large size of action space. The

continuous move is then projected onto the set of legal moves so that the path always leads to valid states. As a proof of concept we demonstrate in experiments that our agent can play well the simplest version of our game for 2-way tables. Our work highlights the potential to train agents to solve non-trivial mathematical queries through contemporary machine learning methods used to train agents to play games.

In the third work, we develop *PrestoCell* which is a Python-based topological framework for segmenting objects with complex shapes. The main contribution of this work is the use of persistence-based clustering (PBC) to generate segmentations that are topologically correct. Specifically, we use PBC to segment microglia whose 0-d homology is 1 (one connected component), and higher-order homology is 0. PBC, as an unsupervised method, is able to generate high-quality clusters that can be easily improved by some post-processing steps. Our framework is able to take as input very large 3D light microscopy imaging data where a single input volume can contain hundreds of microglia and nuclei. We use PBC to quickly generate candidate microglia clusters which are later refined by the coupled nuclei information. We present the machine-generated segmentation in the free visualization tool Napari. In evaluating and comparing PrestoCell to several existing tools, including a commercial machine-learning implementation, we demonstrate that PrestoCell produces image segmentations that rival or exceed existing solutions. In particular, our use of cell nuclei information resulted in the ability to correctly segment individual cells that were interacting with one another, increasing the accuracy of the segmentation. These benefits are in addition to the simplified graphically based user refinement of cell masks that does not require expensive commercial software licenses. We further demonstrate that PrestoCell can complete image segmentation in large samples from light sheet microscopy, allowing quantitative analysis of these large datasets. As an open-source program that leverages freely available visualization software, with minimum computer requirements, we believe that PrestoCell can significantly increase the ability of users without data or computer science expertise to perform complex image analysis.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor Jesús De Loera for his continuous support and guidance throughout my PhD. His expertise and insights on a variety of research problem have been invaluable for my growth as a researcher. Thank you to my committee members Xin Liu and Shiqian Ma for their fruitful conversation and constructive feedback. I am also very grateful for having the opportunity to collaborate with faculty and peer students. I would like to thank Ingrid Brust-Mascher, Colin Reardon, Hongjing Zhang, Zilong Bai and Yongshuai Liu for a lot of interesting discussions. I am fortunate to have some excellent roommates who have giving me a lot of joyful and fun moments so thank you to Xinyue Hu, Ji Wang, and Likang Yin. Finally, this thesis is dedicated to my parents, Rong Yan and Deyang Wu, for all the years of your love and support.

CHAPTER 1

Introduction

Modern machine learning algorithms have achieved great success since the first deep learning model [52] topped ILSVRC [76] in 2012. The success of deep learning largely depends on the following two factors. The first is the increasing computational power and the second is the amount of data that becomes available to train deep learning models. On the model side, with the invention of Transformer [95], people are building larger and larger models that displayed superior performance over smaller-scale counterparts. For example, ViT [30] is a transformer-based model for vision tasks that shows better performance than the best CNN-based ResNet [40] when trained on large data sets. More notably, GPT-3 has 175 billion parameters which is significantly larger than the initial version of GPT [72] which only has 110 million parameters.

Regardless of how successful the paradigm of training large models with abundant data is, it is beneficial to inject prior (structural) knowledge into the system for the following reasons. First, using prior knowledge makes the training more data-efficient. Second, it remains an open problem if a machine can learn structural knowledge or how to reason from data. In this thesis, we aim to incorporate knowledge from Mathematics, especially geometry, to build better machine learning systems.

Geometry has been playing a crucial rule in machine learning. It helps understand the shape of data and gives insight to the development of new machine learning algorithms. For example, principal component analysis and other manifold learning techniques come from the fact that many data sets have the property that data points lie close to a manifold of much lower dimensionality than that of the original data space [12]. In this thesis, we bring geometry into reinforcement learning and biological image segmentation. In reinforcement learning (RL), an agent interacts with the environment and the goal is to learn a policy that maximizes the cumulative reward from a sequence of decisions. The ability to make sequences of decisions has been the long standing goal of artificial intelligence. With the advancement of deep learning, RL also achieved great sucess by incorporating deep learning into traditional RL algorithms, i.e., Q-learning and actor-critic methods [91, 99]. The

most notable deep RL works include a series of computer Go programs [84, 26, 85] and deep RL agents playing Atari games [60, 59, 94, 98, 79].

1.1. Geometry in Markov Decision Processes

Recently, the research on the geometry of RL received a lot of attention. [22] discovers the *value function polytope* derived from Bellman equation of Markov decision process (MDP). These findings later lead to the direction of using these geometric structures as auxiliary tasks in representation learning in deep RL. [21] improves the representation learning in deep RL by shaping the policy improvement path within the value function polytope. These works on the geometry of MDP become the cornerstone of our work, *geometric policy iteration for Markov decision processes* where we study the geometric properties of discounted MDPs with finite states and actions, and propose a new dynamic programming algorithm inspired by their polyhedral structures.

MDP is the mathematical foundation of reinforcement learning (RL) which has achieved great empirical success in sequential decision problems. Despite RL's success, new mathematical properties of MDPs are to be discovered to better theoretically understand RL algorithms. A large family of methods for solving MDPs is based on the notion of (state) values. The strategy of these methods is to maximize the values, then extract the optimal policy from the optimal values. One classic method is value iteration [46, 7] in which values are greedily improved to optimum using the Bellman operator. It is also well known that the optimal values can be solved by linear programming (LP) [70] which attracts a lot of research interest due to its mathematical formulation. The most efficient algorithms in practice are often variants of policy iteration [46] which facilitates the value improvement with policy updates. The **value function**, which maps policies to the value space, is central to our analysis throughout, and it plays a key role in understanding how values are related to policies from a geometric perspective.

Although policy iteration and its variants are very efficient in practice, their worst-case complexity was long believed exponential [57]. The major breakthrough was made by [102] where the author proved that both policy iteration and LP with Simplex method [25] terminate in $\mathcal{O}\left(\frac{|\mathcal{S}||\mathcal{A}|}{1-\gamma}\log\frac{|\mathcal{S}|}{1-\gamma}\right)$. The author first proved that the Simplex method with the most-negative-reduced-cost pivoting rule is strongly polynomial in this situation. Then, a variant of policy iteration called simple policy iteration was shown to be equivalent to the Simplex method. [39] later improved the complexity of

policy iteration by a factor of $|S|$. The best known complexity of policy iteration is $\mathcal{O}\left(\frac{|\mathcal{A}|}{1-\gamma}\log\frac{1}{1-\gamma}\right)$ proved by [80].

In the LP formulation, the state values are optimized through the vertices of the LP feasible region which is a convex polytope. Surprisingly, it was recently discovered that the space of the value function is a (possibly non-convex) polytopes [22]. We call such object the **value function polytope** denoted by $\mathcal{V}$. As opposed to LP, the state values are navigated through $\mathcal{V}$ in policy iteration. Moreover, the *line theorem* [22] states that the set of policies that only differ in one state is mapped onto the same line segment in the value function polytope. This suggests the potential of new algorithms based on single-state updates.

Our first contribution is on the structure of the value function polytope $\mathcal{V}$. Specifically, we show that a hyperplane arrangement H_{MDP} is shared by $\mathcal{V}$ and the polytope of the linear programming formulation for MDPs. We characterize these hyperplanes using the Bellman equation of policies that are deterministic in a single state. We prove that the boundary of the value function polytope $\partial\mathcal{V}$ is the union of finitely many (convex polyhedral) cells of H_{MDP}. Moreover, each full-dimensional cell of the value function polytope is contained in the union of finitely many full-dimensional cells defined by H_{MDP}. Formally, we have the following theorem.

THEOREM 1.1. *Consider the hyperplane arrangement H_{MDP}, with $|\mathcal{A}||\mathcal{S}|$ hyperplanes, consisting of those of the MDP polytope, i.e.,*

$$H_{MDP} = \left\{ V(s) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) V(s') \mid \forall s \in \mathcal{S}, a \in \mathcal{A} \right\}.$$

Then, the boundary of the value function polytope $\partial\mathcal{V}$ is the union of finitely (convex polyhedral) cells of the arrangement H_{MDP}. Moreover, each full-dimensional cell of the value polytope is contained in the union of finitely many full-dimensional cells defined by H_{MDP}.

We further conjecture that the cells of the arrangement cannot be partial, but they have to be entirely contained in the value function polytope.

The learning dynamic of policy iteration in the value function polytope shows that every policy update leads to an improvement of state values along one line segment of $\mathcal{V}$. Based on this, we propose a new algorithm, **geometric policy iteration** (GPI), a variant of the classic policy iteration with several improvements. First, policy iteration may perform multiple updates on the

same line segment. GPI avoids this situation by always reaching an endpoint of a line segment in the value function polytope for every policy update. This is achieved by efficiently calculating the true state value of each potential policy update instead of using the Bellman operator which only guarantees a value improvement. Second, GPI updates the values for all states immediately after each policy update for a single state, which makes the value function monotonically increasing with respect to every policy update. Last but not least, GPI can be implemented in an asynchronous fashion. This makes GPI more flexible and advantageous over policy iteration in MDPs with a very large state set.

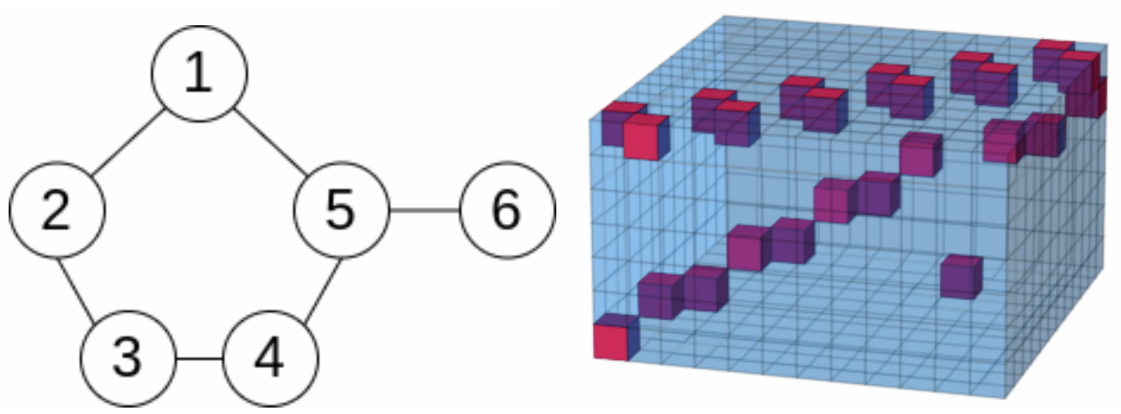
We prove that GPI converges in $\mathcal{O}\left(\frac{|\mathcal{A}|}{1-\gamma}\log\frac{1}{1-\gamma}\right)$ iterations, which matches the best known bound for solving finite discounted MDPs. Although using a more complicated strategy for policy improvement, GPI maintains the same $\mathcal{O}\left(|\mathcal{S}|^2|\mathcal{A}|\right)$ arithmetic operations in each iteration as policy iteration. We empirically demonstrate that GPI takes fewer iterations and policy updates to attain the optimal value.

1.2. Geometry in Integer Programs and Reinforcement Learning

In the second project, we consider the *integer feasibility problem*, a challenge of deciding whether a system of linear equations and inequalities has a solution with integer values. This is a famous NP-complete problem with applications in many areas of Mathematics and Computer Science. We describe a novel algebraic reinforcement learning framework that allows an agent to play a game equivalent to the integer feasibility problem.

Reinforcement learning has seen tremendous success in recent years, especially in playing games at levels that achieve superhuman performances [**60**, **86**, **37**, **96**]). Using machine-learning-based methods to solve Math and NP-hard problems is also an active research area [**4**, **23**, **24**, **64**, **51**, **58**, **65**]. The philosophical principle we introduce is to try to reformulate non-trivial mathematical problems as games and then try to adapt reinforcement learning techniques to play those games. By winning the game we solve the original mathematical problem. Of course this first requires (at least for now) a human creating the right game for the given mathematical problem. As a proof of concept, we investigate the integer feasibility problem. In its simplest form, the IFP is the decision question of whether a polyhedral system $\{x : Ax = b,\ x \geq 0\}$ has an integer vector solution x (note that any system of equations and inequalities can be reduced to this standard

form). This famous NP-complete problem is very important in mathematical optimization, discrete mathematics, algebra, and other areas of mathematics. What we propose here is to turn it into a game on arrays.



(a) (b)

FIGURE 1.1. An illustration of the transformation from a graph instance to a 3-D tensor with enabled entries. (a) A graph matching instance with 6 nodes. (b) The transformed 3-D tensor with red entries as enabled entries.

The theory of the transformation states that any rational polytope $P = \{y \in \mathbb{R}^n_{\geq 0} | Ay = b\}$ can be represented as a 3-way transportation polytope

(1.1)

$$T \;=\; \{\, x \in \mathbb{R}^{r\times r\times h}_{\geq 0} \;\mid\; x_{i,j,k} = 0,\, \forall\,(i,j,k) \notin E\,,\quad \sum_{i,j} x_{i,j,k} = w_k\,,\; \sum_{i,k} x_{i,j,k} = v_j\,,\; \sum_{j,k} x_{i,j,k} = u_i\,\}$$

where E is a set of enabled entries, u, v, and w are 1-margins.

As we explain below we transform every instance of the integer IFP into a positional game over arrays or tables with fixed margin sums belonging to the axial transportation we use an old polynomial-time algorithm from [28] that canonically rewrites an instance of the IFP as the face of a $l \times m \times n$ axial transportation polytope. Axial transportation polytopes are convex polytopes whose points are arrays or tables with fixed axial sums of entries [103]. The second idea is that we know well the toric (polynomial ring) ideals of axial transportation polytopes have an explicit Gröbner basis that connects all lattice points.

An illustration of FG is shown in Fig. 3.1. We consider the problem of graph matching using a simple graph shown in Fig. 3.1a. The problem can be formulated as $\{Ax = b\}$, where A and b are

defined as follows.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

The rows and columns of A denote nodes 1 to 6, and edges $\{(1,2),(2,3),(3,4),(4,5),(1,5),(5,6)\}$, respectively.

Figure 3.1b shows the resulting 3-D integral axial transportation polytope. Only red entries are enabled. 1-margins u, v and w in Eq. (1.1) are as follows

$$u^\top = [1,1,1,1,1,1,1,1,1,1,1,1], \quad v^\top = [1,1,1,1,1,1,1,1,1,1,1,1], \quad w^\top = [1,1,1,1,1,1,6].$$

The game starts with an initial state (an initial array), and by applying a legal move that leaves the margins unchanged, we aim to reach a winning state with zeros in specific positions. To win the game the player must find a path between the initial state and a final terminal winning state. Finding such a winning state is equivalent to solving the integer feasibility problem. As we see later the winning position, if one exists, is essentially a table with very specific zero entries. The Gröbner basis of the toric ideal for the underlying transportation polyhedron as a set generating all connecting moves that the agent can be trained to select the moves. We have the following theorem on the 3D tensor game and integer feasibility problem. Reinforcement learning can then be used to search the game space.

THEOREM 1.2. *Algorithm 3 solves the integer feasibility of any rational polytope P by first transforming it into an 3-way axial transportation polytope Q with specific 1-margins and finding a sequence of Gröbner basis moves from an initial 3-way array in Q to another 3-way array in Q with zeros in specified entries if one exists (equivalent to finding an integer solution for P).*

Reinforcement learning on games has achieved great success and a key point of our work is that the success can be extended to train artificial agents to play these games with a mathematical origin. We make a successful practical demonstration of these ideas with experiments in the simplest form of our games, the situation for 2D tables where we trained an agent to predict the moves. Our

game can be viewed as a variant of the *stochastic shortest path problem* [9, 11] where an agent takes a path to reach pre-defined goal states. However, unlike any of the games like Go, maze, etc, our table game has a very large and highly (algebraically) structured action space. In 2-way tables, the minimum Gröbner basis contains moves of degree 4 (4 non-zero entries) illustrated in Fig. 1.2

FIGURE 1.2. Minimum moves of degree 4.

As the states can be very large, the game horizon can be very long which may require significant exploration before the agent can make any progress. Adding a supervised learning module to the system can increase the learning efficiency comparing to pure RL. These ideas come from the simplest algorithm in RL literature called *behavior cloning* (BC) [67, 78, 74, 73, 105] where an agent learns to copy the behavior of an oracle. In many situations the oracle is imperfect but the performance can be improved when doing RL and BC at the same time.

Another approach to improve learning efficiency is to use a "curriculum" where meaningful intermediate goals are generated so that the agent can obtain reward signals before reaching the final goals. Such strategy is called *curriculum learning* [6, 1, 41, 33, 63]. We also incorporate curriculum learning ideas to speed up the training process.

Although these most basic moves are sufficient for solving the game, the progress they make at each step is limited, which can cause extremely long episodes. We instead consider much more advanced moves that are integer combinations of the minimum moves. These complicated moves will be very effective for solving the table games faster. New techniques are required to compute these advanced moves because they are not only very large in number but also difficult to construct. We treat it as a regression problem and make the agent predict continuous moves which are then projected onto its closest lattice point in a constraint set specified by an integer program. We make the whole framework differentiable and thus can be trained end-to-end.

We conduct experiments on 2-way table and the results are promising, which sheds light on the potential of Gröbner basis approaches for integer feasibility problems.

1.3. Geometry in Biological Image Segmentation

Our third project combines ideas from topology and machine learning. Topological data analysis (TDA) [16] is an emerging field which emphasizes the correctness of the shape of data using methods from algebraic topology and computational geometry. Although some ideas of TDA can be traced back to 90s, it does not receive enough attention from ML community until very recently. It aims to analyze the underlying topological features of complex data sets using tools from topology. The key idea behind TDA is to represent data sets as topological spaces, typically in the form of simplicial complexes. By analyzing topological features of these spaces, TDA can reveal important features such as connected components, loops, voids, and other high dimensional structures that may exist in the data.

Persistence homology (PH) is a powerful method in TDA. PH identifies the most salient features of the data set by using the technique called filtration where a threshold varies from a maximum to a minimum value. Topological features of the data set emerge and die according to different values of the threshold.

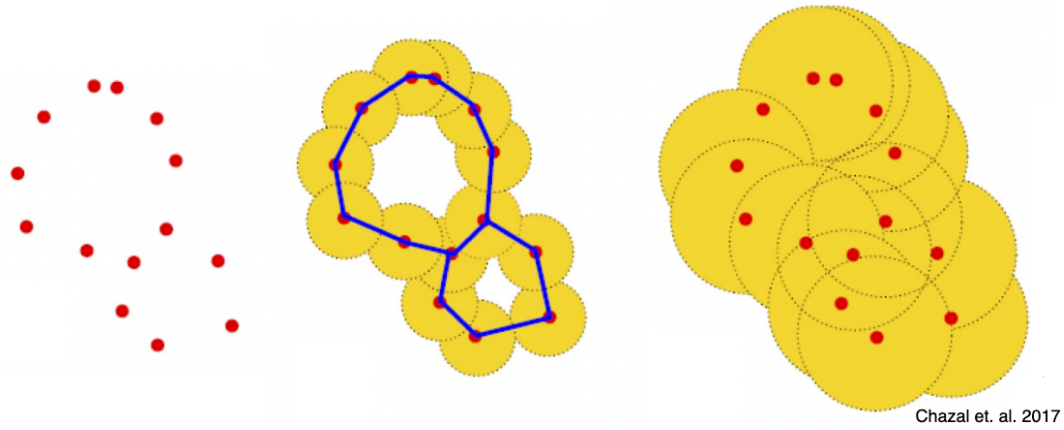


FIGURE 1.3. Filtration with radius on 2-D data points. (Left) Data points are individual connected components. (Middle) Two cycles (blue lines) emerge as the radius increases. (Right) Cycles disappear as the radius keeps increasing.

An illustration of a filtration is shown in Fig. 1.3. The set of points are initially disconnected. We consider that there is a circle centered at each point with the initial radius being 0 (left). The threshold variable used here is the radius of the circle. Then, we increase the radius, and two loops emerge (middle) when each circle overlaps its two adjacent circles. If we further increase the radius,

each loop will disappear when all circles in the loop overlap. Eventually, both loops disappear when the radius is sufficiently large.

The loops are the topological features of the data set emerged and died during the filtration. The number of connected components is also a feature. Initially, each data point is a component, then they get merged when the radius increases. However, the loops are more interesting features as it reveals high level structures of the data set. The difference of the radius value between the death and birth of a structure is called lifespan, and the goal of PH is to find the "persistent" structures in the data set by filtering out structures with short lifespans.

Recently, researchers have started working on developing new machine learning algorithms that generate topologically correct predictions. In image segmentation, topological correctness can be incorporated into the loss function [47, 17]. One can calculate the persistence diagram of the predicted segmentation and groundtruth, and use an appropriate distance metric (i.e., Wasserstein distance) as the loss function. The topological loss can also be used for point cloud segmentation and surface reconstruction [56, 35]. Similar idea is also used in generative models where the generator can learn to generate images that have the correct topology [97]. A more general topological layer is proposed in [36], which suggests that the idea of enforcing topological correctness can be seamlessly incorporated into many machine learning tasks.

In the context of microglia segmentation, we are especially interested in segmenting microglia by identifying components that are persistent using PH. We demonstrate a novel image segmentation approach using persistence homology as part of a workflow to identify and segment microglia as a complex cell type. In brief, persistence homology is a tool in topological data analysis, a field in mathematics that seeks to represent discrete elements in a dataset as a spatial relationship and to understand the relationships between those elements [14, 13]. Persistence homology identifies the most salient features of a data set by using filtration, a technique where a threshold varies from a maximum to a minimum value. Topological features of the dataset "emerge" and "die" depending on the threshold values, with the lifespan being the value of the threshold between the death and birth of a structure. This information is used in persistence homology to find the "persistent" structures in the data set by filtering out structures with short lifespans. This novel approach is capable of identifying complex cell morphologies such as microglia in images representing large physical volumes. Our implementation, that we have named *PrestoCell* can perform segmentation

as well as existing tools, with the added benefit of allowing the user to easily edit the cell masks. We further demonstrate that as PrestoCell makes use of nuclear markers to identify cells, it can accurately segment cells that are physically interacting further increasing accuracy. Finally, we show that PrestoCell can assist users in the segmentation of microglia from large volumes of brain tissue acquired by light sheet microscopy. These features and benefits of PrestoCell are available across computer platforms and require no commercial software.

CHAPTER 2

Geometric Policy Iteration for Markov Decision Processes

2.1. Preliminaries

An MDP has five components $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are finite state set and action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function with $\Delta(\cdot)$ denoting the probability simplex. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and $\gamma = [0, 1)$ is the discount factor that represents the value of time.

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping from states to distributions over actions. The goal is to find a policy that maximizes the cumulative sum of rewards.

Define $V^\pi \in \mathbb{R}^{|\mathcal{S}|}$ as the vector of state values. $V^\pi(s)$ is then the expected cumulative reward starting from a particular state s and acting according to π:

$$V^\pi(s) = \mathbb{E}_{P^\pi} \Big(\sum_{i=0}^{\infty} \gamma^i \mathcal{R}(s_i, a_i) \,|\, s_0 = s \Big).$$

The **Bellman equation** [3] connects the value V^π at a state s with the value at the subsequent states when following π:

$$V^\pi(s) = \mathbb{E}_{P^\pi} \Big(\mathcal{R}(s, a) + \gamma V^\pi(s') \Big). \tag{2.1}$$

Define r^π and P^π as follows.

$$r^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \,|\, s) \mathcal{R}(s, a),$$
$$P^\pi(s' \,|\, s) = \sum_{a \in \mathcal{A}} \pi(a \,|\, s) \mathcal{P}(s' \,|\, s, a),$$

Then, the Bellman equation for a policy π can be expressed in matrix form as follows.

$$\begin{aligned} V^\pi &= r^\pi + \gamma P^\pi V^\pi \\ &= (I - \gamma P^\pi)^{-1} r^\pi. \end{aligned} \tag{2.2}$$

Under this notation, we can define the Bellman operator $\mathcal{T}^\pi$ and the optimality Bellman operator $\mathcal{T}^*$ for an arbitrary value vector V as follows.

$$\mathcal{T}^\pi V = r^\pi + \gamma P^\pi V,$$

$$\mathcal{T}^* V = \max_\pi \mathcal{T}^\pi V.$$

V is optimal if and only if $V = \mathcal{T}^* V$. MDPs can be solved by **value iteration** (VI) [3] which consists of the repeated application of the optimality Bellman operator $V^{(k+1)} := \mathcal{T}^* V^{(k)}$ until a fixed point has been reached.

Let $\mathcal{P}(\mathcal{A})^\mathcal{S}$ denote the space of all policies, and $\mathcal{V}$ denote the space of all state values. We define the **value function** $f_v(\pi) : \mathcal{P}(\mathcal{A})^\mathcal{S} \to \mathcal{V}$ as

$$f_v(\pi) = (I - \gamma P^\pi)^{-1} r^\pi. \tag{2.3}$$

The value function f_v is fundamental to many algorithmic solutions of an MDP. **Policy iteration** (PI) [46] repeatedly alternates between a policy evaluation step and a policy improvement step until convergence. In the policy evaluation step, the state values V^π of the current policy π is evaluated which involves solving a linear system (Eq. (2.2)). In the policy improvement step, PI iterates over all states and update the policy by taking a greedy step using the optimality Bellman operator as follows.

$$\pi'(s) \in \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a) V^\pi(s') \right\}, \forall s \in \mathcal{S}.$$

Simple policy iteration (SPI) is a variant of policy iteration. It only differs from policy iteration in the policy improvement step where the policy is only updated for the state-action pair with the largest improvement over the following advantage function.

$$\tilde{A}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a) V^\pi(s') - V^\pi(s).$$

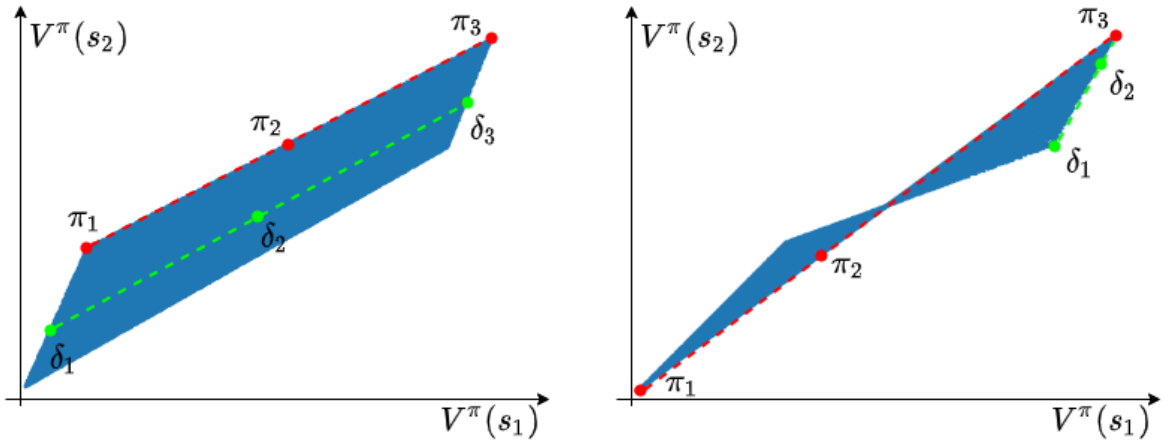
SPI selects a state-action pair from $\operatorname{argmax}_{s,a} \tilde{A}(s,a)$ then updates the policy accordingly.

2.1.1. Geometry of the Value Function. While the space of policies $\mathcal{P}(\mathcal{A})^\mathcal{S}$ is the Cartesian product of $|\mathcal{S}|$ probability simplices, [22] proved that the value function space is a possibly non-convex polytope [106]. Figure 2.1 shows a convex and a non-convex f_v polytopes of 2 MDPs in blue

regions. The proof is built upon the line theorem which is an equally important geometric property of the value space. The line theorem depends on the following definition of policy determinism.

DEFINITION 2.1 (Policy Determinism). *A policy π is*

- s-deterministic *for $s \in \mathcal{S}$ if it selects one concrete action for sure in state s, i.e., $\pi(a|s) \in \{0, 1\}$, $\forall a$;*
- deterministic *if it is s-deterministic for all $s \in \mathcal{S}$.*



(a) (b)

FIGURE 2.1. The blue regions are the value spaces of 2 MDPs with $|\mathcal{S}| = 2$ and $|\mathcal{A}| = 2$. The regions are obtained by plotting f_v of 50,000 random policies. (a): Both $\{\pi_i\}$ and $\{\delta_i\}$ agree on s_1 but differ in s_2. π_1 and π_3 are deterministic. π_2 is s_1-deterministic. δ_1 and δ_3 are s_2-deterministic. (b): $\{\pi_i\}$ and $\{\delta_i\}$ agree on s_1 and s_2, respectively. π_1, π_3, and δ_1 are deterministic while π_2 and δ_2 are s_1 and s_2-deterministic, respectively.

The line theorem captures the geometric property of a set of policies that differ in only one state. Specifically, we say two policies π_1, π_2 *agree* on states $s_1, .., s_k \in \mathcal{S}$ if $\pi_1(\cdot \,|\, s_i) = \pi_2(\cdot \,|\, s_i)$ for each s_i, $i = 1, \ldots, k$. For a given policy π, we denote by $Y^\pi_{s_1,\ldots,s_k} \subseteq \mathcal{P}(\mathcal{A})^\mathcal{S}$ the set of policies that agree with π on $s_1, \ldots, s_k$; we will also write $Y^\pi_{\mathcal{S}\setminus\{s\}}$ to describe the set of policies that agree with π on all states except s. When we keep the probabilities fixed at all but state s, the functional f_v draws a line segment which is oriented in the positive orthant (that is, one end dominates the other). Furthermore, the endpoints of this line segment are s-deterministic policies.

The line theorem is stated as follows:

THEOREM 2.2 (Line theorem [22]). *Let s be a state and π a policy. Then there are two s-deterministic policies in $Y^{\pi}_{\mathcal{S}\setminus\{s\}}$, denoted π_l, π_u, which bracket the value of all other policies $\pi' \in Y^{\pi}_{\mathcal{S}\setminus\{s\}}$:*

$$f_v(\pi_l) \preccurlyeq f_v(\pi') \preccurlyeq f_v(\pi_u).$$

For both Figure 2.1a and 2.1b, we plot policies that agree on one state to illustrate the line theorem. The policy determinism decides if policies are mapped to a vertex, onto the boundary or inside the polytope.

2.2. The Cell Structure of the Value Function Polytope

In this section, we revisit the geometry of the (non-convex) value function polytope presented in [22]. We establish a connection to linear programming formulations of the MDP which then can be adapted to show a finer description of cells in the value function polytope as unions of cells of a hyperplane arrangement. For more on hyperplane arrangements and their structure, see [87].

It is known since at least the 1990's that finding the optimal value function of an MDP can be formulated as a linear program (see for example [70, 10]). In the primal form, the feasible constraints are defined by $\{V \in \mathbb{R}^{|\mathcal{S}|} \mid V \succcurlyeq \mathcal{T}^* V\}$, where $\mathcal{T}^*$ is the optimality Bellman operator. Concretely, the following linear program is well-known to be equivalent to maximizing the expected total reward in Eq. (2.1). We call this convex polyhedron the **MDP-LP polytope** (because it is a linear programming form of the MDP problem).

$$\begin{aligned} \min_{V} \quad & \sum_{s} \alpha(s) V(s) \\ \text{s.t.} \quad & V(s) \geq \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a) V(s'), \ \forall s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned}$$

where α is a probability distribution over $\mathcal{S}$.

Our main new observation is that the MDP-LP polytope and the value polytope are actually closely related, and one can describe the regions of the (non-convex) value function polytope in terms of the (convex) cells of the arrangement.

THEOREM 2.3. *Consider the hyperplane arrangement H_{MDP}, with $|\mathcal{A}||\mathcal{S}|$ hyperplanes, consisting of those of the MDP polytope, i.e.,*

$$H_{MDP} = \left\{ V(s) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) V(s') \mid \forall s \in \mathcal{S}, a \in \mathcal{A} \right\}.$$

Then, the boundary of the value function polytope $\partial\mathcal{V}$ is the union of finitely (convex polyhedral) cells of the arrangement H_{MDP}. Moreover, each full-dimensional cell of the value polytope is contained in the union of finitely many full-dimensional cells defined by H_{MDP}.

PROOF. Let us first consider a point V^π being on the boundary of the value function polytope. Theorem 2 and Corollary 3 of [22] demonstrated that the boundary of the space of value functions is a (possibly proper) subset of the ensemble of value functions of policies, where at least one state has a fixed deterministic choice for all actions. Note that from the value function Eq. (2.3), then the hyperplane

$$V(s) = \mathcal{R}(s, a_l^s) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a_l^s) V(s')$$

includes all policies taking policy $a_l^s = \pi_l(s)$ in state s. Thus the points of the boundary of the value function polytope are contained in the hyperplanes of H_{MDP}. Now we can see how the k-dimensional cells of the boundary are then in the intersections of the hyperplanes too.

The zero-dimensional cells (vertices) are clearly a subset of the zero-dimensional cells of the arrangement H_{MDP} because, by above results, the zero-dimensional cells are precisely in the intersection of $|\mathcal{S}|$ many hyperplanes from H_{MDP}, which is equivalent to choosing a fixed set of actions for all states. This corresponds to solving a linear system consisting of the hyperplanes that bound $\mathcal{V}$ (same as Eq. (2.2)). But more generally, if we fix the policies for only k states, the induced space lies in a $|\mathcal{S}| - k$ dimensional affine space. Consider a policy π and k states $s_1, \ldots, s_k$, and write $C_{k+1}^\pi, \ldots, C_{|\mathcal{S}|}^\pi$ for the columns of the matrix $(I - \gamma P^\pi)^{-1}$ corresponding to states *other* than $s_1, \ldots, s_k$. Define the affine vector space $H_{s_1,..,s_k}^\pi$

$$H_{s_1,\ldots,s_k}^\pi = V^\pi + Span(C_{k+1}^\pi, \ldots, C_{|\mathcal{S}|}^\pi).$$

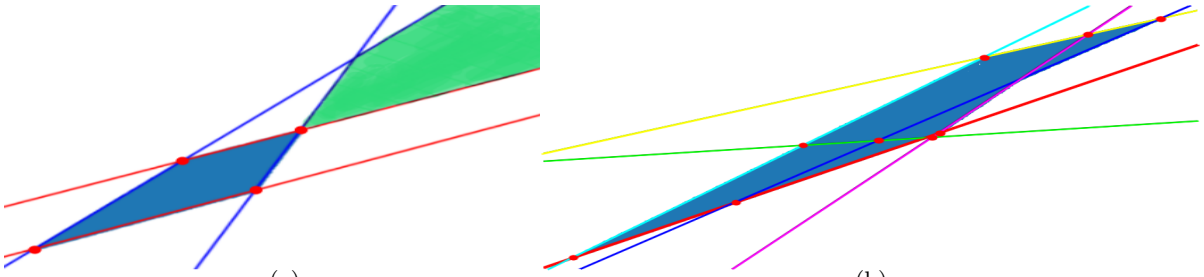
Now For a given policy π, we denote by $Y_{s_1,\ldots,s_k}^\pi \subseteq \mathcal{P}(\mathcal{A})^\mathcal{S}$ the set of policies which agree with π on $s_1, \ldots, s_k$; Thus the value functions generated by $Y_{s_1,..,s_k}^\pi$ are contained in the affine vector space $H_{s_1,\ldots,s_k}^\pi$: $f_v(Y_{s_1,..,s_k}^\pi) = \mathcal{V} \cap H_{s_1,..,s_k}^\pi$.

The points of $H^{\pi}_{s_1,\ldots,s_k}$ in one or more of the H_{MDP} planes (each hyperplane is precisely fixing one policy action pair). This is the intersection of k hyperplanes given by the following equations.

$$\left\{ V(s) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) V(s') \mid \forall s \in \{s_1, \ldots, s_k\}, a \in \mathcal{A} \right\}.$$

Thus we can be sure of the stated containment.

Finally, the only remaining case is when V^{π} is in the interior of the value polytope. If that is the case, because H_{MDP} partitions the entire Euclidean space, it must be contained in at least one of the full-dimensional cell of H_{MDP}. □



(a) (b)

FIGURE 2.2. (a): f_v polytope (blue) and MDP-LP polytope (green) of an MDP with $|\mathcal{S}| = 2$ and $|\mathcal{A}| = 2$. (b) f_v polytope overlapped with the hyperplane arrangement H_{MDP} from Theorem 2.3. This MDP has 3 actions so $|H_{MDP}| = 6$.

Figure 2.2a is an example of the value function polytope in blue, MDP-LP polytope in green and its bounding hyperplanes (the arrangement H_{MDP}) as blue and red lines. In Figure 2.2b we exemplify Theorem 2.3 by presenting a value function polytope with delimited boundaries where H_{MDP} hyperplanes are indicated in different colors. The deterministic policies are those for which $\pi(a|s) \in \{0,1\}\, \forall a \in \mathcal{A}, s \in \mathcal{S}$. In both pictures, the values of deterministic policies in the value space are shown as red dots. The boundaries of the value polytope are indeed included in the set of cells of the arrangement H_{MDP} as stated by Theorem 2.3. These figures of value function polytopes (blue regions) were obtained by randomly sampling policies and plotting their corresponding state values.

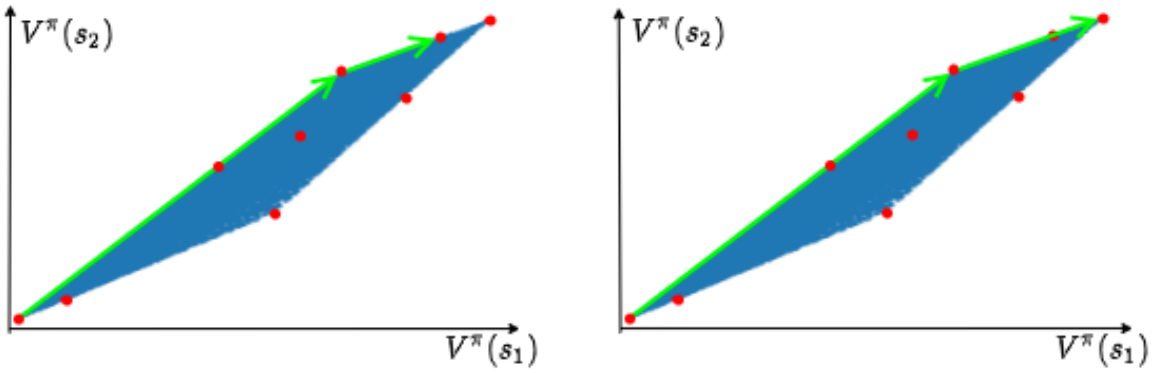
Some remarks are in order. Note how sometimes the several adjacent cells of the MDP arrangement together form a connected cell of the value function polytope. We also observe that for any set of states $s_1, .., s_k \in \mathcal{S}$ and a policy π, V^{π} can be expressed as a convex combination of value functions of $\{s_1, .., s_k\}$-deterministic policies. In particular, $\mathcal{V}$ is included in the convex hull of the value

functions of deterministic policies. It is also demonstrated clearly in Figure 2.2b that the value functions of deterministic policies are not always vertices and the vertices of the value polytope are not always value functions of deterministic policies, but they are always intersections of hyperplanes on H_{MDP}. However, optimal values will always include a deterministic vertex. This observation suggests that it would suffice to find the optimal policy by only visiting deterministic policies on the boundary. It is worthwhile to note that the optimal value of our MDP would be at the unique intersection vertex of the two polytopes. We note that the blue regions in Figure 2.2a are *not* related to the polytope of the dual formulation of LP. Unlike the MDP polytope which can be characterized as the intersection of finitely many half-spaces, we do not have such a neat representation for the value function polytope. The pictures presented here and many more experiments we have done suggest the following stronger result is true:

CONJECTURE 2.1. *if the value polytope intersects a cell of the arrangement H_{MDP}, then it contains the entire cell, thus all full-dimensional cells of the value function polytope are equal to the union of full-dimensional cells of the arrangement.*

Proving this conjecture requires showing that the map from policies to value functions is surjective over the cells it touches. At the moment we can only guarantee that there are no isolated components because the value polytope is a compact set. More strongly [22] shown (using the line theorem) that there is path connectivity from V^{π}, in any cell, to others is guaranteed by a polygonal path. More precisely if we let V^{π} and $V^{\pi'}$ be two value functions. Then there exists a sequence of $k \leq |\mathcal{S}|$ policies, $\pi_1, \ldots, \pi_k$, such that $V^{\pi} = V^{\pi_1}$, $V^{\pi'} = V^{\pi_k}$, and for every $i \in 1, \ldots, k-1$, the set $\{f_v(\alpha\pi_i + (1-\alpha)\pi_{i+1}) \,|\, \alpha \in [0,1]\}$ forms a line segment.

It was observed that algorithms for solving MDPs have different learning behavior when visualized in the value polytope space. For example, policy gradient methods [92, 48, 50, 101, 100] have an improvement path inside of the value function polytope; value iteration can go outside of the polytope which means there can be no corresponding policy during the update process; and policy iteration navigates exactly through deterministic policies. In the rest of this chapter we use this geometric intuition to design a new algorithm.



(a) (b)

FIGURE 2.3. The value sequences of one iteration which involves a sweep over all states looking for policy updates. (a): In PI, we may not reach the end of a line segment for an action switch. (b): An endpoint is always reached in GPI.

2.3. The Method of Geometric Policy Iteration

We now present *geometric policy iteration* (GPI) that improves over PI based on the geometric properties of the learning dynamics. Define an *action switch* to be an update of policy π in any state $s \in \mathcal{S}$. The Line theorem shows that policies agreeing on all but one state lie on a line segment. So an action switch is a move along a line segment to improve the value function. In PI, we use the optimality Bellman operator $\mathcal{T}^*V(s) = \max_\pi(r^\pi + \gamma P^\pi V)(s)$ to decide the action to switch to for state s. However, $\mathcal{T}^*V(s)$ does not guarantee the largest value improvement $V^*(s) - V(s)$ for s. This phenomenon is illustrated in Figure 2.3 where we plot the value sequences of PI and the proposed GPI.

We propose an alternative action-switch strategy in GPI that directly calculates the improvement of the value function for one state. By choosing the action with the largest value improvement, we can always reach the endpoint of a line segment which potentially reduces the number of action switches.

This strategy requires efficient computation of the value function because a naive calculation of the value function by Eq. (2.2) is very expensive due to the matrix inversion. On the other hand, PI only re-evaluates the value function once per iteration. Our next theorem states that the new state-value can be efficiently computed. This is achieved by using the fact that the policy improvement step can be done state-by-state within a sweep over the state set, so adjacent policies in the update sequence only differ in one state.

THEOREM 2.4. *Given* $\mathbf{Q}^\pi = (I - \gamma P^\pi)^{-1}$ *and* $V^\pi = \mathbf{Q}^\pi r^\pi$. *If a new policy* δ *only differs from* π *in state* s *with* $\delta(s) = a \neq \pi(s)$, $V^\delta(s)$ *can be calculated efficiently by*

$$V^\delta(s) = \left(\mathbf{1}_s + \frac{\mathbf{Q}^\pi(s,s)}{1 - \mathbf{w}_a^\top \mathbf{q}_s} \mathbf{w}_a\right)^\top (V^\pi + \Delta r_a \, \mathbf{q}_s)\,, \tag{2.4}$$

where $\mathbf{w}_a = \gamma\left(\mathcal{P}(s,a) - \mathcal{P}(s,\pi(s))\right)$ *is a* $|\mathcal{S}|$*-d vector,* $\Delta r_a = \mathcal{R}(s,a) - \mathcal{R}(s,\pi(s))$ *is a scalar,* $\mathbf{q}_s$ *is the* s^{th} *column of* $\mathbf{Q}^\pi$, *and* $\mathbf{1}_s$ *is a vector with entry* s *being* 1, *others being* 0.

PROOF. We here provide a general proof that we can calculate V^δ given policy π, V^π, and δ differs from π in only one state.

$$V^\delta = \left(I - \gamma P^\delta\right)^{-1} r^\delta = (I - \gamma P^\pi - \gamma \Delta P)^{-1} r^\delta,$$

where $\Delta P = P^\delta - P^\pi$. Assume δ and π differ in state s. ΔP is a rank-1 matrix with row j being $\mathcal{P}(s,\pi(s)) - \mathcal{P}(s,a)$, and all other rows being zero vectors.

We can then express ΔP as the outer product of two vectors $\Delta P = \mathbf{1}_s \mathbf{w}_a^\top$, where $\mathbf{1}_s$ is a one-hot vector

$$\mathbf{1}_s(i) = \begin{cases} 1, & \text{if } i = s, \\ 0, & \text{otherwise,} \end{cases} \tag{2.5}$$

and $\mathbf{w}_a$ is defined above.

Similarly, we have $r^\delta = r^\pi + \Delta r = r^\pi + \Delta r_a \mathbf{1}_s$. Given Sherman-Morrison

Then, we have

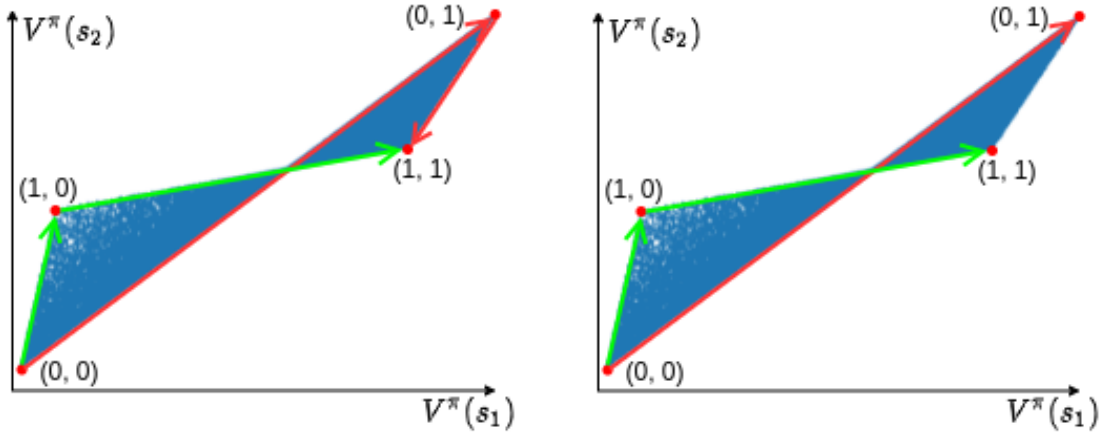
$$\begin{aligned} V^\delta &= \left(I - \gamma P^\delta\right)^{-1} r^\delta \\ &= (I - \gamma P^\pi - \gamma \Delta P)^{-1} (r^\pi + \Delta r) \\ &= \left(I - \gamma P^\pi - \mathbf{1}_s \mathbf{w}_a^\top\right)^{-1} (r^\pi + \Delta r_a \, \mathbf{1}_s) \\ &= \left(\mathbf{Q}^\pi + \frac{\mathbf{Q}^\pi \mathbf{1}_s \mathbf{w}_a^\top \mathbf{Q}^\pi}{1 - \mathbf{w}_a^\top \mathbf{Q}^\pi \mathbf{1}_s}\right) (r^\pi + \Delta r_a \, \mathbf{1}_s) \\ &= \left(I + \frac{\mathbf{q}_s \mathbf{w}_a^\top}{1 - \mathbf{w}_a^\top \mathbf{q}_s}\right) (V^\pi + \Delta r_a \, \mathbf{q}_s)\,. \end{aligned}$$

Thus, for state s, we have

$$V^{\delta}(s) = \left(\mathbf{1}_s + \frac{\mathbf{Q}^{\pi}(s,s)}{1 - \mathbf{w}_a^{\top}\mathbf{q}_s}\mathbf{w}_a\right)(V^{\pi} + \Delta r_a\,\mathbf{q}_s)\,,$$

which completes the proof. □

Theorem 2.4 suggests that updating the value of a single state using Eq. (2.4) takes $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ arithmetic operations which matches the complexity of the optimality Bellman operator used in policy iteration.



(a) (b)

FIGURE 2.4. Two paths are shown for each PI, GPI. The green and red paths denote one iteration with $\pi(s_1)$ and $\pi(s_2)$ updated first, respectively. (a): The policy improvement path of PI. The red path is not action-switch-monotone which will lead to an additional iteration. (b): GPI is always action-switch-monotone. The red path achieves the optimal values in one action switch.

The second improvement over policy iteration comes from the fact that the value improvement path in $\mathcal{V}$ may not be monotonic with respect to action switches. Although it is well-known that the update sequence $\{V^{\pi^{(k)}}\}$ is non-decreasing in the iteration number k, the value function could decrease in the policy improvement step of the policy iteration. An illustration is shown as the red path of PI in Figure 2.4. The possible value decrease is because when the Bellman operator $\mathcal{T}$ is used to decide an action switch, V is fixed for the entire sweep of states. This leads us to the motivation for GPI which is to update the value function after each action switch such that the value function action-switch-monotone. This idea can be seamlessly combined with Theorem 2.4 since the values of all states can be updated efficiently in $\mathcal{O}\left(|\mathcal{S}|^2|\mathcal{A}|\right)$ arithmetic operations. Thus, the complexity of completing one iteration is the same as policy iteration.

Algorithm 1 Geometric Policy Iteration

Input: $\mathcal{P}$, $\mathcal{R}$, γ

1: set iteration number $k = 0$ and randomly initialize $\pi_0^{(k)}$
2: Calculate $\mathbf{Q}_0^{(k)} = (I - \gamma P_0^{(k)})^{-1}$ and $V_0^{(k)} = \mathbf{Q}_0^{(k)} r_0^{(k)}$
3: **for** $i = 1, \ldots, |\mathcal{S}|$ **do**
4: calculate the best action a_i according to Eq. (2.6)
5: update $\mathbf{Q}_i^{(k)}$ according to Eq. (2.9)
6: $\pi_i^{(k)}(i) = a_i$
7: $V_i^{(k)} = \mathbf{Q}_i^{(k)} r_i^{(k)}$
8: **if** $V_{|\mathcal{S}|}^{(k)}$ is optimal **then return** $\pi_{|\mathcal{S}|}^{(k)}$
9: $\mathbf{Q}_0^{(k+1)} = \mathbf{Q}_{|\mathcal{S}|}^{(k)}$, $\pi_0^{(k+1)} = \pi_{|\mathcal{S}|}^{(k)}$, $V_0^{(k+1)} = V_{|\mathcal{S}|}^{(k)}$, $k = k + 1$. Go to step 3

We summarize GPI in Algorithm 1. GPI looks for action switches for all states in one iteration, and updates the value function after each action switch. Let superscript k denote the iteration index, subscript i denote the state index in one iteration. To avoid clutter, we use i to denote the state s_i being updated and drop superscript π in P^π and r^π. Step 2 evaluates the initial policy $\pi_0^{(k)}$. The difference here is that we store the intermediate matrix $\mathbf{Q}_0^{(k)}$ for later computation. From step 3 to step 7, we iterate over all states to search for potential updates. In step 4, GPI selects the best action by computing the new state-value of each potential action switch by Eq. (2.6).

$$a_i \in \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \left(\mathbf{1}_i + \frac{\mathbf{Q}_{i-1}^{(k)}(i,i)}{1 - \mathbf{w}_a^\top \mathbf{q}_i} \mathbf{w}_a \right)^\top \left(V_{i-1}^{(k)} + \Delta r_a \, \mathbf{q}_i \right) \right\}, \tag{2.6}$$

where

$$\mathbf{w}_a = \gamma \left(\mathcal{P}(i,a) - \mathcal{P}(i, \pi_{i-1}^{(k)}(i)) \right), \tag{2.7}$$

$$\Delta r_a = \left(\mathcal{R}(i,a) - \mathcal{R}(i, \pi_{i-1}^{(k)}(i)) \right), \tag{2.8}$$

and $\mathbf{1}_i$ is a vector with i^{th} entry being 1 and others being 0.

Define $\mathbf{q}_i$ to be the i^{th} column of $\mathbf{Q}_{i-1}^{(k)}$. $\mathbf{w}_i^\top$ is obtained by Eq. (2.7) using the selected action a_i. In step 5, we update $\mathbf{Q}_i^{(k)}$ as follows.

$$\mathbf{Q}_i^{(k)} = \mathbf{Q}_{i-1}^{(k)} + \frac{\mathbf{q}_i \, \mathbf{w}_i^\top \mathbf{Q}_{i-1}^{(k)}}{1 - \mathbf{w}_i^\top \mathbf{q}_i}. \tag{2.9}$$

The policy is updated in step 6 and the value vector is updated in step 7 where $r_i^{(k)}$ is the reward vector under the new policy. The algorithm is terminated when the optimal values are achieved.

2.3.1. Theoretical Guarantees. Before we present any properties of GPI, let us first prove the following very useful lemma.

LEMMA 2.4.1. *Given two policies π and π', we have the following equalities.*

$$V^{\pi'} - V^{\pi} = \left(I - \gamma P^{\pi'}\right)^{-1} \left(r^{\pi'} + \gamma P^{\pi'} V^{\pi} - V^{\pi}\right), \tag{2.10}$$

$$V^{\pi'} - V^{\pi} = (I - \gamma P^{\pi})^{-1} \left(V^{\pi'} - r^{\pi} - \gamma P^{\pi} V^{\pi'}\right). \tag{2.11}$$

PROOF. Using Bellman equation, we have

$$V^{\pi'} - V^{\pi} = r^{\pi'} + \gamma P^{\pi'} V^{\pi'} - r^{\pi} - \gamma P^{\pi} V^{\pi}. \tag{2.12}$$

Eq. (2.12) can be rearranged as

$$V^{\pi'} - V^{\pi} = r^{\pi'} - r^{\pi} + \gamma P^{\pi'} \left(V^{\pi'} - V^{\pi}\right) + \gamma \left(P^{\pi'} - P^{\pi}\right) V^{\pi},$$

and Eq. (2.10) follows.

To get Eq. (2.11), we rearrange Eq. (2.12) as

$$V^{\pi'} - V^{\pi} = r^{\pi'} - r^{\pi} + \gamma P^{\pi} \left(V^{\pi'} - V^{\pi}\right) + \gamma \left(P^{\pi'} - P^{\pi}\right) V^{\pi'},$$

and Eq. (2.11) follows. □

Our first result is an immediate consequence of re-evaluating the value function after an action switch.

PROPOSITION 2.1. *The value function is non-decreasing with respect to action switches in GPI, i.e., $V_{i+1}^{(k)} \geq V_i^{(k)}$.*

PROOF. From Eq. (2.10) in Lemma 2.4.1, we have

$$\left(I - \gamma P^{\pi'}\right) \left(V^{\pi'} - V^{\pi}\right) = r^{\pi'} + \gamma P^{\pi'} V^{\pi} - V^{\pi}.$$

Since $P^{\pi} \geq 0$, we have

$$V^{\pi'} - V^{\pi} \geq r^{\pi'} + \gamma P^{\pi'} V^{\pi} - V^{\pi} = \mathcal{T}^{\pi'} V^{\pi} - V^{\pi},$$

which implies that for any π', π,

$$V^{\pi'} \geq \mathcal{T}^{\pi'} V^{\pi}. \tag{2.13}$$

Now, consider $\pi_{i+1}^{(k)}$ and $\pi_i^{(k)}$. According to the updating rule of GPI, for state i we have $V_{i+1}^{(k)}(i) \geq V_i^{(k)}(i)$. For state $j \neq i$, we have

$$V_{i+1}^{(k)}(j) \geq \mathcal{T}_{i+1}^{(k)} V_i^{(k)}(j) = \mathcal{T}_i^{(k)} V_i^{(k)}(j) = V_i^{(k)}(j).$$

Combined, we have $V_{i+1}^{(k)} \geq V_i^{(k)}$, which completes the proof. □

We next turn to the complexity of GPI and bound the number of iterations required to find the optimal solution. The analysis depends on the lemma described as follows.

LEMMA 2.4.2. *Let V^* denote the optimal value. At iteration k of GPI, we have the following inequality.*

$$\left(V^* - V_i^{(k)}\right)(i) \leq \gamma P^* \left(V^* - V_i^{(k-1)}\right)(i).$$

PROOF. From Bellman equation, we have

$$\begin{aligned} V^* - V_i^{(k)} &= \mathcal{T}^* V^* - V_i^{(k)} \\ &= \mathcal{T}^* V^* - \mathcal{T}^* V_i^{(k-1)} + \mathcal{T}^* V_i^{(k-1)} - V_i^{(k)}. \end{aligned}$$

For state i, we have

$$\begin{aligned} &\left(V^* - V_i^{(k)}\right)(i) \\ &= \gamma P^* \left(V^* - V_i^{(k-1)}\right)(i) + \left(\mathcal{T}^* V_i^{(k-1)} - V_i^{(k)}\right)(i) \\ &\leq \gamma P^* \left(V^* - V_i^{(k-1)}\right)(i) + \left(\max_{\pi} \mathcal{T}^{\pi} V_i^{(k-1)} - V_i^{(k)}\right)(i) \\ &\leq \gamma P^* \left(V^* - V_i^{(k-1)}\right)(i) + \left(V_i^{(k)} - V_i^{(k)}\right)(i) \qquad (2.15) \\ &= \gamma P^* \left(V^* - V_i^{(k-1)}\right)(i). \end{aligned} \tag{2.14}$$

Let $\pi' = \text{argmax}_\pi \mathcal{T}^\pi V_i^{(k-1)}$. The inequality (2.15) is because of the updating rule of GPI and (2.13),

$$V_i^{(k)} \geq V_i^{\pi'} \geq \mathcal{T}^{\pi'} V_i^{(k-1)},$$

which completes the proof. □

THEOREM 2.5. *GPI finds the optimal policy in* $\mathcal{O}\left(\frac{|\mathcal{A}|}{1-\gamma} \log \frac{1}{1-\gamma}\right)$ *iterations.*

PROOF. Define $\Delta_k^* \in \mathbb{R}^{|\mathcal{S}|}$ with $\Delta_k^*(s) = V^*(s) - V_i^{(k)}(s)$, $\forall s \in \mathcal{S}$. Then, by Lemma 2.4.2, we have

$$\Delta_k^* \leq \gamma P^* \Delta_{k-1}^*,$$
$$\|\Delta_k^*\|_\infty \leq \gamma^k \|\Delta_0^*\|_\infty .$$

Let j be the state such that $\Delta_0^*(j) = \|\Delta_0^*\|_\infty$, the following properties can be obtained by Eq. (2.11) in Lemma 2.4.1.

$$\begin{aligned}
\|\Delta_k^*\|_\infty &\leq \gamma^k \|\Delta_0^*\|_\infty \\
&\leq \gamma^k \left\| \left(I - \gamma P_j^{(0)}\right)^{-1} \right\|_\infty \left\| V^* - \mathcal{T}_j^{(0)} V^* \right\|_\infty \\
&= \frac{\gamma^k}{1-\gamma} \left(V^* - \mathcal{T}_j^{(0)} V^*\right)(j).
\end{aligned}$$

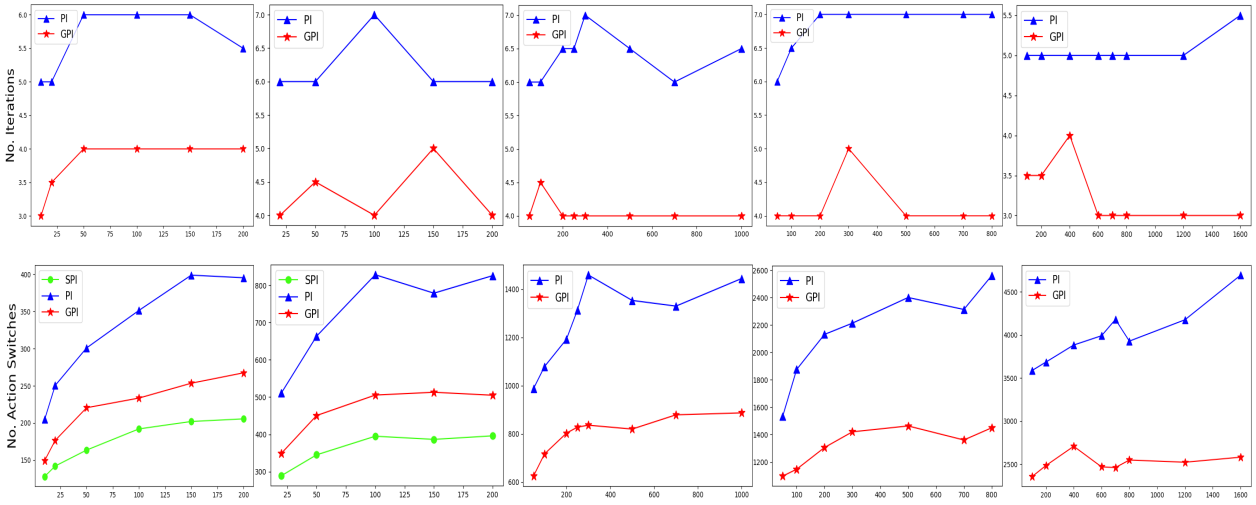
Also from Eq. (2.11), we have

$$\left(V^* - \mathcal{T}_j^{(k)} V^*\right)(j) \leq \Delta_k^*(j) \leq \|\Delta_k^*\|_\infty . \tag{2.16}$$

It follows that

$$\left(V^* - \mathcal{T}_j^{(k)} V^*\right)(j) \leq \frac{\gamma^k}{1-\gamma} \left(V^* - \mathcal{T}_j^{(0)} V^*\right)(j),$$

which implies when $\frac{\gamma^k}{1-\gamma} < 1$, the non-optimal action for j in $\pi^{(0)}$ is switched in $\pi^{(k)}$ and will never be switched back to in future iterations. Now we are ready to bound k. By taking the logarithm



for both sides of $\frac{\gamma^k}{1-\gamma} < 1$, we have

$$k \log \gamma \geq \log(1-\gamma)$$

$$k > \frac{\log(1-\gamma)}{\log \gamma} = \frac{\log \frac{1}{1-\gamma}}{\log \frac{1}{\gamma}}$$

$$k > \frac{1}{1-\gamma} \log \frac{1}{1-\gamma} \qquad \left(\log \frac{1}{\gamma} \geq \frac{\frac{1}{\gamma} - 1}{\frac{1}{\gamma}} = 1 - \gamma \right).$$

Each non-optimal action is eliminated after at most $\mathcal{O}\left(\frac{1}{1-\gamma} \log \frac{1}{1-\gamma}\right)$ iterations, and there are $\mathcal{O}(|\mathcal{A}|)$ non-optimal actions. Thus, GPI takes at most $\mathcal{O}\left(\frac{|\mathcal{A}|}{1-\gamma} \log \frac{1}{1-\gamma}\right)$ iterations to reach the optimal policy. □

2.3.2. Asynchronous Geometric Policy Iteration. When the state set is large it would be beneficial to perform policy updates in an orderless way [91]. This is because iterating over the entire state set may be prohibitive, and exactly evaluating the value function with Eq. (2.2) may be too expensive. Thus, in practice, the value function is often approximated when the state set is large. One example is modified policy iteration [71, 70] where the policy evaluation step is approximated with certain steps of value iteration.

Since GPI avoids the matrix inversion by updating the value function incrementally, it has the potential to update the policy for arbitrary states available to the agent. This property also opens up the possibility of asynchronous (orderless) updates of policies and the value function when the state set is large or the agent has to update the policy for the state it encounters in real-time.

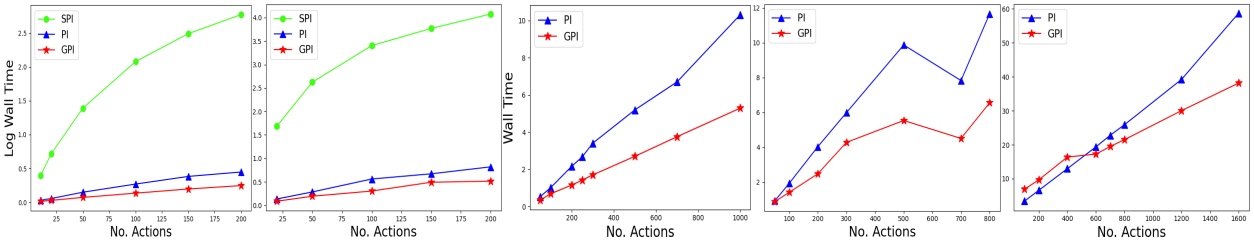


FIGURE 2.5. The results of MDPs with $\{100, 200, 300, 500, 1000\}$ states in (a)-(e). The horizontal axes are the number of actions for all graphs. The vertical axes are the number of iterations, number of action switches, and wall time for the first to the third row, respectively. The performance curves of SPI, PI, and GPI are in green, blue, and red, respectively. The SPI curves are only presented in (a) and (b) to provide a "lower bound" on the number of action switches, and are dropped for larger MDPs due to its higher running time. The number of switches of GPI remains low compared to PI. The proposed GPI consistently outperforms PI in both iteration count and wall time. The advantages of GPI become more significant as the action set size grows.

The asynchronous update strategy can also help avoid being stuck in states that lead to minimal progress and may reach the optimal policy without reaching a certain set of states.

Asynchronous GPI (Async-GPI) follows the action selection mechanism of GPI, and its general framework is as follows. Assume the transition matrix is available to the agent, we randomly initialize the policy $\pi^{(0)}$ and calculate the initial $\mathbf{Q}^{(0)}$ and $V^{(0)}$ accordingly. In real-time settings, the sequence of states $\{s_0, s_1, s_2, \ldots\}$ are collected by an agent through real-time interaction with the environment. At time step t, we search for an action switch for state s_t using Eq. (2.6). Then, we update the $\pi^{(t)}$, $\mathbf{Q}^{(t)}$ with Eq. (2.9), and $V^{(t)}$. Asynchronous value-based methods converge if each state is visited infinitely often [8]. We later demonstrate in experiments that Async-GPI converges well in practice.

2.4. Experiments

We test GPI on random MDPs of different sizes. The baselines are policy iteration (PI) and simple policy iteration (SPI). We compare the number of iterations, actions switches, and wall time. Here we denote the number of iterations as the number of sweeps over the entire state set. Action switches are those policy updates within each iteration. The results are shown in Figure 2.5. We generate MDPs with $|\mathcal{S}| = \{100, 200, 300, 500, 1000\}$ corresponding to Figure 2.5 (a)-(e). For each state size, we increase the number of actions (horizontal axes) to observe the difference in performance. The

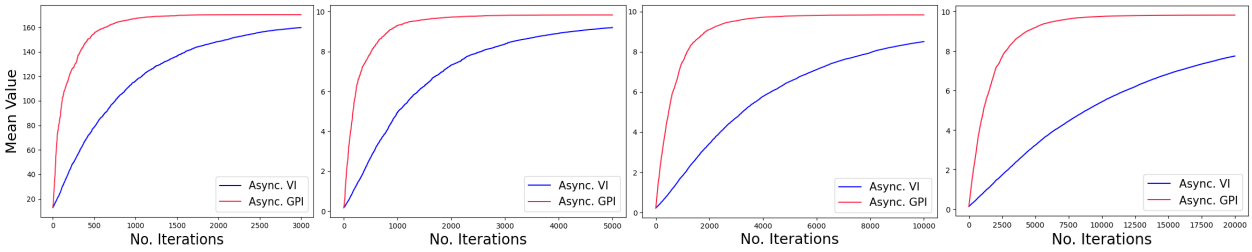


FIGURE 2.6. Comparison between asynchronous geometric policy iteration (red curve) and asynchronous value iteration (blue curve) in 4 MDPs. $|\mathcal{A}| = 100$ for all MDPs and $|\mathcal{S}| = \{300, 500, 1000, 2000\}$ for (a)-(d), respectively. The horizontal axes are the number of updates. The vertical axes show the mean of the value function.

rows from the top to bottom are the number of iterations, action switches and wall time (vertical axes), respectively. Since SPI only performs one action switch per iteration, we only show its number of action switches. The purpose of adding SPI to the baseline is to verify if our GPI can effectively reduce the number of action switches. Since SPI sweeps over the entire state set and updates a single state with the largest improvement, it is supposed to have the least number of action switches. However, SPI's larger complexity of performing one update should lead to higher running time. This is supported by the experiments as Figure 2.5 (a) and (b) show that SPI (green curves) takes the least number of switches and longest time. We drop SPI in Figure 2.5 (c)-(e) to have a clearer comparison between GPI and PI (especially in wall time). The proposed GPI has a clear advantage over PI in almost all tests. The second row of Figure 2.5 (a) and (b) shows that the number of action switches of GPI is significantly fewer than PI and very close to SPI although the complexity of a switch is cheaper by a factor of $|\mathcal{S}|$. And the reduction in the number of action switches leads to fewer iterations. Another important observation is that the margin increases as the action set becomes larger. This is strong empirical evidence that demonstrates the benefits of GPI's action selection strategy which is to reach the endpoints of line segments in the value function polytope. The larger the action set is, the more policies lying on the line segments and thus the more actions being excluded in one switch. The wall time of GPI is also very competitive compared to PI which further demonstrates that GPI can be a very practical algorithm for solving finite discounted MDPs.

We also test the performance of the asynchronous GPI (Async-GPI) on MDPs with $|\mathcal{S}| = \{300, 500, 1000, 2000\}$ and $|\mathcal{A}| = 100$. For each setting, we randomly generate a sequence of states that is larger than $|\mathcal{S}|$. We compare Async-GPI with asynchronous value iteration (Async-VI) which is classic asynchronous dynamic programming algorithm. At time step t, Async-VI performs one step of the optimality Bellman operator on a single state s_t that is available to the algorithm. The results are shown in Figure 2.6. The mean of the value function is plotted against the number of updates. We observe that Async-GPI took significantly fewer updates to reach the optimal value function. The gap becomes larger when the state set grows in size. These results are expected because Async-GPI also has a higher complexity to perform an update and Async-VI never really solves the real value function before reaching the optimality.

2.5. Conclusions and Future Work

We discussed the geometric properties of finite MDPs. We characterized the hyperplane arrangement that includes the boundary of the value function polytope, and further related it to the MDP-LP polytope by showing that they share the same hyperplane arrangement. Unlike the well-defined MDP-LP polytope, it remains unclear which bounding hyperplanes are active and which halfspaces of them belong to the value space. Besides the conjecture stated earlier, we would like to understand in the future which cells of the hyperplane arrangement form the value function polytope, and may derive a bound on the number of convex cells. It is also plausible that the rest of the hyperplane arrangement will help us devise new algorithms for solving MDPs.

Following the fact that policies that differ in only one state are mapped onto a line segment in the value function polytope, and that the only two policies on the polytope boundary are deterministic in that state, we proposed a new algorithm called geometric policy iteration that guarantees to reach an endpoint of the line segment for every action switch. We developed a mechanism that makes the value function monotonically increase with respect to action switches and the whole process can be computed efficiently. Our experiments showed that our algorithm is very competitive compared to the widely-used policy iteration and value iteration. We believe this type of algorithm can be extended to multi-agent settings, e.g., stochastic games [82]. It will also be interesting to apply similar ideas to model-based reinforcement learning.

CHAPTER 3

Integer Feasibility as a Game

3.1. Basic notions: polyhedra and Gröbner bases

We begin describing how testing integer feasibility is actually equivalent to playing a certain game on finite set of arrays inside a polytope. We begin with some notation: Let l,m and n be three positive integers. Let $x = (x_1, \ldots, x_l)$, $y = (y_1, \ldots, y_m)$, and $z = (z_1, \ldots, z_n)$ be three rational vectors of lengths l, m and n, respectively, with non-negative entries. We consider the 3-way transportation polytope $T_{x,y,z}$ with entries $a_{i,j,k}$ defined by 1-marginals,

$$a_{i,j,k} \geq 0,\ \forall i,j,k, \quad \sum_{j,k} a_{i,j,k} = x_i,\ \forall i$$

$$\sum_{i,k} a_{i,j,k} = y_j,\ \forall j, \quad \sum_{i,j} a_{i,j,k} = z_k,\ \forall k.$$

The *margins* are sums of the certain entries of an 3-way array of numbers. We begin with stating the first algorithmic step of this work:

THEOREM 3.1. *Any rational convex polyhedron can be written as a face F of some 3-way transportation polytope Q with 1-marginals x, y, z supported by a hyperplane of the form $\sum_{(i,j,k)\in S} a_{i,j,k} = 0$, where S is a finite subset of indices. The sizes l, m, n, the set S, and marginals x, y, z can be computed explicitly in polynomial time on the size of the input. Specifically, the face F is given by certain entries forced to be zero.*

This theorem is a particular case of a much more general theory further developed in [28, 29]. Given a convex polytope $P = \{x : Ax = b,\ x \geq 0\}$ (for which we wonder whether there is a lattice point), the very first step is to use Theorem 3.1 to transfer the points of P as 3-way tables. We construct 1-marginals for a 3-way transportation polytope Q, and a set S of triples (i, j, k), such that P is the face of Q of those points with $a_{ijk} = 0$ for all i, j, k in S.

Unlike general polytopes, for axial transportation polytopes with given 1-marginals it is trivial to decide integer feasibility and to find an integer vertex of Q. This can be done via the famous greedy

north-west-up corner rule [45, 103]. In fact, integral points of Q will exists as long as the polytope is non-empty and the 1-marginals are integral. To check if Q is nonempty over the reals (i.e., LP feasibility), all we have to do is check whether the sum of the entries in the three margins x, y, z are equal.

The tables inside Q will be the state space for the game. Next, in order to detect whether we have an integral solution of P, we can use another property of axial transportation polytopes, namely we prove that there exists an explicit Gröbner basis for the toric ideal of axial transportation polytopes which will give an integer point in P if one exists. As explained in [90] one can rely on the Gröbner basis elements as moves, which move us from one feasible integer point, a table, in Q to another. We stress that although Gröbner bases have been criticized as having too many elements and potentially many elements of large entries (hence being hard to compute), in our situation we have nicer structure that aids to practical performance. Unlike prior situations our Gröbner bases can be generated on the fly and their elements have entries only $0, 1, -1$ due to the special structure of axial transportation polytopes. We move from table to table that satisfy same margins. In other words, we get around the nasty structure of an arbitrary polyhedron P via the canonical embedding P as a face of an axial transportation polyhedron (which is a larger polyhedron, but it is simpler). By our results in Section 3.2 the pivots will find a feasible point in P if and only if one exists.

Let us explain next more about Gröbner bases and the way to always find an initial table. While general transportation polytopes can be as nasty as arbitrary polytopes, axial transportation polytopes have several noteworthy computational advantages. First, checking real feasibility is trivial, the polytope is feasible if and only if the sum of the entries in each of the 1-margins equal the same value. More strongly, there will be an *integer* solution if the margins are integer.

A special case of axial transportation polytopes that is familiar to most readers, consists of 2-way transportation polytopes. They appear in most college-level optimization courses as bipartite network-flow problems. The $n \times m$ Hitchcock transportation problem: $\min_x c[i,j]x_{ij}$, s.t. $\sum_{i=1}^{n} x_{ij} = a_j^1$ $\sum_{j=1}^{n} x_{ij} = a_i^2$. An initial feasible solution can be obtained from a greedy procedure for certain sequences of the variables, the *northwest-corner rule*. This algorithm proceeds along a sequence of entries S and maximizes each variable in turn with respect to the margins that bound it. A motivating example is shown as follows.

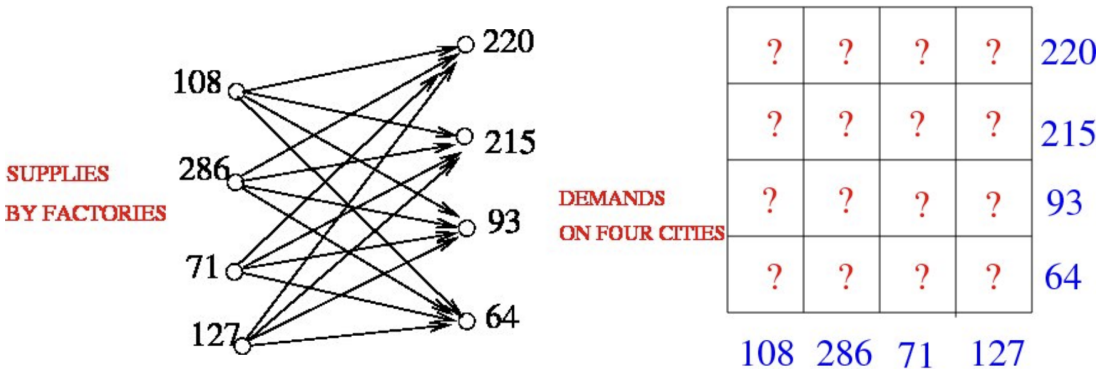


FIGURE 3.1. A motivating example of northwest-corner rule. In (a) there are 4 supplying facilities with different amount of supply, and 4 demanding facilities. The problem can be solved greedily using northwest corner rule by Algorithm 2

Its running time is $O(nm)$. In general the algorithm constructs only a feasible, not an optimal solution. Fortunately, [45] gave a sufficient and necessary condition on S such that the algorithm always constructs an optimal solution for arbitrary demand and supply vectors a^1 and a^2 and cost vector c. But for some cost vectors and special sequences the solution will be optimal. Sequences and cost matrices which fulfill the property above are called *Monge sequences*.

For this problem, we need to find an initial integer table efficiently for 3-way axial transportation polytopes. The good news is that, once again, a very similar greedy algorithm for the classical 2-way problem explained above can be applied to obtain a feasible solution for this more general case. In pseudocode this algorithm will read as follows in the case of a 3-way $l \times m \times n$ axial transportation polytope: Take an arbitrary order of the variables, say the sequence $S := ([i_1, j_1, k_1], [i_2, j_2, k_2], \ldots, [i_{lmn}, j_{lmn}, k_{lmn}])$, and perform the subsequent greedy algorithm:

Algorithm 2 Northwest-corner rule

Input: 3-D tensor $x \in \mathbb{R}^{l \times m \times n}$, 1-margins a^1, a^2, and a^3

1: **for** $s := 1$ to lmn **do**
2: Set $x_{i_s,j_s,k_s} := \min\{a^1_s, a^2_s, a^3_s\}$
3: $a^1_s := a^1_s - x_{i_s,j_s,k_s}$, $a^2_s := a^2_s - x_{i_s,j_s,k_s}$, $a^3_s := a^3_s - x_{i_s,j_s,k_s}$
4: **return** x

Again, given a sequence S of triples of indices, this greedy algorithm maximizes each variable of S in turn. When the algorithm ends it will give always a feasible solution, in fact an integer solution when the margins are integer. In [75] we have a necessary and sufficient condition on S and the

cost matrix c which guarantees that the solution is in fact an optimal LP solution for costs $c_{i,j,k}$ associated to each entry:

LEMMA 3.1.1. *The generalized north-west rule algorithm finds a feasible solution for the three-dimensional axial transportation problem for all right-hand-side vectors a_1, a_2, and a_3 whose sum of entries are equal. The solution is integer if the vectors a_1, a_2, a_3 are integer. Moreover if there is a cost matrix cost matrix $c_{i,j,k}$, which is a three-dimensional Monge sequence in the sense of* [75], *then the solution found is an optimal linear programming solution for the minimization problem.*

3.1.1. Introduction to Gröbner Bases. In the rest of this section, we will briefly introduce the notion of Gröbner bases relevant for our project problems. Recall a *polynomial ideal* I is a set of polynomials in $R = \mathbb{Q}[x_1, \ldots, x_n]$ that satisfies two properties: (1) If f, g are in I then $f + g \in I$ (2) If $f \in I$ and $h \in R$ then fh is in I. a Gröbner basis of an ideal I is a special finite generating set for I with special computational properties. Their computational powers include the ability to answer membership questions for the ideal, computing intersections of ideals, computing projections of ideals, etc. Gröbner bases in general can be computed with the well-known *Buchberger algorithm* [19]. We are only interested in special kinds of ideals, called *toric ideals* whose Gröbner bases are better behaved: Given a matrix A with integer entries, the toric ideal I_A is the ideal generated by the binomials of the form $x^u - x^v$ such that $A(u - v) = 0$. Gröbner bases of toric ideals have been explored in the literature (see [90, 27, 18] and references therein). If we find a Gröbner basis $G_A = \{x^{u_1} - x^{v_1}, x^{u_2} - x^{v_2}, \ldots, x^{u_k} - x^{v_k}\}$ for I_A, it is well known that the vectors $\Gamma = \{u_1 - v_1, u_2 - v_2, \ldots, u_k - v_k\}$ will have the following properties. Let $P = \{x : Ax = b,\ x \geq 0\}$ be any polytope that could be defined by the matrix A and by a choice of an integral right-hand-side vector b. If we form a graph whose vertices are the lattice points of P and we connect any pair x_1, x_2 of them by an edge if there is a vector $u - v$ in Γ such that $x_1 - u + v = x_2$ with $x_1 - u \geq 0$, then the resulting graph is connected [90, Chapter 4]. Moreover if we orient the edges of this graph according to a term order we used to compute the Gröbner basis above (where the tail of an edge is bigger than its head) this directed graph will have a unique sink. Thus from any lattice point of P there is an "augmenting" path to this sink. We will call the process of traversing such an augmenting path a *reduction*. Moreover, we will refer to the elements in Γ as *moves*. It has been shown in [44] that while toric ideals of general transportation polytopes are

as nasty as those for general polytopes, axial transportation polytopes enjoy a rich decomposable structure that essentially allow us to build Gröbner bases from the Gröbner bases of their slices. For instance, for 2-way tables we know everything about the Gröbner bases of their toric ideals

THEOREM 3.2. *Let A be the 0/1-matrix which is the matrix of the linear transformation that computes the row and column sums of a given 2-way table. Then A is (totally) unimodular and hence its minimal universal Gröbner basis consists of its circuits. These circuits are 2-way tables whose row and column sums are zero and with entries in $\{0, +1, -1\}$ of minimal support.*

For axial transportation polytope of size $m \times n \times k$, can we find a similarly nice Gröbner basis for some term order. We explain several ways next.

3.2. Integer Feasibility Testing is a Game on Tables

We have seen that from Theorem 3.1 the (integer) tables with specified margins in the construction represent all the lattice solutions of the original IFP. Those will be the states of the game. In order to check the integer feasibility of P we need to have a Gröbner basis (test set) of the axial transportation polyhedron Q such that the normal form of the North-West-corner rule integer initial solution v is a feasible solution of P (if such a solution exists). Of course, such a Gröbner basis exists right away: a Gröbner basis of the axial transportation arrays with respect to an *elimination* term order where all variables corresponding to the "forbidden" entries of the arrays are bigger than the "enabled" entries will do the job. In principle, we are done. However in the rest of the section we explain how to find a more efficient solution avoiding Buchberger algorithm.

We construct such a Gröbner basis building from the Gröbner bases of 2-way transportation problems slices of 3-way tables (see Theorem 3.2 and discussion there). Moreover, we will prove that we do not need to explicitly compute and store this Gröbner basis in advance (which is a very large set of vectors). It is enough to compute an element of the Gröbner basis "on the fly" that will improve the current feasible solution. For our construction we will follow the ideas presented in [44]. There a similar construction was given for any *decomposable* statistical problem. Fortunately, the 3-way axial problems are decomposable, so we can use those techniques. But we will do this in a slightly more general way. The first set of moves that will make up a Gröbner basis is obtained as follows. Let T be, for a fixed index value k, the 2-way transportation problem defined as $\{a_{i,j,k} \geq 0, \forall i, j, k, \sum_j a_{i,j,k} = u_{i,k}, \forall (i, k), \sum_i a_{i,j,k} = v_{j,k}, \forall (j, k).\}$

Let $G_{\succ_1}, \ldots, G_{\succ_n}$ be n different Gröbner bases of 2-way $l \times m$ transportation problems. Note that if the $l \times m$ table X with entries $X[i,j]$ is a Gröbner basis element, then for each fixed k the 3-way table Y with entries $Y[i,j,k] = X[i,j]$ and $Y[i,j,t] = 0$ for $t \neq k$ is a valid move for the transportation problem T. Now let $\mathcal{F}(G_{\succ_1}, \ldots, G_{\succ_n})$ be the set of moves obtained this way from all elements in $G_{\succ_k}$ for all values of k. The following theorem is a modification of Theorem 4.13 in [44], we omit its proof here:

THEOREM 3.3. *Let $G_{\succ_1}, \ldots, G_{\succ_n}$ be n Gröbner bases of the 2-way $l \times m$ transportation problem. Let $\succ'$ be the term order on the entries of the $l \times m \times n$ axial transportation problem where $\{Y[i,j,1]\} \succ' \{Y[i,j,2]\} \succ' \cdots \succ' \{Y[i,j,n]\}$ and the entries in the kth horizontal slice $\{Y[i,j,k]\}$ are ordered with respect to the term order $\succ_k$. Then $\mathcal{F}(G_{\succ_1}, \ldots, G_{\succ_n})$ is a Gröbner basis with respect to $\succ'$.*

There is a second set of moves for the original transportation problem Q obtained from 2-way transportation problems. Now we will describe these moves. This time let $T_{x,z}$ be the 2-way planar transportation problem. Let $G_{x,z}$ be a Gröbner basis for this problem. Suppose $X_1 - X_2$ is an element in $G_{x,z}$ where X_1 and X_2 are nonnegative tables with entries $X_1[i,k]$ and $X_2[i,k]$. We can "lift" such an element to a move for the original problem Q as follows. First note that $X_1 - X_2$ is homogeneous, i.e., $\sum_{i,k} X_1[i,k] = \sum_{i,k} X_2[i,k] = t$. Second because we are in the setting of a 2-way transportation problem, for each k we have $\sum_i X_1[i,k] = \sum_i X_2[i,k]$. Hence we can represent $X_1 - X_2$ as

$$([i_1,k_1],[i_2,k_2],\ldots,[i_t,k_t]) - ([i'_1,k_1],[i'_2,k_2],\ldots,[i'_t,k_t]).$$

Here we allow that some indices $[i_s,k_s]$ repeated if the corresponding entry $X_1[\cdot,\cdot]$ (or $X_2[\cdot,\cdot]$) in the table is bigger than one. Now given a sequence of indices (again repetitions are allowed) $j_1, \ldots, j_t$ we get a move $Y_1 - Y_2$ for the transportation problem Q: $([i_1,j_1,k_1],[i_2,j_2,k_2],\ldots,[i_t,j_t,k_t]) - ([i'_1,j_1,k_1],[i'_2,j_2,k_2],\ldots,[i'_t,j_t,k_t])$.

We let $\mathcal{L}(G_{x,z})$ to be the set of all moves obtained from all Gröbner basis elements in $G_{x,z}$ using all possible liftings. Similarly we can define $\mathcal{L}(G_{y,z})$. Now we claim we can put together $\mathcal{F}(G_{\succ_1}, \ldots, G_{\succ_n})$, $\mathcal{L}(G_{x,z})$, and $\mathcal{L}(G_{y,z})$ to get a Gröbner basis for the toric ideal of $l \times m \times n$ 3-way axial transportation problem. However, we first need to describe the appropriate term order.

Given a 3-way table X we can compute "projection" tables (marginals) in any axial direction. For instance $Proj_{x,z}(X)$ would be the 2-way table whose (i,k) entry is $\sum_j X[i,j,k]$.

LEMMA 3.3.1. *Let $\succ^1$ and $\succ^2$ be term orders for $l \times n$ and $m \times n$ planar tables and let $\succ_1, \ldots, \succ_n$ be n term orders forthe $l \times m$ planar tables. If $\succ'$ is the term order for $l \times m \times n$ 3-way tables described in Theorem 3.3, then the relation $\succ_*$ on such 3-way tables given by $X \succ_* X'$ if*

$$Proj_{x,z}(X) \succ^1 Proj_{x,z}(X') \text{ or}$$

$$Proj_{x,z}(X) = Proj_{x,z}(X') \text{ and } Proj_{y,z}(X) \succ^2 Proj_{y,z}(X') \text{ or}$$

$$Proj_{x,z}(X) = Proj_{x,z}(X') \text{ and } Proj_{y,z}(X) = Proj_{y,z}(X') \text{ and}$$

$$X \succ' X'$$

is a term order.

PROOF. If $X \neq X'$ it is clear that either $X \succ_* X'$ or $X' \succ_* X$, and the relation is compatible with adding the same table Y to X and X'. So we just need to show that $\succ^*$ is transitive. So let $X \succ_* X'$ and $X' \succ_* X''$. There is a total of nine possibilities to be checked depending on how the tables are aligned, so we give one of these for illustration. Suppose $X \succ_* X'$ because $Proj_{x,z}(X) = Proj_{x,z}(X')$ but $Proj_{y,z}(X) \succ^2 Proj_{y,z}(X')$, and also suppose that $X' \succ_* X''$ because $Proj_{x,z}(X') \succ^1 Proj_{x,z}(X'')$. Then $Proj_{x,z}(X) = Proj_{x,z}(X') \succ^1 Proj_{x,z}(X'')$. Hence $X \succ_* X''$. □

THEOREM 3.4. *Let $G_{x,z}$ and $G_{y,z}$ be Gröbner bases for the $l \times n$ and $m \times n$ planar transportation problems with respect to the term orders $\succ^1$ and $\succ^2$, respectively. Also let $G_{\succ_1}, \ldots, G_{\succ_n}$ be n Gröbner bases for $l \times m$ planar transportation problems with respect to the term orders $\succ_1, \ldots, \succ_n$. Let $\succ'$ be the term order for the $l \times m \times n$ tables given in Theorem 3.3. Then the set*

$$G \quad = \quad \mathcal{L}(G_{x,z}) \cup \mathcal{L}(G_{y,z}) \cup \mathcal{F}(G_{\succ_1}, \ldots, G_{\succ_n})$$

is a Gröbner basis for the 3-way axial transportation problems with respect to the term order $\succ_*$ of Lemma 3.3.1.

Now we are ready to construct a Gröbner basis for 3-way axial transportation problems with which we will solve the integer feasibility problem for any polytope P after it has been encoded as a 3-way axial transportation polytope Q. In order to do this we will describe term orders $\succ^1$, $\succ^2$ and $\succ_1, \ldots, \succ_n$ and then use Theorem 3.4. Recall that by the construction of Q, the tables

corresponding to the feasible solutions of P will be a face F given by forbidden entries which need to be set to zero. If X is such a table where all the enabled entries are positive, then we get forbidden entries for tables $Proj_{x,z}(X)$, $Proj_{y,z}(X)$ which are forced to be equal to zero. Also each horizontal slice $X_1, \ldots, X_n$ will have its own forbidden entries. Let $F_{x,z}$, $F_{y,z}$ and $F_1, \ldots, F_n$ be these forbidden entries, and let $E_{x,z}$, $E_{y,z}$, and $E_1, \ldots, E_n$ be their complements, namely the enabled entries. We let $w_{x,z}$ be the weight vector where $w_{x,z}(i,k) = 1$ if $(i,k) \in F_{x,z}$ and $w_{x,z}(i,k) = 0$ if $(i,k) \in E_{x,z}$. We define $w_{y,z}$ and $w_1, \ldots, w_n$ in a similar way. We define the term order $\succ^1$ to be any elimination term order where $F_{x,z} \succ^1 E_{x,z}$ (and the entries $F_{x,z}$ and $E_{x,z}$ within themselves are ordered in an arbitrary but fixed way) which refines the ordering giving by $w_{x,z}$. In other words, given two $l \times m \times n$ tables X and Y, if the weight of $Proj_{x,z}(X)$ is bigger than the weight of $Proj_{x,z}(Y)$ then we declare $Proj_{x,z}(X) \succ^1 Proj_{x,z}(Y)$. In the case of equality, we resort to the elimination term order that breaks the tie. Note that if the support of $Proj_{x,z}(Y)$ is contained in $E_{x,z}$ and that of $P_{x,z}(X)$ is not, then we immediately declare $Proj_{x,z}(X) \succ^1 Proj_{x,z}(Y)$. We define $\succ^2$ and $\succ_1, \ldots, \succ_n$ similarly in which the forbidden entries are eliminated.

3.2.1. Preprocessing: Coefficient Reduction. Let $P = \{y \geq 0 : Ay = b\}$ where $A = (a_{i,j})$ is an integer matrix and b is an integer vector. We represent it as a polytope $Q = \{x \geq 0 : Cx = d\}$, in polynomial-time, with a $\{-1, 0, 1, 2\}$-valued matrix $C = (c_{i,j})$ of coefficients, as follows. Consider any variable y_j and let $k_j := \max\{\lfloor \log_2 |a_{i,j}| \rfloor : i = 1, \ldots m\}$ be the maximum number of bits in the binary representation of the absolute value of any $a_{i,j}$. We introduce variables $x_{j,0}, \ldots, x_{j,k_j}$, and relate them by the equations $2x_{j,i} - x_{j,i+1} = 0$. The representing injection σ is defined by $\sigma(j) := (j, 0)$, embedding y_j as $x_{j,0}$. Consider any term $a_{i,j}\, y_j$ of the original system. Using the binary expansion $|a_{i,j}| = \sum_{s=0}^{k_j} t_s 2^s$ with all $t_s \in \{0, 1\}$, we rewrite this term as $\pm \sum_{s=0}^{k_j} t_s x_{j,s}$. To illustrate, consider a system consisting of the single equation $3y_1 - 5y_2 + 2y_3 = 7$. Then the new system is

$$\begin{array}{rrrrrrrrrrr}
2x_{1,0} & -x_{1,1} & & & & & & & & = & 0 \\
 & & 2x_{2,0} & -x_{2,1} & & & & & & = & 0 \\
 & & & & 2x_{2,1} & -x_{2,2} & & & & = & 0 \\
 & & & & & & 2x_{3,0} & -x_{3,1} & & = & 0 \\
x_{1,0} & +x_{1,1} & -x_{2,0} & & & -x_{2,2} & & +x_{3,1} & & = & 7
\end{array} .$$

It is easy to see that this procedure provides a new representation, and we get the following.

LEMMA 3.4.1. *Any rational polytope* $P = \{y \geq 0 : Ay = b\}$ *is polynomial-time representable as a polytope* $Q = \{x \geq 0 : Cx = d\}$ *with* $\{-1, 0, 1, 2\}$*-valued defining matrix* C*.*

3.2.2. Representing Polytopes as 3-way Transportation Polytopes with 1-marginals and forbidden entries. Let $P = \{y \geq 0 : Ay = b\}$ where $A = (a_{i,j})$ is an $m \times n$ integer matrix and b is an integer vector: we assume that P is bounded and hence a polytope, with an integer upper bound U (which can be derived from the Cramer's rule bound) on the value of any coordinate y_j of any $y \in P$.

For each variable y_j, let r_j be the largest between the sum of the positive coefficients of y_j over all equations and the sum of absolute values of the negative coefficients of y_j over all equations,

$$r_j := \max\left(\sum_k \{a_{k,j} : a_{k,j} > 0\}, \sum_k \{|a_{k,j}| : a_{k,j} < 0\}\right) .$$

Let $r := \sum_{j=1}^n r_j$, $R := \{1, \ldots, r\}$, $h := m + 1$ and $H := \{1, \ldots, h\}$. We now describe how to construct vectors $u, v \in \mathbb{Z}^r, w \in \mathbb{Z}^h$, and a set $E \subset R \times R \times H$ of triples - the "enabled", non-"forbidden" entries - such that the polytope P is represented as the corresponding transportation polytope of $r \times r \times h$ arrays with plane-sums u, v, w and only entries indexed by E enabled,

$$T = \{\, x \in \mathbb{R}_{\geq 0}^{r \times r \times h} \mid x_{i,j,k} = 0 \text{ for all } (i,j,k) \notin E, \text{ and}$$
$$\sum_{i,j} x_{i,j,k} = w_k \,,\ \sum_{i,k} x_{i,j,k} = v_j \,,\ \sum_{j,k} x_{i,j,k} = u_i \,\} .$$

We also indicate the injection $\sigma : \{1, \ldots, n\} \longrightarrow R \times R \times H$ giving the desired embedding of the coordinates y_j as the coordinates $x_{i,j,k}$ and the representation of P as T (see paragraph following Theorem 3.1).

Basically, each equation $k = 1, \ldots, m$ will be encoded in a "horizontal plane" $R \times R \times \{k\}$ (the last plane $R \times R \times \{h\}$ is included for consistency and its entries can be regarded as "slacks"); and each variable y_j, $j = 1, \ldots, n$ will be encoded in a "vertical box" $R_j \times R_j \times H$, where $R = \biguplus_{j=1}^n R_j$ is the natural partition of R with $|R_j| = r_j$, namely with $R_j := \{1 + \sum_{l<j} r_l, \ldots, \sum_{l \leq j} r_l\}$.

Now, all "vertical" plane-sums are set to the same value U, that is, $u_j := v_j := U$ for $j = 1, \ldots, r$. All entries not in the union $\biguplus_{j=1}^n R_j \times R_j \times H$ of the variable boxes will be forbidden. We now describe the enabled entries in the boxes; for simplicity we discuss the box $R_1 \times R_1 \times H$, the others being similar. We distinguish between the two cases $r_1 = 1$ and $r_1 \geq 2$. In the first case,

$R_1 = \{1\}$; the box, which is just the single line $\{1\} \times \{1\} \times H$, will have exactly two enabled entries $(1, 1, k^+), (1, 1, k^-)$ for suitable k^+, k^- to be defined later. We set $\sigma(1) := (1, 1, k^+)$, namely embed $y_1 = x_{1,1,k^+}$. We define the *complement* of the variable y_1 to be $\overline{y}_1 := U - y_1$ (and likewise for the other variables). The vertical sums u, v then force $\overline{y}_1 = U - y_1 = U - x_{1,1,k^+} = x_{1,1,k^-}$, so the complement of y_1 is also embedded. Next, consider the case $r_1 \geq 2$. For each $s = 1, \ldots, r_1$, the line $\{s\} \times \{s\} \times H$ (respectively, $\{s\} \times \{1 + (s \mod r_1)\} \times H$) will contain one enabled entry $(s, s, k^+(s))$ (respectively, $(s, 1 + (s \mod r_1), k^-(s))$. All other entries of $R_1 \times R_1 \times H$ will be forbidden. Again, we set $\sigma(1) := (1, 1, k^+(1))$, namely embed $y_1 = x_{1,1,k^+(1)}$; it is then not hard to see that, again, the vertical sums u, v force $x_{s,s,k^+(s)} = x_{1,1,k^+(1)} = y_1$ and $x_{s,1+(s \mod r_1),k^-(s)} = U - x_{1,1,k^+(1)} = \overline{y}_1$ for each $s = 1, \ldots, r_1$. Therefore, both y_1 and $\overline{y}_1$ are each embedded in r_1 distinct entries.

To clarify the above description it is helpful to visualize the $R \times R$ matrix $(x_{i,j,+})$ whose entries are the vertical line-sums $x_{i,j,+} := \sum_{k=1}^{h} x_{i,j,k}$.

Next we encode the equations by defining the horizontal plane-sums w and the indices $k^+(s), k^-(s)$ above as follows. For $k = 1, \ldots, m$, consider the kth equation $\sum_j a_{k,j} y_j = b_k$. Define the index sets $J^+ := \{j : a_{k,j} > 0\}$ and $J^- := \{j : a_{k,j} < 0\}$, and set $w_k := b_k + U \cdot \sum_{j \in J^-} |a_{k,j}|$. The last coordinate of w is set for consistency with u, v to be $w_h = w_{m+1} := r \cdot U - \sum_{k=1}^{m} w_k$. Now, with $\overline{y}_j := U - y_j$ the complement of variable y_j as above, the k-th equation can be rewritten as

$$\sum_{j \in J^+} a_{k,j} y_j + \sum_{j \in J^-} |a_{k,j}| \overline{y}_j = \sum_{j=1}^{n} a_{k,j} y_j + U \cdot \sum_{j \in J^-} |a_{k,j}| = b_k + U \cdot \sum_{j \in J^-} |a_{k,j}| = w_k.$$

To encode this equation, we simply "pull down" to the corresponding kth horizontal plane as many copies of each variable y_j or $\overline{y}_j$ by suitably setting $k^+(s) := k$ or $k^-(s) := k$. By the choice of r_j there are sufficiently many, possibly with a few redundant copies which are absorbed in the last hyperplane by setting $k^+(s) := m + 1$ or $k^-(s) := m + 1$. For instance, if $m = 8$, the first variable y_1 has $r_1 = 3$ as above, its coefficient $a_{4,1} = 3$ in the fourth equation is positive, its coefficient $a_{7,1} = -2$ in the seventh equation is negative, and $a_{k,1} = 0$ for $k \neq 4, 7$, then we set $k^+(1) = k^+(2) = k^+(3) := 4$ (so $\sigma(1) := (1, 1, 4)$ embedding y_1 as $x_{1,1,4}$), $k^-(1) = k^-(2) := 7$, and $k^-(3) := h = 9$.

This way, all equations are suitably encoded, and Theorem 3.1 follows from the construction outlined above and Lemma 3.4.1.

Algorithm 3 Integer Feasibility Testing Game

Input: A rational polytope $P \subset \mathbb{R}^d$ presented in its representation $P = \{x : Ax = b,\ x \geq 0\}$.
Output: *True* or *False* depending on whether P contains an integer lattice point.

1: Compute the encoding of P as a face of a 3-way axial transportation polytope Q.
2: Find an initial table V that is a feasible integer solution in Q by Northwest-corner rule.
3: Use Gröbner basis elements with respect to $\succ_*$ constructed above to get the unique sink W.
4: **if** weight of $Proj_{x,z}(W) \neq 0$ or weight of $Proj_{y,z}(W) \neq 0$ **then**
5: **return** *False*
6: Using the Gröbner basis elements in $G_{x,z}$ and $G_{y,z}$ of weight zero (and their liftings) to generate a set $\mathcal{S}$ of tables such that $\{Proj_{x,z}(T) : T \in \mathcal{S}\}$ and $\{Proj_{y,z}(T) : T \in \mathcal{S}\}$ are the set of 2-way $l \times n$ and $m \times n$ tables with the same row and column sums as $Proj_{x,z}(W)$ and $Proj_{y,z}(W)$, and with their support in $E_{x,z}$ and $E_{y,z}$ respectively.
7: **for** $T \in \mathcal{S}$ **do**
8: reduce T using the Gröbner basis elements in $\mathcal{F}(G_{\succ_1}, \ldots, G_{\succ_n})$ to obtain X.
9: **if** weight of X with respect to w_k is zero for all $k = 1, \ldots, n$ **then**
10: **return** *True*
11: **return** *False*

THEOREM 3.5. *Algorithm 3 solves the integer feasibility of any rational polytope P by first transforming it into an 3-way axial transportation polytope Q with specific 1-margins and finding a sequence of Gröbner basis moves from an initial 3-way array in Q to another 3-way array in Q with zeros in specified entries if one exists (equivalent to finding an integer solution for P).*

PROOF. First we show that if P is feasible then the $w_{x,z}$-weight of $Proj_{x,z}(W)$ and the $w_{y,z}$-weight of $Proj_{y,z}(W)$ must be zero. Suppose U is a table corresponding to a feasible solution of P. Then clearly the $w_{x,z}$-weight of $Proj_{x,z}(U)$ and the $w_{y,z}$-weight of $Proj_{y,z}(U)$ are zero. But then if one of these weights for W were bigger than zero we would get a contradiction to the assumption that W is the unique sink obtained by reduction of V with respect to $\succ_*$. Now because the $w_{x,z}$-weight of both $Proj_{x,z}(W)$ and $Proj_{x,z}(U)$ are zero, and since $Proj_{x,z}(W)$ is the unique sink of 2-way $l \times n$ tables with row and column sums equal to those of $Proj_{x,z}(V)$, there is a sequence of Gröbner basis elements in $G_{x,z}$ which reduce $Proj_{x,z}(U)$ to $Proj_{x,z}(W)$. These elements must have $w_{x,z}$-weight zero. This means that we can reverse these moves and use their liftings to obtain a table W' such that $Proj_{x,z}(W') = Proj_{x,z}(U)$. Note that $Proj_{y,z}(W') = Proj_{y,z}(W)$, and we can use the same argument above to conclude that one can reach to a table T using lifted elements of $G_{y,z}$ with $w_{y,z}$-weight zero such that $Proj_{y,z}(T) = Proj_{y,z}(U)$. Of course, we also have $Proj_{x,z}(T) = Proj_{x,z}(U)$. The table T will be an element of $\mathcal{S}$ in Step 4 above, if we use the (reversed) elements of $G_{x,z}$ and $G_{y,z}$ of respective weights zero to generate all possible 2-way $l \times n$ and $m \times n$ tables with the same

row and column sums as $Proj_{x,z}(W)$ and $Proj_{y,z}(W)$. By this construction, for each $1 \leq k \leq n$ the horizontal slices T_k and U_k have identical row and column sums. Since the w_k-weight of U_k is zero, if the same weight of T_k is not zero, one can find a sequence of Gröbner basis elements in $G_{\succ_k}$ (and hence in $\mathcal{F}(G_{\succ_1}, \ldots, G_{\succ_n})$) to obtain X where the w_k-weight of X_k is zero. Because $Proj_{x,z}(X) = Proj_{x,z}(T) = Proj_{x,z}(U)$ and $Proj_{y,z}(X) = Proj_{y,z}(T) = Proj_{y,z}(U)$, the table X corresponds to a feasible solution of P. □

3.3. Learning to Play Games on 2-way Tables

As a proof of concept, we now describe our method of playing table games using reinforcement learning on the polyhedral Gröbner bases systems we presented. Here we only consider the simpler special case of 2-way tables because we have an explicit list of all Gröbner basis moves already (See Theorem 3.2). Similar ideas can be extended to 3-way axial transportation polytopes. In this family, the state space $\mathcal{S}$ consists of $m \times n$ tables of a 2-way transportation polytope with fixed 1-margins,

$$\mathcal{S} = \{\mathbf{s} \in \mathbb{Z}^{m \times n} \mid \sum_i \mathbf{s}_{i,j} = x_i, \sum_j \mathbf{s}_{i,j} = y_j, \mathbf{s} \geq 0\}.$$

The action space is the Gröbner basis generators that connect any two $m \times n$ tables in $\mathcal{S}$. They are computed using Theorem 3.2. It is noteworthy that the action space can be defined with more flexibility because one can consider the minimum set of actions that connects all tables, or with more complicated moves that are linear combinations of the minimum moves. There is a trade-off between the action space and the efficiency of solving the game. We refine our action space $\mathcal{A}$ as follows.

$$\mathcal{A} = \{\mathbf{a} \in \{-1, 0, 1\}^{m \times n} \mid \sum_i \mathbf{a}_{i,j} = 0, \sum_j \mathbf{a}_{i,j} = 0\}.$$

Note that any element of the action space $\mathcal{A}$ is generated as integer combination of the Gröbner basis of Theorem 3.2, which suggests that for any two states $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{S}$, there exists a path connecting $\mathbf{s}_1, \mathbf{s}_2$ using moves from $\mathcal{A}$. In the most general form, our table games can be formulated as follows.

$$\begin{aligned} \max_{\pi} \quad & \sum_t \mathbf{r}_t(\mathbf{s}_t, \mathbf{a}_t)|_{\pi(\mathbf{s}_t)=\mathbf{a}_t} \\ \text{s.t.} \quad & \mathbf{a}_t \in \Omega(\mathbf{s}_t) \\ & \mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{a}_t. \end{aligned}$$

where $\Omega(\mathbf{s})$ is the set of valid moves for (table) $\mathbf{s}$: $\Omega(\mathbf{s}) = \{\mathbf{a} \,|\, \mathbf{a} \in \mathcal{A}, \mathbf{s} + \mathbf{a} \geq \mathbf{0}\}$.

We pre-defined the terminating state of the game by specifying a set of entries $\mathcal{G}$ such that $\mathbf{s}_{(i,j)} = 0, \forall (i,j) \in \mathcal{G}$ as the terminating condition which mimics the forbidden entries of the axial transportation polytope computed in Theorem 3.5. The reward function penalizes every step when the goal state is not reached:

$$\mathbf{r}_t = \begin{cases} -1, & \text{if} \quad \exists\, (i,j) \in \mathcal{G},\, \mathbf{s}_{(i,j)} \neq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

The reward function encourages a policy π that can reach a terminating state with the minimum number of steps.

3.3.1. Demonstration Generating Algorithms. As we stated earlier, the state and action spaces are large and the rewards are constantly negative until the games end, thus very sparse. Learning from demonstrations can effectively increase the learning efficiency and mitigate the sparse-reward problem. Here we discuss some strategies to generate demonstrations.

For every non-goal state $\mathbf{s}$, we use a greedy algorithm to generate a move that reduces the distance to the goal state.

$$\begin{aligned} \min_{\mathbf{u}^+, \mathbf{u}^-} \quad & \sum_{(i,j) \in \mathcal{E}} (\mathbf{s} + \mathbf{u}^+ - \mathbf{u}^-)(i,j) \\ \text{s.t.} \quad & \sum_i \mathbf{u}^+_{i,j} - \mathbf{u}^-_{i,j} = \mathbf{0}, \sum_j \mathbf{u}^+_{i,j} - \mathbf{u}^-_{i,j} = \mathbf{0}, \\ & \mathbf{u}^+ + \mathbf{u}^- \leq 1, \sum_{i,j} \mathbf{u}^+_{i,j} \geq 1, \\ & \mathbf{u}^+,\, \mathbf{u}^- \in \{0,1\}^{m \times n}. \end{aligned} \tag{3.2}$$

The number of non-zero elements in the move can be controlled by replacing $\mathbf{u}^+_{i,j} \geq 1$ with $\mathbf{u}^+_{i,j} = k$ to have $2k$ non-zero elements. Eq. (3.2) always finds the most complicated moves so that every move makes significant amount of progress and the episode length can be reduced. However, such strategy may be prohibited in 3-way tables due to the computational cost.

It is worth noting that the greedy algorithm does not return the shortest path to the goal state. With the RL loss, the agent can learn to outperform the demonstration policy by trial-and-error as

we show in the experiment. The reason of using this greedy policy as demonstration is to reduce the computational cost. To exactly solve for the optimal path is computationally prohibited.

3.3.2. Reinforcement Learning Framework. Our RL framework is based on Twin Delayed DDPG (TD3 [34, 54]). TD3 is an off-policy method which requires a replay buffer $\mathcal{B}$ to collect a set of transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{r})$. There are two critic networks Q_{θ_1} and Q_{θ_2} and an actor network π_ϕ. Both critic networks are trained by minimizing the mean squared Bellman error:

$$L_Q(\theta_i) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},\mathbf{s}',\mathbf{r})\sim\mathcal{B}} \left\| y(\mathbf{s}', \mathbf{r}) - Q_{\theta_i}(\mathbf{s}, \mathbf{a}) \right\|^2, \; i = 1, 2.$$

The target y is defined as the minimum of two fixed target critic networks to prevent overestimation of the Q function.

$$y(\mathbf{s}', \mathbf{r}) = \mathbf{r} + \gamma \min_{i=1,2} Q_{\theta_i'}(\mathbf{s}', \pi_{\phi'}(\mathbf{s}')).$$

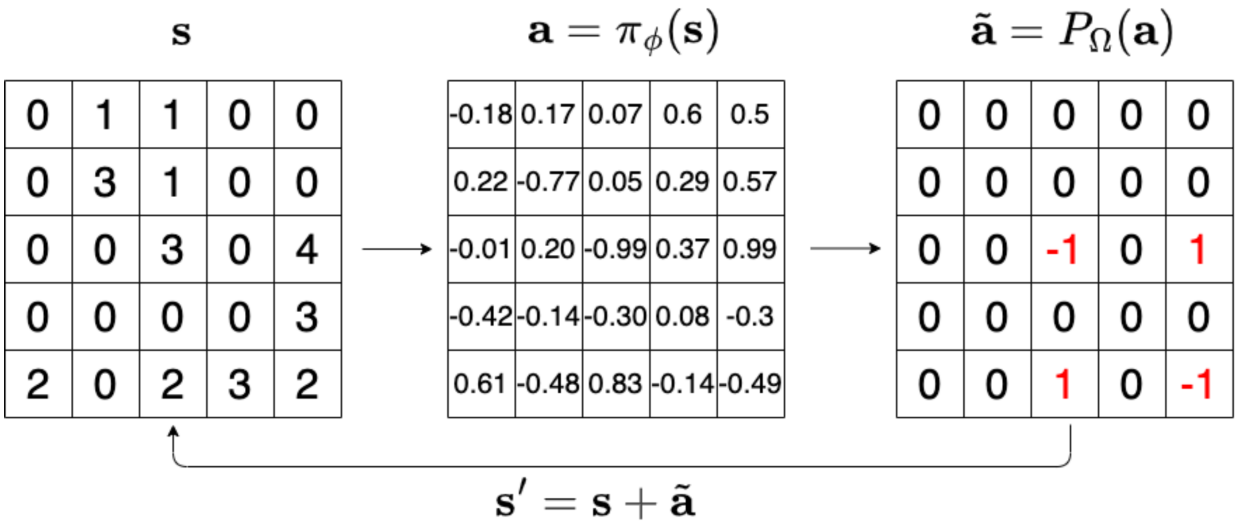


FIGURE 3.2. The interaction between the actor and the environment. The actor network π_ϕ predicts a continuous move $\mathbf{a}$, which is then projected to a discrete $\tilde{\mathbf{a}}$ in the set $\Omega(\mathbf{s})$ before being applied to the environment.

3.3.3. Structured Action Prediction. Due to the fact that our action space is discrete, one natural way is to predict the action as a classification problem at each time step. However, its feasibility is hindered by the large number of actions to compute and store in advance. We instead make the actor network predict a continuous action $\mathbf{a} = \pi_\phi(\mathbf{s})$. Then, we obtain the integral solution $\tilde{\mathbf{a}} = P_{\Omega(\mathbf{s})}(\mathbf{a})$ by projecting $\mathbf{a}$ to the feasible set of discrete actions encoded by the following

integer program.

$$
\begin{aligned}
\min_{\mathbf{a}^+,\mathbf{a}^-} \quad & \sum_{i,j} ||\mathbf{a}^+ - \mathbf{a}^- - \mathbf{a}||_d \\
\text{s.t.} \quad & \sum_i \mathbf{a}^+_{i,j} - \mathbf{a}^-_{i,j} = \mathbf{0}, \sum_j \mathbf{a}^+_{i,j} - \mathbf{a}^-_{i,j} = \mathbf{0}, \\
& \mathbf{a}^+ + \mathbf{a}^- \leq 1, \sum_{i,j} \mathbf{a}^+_{i,j} \geq 1, \qquad (3.3) \\
& \mathbf{a}^+, \mathbf{a}^- \in \{0,1\}^{m \times n},
\end{aligned}
$$

and we obtain the projected discrete action by $\tilde{\mathbf{a}} = \mathbf{a}^+ - \mathbf{a}^-$. We add constraint (3.3) to exclude the action with all zeros. An illustration of the process is shown in Figure 3.2. Splitting $\tilde{\mathbf{a}}$ into two binary variables $\mathbf{a}^+$ and $\mathbf{a}^-$ not only speeds up the projection but also provides more flexibility on the constraints of the actions. For instance, we can bound the number of non-zero elements in the action by adding $c_1 \leq \sum_{i,j} \mathbf{a}^+_{i,j} \leq c_2$ to the system.

To train the critic network, we consider the projection operator a deterministic part of the environment and directly learn $Q(\mathbf{s}, \mathbf{a})$. Similar strategies can be found in [31]. It is worthwhile to mention that one can also learn $Q(\mathbf{s}, \tilde{\mathbf{a}})$. The non-differentiable projection layer can be tackled by straight-through estimators [5, 93]. However, for every mini-batch update, calculating a target in the temporal difference learning requires the projection operations to obtain $\tilde{\mathbf{a}}$. The projection will become a bottleneck and significantly slows down the whole training process. We also observed in experiments that learning $Q(\mathbf{s}, \tilde{\mathbf{a}})$ has no obvious performance gain. The actor network is updated for each mini-batch B by gradient ascent

$$
\nabla J(\phi) = \frac{1}{|B|} \sum \nabla_{\mathbf{a}} Q_\theta(\mathbf{s}, \mathbf{a}) \nabla_\phi \pi_\phi(\mathbf{s}). \qquad (3.4)
$$

3.3.4. Learning from Demonstrations. Learning from demonstrations [?, 43] is an effective strategy to improve sample efficiency. Due the the large action space of our problem, we generate a set of demonstrations $\mathcal{B}_D$ using a greedy algorithm to speed up the learning. We make sure that a fixed portion of samples in the mini-batch are drawn from the demonstration during each training step. Since the demonstrations generated by the greedy algorithm are not perfect, we adopt the Q-filter strategy [62] to only enforce a supervised learning loss when the demonstrated action $\mathbf{a}$ has a higher Q value than the actor's action. Thus, the demonstration loss on mini-batch

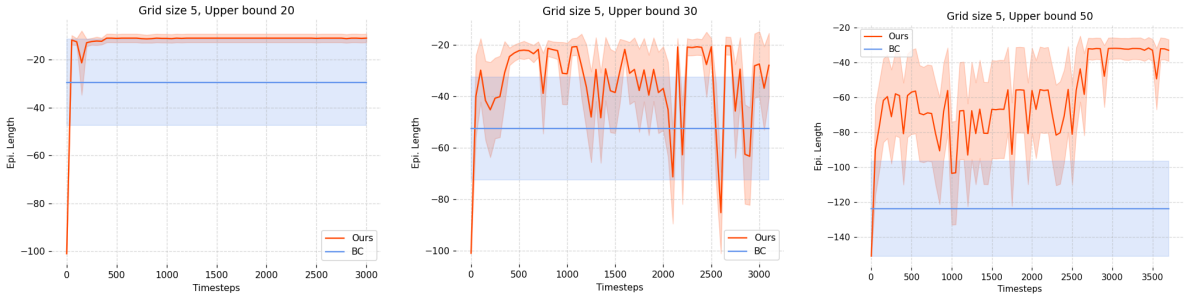


FIGURE 3.3. The learning curve of our RL agent compared with the behavior cloning baseline in 5×5 table games.

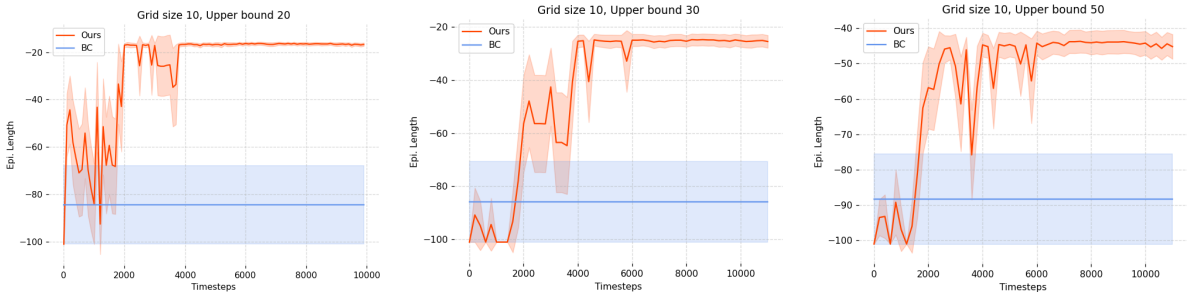


FIGURE 3.4. The learning curve of the RL agent compared to BC model on 10×10 tables.

B_D using demonstrated action $\mathbf{a}_D$ can be summarized as:

$$L_D = \frac{1}{|B_D|} \sum ||(\mathbf{a}_D - \pi_\phi(\mathbf{s}))||^2 \cdot \mathbb{1}_{Q_\theta(\mathbf{s},\mathbf{a}_D) > Q_\theta(\mathbf{s},\pi_\phi(\mathbf{s}))}.$$

3.4. Experiments

3.4.1. Data Collection. The goal entries in $\mathbf{g}$ are selected randomly and remain fixed. The initial state is also randomly generated, and we randomly generate forbidden entries and fill in other entries with random numbers. To start an episode, we randomly initialize the starting table $\mathbf{s}$. To improve the training stability, we add lower and upper bounds on the 1-margins, i.e., $LB \leq \sum_i \mathbf{s}_i \leq UB$. As a result, the 1-margins are fully specified by the initial state and remain the same throughout the episode. The moves of the demonstrating greedy algorithm are calculated state-by-state by selecting a valid move that maximizes the values of the entries in $\mathcal{G}$.

3.4.2. Network Architecture and Baseline. The actor and critics are parameterized by convolutional neural networks. networks consist of multiple convolutional blocks where each block

has a convolution layer followed by batch normalization (BN) and ReLU. The number of blocks depends on the size of the board. All convolution layers maintain the same spatial dimensions. The final block of the actor network has a convolution layer followed by tanh. In the critic network, we apply a global average pooling layer before the final linear layer to predict the Q value.
We also train a baseline model on the demonstration set using behavior cloning (BC) [74, 67]. The BC model is also a neural network with the same architecture as the actor network.

3.4.3. Training Details. We collect 100 demonstrations prior to the RL training and we sample 10% of transitions from the demonstration buffer for every mini-batch update. We use Adam [49] as the optimizer with a learning rate 10^{-4} for both actor and critics. The discount factor γ is 0.99. We use a batch size of 32. The exploration of DDPG-styled algorithm is achieved by adding a Gaussian noise to the actor $\mathbf{a} = \pi_\phi(\mathbf{s}) + \epsilon$. We use $\epsilon \sim \mathcal{N}(0, 0.2)$ throughout the experiment.

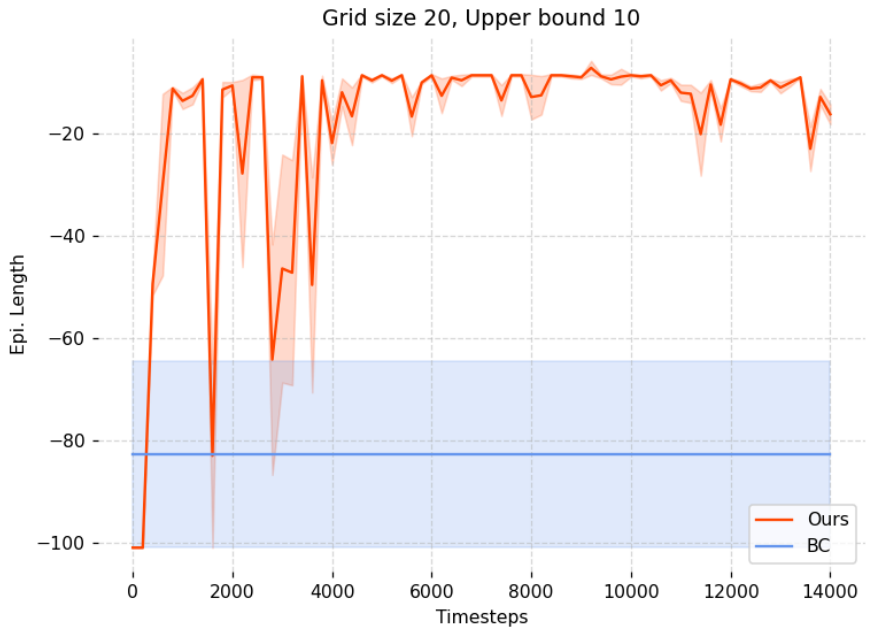


FIGURE 3.5. The learning curve of the RL agent compared to BC model on 20×20 tables.

3.4.4. Results. To demonstrate the ability of our method to learn complex moves, we test our agent on 2-way table of various grid (table) sizes with increasing maximum elements. Larger maximum entries in the table would lead to longer horizons. The results on 5×5 table games is

shown in Figure 2. The mean negative episode length is shown in as the curve and the shaded region denotes the half standard deviation. The BC model can achieve relatively good performance but deteriorates as the upper bound on the entry increases. The RL model outperforms the BC model by a margin and the advantage becomes more significant with larger upper bounds. In Figure 3 we observe similar performance on 10×10 games. We tested three 1-margin bounds $\{20, 30, 50\}$. The BC model does not perform well even with $UB = 20$. On the other hand, our RL agent is able to consistently reach the goal tables with relatively short paths. The results on 20×20 tables are shown in Figure 3.5. The increase of the grid size also increases the action space exponentially. The RL agent can solve the problem whereas the BC model has a very low success rate and it takes longer steps to reach the goal state.

UB	20	40	60	80	100	140
$GS = 5$	1.0	1.0	0.90	0.60	0.72	0.39
$GS = 10$	1.0	1.0	0.79	0.49	0.34	0.32

TABLE 3.1. We test the trained model with margin bound 20 on increased margins and compare the how the success rate varies. The success rates drop as we increase the bound, but the model is able to maintain a decent success rate which shows its generalization ability.

3.4.5. Generalization Test. In this section we wonder if the model can solve games that are unseen in the training data. We set up the experiment by training the model on the fixed upper 1-margin bound 20 with grid size 5 and 10, and test its success rates on larger bounds with the same corresponding grid size. The results in Table 3.1 demonstrate that the trained model can solve games that it never experiences during the policy training step. This further suggests that the model can successfully recognize patterns in the table and produces corresponding moves regardless of their magnitudes.

3.5. Learning to Play Games on 3-way Tables

The FG in 3-way tables can be formulated as a goal-conditioned reinforcement learning framework as follows.

- $S = \{s \in \mathbb{Z}_{\geq 0}^{r \times r \times h} \mid \sum_{ij} s_{ijk} = w_k, \sum_{ik} s_{ijk} = v_j, \sum_{jk} s_{ijk} = u_i\}$.
- $A = \{a \in \{-1, 0, 1\}^{r \times r \times h} \mid \sum_{ij} a_{ijk} = 0, \sum_{jk} a_{ik} = 0, \sum_{jk} a_{ijk} = 0\}$.
- $G = \{s \in S \mid s_{ijk} = 0, \forall (i, j, k) \notin E\}$

- $r(s, a, s') = 0$ if $s' \in G$, -1 otherwise.

S, A, G, and r denote the state space, action space, goal states and reward function, respectively. The game starts at a random valid state and ends when any goal state is reached. The agent is trained to find the shortest path to the goal state.

Unlike FG in 2-way tables, finding complex moves in 3-way tables is computationally prohibited. In addition, generating demonstrations also becomes more challenging and crucial. We therefore consider the minimal moves that are feasible in 3-way tables. These moves are length-4 (4 non-zero entries), which are essentially 2-D slices in 3-way tables. These moves can be combined to generate arbitrary complex moves. We describe an efficient algorithm for generating 3-D demonstration in the following. The idea is to decompose the difference tensor $D = G - S$ into simpler tensors (less non-zero entries) given a goal state G and a start state S. G can be solved by an integer program that satisfies the 1-margin constraints and S can be solved by the north-west corner rule.

3.5.1. Fixed-Step Algorithm for Solving 3-D Feasibility Game. We begin by defining *move complexity* β as the number of non-zero entries in a move. Given β and maximum number of moves T, we can decompose D into much simpler moves $X \in \mathbb{Z}_{-1/0/-1}^{T \times r \times r \times h}$ where $|X^{(t)}| \leq \beta$. Define $X = \mathit{decompose}(D, \beta, T)$ as the solution of the following system

$$
\begin{aligned}
&\sum_t X^{(t)} = D, \\
&\sum_{ijk} X_{ijk}^{(t)} \leq \beta, \ \forall t, \\
&\sum_{ij} X_{ijk}^{(t)} = 0, \ \forall k, \ t, \\
&\sum_{jk} X_{ijk}^{(t)} = 0, \ \forall i, \ t, \\
&\sum_{ik} X_{ijk}^{(t)} = 0 \, \forall j, \ t.
\end{aligned}
\tag{3.5}
$$

The main idea of our algorithm is to guess the number of steps T required to successfully decease to the number of non-zero entries to β. If there is no solution at current β, we increase T until a solution is found. Then, we further decrease β until minimal moves are found. This is essentially trying to achieve minimal action space by sacrificing the game length. A detailed procedure is described in Algorithm 4.

Algorithm 4 3-D Demonstration Generating Algorithm

Input: $D \in \mathbb{Z}^{r\times r\times h}$: difference tensor; β: initial length complexity; $\beta^\star$: target length complexity; T: estimated number of steps
Output: $X^\star$ such that $D = \sum_{t=1}^{T} X^{(t)\star}$

1: $q = \{(D, \beta, T)\}$
2: $X^\star = \emptyset$
3: **while** $|q| > 0$ **do**
4: $D,\ \beta,\ T = q.pop()$
5: $D = submatrix(D)$ // remove empty slices
6: **while** True **do**
7: $X = decompose(D, \beta, T)$
8: **if** X not found **then**
9: $T \leftarrow 2 \cdot T$ // increase the number of steps by 2
10: **else**
11: **break**
12: **for** t = 1,..., T **do**
13: **if** $|X^{(t)}| = \beta^\star$ **then**
14: $X^\star \leftarrow X^\star \cup X^{(t)}$ // add $X^{(t)}$ to demonstration if it reaches the target move complexity
15: **else if** $|X^{(t)}| > 0$ **then**
16: $q \leftarrow q \cup \{(X^{(t)}, \beta/2, T)\}$ // add the new tensor to list and decrease the move complexity by 2
17: **return** X^*

Algorithm 4 maintains a list q of tensors to decompose. Initially, q only has the difference tensor D and two parameters β, T for running *decompose* algorithm as described in Eq. 3.5. If a decomposition cannot be found, we increase T by 2 and re-run the *decompose* algorithm. If a decomposition can be found then we iterate over all decomposed tensors and check if they reach the target move complexity $\beta^\star$. If yes, we add them to the demonstration, otherwise, we add them to q and decrease the move complexity by 2.

We argue that Algorithm 4 can successfully decompose arbitrary tensors into T tensors of minimal move complexity 4 given a starting *beta* being a power of 2. The algorithm is very efficient thanks to several tricks. First, empty slices are removed (line 5). Second, we check if a tensor is empty before adding it to q (line 15). If we increase T by 1 when no solution is found, we can solve for the optimal T by returning the first T that finds a solution. However, it requires too much computation as searching for a integral solution using *decompose* is every expensive. By increasing T by 2 we reduce the times we call *decompose* by may use extra steps ($2T > T^\star$). Thus, we remove empty tensors to improve efficiency. The first trick is even more crucial because it further removes empty slices. At later stages there are very few non-zero entries, e.g., reducing move complexity from 8

to 4. The submatrix operator is implemented by taking all non-zero entry coordinates and slicing each dimension. Its Python implementation is provided in Algorithm 5.

Algorithm 5 3-D Submatrix Slicing

Input: 3-D tensor X

$\mathcal{I} = np.nonzero(X)$
$\mathcal{I}_x, \mathcal{I}_y, \mathcal{I}_z = np.unique(\mathcal{I}[0]), np.unique(\mathcal{I}[1]), np.unique(\mathcal{I}[2])$
$X = X[\mathcal{I}_x, :, :]$
$X = X[:, \mathcal{I}_y, :]$
$X = X[:, :, \mathcal{I}_z]$
return X

The learning framework of 3-D game will comprise learning from demonstrations and reinforcement learning that improves the demonstration policy by trial-and-error. We will only consider minimal moves with 4 non-zero entries for both RL and behavior cloning.

3.6. Conclusions and Future Work

We proposed an algorithm that converts the integer feasibility problem into a game on tables. We formulated the table game as a reinforcement learning problem and developed novel techniques to tackle the algebraic structure of the action space. The experimental results on 2-way tables show the potential of solving integer feasibility problems using the Gröbner bases approach. One remaining component of this framework is the ability to predict "no solution" since the feasible lattice point may not exist. Since IFP is NP-complete, it takes exponential time to visit all lattice points before concluding "no solution". One potential solution is to train a classifier that predicts the winning probability for every state of the game. When testing, we can terminate the program when the predicted winning probability drops below a threshold.

Training on 3-way tables is the next stage of our work. The challenge comes from the enormous state and action spaces. We have proposed several advanced techniques to make the training possible. We will present new results in our followup works.

CHAPTER 4

Persistence-based Clustering for Biological Image Segmentation

4.1. Background

Neuroimmunology in the central nervous system includes a diverse array of biology ranging from traumatic brain injury to autism spectrum disorder, a commonality of these studies is the assessment of resident immune cell activation. Microglia are resident immune cells in the brain that survey CNS tissue for damage, or perturbations to homeostasis due to infection or injury [53, 69]. Under homeostatic conditions, microglia are highly ramified having small cell bodies with multiple processes that are surveying the surrounding space. Activation of microglia by danger-associated or pathogen-associated molecular patterns such as extracellular ATP, or LPS respectively not only induces the expression of pro-inflammatory genes, such as cytokines but also results in a less ramified morphology [38]. This distinct change in morphology has therefore been used to interrogate neuroinflammation in a variety of models and animal species. Although several technologies are capable of ascertaining cell type identity and the gene transcription profile of a single cell [38], dissociative techniques not only result in the loss of spatial information but can also induce activation during isolation [38, 77]. Consequently, light microscopy-based methods such as confocal, two-photon, or light sheet microscopy remain standard methodologies to assess microglial activation within defined tissue structures. Despite this "gold standard" approach, there are several variations used for determining cellular activation based on the amount of ramification. These include wide-field microscopy, or confocal image data that is simplified by data reduction with the projection of three-dimensional space down to a two-dimensional representation. While these approaches produce image data of the cells of interest in both cases, all three-dimensional data is lost. Typically these data sets are then assessed with the degree of microglial ramification measured in an indirect manner using Sholl analysis [83, 66]. Although this reductionist approach has been the standard for many years, restricting the analysis to 2D data results in significant data reduction and losses in sensitivity.

While the activation state could be better assessed by image segmentation in three-dimensions, to discriminate each microglia from the background tissue image, this approach has proven to be technically difficult. Typically, microglia are identified by either indirect immunofluorescence with a primary antibody that detects proteins that are highly expressed by microglia, or by using genetically encoded fluorescent proteins with expression controlled by microglial specific promoters. Image processing routines based on simple thresholding to identify the cell type of interest from the background are usually not sufficient for images of cells within tissues due to different background intensities and an increased potential for noise. Several recent advances have been made including 3Dmorph that seeks to extract information on defined cell types, such as microglia, in three-dimensions [104]. Despite this advance, this process relies on user-determined OTSU thresholding, and decisions on parameters such as maximum cell size and process lengths for each object detected. As a further limitation, these prior works cannot handle large image files that represent large physical regions of a tissue. New technological advancements, including tissue clearing methods such as CLARITY [32] and iDICSCO [55], coupled with Light sheet microscopy [88] have further increased these difficulties simply due to the ability to acquire data from an entire intact organ, such as the brain, or small model organisms at sub-cellular resolution. Despite this explosive growth in imaging technology, robust quantitative analysis of increasingly large datasets has lagged behind. Image analysis of these sample types typically is difficult if not impossible with these software tools. Although elegant segregation modalities continue to be developed that leverage machine learning algorithms for the identification of cells, such as Stardist [81] and Cellpose [89], these neural network based tools do not function for non-uniform cell types such as microglia. Segmentation of objects with irregular morphology, such as bacteria, using machine learning algorithms has been implemented in Omnipose [20], although application to other cell types typically requires extensive training using expert annotated training data. Commercial analysis software such as Imaris (Oxford Instruments), and Neurolucida (MBF Bioscience) also offer these capabilities, however, in addition to limitations of training, there is often a substantial cost associated with these software licenses. All these issues contribute to an inability to perform robust analysis of non-uniform cell types in physically large regions, whole organs, or organisms from biological samples.

4.2. Persistence-based Clustering

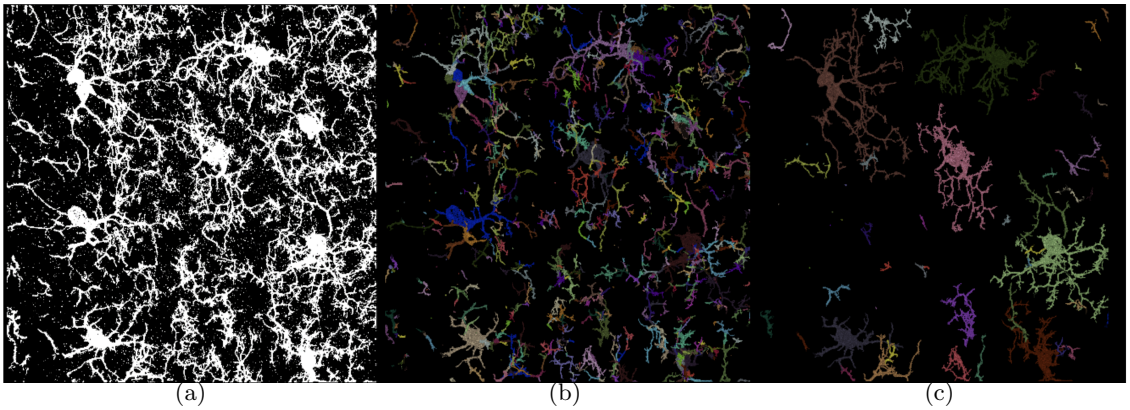
An overview of PrestoCell workflow is illustrated in Figure . Our input to the pipeline is the imaging data of dimension Z * 2 C * X * Y. The input contains microglia and nuclei in two separate channels, each with dimension Z * X * Y. PrestoCell segments both microglia and nuclei, corresponding to (b) and (d) in Figure overview. The nuclei segmentation is used to refine microglia segmentation. The input is denoised using Otsu thresholding to remove low-intensity voxels before proceeding to segmentation algorithms described in following sections.

The microglia are segmented using persistence-based clustering (PBC), which is a topological segmentation method based on persistence homology. Its main idea comes from PH where a filtration is applied on the input data to create lifespan of topological features. Prominent structures, whose lifespans exceed a persistence threshold δ, are declared clusters. Those with shorter lifespans will be seeking to merge into its adjacent prominent structures. If there is no prominent structure in their neighborhood, they will be considered noise. For microglia segmentation, the only topological structure we consider is the connect component. An illustration on 1-D data is shown in Fig. 4.3.

Mathematically, the procedure of PBC is summarized as follows. Let $X = \{x_1, \ldots, x_N\} \in \mathbb{R}^d$ denote a set of data points, $f : X \to \mathbb{R}$ denote a mapping from a point to its function value (the intensity of data points in this work), $\mathcal{N}(x_i)$ denote the neighborhood of x_i. PBC has two phases. (1) *Mode-seeking.* We iterate over all data points X in decreasing f. If $x_i \in X$ is has the highest f within $\mathcal{N}(x_i)$, x_i is declared as a peak. Otherwise, x_i is assigned to the data point with the highest f in $\mathcal{N}(x_i)$. (2) *Merging.* In (1), each peak and every other point connected to it can be considered a cluster. During merging, peaks will be merged according to the persistence threshold τ. Again, we iterate over X in decreasing f. If x_i is a peak, we create a new entry e_i. If x_i is not a peak, it must belong to some entry e_i. we search over $x_k \in \mathcal{N}(x_i)$ such that $f(x_k) > f(x_i)$. We merge e_k into e_i if $f(r(e_k)) < \min\{f(r(e_i)), f(x_i) + \tau\}$, where $r(e_i)$ is the root entry of e_i. Conversely, if $f(r(e_i) < \min\{f(r(e_k)), f(x_k) + \tau\}$, we merge e_i into e_k.

PBC naturally brings many advantages over other clustering methods. First, PBC correctly captures the topological features of microglia, i.e., always a single component. Second, PBC is very flexible because the user can adjust the number and size of clusters by tuning δ. Also using higher δ can have a denoising effect. Third, the first two advantages allow us to build a human-in-the-loop

framework where domain experts can interact with the algorithm to generate better microglia segmentations. Figure 4.1 shows PBC with different persistence thresholds. Figure 4.1a is the input



(a) (b) (c)

FIGURE 4.1

to PBC. Figure 4.1b and Figure 4.1c are the outputs of PBC using a lower and higher δ. Each color represents a cluster. The higher δ results in more clusters merging into bigger ones, and some clusters are removed due to the lack of adjacent prominent clusters. It is worth noting that some smaller clusters omitted by Figure 4.1c are microglia and these microglia are detected by a lower persistence threshold in Figure 4.1c. Thus, it is recommended to run PBC with different thresholds and PrestoCell will combine microglia from all clustering results.

The radius δ also plays a crucial role PBC as it decides if two clusters get merged. In microglia segmentation, cells have to be single connect components. We set $\delta = 1.8$ to consider 26 adjacent points around a voxel, which says if two voxels are separated by at least one voxel, they will not be clustered into one group, thus they do not belong to the same cell.

4.3. Nuclei Segmentation

We segment nuclei using Cellpose [89]. Cellpose is good at segmenting roundish objects. We use a 2-D version of Cellpose to efficiently generate nuclei masks. We make batches across dimension z and create a 2-D image for each batch by taking the maximum along dimension z. Then we lift the 2-D prediction back to 3-D by stacking the 2-D image and taking the overlap between the input nuclei image.

4.4. Nuclei Matching

The nuclei are segmented using Cellpose [89], a deep neural network trained on very large, annotated data sets to predict cell boundaries. It has a superior performance in predicting round-shape objects. PrestoCell directly uses Cellpose to produce nuclei segmentation. After removing the nuclei that do not overlap with predicted microglia, we obtain the nuclei masks shown in Figure 4.3.

The main purpose of nuclei segmentation is to refine the microglia segmentation given by PBC, since PBC is an unsupervised method. The matching step does three refinements of the PBC's prediction. First, clusters that do not overlap any nuclei are removed. Second, clusters that overlap with the same nucleus are merged. Lastly, clusters that overlap with more than one nucleus are split. The first two refinements are straightforward to implement. For cluster splitting, we implement a heuristic algorithm that splits the cluster into subclusters such that there is a one-one correspondence between nuclei and clusters. The splitting algorithm is described in Algorithm 6 where we consider splitting a cluster that overlaps two nuclei. The algorithm can be easily generalized to multiple nuclei situations.

Algorithm 6 Cluster Splitting

Input: cluster $\mathcal{X}_i$ to split, nuclei $\mathcal{N}_1$, $\mathcal{N}_2$ overlapping $\mathcal{C}$.

1: $\mathcal{C} = persistence\_based\_clustering(\mathcal{X}_i, \delta, \tau)$
2: **while** $\nexists\, \mathcal{S}_1 \in \mathcal{C}, \mathcal{S}_2 \in \mathcal{C}$, $i \neq j, \mathcal{S}_1 \cap \mathcal{N}_1 = \emptyset$, $\mathcal{S}_2 \cap \mathcal{N}_2 = \emptyset$ **do**
3: $\quad \{\mathcal{C}_i\} = persistence\_based\_clustering(\mathcal{X}_i, \delta, \tau/2)$
4: **while** $\mathcal{S}_1 \cup \mathcal{S}_2 \neq C$ **do**
5: $\quad$ **for** $\mathcal{C}_i$ in $\mathcal{C}$ **do**
6: $\quad\quad$ **if** $\mathcal{C}_i$ is adjacent to $\mathcal{S}_k$, $k = \{1, 2\}$ **then**
7: $\quad\quad\quad \mathcal{S}_k \leftarrow \mathcal{S}_k \cup \mathcal{C}_i$
8: $\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}_i$
9: **return** $\mathcal{S}_1$, $\mathcal{S}_2$

The splitting algorithm takes as input all data points $\mathcal{X}_i$ of cluster i. The main idea of the splitting algorithm is to break $\mathcal{X}_i$ into smaller clusters so that we can find two anchor clusters $\mathcal{S}_1$ and $\mathcal{S}_2$, each of which overlaps one nucleus. We keep reducing τ by half if we cannot find such anchor clusters for each nucleus overlapped (line 2-3). A good initial τ is critical to the efficiency of the algorithm. We first calculate the lifespan of all clusters and then use the median as the initial guess.

Then, we greedily merge the rest of clusters to an anchor cluster if they are adjacent. The anchor cluster is the union of merged clusters and the original anchor cluster (line 4-8). After all clusters get merged, we get 2 clusters each of which overlaps one nucleus.

4.5. Materials and Methods

Animals. Samples were obtained from mice that were originally purchased from The Jackson laboratories. All animals had ad libitum access to food and water, and all procedures were approved by the UC Davis Institutional Animal Care and Use Committee.

4.5.1. Tissue processing. Following euthanasia, trans-cardiac perfusion was performed with ice cold saline, followed by 10% normal buffered formalin. Brains were carefully dissected and placed in formalin for an additional 24 hours. Samples for confocal microscopy were immersed in 30% sucrose solution as part of a standard cryoprotection regimen. Brains were then placed in molds with optimum cutting temperature (OCT, Fisher Scientific), and rapidly frozen in a dry ice chilled isopentane bath. Blocks were mounted in a cryostat allowing for cutting of 40 μm thick coronal tissue sections and frozen until use.

4.5.2. Antibody staining. A standard tissue staining protocol was used to label specific cell types. In brief, slides were washed with PBS, and incubated in blocking solution (5% BSA, 5% normal donkey serum) for 1h before incubation with rabbit anti-IBa-1 antibody (catalog number 019-19741; Wako) for 16 h at 4°C, 1:300). After extensive washing donkey anti-rabbit Alexa Fluor 546 was applied for 1h at RT, 1:500 (Invitrogen, cat#), washed extensively and sections were stained with DAPI (1:5000, PBS TritonX100 0.1% v/v). Coverslips were mounted with prolong gold antifade reagent (Invitrogen), allowed to cure and kept at 4°C until imaged.

4.5.3. Data acquisition. Confocal microscopy: Data from samples mounted on slides were acquired with a Leica SP8 STED 3X confocal microscope, equipped with a white light laser (using a 557 nm excitation for Alexa Fluor 546, and a 405 laser for excitation of DAPI). Images are acquired using a 63x/1.4NA objective, with areas larger than the field of view captured by imaging of multiple overlapping segments (10% overlap), with Z planes acquired at a 0.3 μm step size. Tiles were either processed individually or merged into a larger image prior to microglia segmentation. Light sheet microscopy: After necropsy and perfusion brains were removed and placed into 10% normal buffered formalin to ensure proper fixation before tissue clearing using iDISCO(ace) as previously reported [55]. In brief, tissues were pretreated with incubations in (i) 25% acetone, (ii) 50% acetone, (iii) 25% acetone, (iv) PBS, (v) PBS/30% sucrose at room temperature, followed by incubation in PBS/30% sucrose/1% H2O2/10 mM EDTA-Na (pH 8.0) at 4°C overnight. Tissues

were permeabilized (PBS/0.2% Triton X-100/0.1% deoxycholate/10% DMSO/10 mM EDTA (pH 8.0) overnight), blocked (PBS/0.2% Triton X-100/10% DMSO/5% normal donkey serum at room temperature for two days), and incubated with primary antibody (rabbit anti-Iba1 1:500 dilution in PBS/0.1% Tween 20/heparin (10 $\mu g/ml$) / 5% normal donkey serum) at 37°C for 4 days. Tissues were washed in PBS/0.1% Tween 20/heparin (10 $\mu g/ml$) at 37°C for 48 hours with multiple buffer changes, then incubated in secondary antibody (donkey anti-rabbit Alexa Fluor 546 diluted 1:1000 in PBS/0.1% Tween 20/heparin (10 $\mu g/ml$)/ 5% normal donkey serum) at 37°C for 72 hrs. Tissues were washed in PBS/0.1% Tween 20/heparin (10 $\mu g/ml$) at 37°C for 48 hours with multiple buffer changes, then incubated in Sytox Deep Red for 24 hrs. After extensive washing, clearing was performed by incubating overnight in each of the following: PBS, 20% methanol twice, 40% methanol, 60% methanol, 80% methanol, and 100% methanol twice. Finally, tissues were incubated in 33% methanol / 67% DCM for 3 hours, 100% DCM twice for 1 hr each, and placed in Ethyl cinnamate (CAS-No: 4192-77-2, EMD Millipore Corporation), before image acquisition with a Zeiss Lightsheet 7 microscope and a 20 X immersion objective.

4.6. PrestoCell

4.6.1. Theory of persistence-based clustering applied to image data. Putative microglia are segmented using confocal microscopy data where immunoreactivity for the IBA-1 protein has been detected. Segmentation of these cells is performed using persistence-based clustering (PBC) software Tomato [15], which is a topological segmentation code based on persistence homology. This PBC is derived from Persistence homology where a filtration is applied to the input data creating a lifespan of topological features. Prominent structures, with lifespans exceeding a persistence threshold δ, are declared clusters. Structures with lifespans smaller than δ will be potentially merged with adjacent prominent structures. If there are no prominent structures in their immediate neighborhood, they will be considered noise. As an example with 1D data, filtration starts by setting the threshold ($alpha$) to the maximum value of the data resulting in a single component. The threshold value is then decreased, and at this new value (a1), a new component emerges for a total of two components. Further reducing the threshold (a2) causes the death of this component. The persistence threshold δ will determine if components are discrete clusters or

should be merged with an adjacent cluster, as each microglia is a group of connected components (Fig1A).

4.7. Segmentation of confocal images using PrestoCell

4.7.1. Overview. PrestoCell performs segmentation on a Tag Image File Format (tiff) file that contains two channels, the first representing nuclei (DAPI), and the second identifying channel corresponding to the cellular marker (Iba1) of interest (Fig 1B). PrestoCell uses Cellpose to segment nuclei, a deep neural network trained on very large, annotated data sets, that has superior performance in predicting objects with a uniform morphology [89]. (Fig 1B). Since the preceding PBC is an unsupervised method, the nuclear segmentation step serves to refine the microglia segmentation in three discrete ways. First predicted microglia that lack nuclei, and therefore are likely incomplete cells, or potential artifacts are removed. Second, clusters from the microglia segmentation that overlap with the same nucleus are merged, as they belong to the same cell. Lastly, clusters that overlap with more than one nucleus are split using a heuristic algorithm to split the cluster into subclusters such that there is a one-on-one correspondence between nuclei and clusters. The image channel containing microglia is subjected to OTSU thresholding, followed by analysis of this result by persistence-based clustering. These are matched the identified nuclei, aiding the editing of the predicted masks by the user (Fig 1B).

4.7.2. Using PrestoCell. Users begin image segmentation by starting PrestoCell from the command line in a Python environment. A graphical user interface (GUI) directs the user to select the desired file, and what operation is desired (Fig 2A). Here we provide the user the ability to segment an image, or refine a previous segmentation if one exists (Fig 2B). The number of Z planes, channels, and dimensions (x and y) of the selected input image are displayed, and the user is asked to provide additional information for segmentation (Fig 2C). PrestoCell leverages the open-source Python-based image viewer Napari, providing the user with an interactive editing feature in two and three-dimensions (Fig 2D). This unique feature allows the user to rotate and modify the segmentation. To allow easy editability, PrestoCell shows each cell with the largest persistence value that produced a putative mask, as the largest possible cell mask. The lowest persistence value is then used to display the identified clusters for editing. This allows for the user to visually inspect the PrestoCell suggested mask, refining the segmentation by adding or removing component

clusters while viewing the mask and original data in 3D space. In the depicted 2D editing window within Napari, the extent of the cell process can be added by clicking the mouse button on the desired adjacent structure (Fig 2E). This is the key to the user-friendly interactive interface with mask refinement at a cluster level, making editing efficient and easy.

4.7.3. Comparison to Segmentation with other tools. Imaris: Segmentation in Imaris (v10) was performed using "Surface Creation" and selecting LABKIT for the thresholding step. This thresholding uses the machine learning classification kit in FIJI[17], where the user trains a model to distinguish between the foreground and background. These results are returned to Imaris to complete surface creation, where each cell is manually selected, and individual masks are saved as a separate surface. Cells that are touching are split using the "cut surface" option. Cells requiring multiple splitting events or complex cutting geometries, with multiple planes of section were not split.

3DMorph: Segmentation of microglia was modified from the original publication [104] using Imaris to aid validation of the segmentation process [42]. In brief, a custom modification to the MATLAB script was made to allow the export of 3D mask coordinates. We have used these in our comparison to ground state truth as "unedited 3DMorph" segmentation results. User refinement of the suggested masks was also generated, and these "edited 3DMorph" masks were produced by overlaying the suggested mask on the original data set, allowing for manual joining or splitting of structures in Imaris.

Omnipose: Installation with GPU processing enabled was performed as described in the GitHub repository [20]. Segmentation was performed using Jupyter Notebook to generate segmentations with values for "mask_threshold" ranging from [-4 to 4] in increments of 0.5. Values for "flow_threshold" were set to 0.4 as indicated in the documentation for this tool. Cell diameter was adjusted to control the scaling of the image to meet the constraints of the trained model as described. Results of this iteration were saved and evaluated visually for segmentation with masks evaluated visually and using mathematical tools as described below.

4.7.4. Evaluation of segmentation. With the rise of machine learning techniques, a plethora of methodologies for the rigorous evaluation of segmentation results have been developed. Although the various analysis modalities seek to compare pixels in a ground state truth image, an annotated

image that identifies the structure or feature or interest, each of these measures is subject to different effects of the underlying data set and has unique limitations [61]. With this in mind, we have used multiple comparisons to assess segmentation compared to a ground state truth for each identified cell including Jaccard index (Intersection over Union), F1 score (a.k.a. Dice coefficient, or Sørensen-Dice coefficient) [68], and the percent of false positive and false negative pixels.

4.8. Results

4.8.1. Image segmentation with PrestoCell. PrestoCell was capable of image segmentation and mask generation from 3D confocal images from tiles of confocal Z-stacks up to 435 x 435 x 42 um. Using the workflow described above, we performed image segmentation using PrestoCell (Fig. 2A, top). To quantify the accuracy of the segmentation, we compared segmented signal cells from the unedited and user-edited PrestoCell outputs to a ground state truth segmentation that was previously prepared (Fig. 2A bottom). To determine how PrestoCell compared to other current tools, we also compared segmentations conducted on the same dataset using 3DMorph, and a machine learning-based approach in Imaris (Fig. 2A top), and compared these to the ground state truth as well (Fig. 2A bottom). Visual representation of false positives and false negatives compared to ground state revealed that the PrestoCell outputs without user refinement or editing typically have increased false negatives compared to the other tools. Critically, user refinement of the PrestoCell segmentation appeared to reduce this type of error. We quantified the performance of each of these tools by comparing the segmentation to the ground state using F1, Jaccard scores, and the False positive and false negative rate for each cell in our data set. As expected from the graphical visualization, unedited PrestoCell segmentations have a lower F1 and Jaccard score, compared to either 3DMorph or Imaris (Fig. 2B & C). These deficiencies were absent in the user-edited PrestoCell segmentations demonstrating that with simple user feedback, there is no difference in the quality of segmentation produced based on F1 and Jaccard scores. PrestoCell and the edited PrestoCell segmentations had significantly lower false positivity compared to the other tools with Imaris producing the most false positives (Fig. 2D). False negatives were similar between user-edited PrestoCell segmentations and 3D morph (Fig. 2E). These analyses further demonstrated that Imaris platform resulted in significantly less False negatives compared to the other methods.

As a machine learning-based tool, Omnipose has shown a remarkable ability to perform segmentation of light microscopy images of non-uniform cells. Our use of this Omnipose was
however constrained by an inability to train a custom model on our microglial data due to the sparsity of microglial features in three-dimensions. Although the pretrained cell models produced segmentation this process resulted in masks, these masks were fragments of the individual cells previously identified in our ground state truth (Fig 3). These results suggest that the pre-trained Omnipose model is not capable of segmenting microglia without prior knowledge of the expected result.

Together these data demonstrate that PrestoCell produces image segmentations that are comparable or exceed the results from existing methods. In particular, with the reduced false positives generated from PrestoCell segmentations, PrestoCell will outperform these existing methodologies.

4.8.2. PrestoCell-based segmentation of physically interacting cells. Our results suggested that PrestoCell can provide high quality image segmentation of cells where other software tools may produce errors. In the evaluation of our data set, we noted that PrestoCell was well suited to perform segmentation of cells in close proximity to one another or cells that appeared to be physically interacting, whereas 3DMorph produced a single segmented cell (Fig. 4).

4.8.3. PrestoCell segmentation of light sheet microscopy data. Continued development of light imaging modalities capable of sub-cellular resolution throughout a whole organ or even a small model organism has produced remarkable results with few tools to aid quantitative analysis. PrestoCell was able to segment microglia within a 664 x 664 x 384 μm region (0.169 mm3) from a mouse brain subjected to tissue clearing with iDISCO and light sheet microscopy (Fig. 5). Attempting to generate segmentations using 3DMorph resulted in the software becoming rapidly resource-constrained and yielded no segmented cells.

4.8.4. Discussion. Rapid advances in light microscopy technologies have brought about an unprecedented increase in the amount of data generated in the life sciences. Despite the development of techniques to acquire data from entire intact organs including the brain, the ability to assess these data rapidly and quantitatively has been limited by a lack of tools or approaches that are time intensive. Traditionally, this analysis had been performed on 2D datasets or maximum intensity projection to generate 2D datasets and is based on Sholl analysis. These approaches result in

the loss of data, therefore the ability to detect small and potentially meaningful biological effects. Better representations of the complexities of cells in 3D can also be achieved by computerized tracing of the raw data. Perhaps unsurprisingly this approach must be completed with great care, is laborious and painstaking when dealing with many individual cells and many biological replicates, and may not be optimized for all cell types. While many methodologies and segmentation techniques have been developed for microscopy, few can provide segmentation of irregularly shaped cell types that can be closely juxtaposed. Recent tools such as Cellpose [89], or histopathology samples with QuPath [2] generally do not perform well for analysis of non-uniform cells such as microglia. Other approaches include 3DMorph, a MATLAB-based script designed for the segmentation and analysis of microglia in 3D datasets. Although 3DMorph offers substantial improvements in the segmentation of these cells under specific conditions, there are clear limitations and design choices that can preclude generalized use. Our experience with these prior tools led us to create PrestoCell allowing biologists to rapidly and accurately segment non-uniform microglial Here we document that cell masks generated by PrestoCell and user-refined PrestoCell masks are equivalent to, or exceed the accuracy of either 3DMorph and even a commercial machine learning approach. We further demonstrate that although the ability of 3DMoprh to segment cells without requiring a nuclear marker, such as DAPI is touted as a benefit, there are instances where this choice leads to inaccuracies. The requirement of nuclei for PrestoCell to segment cells allows for the ability to confidently segment microglia that are closely juxtaposed or interacting in 3D. In stark contrast, 3DMorph typically groups microglia with these characteristics together. The resulting segmentation mask remains less than the user defined maximum physical size but with two discrete nuclei. While it is possible to separate these cells post-segmentation, the output of cell masks from 3DMorph was not implemented in the original script, preventing users not familiar with MATLAB from achieving this easily. In addition, 3D visualization software with the capacity to edit, such as Imaris, is required to separate aberrant microglia masks. We have previously used this approach, although identification of where segmentations should be split or joined in 3D is entirely up to the user, without any guidance to indicate points or regions where two cells are interacting.

PrestoCell has been designed to allow visualization and refinement during the mask generation process. To achieve this, we have leveraged existing tools in the Python language, with visualization and editing performed in 3D using the user-friendly multidimensional Napari viewer. The use of

Python and the various tools was purposefully selected to ensure that broad multiplatform use, that does not require a MATLAB license or purchase of expensive visualization software such as Imaris. During the mask refinement process we further specifically highlight "critical points" allowing a user to review and determine if those regions should be included in the final mask, that are based on the PBC outcomes. This reduces the subjectivity of splitting or adding during mask refinement. Another clear benefit of PrestoCell over current tools is the ability to perform segmentation on large datasets where 3DMorph simply fails to load the data. These datasets include not only standard confocal image tiles but also the ability to perform segmentation from large pieces of tissue acquired by light sheet imaging. This attribute will allow PrestoCell to aid users in performing quantitative analysis on large volumes of data obtained with new technologies.

Analysis tools have recently been developed to make use of machine-learning approaches. In general, these machine learning algorithms using statistical based models that are trained on datasets containing cell type of interest. Training and model development to reduce the potential for bias can be difficult to achieve in practice, and non-commercial software may require a significant knowledge base in data science or machine learning. Given the interest and potential power of these approaches, many commercial image analysis platforms are offering fully integrated solutions or the ability to use open-source tools. These include the use of the LABKIT Fiji plugin for machine learning pixel classification as an extension for Imaris. Segmentations produced using this LABKIT extension generated robust F1 and Jaccard scores. Despite this performance, the number of false positives was significantly increased compared to PrestoCell and 3DMorph. These results suggest that PrestoCell performs as well, or better under select conditions compared to this Imaris extension. Our results to date demonstrate that as a novel of persistence-based clustering, PrestoCell can provide users across different operating systems with a robust segmentation tool that is easy to use and able to handle large datasets. Critically this implementation is accurate, and in some cases outperforms existing segmentation tools, and will allow users of varying skill levels to begin quantitative analysis of non-uniform cell types including microglia.

4.9. Software Release

The code and user guide can be found in our Github repo https://github.com/euyuw2/PrestoCell.

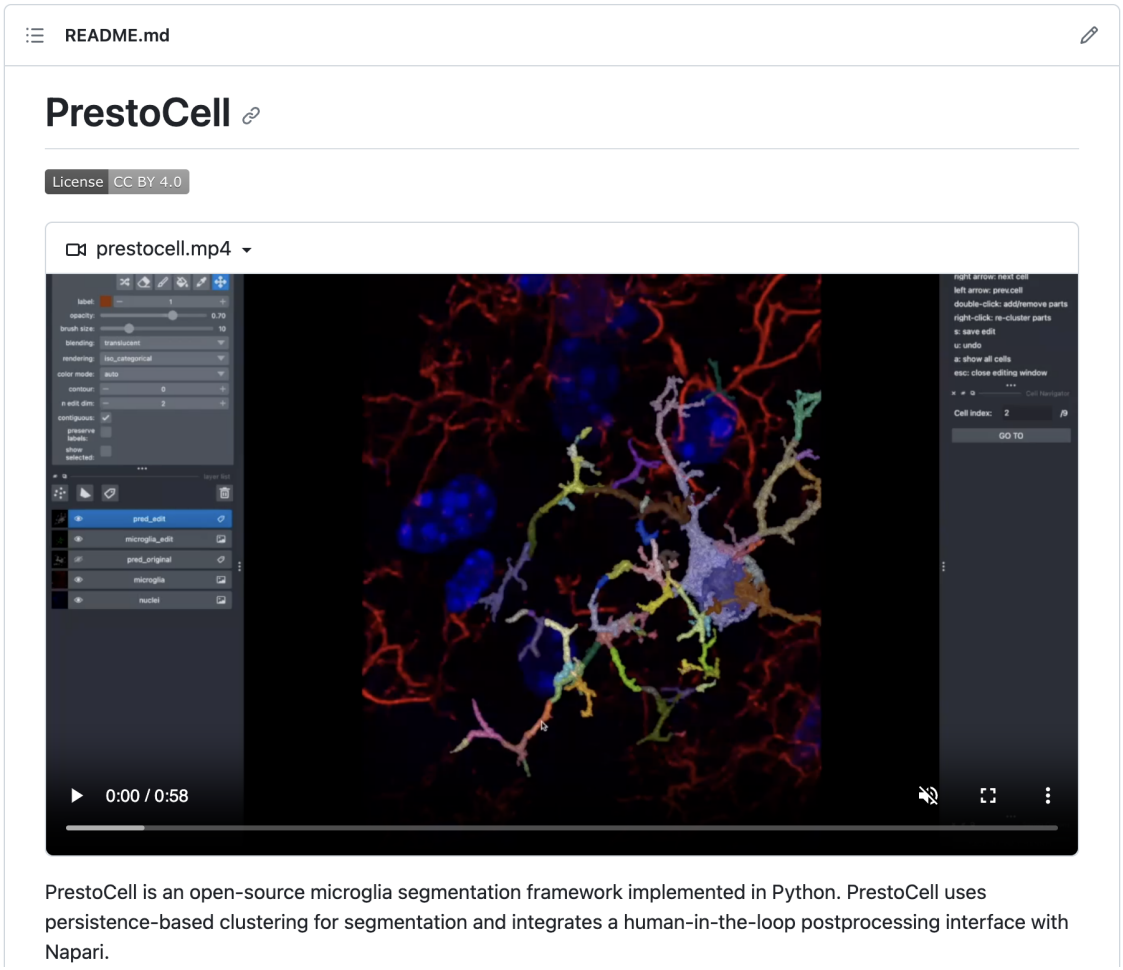


FIGURE 4.2. Caption

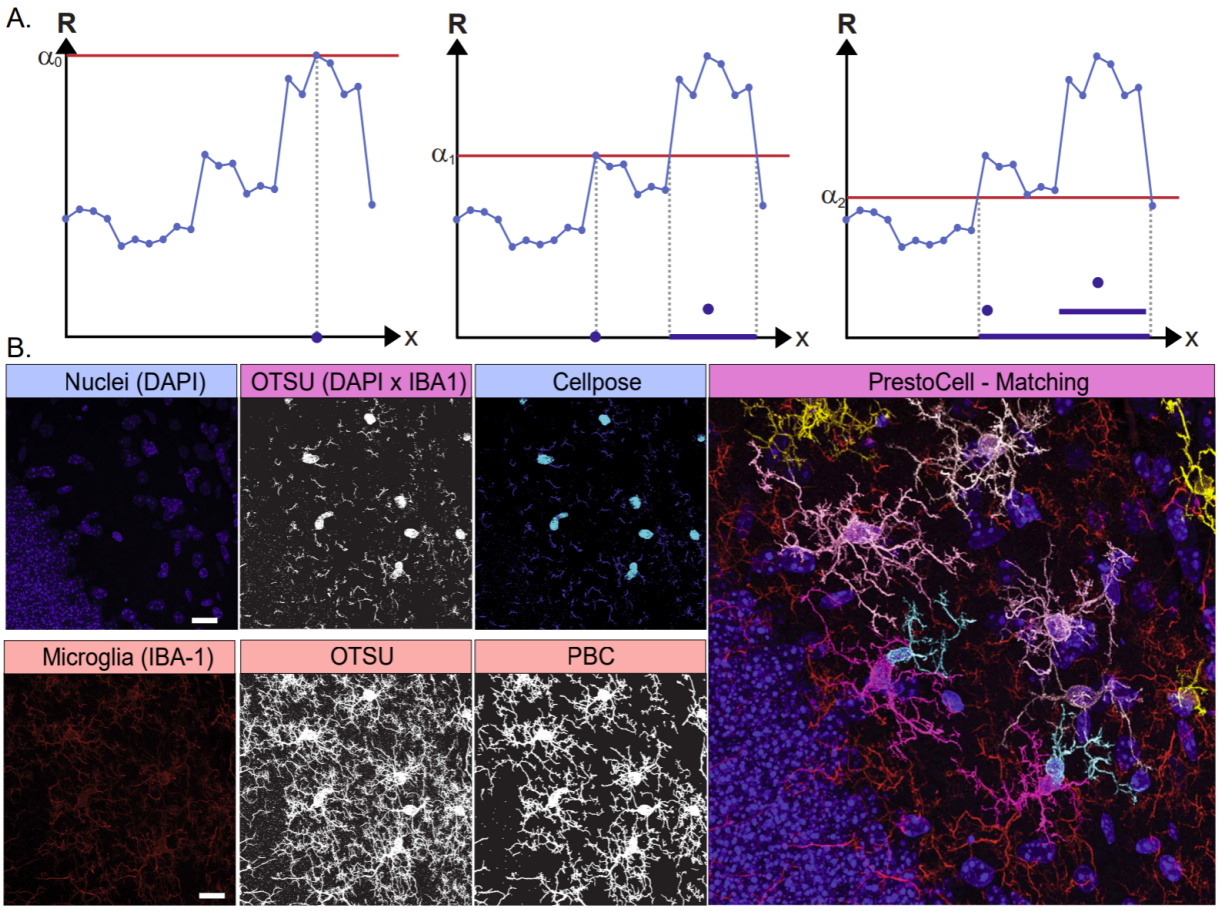


FIGURE 4.3. Persistence Based Clustering and an Overview of PrestoCell.

Figure 4.3 shows an 1D example of PBC. The filtration in PH starts with the threshold α being the maximum value of the data (α_1) with a single component, then decreasing α to the lowest value. When a is decreased to a1, a new component emerges (there are now 2 components). This component dies when a reaches the value α_2. A persistence threshold d is then used to decide if a component is an independent cluster or should be merged with another cluster. When looking for potential merges, we only consider adjacent components because all microglia have 1 connected component. The overview of the PrestoCell process is diagramed (B). PrestoCell splits the raw tiff image z-stack into channels containing the nuclei (DAPI) and microglia (IBA-1). PrestoCell then multiplies the nuclei and microglia channel to identify candidate microglia. Nuclear segmentation is performed using Cellpose and allows for the use to refine the predicted nuclear masks. PrestoCell

then performs PBC on the channel containing the microglia data. The nuclear masks and predicted cells are then matched and user refinement of the IBA-1 mask can be performed

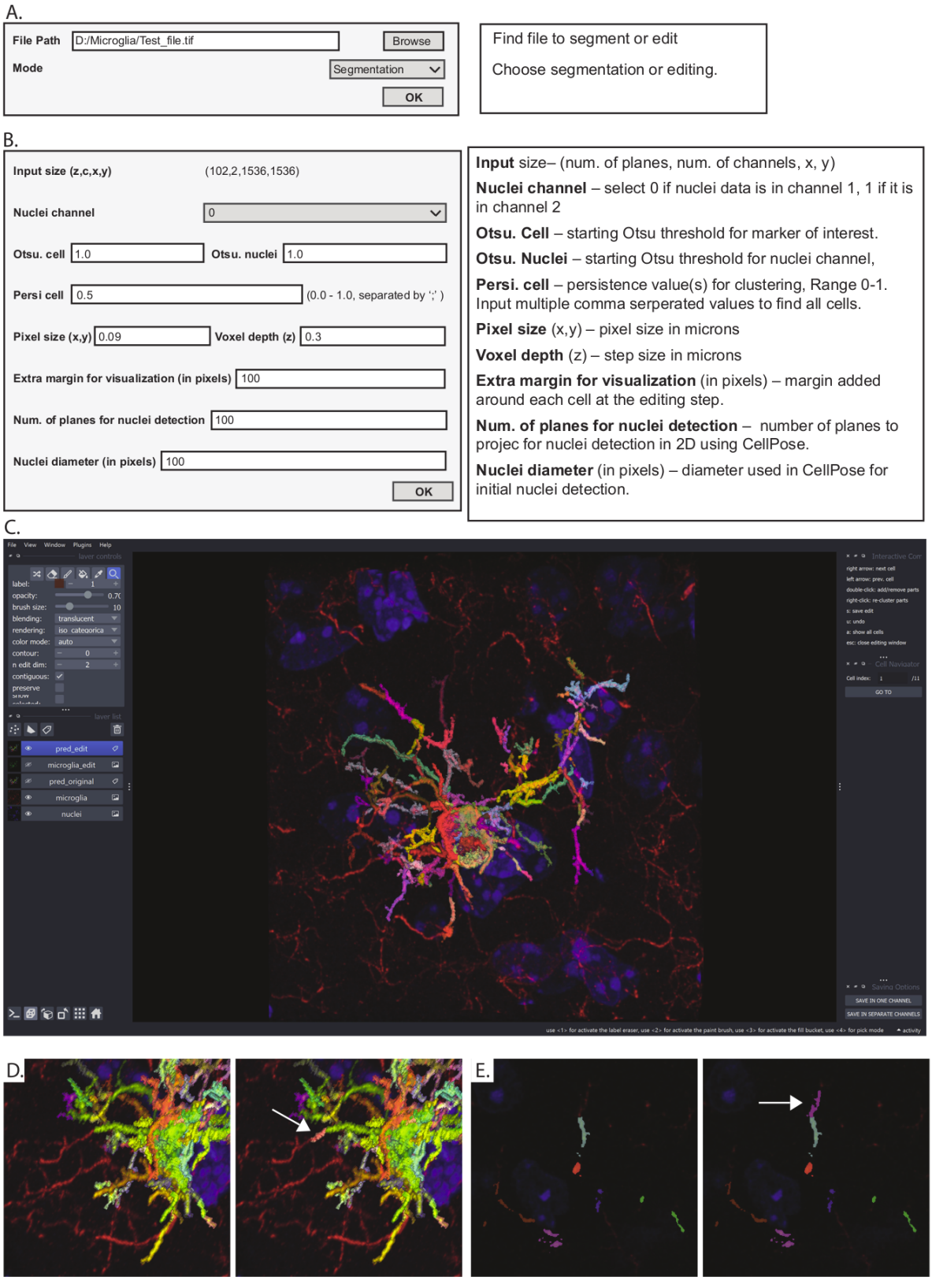


FIGURE 4.4. Use of PrestoCell.

Users interact with PrestoCell through a graphical user interface to select a file they wish to segment (a.B?). The user is promoted to enter additional information about the file and add parameters for the segmentation routine (C). Editing of the segmentation can be performed using Napari in 3D (D), or 2D by using the mouse to indicate what elements should be included with the cell. Here we show that the segmentation mask predicted by PrestoCell is missing elements that can be added simply to build a segmentation acceptable to the user (E).

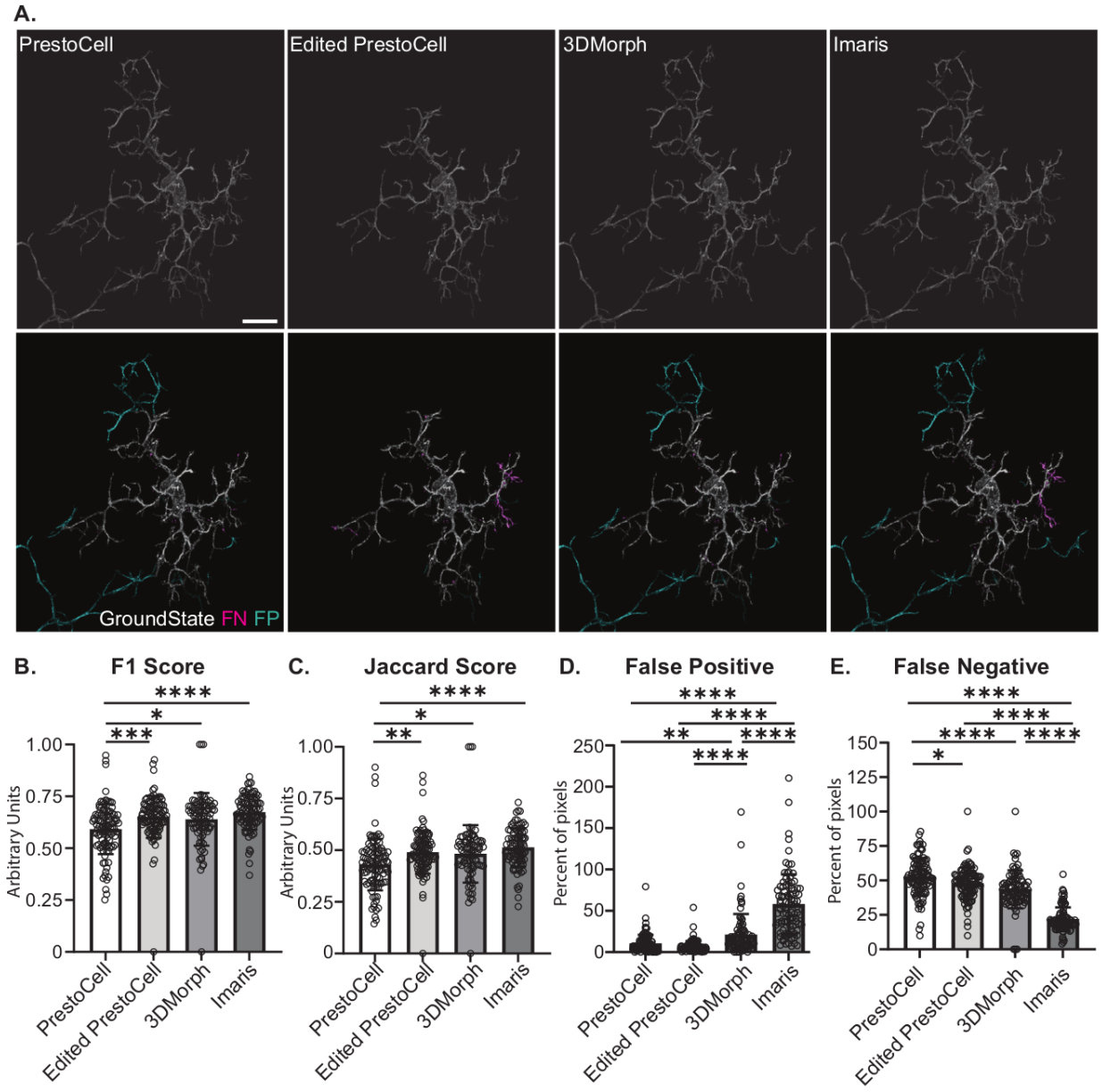


FIGURE 4.5. Comparison of PrestoCell segmentation to other tools.

Segmentation of microglia from light microscopy data produced by PrestoCell, user-edited PrestoCell, 3DMorph, and the LABKIT extension in Imaris (A: top row). The mask generated by each tool is then compared visually to the ground state (A: bottom row, white), and image math is performed to identify false positives ("FP", red) and false negatives ("FN", cyan). Quantitative analysis comparing each segmentation to the ground state truth was performed using F1 score (B), Jaccard score (C), and the percent of false positive (D) and false negative pixels (E).

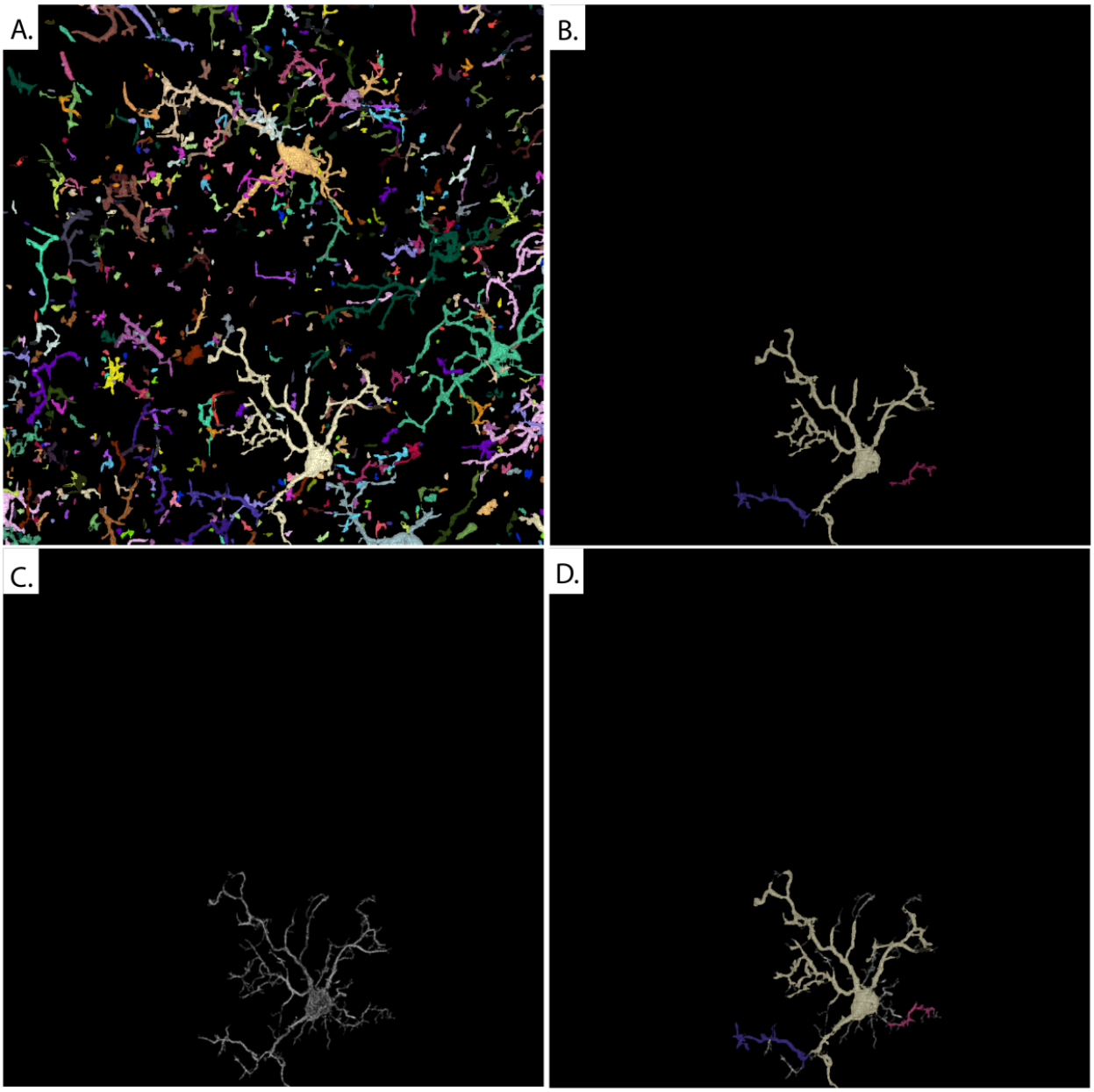


FIGURE 4.6. Incomplete segmentation of microglia with Omnipose.

Incomplete segmentation of microglia with Omnipose. Using Omnipose on our microglia dataset produced segmentation masks throughout the image (A). In general, these masks were discontinuous and heavily fragmented (B). Comparison to an individual cell in the ground state truth (C) revealed that this cell would be represented by many different Omnipose masks (D). Scale bar: 10 μm

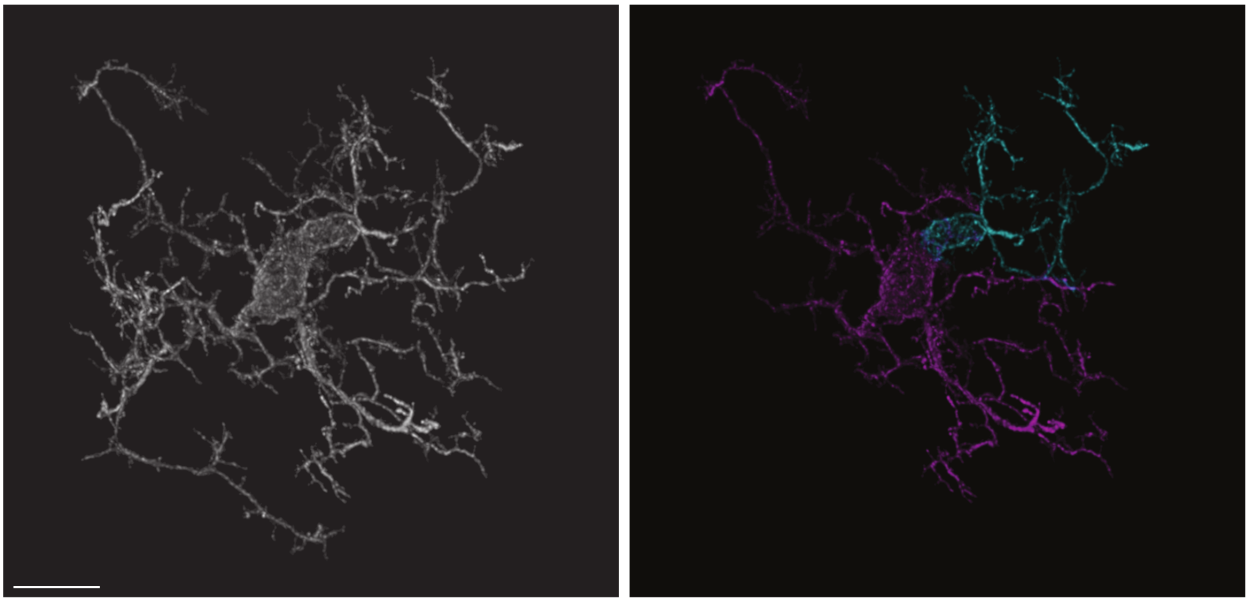


FIGURE 4.7. PrestoCell excels in the segmentation of interacting microglia.

Comparison of microglia that are interacting that were segmented and identified as single cells by 3DMorph (left white). The unedited PrestoCell output demonstrates the power of using nuclear identification, allowing two interacting cells to be resolved (right, pseudo-colored purple, and cyan). Scale bar: 10 μm, representative image of 41 interacting cells identified in 245 discrete masked microglia.

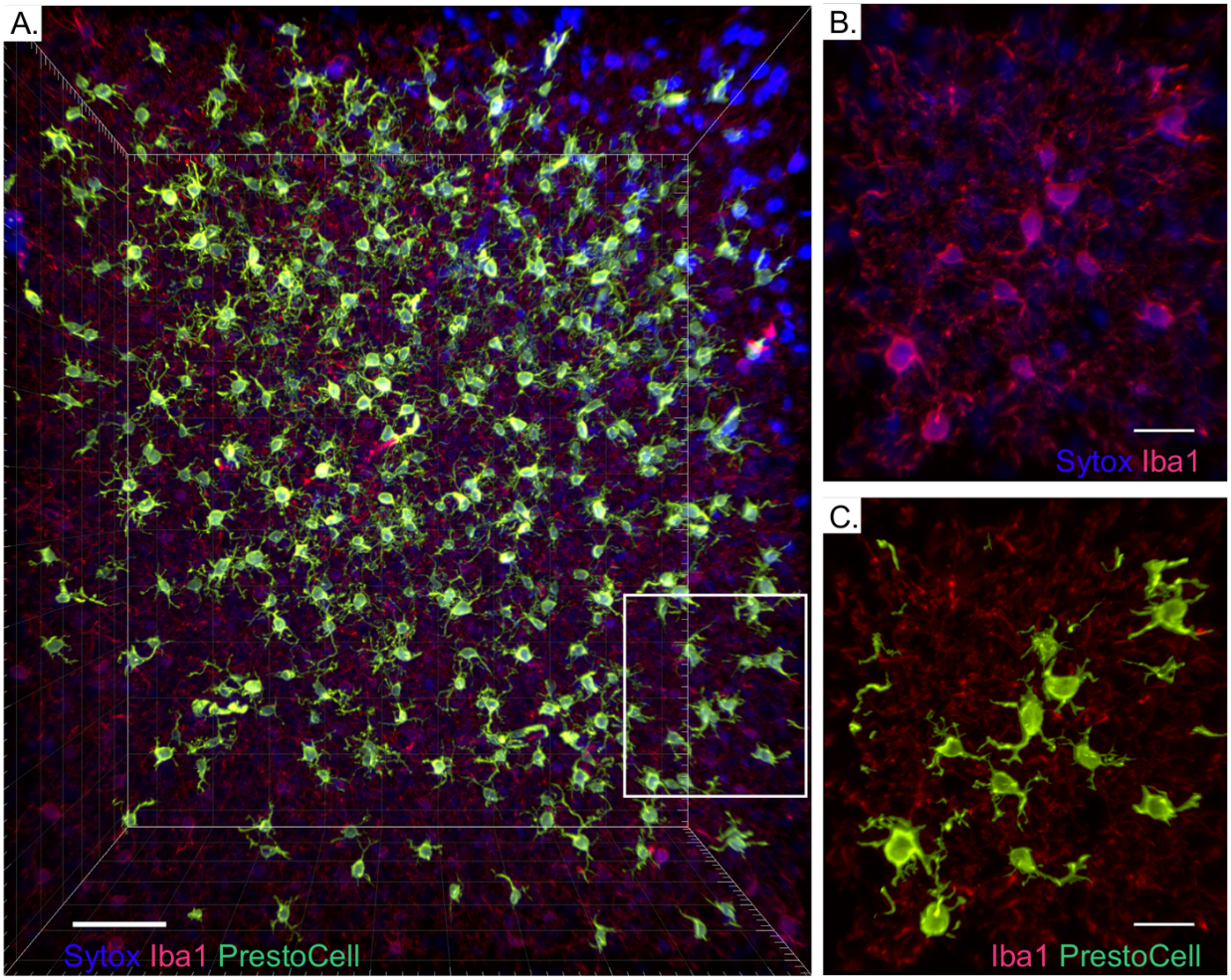


FIGURE 4.8. Segmentation of cells within data acquired by light sheet imaging of the brain.

An overview of the 664 x 664 x 384 μm of brain volume imaged by light sheet microscopy showing the nuclei (DAPI, blue), microglia (IBA-1, red), and PrestoCell segmentations (green). The indicated subregion shows the concordance between DAPI+ microglia (B) and the corresponding segmentations (C). Scale bar: 70 μm for (A); 30 μm for (B) and (C).

4.10. Conclusion and Future Work

We developed PrestoCell, a python-based framework for microglia segmentation. We used persistence-based clustering as the core algorithm to segmentation raw input 3-D imaging data. We used Cellpose to generate nuclei masks which are then used to refine PBC clusters. Clusters that do not overlap any nuclei are removed, clusters that overlap the same nucleus are merged, and clusters

that overlap multiple nuclei are split. We visualize the final clean output in Napari with abundant interactive commands that users can use to modify the predicted cells.

One limitation of the existing pipeline is the identification of branches. Deciding the branch length is especially difficult. We believe that leveraging a large annotated set of training data would be beneficial to producing a better initial machine generated cell predictions. Another direction is to use a shared deep neural network to predict nuclei and microglia simultaneously. Such joint learning strategy can produce more unified nuclei and microglia prediction to remove hard-coded nuclei matching steps.

Bibliography

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, *Hindsight experience replay*, CoRR, abs/1707.01495 (2017).

[2] P. Bankhead, M. Loughrey, J. Fernández, Y. Dombrowski, D. McArt, P. Dunne, et al., *Qupath: Open source software for digital pathology image analysis*, Scientific Reports, 7 (2017), p. 16878.

[3] R. E. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[4] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, *Neural combinatorial optimization with reinforcement learning*, arXiv preprint arXiv:1611.09940, (2016).

[5] Y. Bengio, N. Léonard, and A. C. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, CoRR, abs/1308.3432 (2013).

[6] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, *Curriculum learning*, in Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09, Montreal, Quebec, Canada, 2009, ACM Press, pp. 1–8.

[7] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Inc., 1987.

[8] D. P. Bertsekas and J. N. Tsitsiklis, *Convergence rate and termination of asynchronous iterative algorithms*, in Proceedings of the 3rd International Conference on Supercomputing, Association for Computing Machinery, 1989, p. 461–470.

[9] ———, *An analysis of stochastic shortest path problems*, 16 (1991), p. 580–595.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*, vol. 3 of Optimization and neural computation series, Athena Scientific, 1996.

[11] D. P. Bertsekas and H. Yu, *Stochastic shortest path problems under weak conditions*, 2013.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag, 2006.

[13] J.-D. Boissonnat, F. Chazal, and B. Michel, *Topological data analysis*, in Novel Mathematics Inspired by Industrial Challenges, M. Günther and W. Schilders, eds., Springer International Publishing, Cham, 2022, pp. 247–269.

[14] J.-D. Boissonnat, F. Chazal, and M. Yvinec, *Geometric and topological inference*, vol. 57 of Cambridge Texts in Applied Mathematics, 2018.

[15] F. Chazal, L. Guibas, S. Oudot, and P. Skraba, *Persistence-based clustering in riemannian manifolds*, Journal of the ACM (JACM), 60 (2013), p. Article 41.

[16] F. Chazal and B. Michel, *An introduction to topological data analysis: fundamental and practical aspects for data scientists*, 2021.

[17] J. R. Clough, I. Öksüz, N. Byrne, V. A. Zimmer, J. A. Schnabel, and A. P. King, *A topological loss function for deep-learning based image segmentation using persistent homology*, CoRR, abs/1910.01877 (2019).

[18] D. A. Cox, J. Little, and D. O'Shea, *Using algebraic geometry*, vol. 185 of Graduate Texts in Mathematics, Springer, New York, second ed., 2005.

[19] ———, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer Publishing Company, Incorporated, 3rd ed., 2010.

[20] K. Cutler, C. Stringer, T. Lo, L. Rappez, N. Stroustrup, S. Brook Peterson, et al., *Omnipose: a high-precision morphology-independent solution for bacterial cell segmentation*, Nature Methods, 19 (2022), pp. 1438–1448.

[21] W. Dabney, A. Barreto, M. Rowland, R. Dadashi, J. Quan, M. G. Bellemare, and D. Silver, *The value-improvement path: Towards better representations for reinforcement learning*, vol. 35, 2021, pp. 7160–7168.

[22] R. Dadashi, A. A. Taiga, N. L. Roux, D. Schuurmans, and M. G. Bellemare, *The value function polytope in reinforcement learning*, in Proceedings of the 36th International Conference on Machine Learning, vol. 97 of Proceedings of Machine Learning Research, PMLR, 2019, pp. 1486–1495.

[23] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, *Learning combinatorial optimization algorithms over graphs*, in Advances in Neural Information Processing Systems, 2017.

[24] H. Dai, Z. Kozareva, and B. Dai, *Combinatorial optimization with graph convolutional networks and guided tree search*, in Advances in Neural Information Processing Systems, 2018.

[25] G. Dantzig, *Linear programming and extensions*, Princeton Univ. Press, 1963.

[26] S. David, S. Julian, and S. et al., *Mastering the game of go without human knowledge*, Nature, 550 (2016).

[27] J. A. De Loera, R. Hemmecke, and M. Köppe, *Algebraic and geometric ideas in the theory of discrete optimization*, vol. 14 of MOS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA; Mathematical Optimization Society, Philadelphia, PA, 2013.

[28] J. A. De Loera and S. Onn, *All rational polytopes are transportation polytopes and all polytopal integer sets are contingency tables*, in Integer Programming and Combinatorial Optimization, 10th International IPCO Conference, New York, NY, USA, June 7-11, 2004, Proceedings, G. L. Nemhauser and D. Bienstock, eds., vol. 3064 of Lecture Notes in Computer Science, Springer, 2004, pp. 338–351.

[29] ———, *The complexity of three-way statistical tables*, SIAM J. Comput., 33 (2004), pp. 819–836.

[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, CoRR, abs/2010.11929 (2020).

[31] G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, *Reinforcement learning in large discrete action spaces*, CoRR, abs/1512.07679 (2015).

[32] J. Epp, Y. Niibori, H. Liz Hsiang, V. Mercaldo, K. Deisseroth, S. Josselyn, et al., *Optimization of clarity for clearing whole-brain and other intact organs*, eNeuro, 2 (2015).

[33] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, *Reverse curriculum generation for reinforcement learning*, CoRR, abs/1707.05300 (2017).

[34] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, CoRR, abs/1802.09477 (2018).

[35] R. B. Gabrielsson, V. Ganapathi-Subramanian, P. Skraba, and L. J. Guibas, *Topology-aware surface reconstruction for point clouds*, CoRR, abs/1811.12543 (2018).

[36] R. B. Gabrielsson, B. J. Nelson, A. Dwaraknath, P. Skraba, L. J. Guibas, and G. E. Carlsson, *A topology layer for machine learning*, CoRR, abs/1905.12200 (2019).

[37] X. Guo, S. Singh, R. L. Lewis, and H. Lee, *Deep learning for reward design to improve monte carlo tree search in ATARI games*, in Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, S. Kambhampati, ed., IJCAI/AAAI Press, 2016, pp. 1519–1525.

[38] T. Hammond, C. Dufort, L. Dissing-Olesen, S. Giera, A. Young, A. Wysoker, et al., *Single-cell rna sequencing of microglia throughout the mouse lifespan and in the injured brain reveals complex cell-state changes*, Immunity, 50 (2019), pp. 253–71.e6.

[39] T. D. Hansen, P. B. Miltersen, and U. Zwick, *Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor*, J. ACM, 60 (2013).

[40] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, CoRR, abs/1512.03385 (2015).

[41] D. Held, X. Geng, C. Florensa, and P. Abbeel, *Automatic goal generation for reinforcement learning agents*, CoRR, abs/1705.06366 (2017).

[42] C. Hennessey, C. E. Keogh, M. Barboza, I. Brust-Mascher, T. A. Knotts, J. A. Sladek, et al., *Neonatal enteropathogenic escherichia coli infection disrupts microbiota-gut-brain axis signaling*, Infection and Immunity, 89 (2021), pp. e00059–21.

[43] T. Hester, M. Vecerík, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. P. Agapiou, J. Z. Leibo, and A. Gruslys, *Learning from demonstrations for real world reinforcement learning*, CoRR, abs/1704.03732 (2017).

[44] S. Hoşten and S. Sullivant, *Gröbner bases and polyhedral geometry of reducible and cyclic models*, J. Combin. Theory Ser. A, 100 (2002), pp. 277–301.

[45] A. J. Hoffman, *On simple linear programming problems*, in Proc. Sympos. Pure Math., Vol. VII, Amer. Math. Soc., Providence, R.I., 1963, pp. 317–327.

[46] R. A. Howard, *Dynamic Programming and Markov Processes*, MIT Press, 1960.

[47] X. Hu, F. Li, D. Samaras, and C. Chen, *Topology-preserving deep image segmentation*, in Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.

[48] S. M. Kakade, *A natural policy gradient*, in Proceedings of the 14th International Conference on Neural Information Processing Systems, T. Dietterich, S. Becker, and Z. Ghahramani, eds., vol. 14, MIT Press, 2002, p. 1531–1538.

[49] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, 2015.

[50] V. Konda and J. Tsitsiklis, *Actor-critic algorithms*, in Proceedings of the 12th International Conference on Neural Information Processing Systems, S. Solla, T. Leen, and K. Müller, eds., MIT Press, 1999, pp. 1008–1014.

[51] W. Kool, H. van Hoof, and M. Welling, *Attention, learn to solve routing problems!*, in International Conference on Learning Representations, 2019.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012.

[53] Q. Li and B. Barres, *Microglia and macrophages in brain homeostasis and disease*, Nature Reviews Immunology, 18 (2018), pp. 225–242.

[54] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning.*, in ICLR, 2016.

[55] T. Liu, L. Yang, X. Han, X. Ding, J. Li, and J. Yang, *Local sympathetic innervations modulate the lung innate immune responses*, Science Advances, 6 (2020).

[56] W. Liu, H. Guo, W. Zhang, Y. Zang, C. Wang, and J. Li, *Toposeg: Topology-aware segmentation for point clouds*, in Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, L. D. Raedt, ed., International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 1201–1208. Main Track.

[57] Y. Mansour and S. Singh, *On the complexity of policy iteration*, in Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999, p. 401–408.

[58] S. McAleer, F. Agostinelli, A. Shmakov, and P. Baldi, *Solving the rubik's cube with deep reinforcement learning and search*, Nature Machine Intelligence, 1 (2019), pp. 269–277.

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, *Playing atari with deep reinforcement learning*, CoRR, abs/1312.5602 (2013).

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Human-level control through deep reinforcement learning.*, Nature, 518 (2015), pp. 529–533.

[61] D. Müller, I. Soto-Rey, and F. Kramer, *Towards a guideline for evaluation metrics in medical image segmentation*, BMC Research Notes, 15 (2022), p. 210.

[62] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, *Overcoming exploration in reinforcement learning with demonstrations*, CoRR, abs/1709.10089 (2017).

[63] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, *Curriculum learning for reinforcement learning domains: A framework and survey*, CoRR, abs/2003.04960 (2020).

[64] M. Nazari, M. Norouzi, and O. Vinyals, *Learning tsp requires rethinking generalization*, arXiv preprint arXiv:1810.01222, (2018).

[65] T. Nogueira, S. Rao, C. Zhang, A. Parameswaran, and J. Jiang, *Reinforcement learning on web interfaces using workflow-guided exploration*, in Proceedings of the 13th ACM International Conference on Web Search and Data Mining, 2020.

[66] G. Norris, N. Derecki, and J. Kipnis, *Microglial sholl analysis*, 2014.

[67] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, *An algorithmic perspective on imitation learning*, CoRR, abs/1811.06711 (2018).

[68] F. G. Pedregosa, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay, *Scikit-learn: Machine learning in python*, Journal of Machine Learning Research, 12 (2011), p. 5.

[69] M. Prinz, T. Masuda, M. Wheeler, and F. Quintana, *Microglia and central nervous system–associated macrophages—from origin to disease modulation*, Annual Review of Immunology, 39 (2021), pp. 251–277.

[70] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc, 1994.

[71] M. L. Puterman and M. C. Shin, *Modified policy iteration algorithms for discounted markov decision problems*, Management Science, 24 (1978), pp. 1127–1137.

[72] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, 2018.

[73] S. Ross and J. A. Bagnell, *Reinforcement and imitation learning via interactive no-regret learning*, CoRR, abs/1406.5979 (2014).

[74] S. Ross, G. J. Gordon, and J. A. Bagnell, *No-regret reductions for imitation learning and structured prediction*, CoRR, abs/1011.0686 (2010).

[75] R. Rudolf, *On Monge sequences in d-dimensional arrays*, Linear Algebra Appl., 268 (1998), pp. 59–70.

[76] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, *Imagenet large scale visual recognition challenge*, CoRR, abs/1409.0575 (2014).

[77] R. Sankowski, G. Monaco, and M. Prinz, *Evaluating microglial phenotypes using single-cell technologies*, Trends in Neurosciences, 45 (2022), pp. 133–144.

[78] S. Schaal, *Learning from demonstration*, in Advances in Neural Information Processing Systems, M. Mozer, M. Jordan, and T. Petsche, eds., vol. 9, MIT Press, 1996.

[79] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay*, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[80] B. Scherrer, *Improved and generalized upper bounds on the complexity of policy iteration*, in Proceedings of the 26th International Conference on Neural Information Processing Systems, 2013, p. 386–394.

[81] U. Schmidt, M. Weigert, C. Broaddus, and G. Myers, *Cell detection with star-convex polygons*, in Medical Image Computing and Computer Assisted Intervention – MICCAI 2018, Cham, 2018, Springer International Publishing.

[82] L. S. Shapley, *Stochastic games*, Proceedings of the National Academy of Sciences, 39 (1953), pp. 1095–1100.

[83] D. Sholl, *Dendritic organization in the neurons of the visual and motor cortices of the cat*, Journal of Anatomy, 87 (1953), pp. 387–406.

[84] D. Silver, A. Huang, and C. J. M. et al., *Mastering the game of go with deep neural networks and tree search*, Nature, 529 (2016).

[85] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, and A. G. et al., *A general reinforcement learning algorithm that masters chess, shogi, and go through self-play*, Science, 362 (2018), pp. 1140–1144.

[86] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, *Mastering the game of go without human knowledge*, Nat., 550 (2017), pp. 354–359.

[87] R. P. Stanley, *An introduction to hyperplane arrangements*, in Geometric combinatorics, vol. 13 of IAS/Park City Math. Ser., Amer. Math. Soc., Providence, RI, 2007, pp. 389–496.

[88] E. Stelzer, F. Strobl, B.-J. Chang, F. Preusser, S. Preibisch, K. McDole, et al., *Light sheet fluorescence microscopy*, Nature Reviews Methods Primers, 1 (2021).

[89] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, *Cellpose: a generalist algorithm for cellular segmentation*, Nature Methods, 18 (2021), pp. 100–106.

[90] B. Sturmfels, *Gröbner bases and convex polytopes*, vol. 8 of University Lecture Series, American Mathematical Society, Providence, RI, 1996.

[91] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, second ed., 2018.

[92] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, *Policy gradient methods for reinforcement learning with function approximation*, in Proceedings of the 12th International Conference on Neural Information Processing Systems, MIT Press, 1999, p. 1057–1063.

[93] A. van den Oord, O. Vinyals, and k. kavukcuoglu, *Neural discrete representation learning*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017.

[94] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, CoRR, abs/1509.06461 (2015).

[95] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, CoRR, abs/1706.03762 (2017).

[96] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, *Grandmaster level in starcraft II using multi-agent reinforcement learning*, Nat., 575 (2019), pp. 350–354.

[97] F. Wang, H. Liu, D. Samaras, and C. Chen, *Topogan: A topology-aware generative adversarial network*, Berlin, Heidelberg, 2020, Springer-Verlag, p. 118–136.

[98] Z. Wang, N. de Freitas, and M. Lanctot, *Dueling network architectures for deep reinforcement learning*, CoRR, abs/1511.06581 (2015).

[99] C. J. C. H. Watkins and P. Dayan, *Q-learning*, Machine Learning, 8 (1992), pp. 279–292.

[100] R. Williams and J. Peng, *Function optimization using connectionist reinforcement learning algorithms*, Connection Science, 3 (1991), pp. 241–268.

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

[102] Y. Ye, *The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate*, Math. Oper. Res., 36 (2011), p. 593–603.

[103] V. A. Yemelichev, M. M. Kovalëv, and M. K. Kravtsov, *Polytopes, graphs and optimisation*, Cambridge University Press, Cambridge, 1984. Translated from the Russian by G. H. Lawden.

[104] E. York, J. LeDue, L. Bernier, and B. MacVicar, *3dmorph automatic analysis of microglial morphology in three dimensions from ex vivo and in vivo imaging*, eNeuro, 5 (2018).

[105] J. Zhang and K. Cho, *Query-efficient imitation learning for end-to-end autonomous driving*, CoRR, abs/1605.06450 (2016).

[106] G. Ziegler, *Lectures on Polytopes*, Graduate Texts in Mathematics, Springer New York, 2012.