VALUE METHODS FOR EFFICIENTLY SOLVING STOCHASTIC GAMES OF COMPLETE AND INCOMPLETE INFORMATION

A Thesis
Presented to
The Academic Faculty

by

Liam Charles MacDermed

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
December 2013

VALUE METHODS FOR EFFICIENTLY SOLVING STOCHASTIC GAMES OF COMPLETE AND INCOMPLETE INFORMATION

Approved by:

Charles L. Isbell Jr., Advisor
School of Interactive Computing
Georgia Institute of Technology

Geoff Gordon
School of Computer Science
Carnegie Mellon University

Maria-Florina Balcan
School of Computer Science
Georgia Institute of Technology

Lora Weiss
Georgia Tech Research Institute
Georgia Institute of Technology

C.Karen Liu
School of Interactive Computing
Georgia Institute of Technology

Date Approved: August 2013

To my parents, Charles Mac Dermed and Varya Simpson.

Thank you for giving me the world.

ACKNOWLEDGEMENTS

This dissertation is the culmination of a long and arduous journey made navigable by my friends, family, colleagues, and mentors. Among them I thank: My advisor, Charles Isbell, who gave me the precious gift of intellectual freedom to pursue my own research agenda while giving me the guidance and support I needed to succeed. My Georgia Tech Research Institute advisor Lora Weiss, who spent many hours over many years keeping my research focused, and my publications crisp. Jerry Feldman, who started me on my path towards research. My lab-mates, Michael Holmes, Chip Mappus, David Roberts, Sooraj Bhat, Peng Zang, Arya Irani, Chris Simpkins, Jon Scholz, Luis Carlos Cobo, Ashley Edwards, and Kaushik Subramanian, who improved both the quality of my life and the quality of my work. I am grateful to all my co-authors but would particularly like to thank Karthik Narayan and Josh Letchford. I am also grateful to have had as a committee member Geoff Gordon, whose research inspired my own. Above all I would like to thank Yee Chieh Chew, who's patience, love, and optimism gave me the fortitude to complete this expedition.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Multi-agent reinforcement learning (MARL) poses the same planning problem as traditional reinforcement learning (RL): What actions over time should an agent take in order to maximize its rewards? MARL tackles a challenging set of problems that can be better understood by modeling them as having a relatively simple environment but with complex dynamics attributed to the presence of other agents who are also attempting to maximize their rewards. A great wealth of research has developed around specific subsets of this problem, most notably when the rewards for each agent are either the same or directly opposite each other. However, there has been relatively little progress made for the general problem. This thesis address this lack.

Our goal is to tackle the most general, least restrictive class of MARL problems. These are general-sum, non-deterministic, infinite horizon, multi-agent sequential decision problems of complete and incomplete information. Towards this goal, we engage in two complementary endeavors: the creation of tractable models and the construction of efficient algorithms to solve these models. We tackle three well known models: stochastic games, decentralized partially observable Markov decision problems, and partially observable stochastic games. We also present a new fourth model, Markov games of incomplete information, to help solve the partially observable models.

For stochastic games and decentralized partially observable Markov decision problems, we develop novel and efficient value iteration algorithms to solve for game theoretic solutions. We empirically evaluate these algorithms on a range of problems, including well known benchmarks and show that our value iteration algorithms perform better than current policy iteration algorithms. Finally, we argue that our approach is easily extendable to new models and solution concepts, thus providing a foundation for a new class of multi-agent value iteration algorithms.

CHAPTER I

INTRODUCTION

Multi-agent reinforcement learning (MARL) [34] poses the same learning problem as traditional reinforcement leaning: How can agents learn to maximize their rewards through interaction with their environment? Traditional reinforcement learning has formalized this problem by modeling the environment as a Markov decision process (MDP) where the outcome of an agent's actions are fully explained by the state the world is in. MDPs work well for modeling simple dynamical systems but have a hard time modeling the near boundless complexity of the real world. A particularly interesting class of environments that MDPs model poorly can be understood, instead, by modeling them as a relatively simple process, but with complex dynamics attributed to the presence of other agents who are also attempting to maximize their rewards. MARL addresses the reinforcement learning problem in these environments.

We start by introducing the field of multi-agent learning and explain why game theoretic solutions are both reasonable and useful for our problem (Chapter 1). After reviewing relevant background material (Chapter 2), we present achievable set methods under full observability (Chapter 3). We then extend our results to games of incomplete information, first by defining a new model which explicitly enumerates the beliefs of all agents (Chapter 4). We then show how to solve this new model for the case when rewards are shared (Chapter 5).

1.1 Motivation

The field of game theory has long addressed the question of how to model and predict the behavior of multiple self interested agents acting in the same environment. Game theory is a mature discipline containing some of the best answers we have to how competing agents will act. MARL has taken much, as it should, from game theory. However, there are two fundamental problems preventing game theory from being directly applied to the MARL

problem.

1. Few algorithms that compute general-sum game theoretic solutions efficiently scale beyond toy problems.

2. Game theory assumes agents are fully rational, meaning they have infinite intelligence and computational power, an assumption that clearly does not hold in the real world.

Due in large part to these two problems, game theory focuses on describing the properties of an optimal joint-policy, while MARL is interested in prescribing a particular optimal policy for an individual agent. This problem is even more acute in unknown environments. This thesis primarily attempts to address the first shortcoming and utilize game theory's strong theoretical foundation to solve the multi-agent reinforcement learning problem. We accomplish this by designing scalable algorithms to compute game theoretic solutions for scalable models.

MARL has very similar goals to traditional game theory but deals with the unattainability of rationality in a principled yet pragmatic way. Rationally leads to a number of conclusions that are unreasonable for practical MARL problems. However, we should be careful not to immediately throw out such an assumption. MARL is predicated on understanding the world in terms of other agents maximizing their own rewards. Rational agents inhabit the idealization of this model. If we believe that other agents have bounded-rationality we need to be careful to clearly redefine our model. On one hand, rationality is the goal all agents should aspire to. On the other hand, rationality is unattainable, so when designing algorithms for the real world we aim for robust algorithms - algorithms that perform well even when our model is inaccurate and our computing power insufficient.

The first and most obvious failing of rationality is that agents may not be able to compute their optimal course of action. Worse agents may not be able to accurately compute the value of a particular policy (even when it's presented to them) and may erroneously conclude suboptimal policies are in equilibrium, or break equilibria by taking suboptimal actions. To mitigate these problems this thesis assumes agents are willing to tolerate some small ϵ error in their utility calculations and therefore will accept ϵ-equilibria as solutions. A more formal

account of this assumption is given in Section 3.4.1.

Another failing of rationality, is that it assumes that the world model (*a.k.a.* environment dynamics) is commonly known to all agents. These problems are known as model-based. MARL is often interested in cases where agents have to simultaneously learn about the world while maximizing rewards, leading to the well known exploration vs. exploitation trade off [39]. These problems are known as model-free. While game theory can by applied to model-free problems by viewing them as an incomplete information game with a type for each possible observation trace, such an exact approach (*a.k.a.* full Bayesian policy [43]) is impractical even in the single agent case. We believe that approximation techniques similar to those used for traditional reinforcement learning will transfer to MARL but before such approaches can be used we need to solve the model-based problem.

Third, rationality implies an unambiguous consensus about the definition of optimal behavior. This implies that agents will inherently agree on a course of action. Ironically, while game theorists agree that rational agents will agree on the definition of optimality the theorists themselves can not agree on a definition, with many different competing views of ideal behavior and solution concepts. It is therefore unreasonable to expect sub-rational agents to perfectly predict each other's behavior. It seems reasonable that with communication (either explicitly or implicitly through repeated interactions) agents would settle into policies in equilibrium (where no player can unilaterally improve their utility). However, there may be many equilibria so which particular equilibrium is targeted and agreed upon (equilibrium selection) poses a huge problem. Game theory is able to tackle the problem through axioms of fairness leading to bargaining solutions [56]. However, once again, many such bargaining solutions exist leading to further question the applicability of the approach. In the face of this uncertainly a large open research question for multi-agent learning is how to efficiently and effectively converge onto an equilibrium. This thesis does not attempt to directly answer this question, instead we compute the *entire* set of equilibria (the achievable set), guaranteeing that we find the solution no matter which definition of optimal is used.

Despite the shortcomings of game theory it is the most vetted and well grounded method for understanding multi-agent interactions. Ultimately it is the disconnect between the

assumption of rationality and the intractability of the resulting solutions that most harms the credibility of game theoretic solutions. It is the attempt of this thesis to bridge this gap by developing efficient algorithms and compact models that compute game theoretic solutions while respecting the limitations of real world agents.

1.2 Problem

Our primary objective in this thesis is to make it feasible to solve larger and more complex sequential multi-agent reinforcement learning problems with an infinite horizon. Towards this goal we engage in two complementary endeavors: the creation of tractable models and the construction of efficient algorithms to solve these models.

Stochastic or Markov games [48] are a natural extension to Markov decision processes (MDPs) for multi-agent reinforcement learning, with actions being joint actions and rewards being a vector of rewards (one to each player). They can model a broad range of interesting problems including oligopoly and monetary policy [5], network security and utilization [58], and phenotype-expression patterns in evolving microbes [91]. Unfortunately, typical applications of stochastic games are currently limited to very small and simple games. The primary bottleneck for solving larger stochastic games games is the efficiency and efficacy of current algorithms. In addition, previous attempts at solving MARL problems have focused on limited subsets of the general models. For example, there has been a great deal of work on zero-sum games and common-payoff games [82, 67]. In this dissertation we solve both general sum problems and, when there is only partial observability, problems with common payoff. We tackle solving stochastic games in Chapter 3.

While stochastic games are useful they assume complete knowledge of the environment. In most real world problems agents are not able to completely observe the world around them. This is particularly true when reasoning about other agent's goals. The natural extension of stochastic games to these incomplete information scenarios is the model known as partially observable stochastic games (POSG). POSGs are extremely complicated and attempts to solve POSGs have all met large computational barriers. In order to solve POSGs we address three complementary questions: Why are POSGs very challenging to

solve efficiently? How can we go about approximating POSGs in order to tractably solve them? How can we solve an approximation efficiently? Chapter 4 addresses the first two questions, and Chapter 5 addresses the final question.

For all models, we assume each agent is ϵ-rational, in that they are willing to be within ϵ utility from optimal in order to maintain equilibrium. Our goal is to produce a joint policy that is Pareto-optimal (no other viable joint policy gives a player more utility without lowering another player's utility), fair (players agree on the joint policy), and in equilibrium (no player can gain by deviating from the joint policy). The precise meaning of fair is intentionally left unspecified for generality. This solution concept is the game-theoretic solution. While there are many potential equilibrium solution concepts the final algorithms in this dissertation are only directly applicable to concepts with convex solution spaces. In particular we target correlated equilibria in the fully observable case, and agent normal-form correlated equilibria in the partially observable case. This solution is both broadly applicable and useful.

1.3 Approach

The algorithms developed in this dissertation are value-based dynamic programming algorithms, taking their heritage from value iteration and the Bellman-equation. At each iteration we compute an estimate for the value (*a.k.a.* utility) that can be achieved in each state. This estimate is improved over successive iterations until we can make guarantees about the high accuracy of the values computed.

Our value-based approach is in contrast to the current dominate policy based lines of MARL research which attempts to directly optimize agents' policies [59, 64, 87]. Unlike policy-based approaches, we do not need to remember policy information between iterations. For MARL, value based approaches can have significant advantages over their policy-based counterparts. With multiple agents the joint-policy space tends to be exponentially larger than in the single-agent case and attempting to optimize each agent's policy independently is typically a non-convex problem leading to local optima.

We extend the Bellman heritage to stochastic games by utilizing the notion of an

achievable-set function – the multi-agent analogy to the single agent value-function. In MDPs, the value-function represents the optimal achievable utility for every state. When there are multiple agents it is not always possible to simultaneously maximize all agent's utilities, so the multi-agent achievable-set-function must represent all possible combinations of values. Because we now operate on sets for each state instead of scalar values the Bellman backup becomes more complicated, although conceptually it remains unchanged. Along with values transforming into sets, there are two additional major difficulties that arise in the multi-agent problem. We must ensure that policies remain in equilibrium and that the complexity of the sets does not overwhelm the problem. We solve the first problem through linear programming and the second through polytope approximation. In each case we are careful to prove correctness and bound the error introduced, allowing us to have strong guarantees about the quality of our solutions.

When the world is only partially observable the problem becomes much harder. We argue that the primary complicating feature of POSGs is that they deny a compact belief. This means that agents must remember their entire history and reason about other agent's entire histories. Because of this, we believe POSGs are too general. Instead we develop a slightly weaker model which still captures the important aspects of POSGs, but permits tractable solution algorithms. Our new model – Markov games of incomplete information (MaGII)– is based on making the assumption that beliefs are bounded. While this assumption is simple, the consequences are non-trivial and allows us to extend existing algorithms to solve new classes of problems. While the theory behind MaGIIs is developed for the general-sum problem, we design an efficient algorithm for the common-payoff case based on point based value iteration for POMDPs. This dissertation results in both a framework to think about how to tackle general multi-agent problems, as well as concrete value-based algorithms able to efficiently compute solutions to previously unsolved problems.

1.4 Thesis

This dissertation presents theory, algorithms, and empirical results to support the following thesis:

> **Value methods can efficiently compute useful game theoretic solutions to general multi-agent reinforcement learning problems.**

We clarifying the meaning of each part of this thesis as follows:

- "Value methods ..."

  Value-function based approaches have proven effective for single agent problems, but have proven to be less effective for general-sum multi-agent problems. We extend the notion of value-functions to multi-agent domains and develop value-iteration algorithms to operate on these extensions.

- "... can efficiently compute ..."

  We show that our algorithms scale polynomially with respect to the size of the model, while having low enough degree for it to be practical to solve interesting problems.

- "... useful game theoretic solutions ..."

  In this thesis we target correlated equilibria in the fully observable case, and agent normal-form correlated equilibria in the partially observable case. We provide tight guarantees about our error, and we argue that these solution concepts are viable and useful for real world agents.

- "... to general multi-agent reinforcement learning problems."

  The goal of this thesis is to tackle the most general, least restrictive class of MARL problems. These are general-sum, non-deterministic, infinite horizon, multi-agent sequential decision problems of complete and incomplete information. To the author's knowledge, no other body of work has directly addressed this broad class of problems. In order to tackle partially observable problems we develop several new models that better capture and illuminate the complexities involved with these problems.

CHAPTER II

BACKGROUND

Sequential decision problems are, at their heart, optimization problems. The goal is to find a policy which maximizes an agent's utility. Two factors make this problem particularly challenging: (1) The number of distinct policies grows exponentially with time. (2) The relationship between policies and their utility can be quite complex or even partially unknown. Because of these complications, standard general optimization techniques are typically not directly applicable.

Sequential decision problems use various formal models to describe the relationship between policies and utility. These models typically describe the dynamics of the environment along with the agent's goals in the form of rewards. Such models assume agents are rational and that their Von NeumannMorgenstern utility [89] can be broken down into a sum of (possibly discounted) rewards. This model is either specified ahead of time (a model-based problem) or has an assumed form with unknown values that is learned over time (a model-free problem). This thesis deals with model-based problems. Models should be able to accurately and compactly represent the problem at hand, while permitting tractable algorithms to solve them. This has led to research into a number of different formal models for different decision problems. This thesis utilizes and extends many of these models, which are defined and discussed in detail in this chapter.

There are many classes of sequential decision problems. In this thesis we are interested in solving problems that are: multi-agent (more than one agent simultaneously making decisions and optimizing their policy), model-based (the world dynamics are known ahead of time to all agents), infinite horizon with discounted reward (there is no planning horizon, but rewards decay exponentially over time). We also assume all agents are ϵ-rational (meaning that all agents are able to maximize their own utility to within ϵ of optimal, and are able to predict the thought process of all other agents). While we do not assume there is any means

of communication beyond what is explicitly given by the model, we do assume that agents can observe a shared random variable equivalent to having access to the same random number generator with the same seed. Rationality could be seen as implying a shared random variable.

This chapter presents background on models that we use to help solve our class of problems. First, in Section 2.1, we examine multi-agent non-sequential decision problems (stage games). We also describe what a solution should look like in these problems because when there are multiple agents the definition of an optimal solution is not self-evident. We look at three basic stage games: normal form games, Bayesian games, and decentralized decisions. In Section 2.2 we discuss the sequential version of each of these models. Because a sequential problem can be viewed as a single shot problem where the decision is an entire policy, the solutions concepts described for stage games translate well (with minor additional considerations) over to their sequential counterparts.

2.1 Stage games

Stage games are one shot multi-agent decision problems. Each agent chooses an action, and then receives a final payoff. Agents attempt to maximize this single payoff. Naively these can be represented in tabular form. For each possible *joint*-decision (one for each agent) the model can describe the utility each agent obtains. Throughout this thesis we use the terms "agent" and "player" interchangeably.

A decision can come in many different forms depending on the problem. When the world is fully observable, these decisions are actions and the model is a *normal form game* (Section 2.1.1). When the world is only partially observable each agent may receive different *types* of information before making their decision. Such a problem is known as a Bayesian game (Section 2.1.2). An agent in these problems may choose a different action for each information-state they find themselves in, and thus their decisions are not just actions but *strategies* (an action choice for each information-type).

Unfortunately, agents often have multiple independent decisions which can make a tabular representation unwieldy. For example, a centralized agent controlling multiple actors

may have to make a decision for each actor. Or, even worse, an agent may have to make a series of decisions over a number of time steps after receiving one of many possible observation. While this Any decision problem can be thought of as a stage game, with multiple decisions being condensed into a single action representing that player's policy. The problem with such models is that these policies grow exponentially with the horizon. These problems are better represented by an explicit decentralized and/or sequential model such as those presented later in Section 2.2. However, it is useful to view these models as a series of stage-games which can be solved individually, thus we begin our tour of formal models with single shot stage games.

Multi-agent models extend single agent decision problems by using vectors (one element per player) to represent actions, rewards, beliefs, or observations rather than the scalars used in single agent models. These vectors are often referred to as *joints* of the individual variables. It is useful to refer to all elements of the joint except one particular player. For a joint-action $\vec{a}$ where $\vec{a}_i \in A_i$ we refer to all actions except for player i's as $\vec{a}_{-i}$ and the set of possible such joint-actions as $A_{-i} = \prod_{j \in A | j \neq i} A_j$. If all other players besides player i take joint-action $\vec{a}_{-i}$ and player i takes action α then we write the full joint-action as $\vec{a}_{-i,\alpha}$. A similar notation is used for types and observations.

2.1.1 Normal form games

A normal form game is the simplest model of a multi-agent decision problem. Players simultaneously take actions and then receive individual rewards (*a.k.a.* payoffs) based on their joint-action. Players care only about maximizing their own personal reward and not about what rewards the other players receive. It is assumed that all players have common knowledge of the game being played, and that those players are rational. A normal form game consists of a tuple $\langle N, A, R \rangle$

- N is the set of players. $n = |N|$
- $A = \prod_{i=1}^{n} A_i$ where A_i is the set of actions for player i
- $R : A \to \mathbb{R}^n$ is the reward function

Rational players can predict the behavior of other agents, and take optimal actions. When all players are rational their strategies will be in equilibrium, meaning that no player has an incentive to change their action given the equilibrium solution. This solution can come in a variety of different *solution concepts* depending on small differences about how the game and the players are defined.

A *Nash equilibrium* [37] takes the form of n independent probability distributions (strategies) over each player's actions, where each player can not gain by changing their strategy given that the other players don't change their strategies. This solution concept is the most commonly used and has the longest history within the field of game theory, but has high computational complexity (PPAD-complete [20]), which means it most likely is not polynomially solvable. It is also not a convex solution concept, meaning that the interpolation of two joint-strategies in Nash equilibrium may not be a Nash equilibrium. For these reasons we shy away from the Nash equilibrium solution concept.

A *correlated equilibrium* [7], which generalizes a Nash equilibrium, takes the form of a single probability distribution across joint-actions. Agents observe a shared random variable corresponding to joint-actions drawn from the correlated equilibrium's distribution. This informs agents of their prescribed action without directly revealing other agents' actions; however, information about other agents' actions is revealed in the form of a posterior probability after viewing the recommendation. The distribution is a correlated equilibrium when no agent has an incentive to unilaterally deviate. One can think of a CE as having a mediator (although one is not needed) who draws a joint-action from a known distribution and tells each player privately which action they should take. If players can't gain by deviating from the mediator's recommendation than the known distribution is a correlated equilibrium.

Formally, a probability distribution $p \in \Delta(\vec{A})$ is a correlated equilibrium iff:

For each player i and distinct actions α, $\beta \in A_i$,

$$\sum_{\vec{a}_{-i} \in A_{-i}} p((\vec{a}_{-i}, \alpha)) R((\vec{a}_{-i}, \alpha))_i \geq \sum_{\vec{a}_{-i}} p((\vec{a}_{-i}, \alpha)) R((\vec{a}_{-i}, \beta))_i$$

As an example, take the game of chicken (Figure 1). This game has three Nash equilibria. Two are pure strategy equilibria (each player takes exactly one action): (Brave, Chicken) and (Chicken, Brave). One is a mixed strategy with each player taking 'Chicken' with probability $\frac{2}{3}$ and 'Brave' with probability $\frac{1}{3}$. This mixed strategy yields an expected payoff of $4.6\overline{6}$ for each player, which is the greatest average reward that the agents can achieve using a Nash equilibrium. On the other hand, the game of chicken has an infinite number of correlated equilibria. Because Equation 1 is linear, the constraints on distributions over joint-actions forms a closed convex set. Any distribution in this set is a correlated equilibrium (Section 3.2 goes into more detail about these sets). The correlated equilibrium with the highest payoff has the agents taking (Brave, Chicken) and (Chicken, Brave) each with probability $\frac{1}{4}$ and (Chicken, Chicken) with probability $\frac{1}{2}$. This yields an expected payoff of 5.25 to each player which is better than the best Nash equilibrium.

		Player 2	
		Dare	Chicken
Player 1	Dare	0, 0	7, 2
	Chicken	2, 7	6, 6

Figure 1: The game of chicken.

2.1.2 Bayesian games

A Bayesian game is the partially observable equivalent of a normal form game. Before the beginning of the game, each player observes a private signal (*a.k.a.* their type). The players then take actions and receive rewards. The joint-signal (of which players only have partial knowledge) is drawn from a commonly known distribution. Instead of knowing the rewards for every joint-action, players know the rewards for every joint-action-type pair. A Bayesian game consists of a tuple $BG = \langle N, A, \Theta, \tau, R \rangle$

- N is the set of players. $n = |N|$
- $A = \prod_{i=1}^{n} A_i$ where A_i is the set of actions for player i
- $\Theta = \prod_{i=1}^{n} \Theta_i$ is the set of types (one for each different possible private signal $\theta_i \in \Theta_i$

)

- $\tau \in \Delta(\Theta)$ is a probability distribution over joint types Θ assigning θ w.p. $\tau(\theta)$
- $R : \Theta \times A \to \mathbb{R}^n$ is the reward function

Like for actions, it is convenient to refer to the types of all players except player i. We use a similar notation for types as we did for actions. $\theta_{-i} \in \Theta_{-i}$ is a joint-type excluding player i and $\Theta_{-i} = \prod_{j \neq i} \Theta_j$ is the type space of all players expect player i. (θ_{-i}, θ_i) is a full joint-type.

There are a number of different game theoretic solutions to Bayesian games, each of which make slightly different assumptions [25]. We distinguish between three different dimensions of solutions and argue for a particular solution concept based on which assumptions we believe to be most general and applicable.

The first distinction we make is between Nash and correlated equilibria. Nash equilibria assume players choose actions independently while correlated equilibria allow player's actions to be dependent via a shared random variable (and thus conditionally independent given the random variable). Correlated equilibria are more general than Nash equilibria (all Nash equilibria are correlated equilibria) and thus will produce better results. Correlated equilibria are also computationally much less demanding to compute. Therefore agents have an incentive to use correlated equilibria instead of Nash equilibria when available. While Correlated equilibria require a shared random variable (*a.k.a.* correlation device), such a requirement is not particularly demanding as agents in most real world scenarios can find or create such variables (*e.g.,* electromagnetic or sound noise at a particularly volatile frequency, sunspots, a dice roll, a mediator, or cryptographic techniques using communication). It might seem implausible for agents to have the ability to agree on a particular shared random variable but if they can't select a random variable then they also won't be able to perform equilibrium selection or for that matter even agree on a solution concept. Determining a correlation device as well as equilibrium selection and solution concept agreement are thus all part of the same problem and out of the scope of this thesis. Because the requirements are low and reasonable while providing superior results at lower computational

complexity, we will focus on correlated equilibria.

Within correlated equilibria concepts a second distinction is made with regard to the level of communication permitted. Unlimited communication which incurs no cost to the agents (*cheap-talk*) is sometimes present. In games of incomplete information cheap-talk can expand the set of equilibria as agents can potentially share their private information with each other. This leads to a solution concept known as *communication equilibrium* [25]. However the problem of determining a communication policy (without an unbiased mediator) is itself a complicated problem with doubts that it is possible at all [42]. We also want to maintain as general a solution as possible so we believe that any communication should be explicitly included in the model. While we still assume agents have access to a shared random variable, we will not assume the presence of cheap-talk (thus agents can not inform each other about their private information within a stage-game, only explicitly through their actions across stages).

The third distinction we make is between strategic and agent form solutions. Strategic form solutions assume that a player simultaneously controls the policy of all types of that player while agent form treats each player type as a separate player. Strategic form equilibria must guarantee that there is no incentive for multiple types of the same player to simultaneously switch their policies, while agent-normal form only requires that each individual type of a player does not have an incentive to change their policy. In effect, strategic form must guard against coalitions between types of the same player, and is thus more restrictive than agent form (all strategic form equilibria are agent form equilibria, but not visa-versa). Agent-normal form will therefore provide better results. Agent-normal form also seems more reasonable as an agent has no reason to help hypothetical versions of themselves. Even if an agent might be a different type next time around, a sequential solution accounts for this future possibility. Agent-form is also easier to compute. For these reasons we believe the agent-normal form is a more appropriate solution concept.

Taking into account the three distinctions listed above we believe the appropriate solution concept for each Bayesian stage-game is the agent-normal form correlated equilibria. For a description of various Bayesian-Nash solution concepts see [56], while other correlated

solution concepts are examined in [25].

Definition: An agent normal form correlated equilibrium takes the form of a probability distribution across actions and types such that agent-types don't have an incentive to deviate, and that a player's action is conditionally independent of the types of other players (enforcing that no private information is revealed by the shared random variable). Formally, a probability distribution $p \in \Delta(\Theta \times A)$ is an agent normal form correlated equilibrium iff:

For each player i with type t_i, distinct actions α, $\beta \in A_i$,

$$\sum_{\vec{a}_{-i}} \sum_{\vec{\theta} \in \Theta | \theta_i = t_i} \tau(\vec{\theta}) p_{\vec{\theta},(\vec{a}_{-i},\alpha)} R(\vec{\theta},(\vec{a}_{-i},\alpha)) \quad \geq \quad \sum_{\vec{a}_{-i}} \sum_{\vec{\theta} \in \Theta | \theta_i = t_i} \tau(\vec{\theta}) p_{\vec{\theta},(\vec{a}_{-i},\alpha)} R(\vec{\theta},(\vec{a}_{-i},\beta)) \qquad (1)$$

For all players i, j with types t_i, t_j, and action α

$$\sum_{\vec{a}_{-i}} \sum_{\theta \in \Theta | \theta_i = t_i, \theta_j = t_j} \left[p_{\theta,(\vec{a}_{-i},\alpha)} \right] \; = \; \tau(t_i) \sum_{\vec{a}_{-i}} \sum_{\theta \in \Theta | \theta_j = t_j} \left[p_{\theta,(\vec{a}_{-i},\alpha)} \right] \qquad (2)$$

An agent normal form correlated equilibrium will yield an expected payoff to each agent i (utility) of:

$$V_i(p) = \sum_{\vec{\theta}} \tau(\vec{\theta}) \sum_{\vec{a}} p_{\vec{\theta},\vec{a}} R_i(\vec{\theta},\vec{a})$$

2.1.3 Decentralized decisions

If all the agents of a stage game share the same reward function then the agents are cooperative and not competitive. If the state of the world is fully observable (*i.e.,* the agents are playing a normal form game with shared reward) then the problem can be viewed as a centralized single-agent decision problem where the single agent must choose actions for each decentralized-agent. Recall that we don't assume agents are able to communicate private information, but because there is no uncertainty about the state of the world and agents are rational they are able to predict the actions of other agents and act as though there is a centralized decision maker. The action space for the centralized decision maker is the Cartesian product of the action spaces for each decentralized agent.

The problem is more complicated when each cooperative agents is given private information before they make their decision (*i.e.,* the agents are playing a Bayesian game with shared reward). However, we can still view the problem through the lens of a single centralized decision maker. Now, instead of only one action per decentralized agent, the decision maker must assign an action to each agent for each information-state (type) of that agent. This is equivalent to a solution to the corresponding Bayesian game and can be thought of as a probability distribution across actions and types that satisfies Equation 1 except because the agents share rewards there is no need for the rationality constraints. Without the rationality constraints there will always exist an optimal integral (all variables either 0 or 1) solution to Equation 1. This means that without loss of generality we can assume an optimal solution will be a pure strategy (no randomization in the actions taken).

2.2 Sequential decision models

Stage games represent a single decision at a single point in time. However, most agents must make multiple decisions over time after taking observations and gaining information. These problems are referred to as sequential decision problems. A sequential decision maker follows a policy, which maps possible information states to actions. While its possible to view sequential problems as a single decision over policies, the number of policies is exponential with respect to the number of information states. It is therefore useful to have models tailored to sequential decision problems.

Single agent reinforcement learning models the world as a Markov decision process (MDP) where the objective is to maximize the long term expectation of a reward signal ([86] for more information). When there is uncertainty in the world, it becomes more efficient to modify the MDP model by explicitly separating observations from underlying dynamics, resulting in a partially observable Markov decision process (POMDP). When the world can be be understood as containing multiple self interested agents we must once again use different models (these are referred to as *games*). Game theory uses a number of different models. For non-sequential problem (*a.k.a. stage games*) the models of normal form games and stochastic games are used for fully observable and partially observable

scenarios respectively. The sequential formulation of these problems become stochastic games (SGs) and partially observable stochastic games (POSGs) for the fully observable and partially observable scenarios respectively.

Each model efficiently captures otherwise unwieldy aspects of the environment (hidden state, multiple agents, and/or sequential decisions) by making a slightly more complex model and/or including additional assumptions. The models listed are widely used and have been studied by a variety of fields for many decades. We use these models and previous research on them as the foundation for this thesis and continue the tradition of extending these models to handle additional complexity. In this section we review the well known models listed above, what it means to solve them, and what previous work has been done towards computing these solutions.

2.2.1 Markov decision processes (MDP)

Markov decision processes (MDPs) model fully observable single agent sequential decision problems. In every state an agent takes an action, receives rewards and then observes the new state they end up in (the successor state). The dynamics of this series of states is described by the transition function which obeys a Markov property. The Markov property guarantees that the probability of a particular successor state depends only on the previous state and the action taken, not on any previous state. MDPs consist of a tuple $MDP = \langle S, s^{(0)}, A, P, R \rangle$

- S is the set of states
- $s^{(0)} \in \Delta(S)$ is the initial state distribution
- A is the set of actions
- $P : S \times A \to \Delta(S)$ is the probability transition function with $P(s'|s, a)$ being the probability of ending up in state s' after taking action a in state s
- $R : S \times A \to \mathbb{R}$ is the function

MDPs have proved to be very useful and much research has gone into developing efficient algorithms for computing optimal policies for MPDs. A major class of these algorithms operates by computing the utility (*a.k.a.* value) of every state. This approach is based off of the observation that if the utility of all successor states is known, then the utility of the current state can be computed by finding the optimal action for that state. A states value $V(s)$ can be described recursively in terms of the value of successor states by the Bellman equation:

$$V(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\} \tag{3}$$

If an agent knows the value of every state (the function $V : S \to \mathbb{R}$ *a.k.a.* the value-function) then greedily taking actions (based on which one has the greatest expected value) will result in an optimal policy.

One well known algorithm to compute the value-function is *value-iteration* [86] which uses the Bellman equation as an update rule to sequentially improve the estimated utility of each state. This is a dynamic programming approach. We start the process with an arbitrary initialization of each value. Each time the Bellman equation is used to update a state's value estimate (a Bellman update) the horizon of the initial value is backed up and its relative contribution to the current value estimate is reduced by the discount factor. In this way the error of our estimates is driven down exponentially. In order to estimate the value of state to within ϵ we only need to perform $\log_\gamma(\epsilon)$ backups for each state. It can also be shown that the induced policies after each iteration of this process will not repeat, so because there are only a finite number of policies, only a finite number of iterations are needed to compute the exact values.

2.2.2 Partially observable Markov decision processes (POMDP)

While MDPs are useful for modeling a broad range of problems, it is often the case that past observations can inform an agent's current decision. A sequential decision problem with this property is said to be partially observable as the current observation (which corresponds to the current state in an MDP) does not completely dictate the probabilities of successor states, and a decision maker must remember past states in order to perform optimally.

This large class of problems is modeled using partially observable Markov decision processes (POMDPs) [53]. A POMDP consists of a tuple $POMDP = \langle S, O, A, P, R, s^{(0)} \rangle$.

- S is the set of states
- O is the set of possible observations
- A is the set of actions
- $P : S \times A \to \Delta(S \times O)$ is the probability transition function with $P(s', o|s, a)$ being the probability of ending up in state s' with observations $o \in O$ after taking action $a \in A$ in state s
- $R : S \times A \to \mathbb{R}$ is the function
- $s^{(0)} \in \Delta(S)$ is the initial state distribution

A number of different algorithms have been developed to solve POMDPs [15]. Most of them rely on the idea of a belief space. At any point in time an agent may not know the true underlying state, but instead will know a distribution across possible states. This distribution captures all the agents knowledge about the state of world and is that agent's *belief*. The set of possible beliefs ($\Delta(S)$) is the *belief-space*. As the belief-space completely determines what an agent needs to know, we can convert a POMDP into an MDP over belief-space (the states in the MDP are all possible distributions over the states in the POMDP). The resulting MDP is continuous so traditional value based approaches will not work. Belief-space MDPs have regular structure (such as the value-function being concave across beliefs) that can be exploited to produce efficient algorithms.

A policy of a POMDP is a mapping from belief to action. This can take a number of different forms. Naively this can be a mapping from observation histories to actions. We can view this map as a tree where each node is an action and each branch an observation. This is described as a *policy-tree*. If a policy is represented as a tree, then the number of nodes at level h grows exponentially ($|O|^h$). Often this mapping has redundant branches and can be more efficiently represented as a directed multigraph known as a *policy-graph*.

Such a graph acts like a finite-state controller, with the current node being the state of the controller outputting an action and transitioning based on the observation seen. These policy representations are useful for policy-iteration algorithms which attempt to optimize the policy directly.

The value of a policy-tree $\pi : \vec{O} \rightarrow a$ of POMDP $\langle S, O, A, P, R, s^{(0)} \rangle$ when starting in state s achieves utility:

$$V_\pi(\vec{O}, s) = R(s, a) + \gamma \sum_{s' \in S} \sum_{o' \in O} P(s', o'|s, a) V_\pi(\left\langle \vec{O}, o' \right\rangle, s')$$

A policy-tree depends only on the observations made, not the true underlying state (as this is unobservable to the agent). This means that if a policy-tree is fixed, the utility it receives is equal to the utility it would receive starting from each state weighted by the probability of that state for the current belief (*i.e.,* $V_\pi(\vec{O}, b^{\vec{O}}) = \sum_s b^{\vec{O}}(s) V_\pi(\vec{O}, s)$)

When viewing a POMDP as a belief-MDP the policy is a continuous mapping from a probability distribution over states to action. This is often represented by mapping actions to hyperplanes in the belief/value space and specifying the policy as mapping to the action with the maximum valued hyperplane for any belief. Such a representation acts as both a value function (Section 3.1) and a policy.

2.2.3 Stochastic games

Stochastic or Markov games [48] games extend MDPs to multiple agents by using vectors (one element per player) to represent actions and rewards rather than the scalars used in MDPs. Alternatively, stochastic games can be be viewed as a sequence of normal-form games. They generalize both repeated games (a stochastic game with one state) and extensive-form games (a stochastic game with no cycles). Each state corresponds to a normal form game where joint-actions dictate not only the rewards, but also which normal-form game will be played next. Agents start in initial state s^0. They then repeatedly choose joint-actions, receive rewards, and transition to new states.

In our work, we assume agents are rational and attempt to maximize their long term expected utility with discount factor γ. We assume that agents have perfect recall of all

past joint-actions, allowing agents to change behavior and threaten or punish based on each other's past actions (e.g. "if you do X, I'll do Y"). We formally represent a stochastic game by the tuple $\langle N, S, s^0, A, P, R \rangle$.

- N is the set of players. $n = |N|$
- S is the set of states
- $s^{(0)} \in \Delta(S)$ is the initial state distribution
- $A = \prod_{i=1}^{n} A_i$ is the set of joint-actions
- $P : S \times A \to \Delta(S)$ is the probability transition function describing the probability of transitioning from one state to another given a joint-action $\vec{a} = \langle a_1, \cdots, a_n \rangle$;
- $R : S \times A \to \mathbb{R}^n$ is the joint-reward function describing the reward a state provides to each player after a joint-action.

A solution to a stochastic game takes the form of a set of policies in equilibrium. This means that after observing all other players' policies no player has incentive to unilaterally change their own policy. It is not sufficient for players to be in equilibrium for each stage-game. Instead, players must consider their long-term utility instead of their immediate reward. This utility is determined by the long-term policy of all players. If for every stage game, players don't have an incentive to change their mixed-action given a set of policies, then those policies are in equilibrium. Therefore, in a similar way to how a single agent policy is optimal if that agent can't improve their policy by changing any one action, a multi-agent joint-policy is in equilibrium if no agent can improve by changing any one stage-game strategy given the joint-policy. This property can be used to verify that policies are in equilibrium.

2.2.4 Threats

An important difference between policies of stochastic and normal-form games is how agents deal with an agent that deviates from the prescribed policy. For a normal-form game, once actions are taken the game is over. Therefore, if an agent deviates there is nothing the other

agents can do about it. This is not the case in a sequential game. Other agents may observe a deviation and change their policy in order to punish the deviator. Recall that we define an equilibrium of a sequential game as a joint-policy where agents don't have an incentive to unilaterally deviate. The value of deviation is therefore very important when computing equilibria. A *threat* is a contingency policy that agents will follow if another agent deviates from the equilibrium. When such a threat is carried out, the value that each agent receives is called the *threat-point*. While agents using policies in equilibrium should never deviate, the threat of punishment can decrease the value of deviation and thus allow a richer set of equilibrium policies.

There are many different types of threats depending on subtle differences in how the game and rationality is defined. One big distinction is between credible and non-credible threats [32]. A threat is *credible* if it is itself an equilibrium. A *non-credible* threat is one that isn't credible. If a threat isn't an equilibrium, then a rational agent will have an incentive to deviate from the threat and will therefore not follow through with it. However, if a rational agent has the ability to somehow commit ahead of time to a threat, thereby guaranteeing that they will follow through with it, then the a non-credible threat can be believed and used. An equilibrium that uses only credible threats is said to be *sub-game perfect* because the policy is an equilibrium to matter which state an agent ends up in.

One famous example of credible and non-credible threats is the idea of mutually assured destruction. During the cold war both the U.S. and Russia had nuclear missiles pointed at each-other. Each threatened the other country with nuclear annihilation if they were ever attacked. At the time, if one country launched their missiles at the other there was no way to prevent them from hitting their targets. The only option would be to retaliate despite it being useless to save one's own country. Such a retaliation is an example of a non-credible threat - your country will be destroyed no matter what you do and you must now decide to launch missiles in retaliation, but killing hundreds of millions of people more will have no positive value. However, if instead of relying on human operators to launch missiles, a machine is used that automatically retaliates if it detects an incoming nuclear strike, then the non-credible threat can be used and it would be rational to build such a machine.

2.2.5 Partially observable stochastic games (POSG)

A partially observable stochastic game (POSG) is the multi-agent equivalent to a POMDP and the partially observable extension to a stochastic game. A POSG is identical to a POMDP except instead of a single action, observation, and reward there is one for each player which are expressed together as a joint-action, joint-observation, and joint-reward. POSGs operate similarly to all previous sequential models. Players repeatedly choose actions, receive observations (including any reward signal), and transition to new (unobserved) states. POSGs consist of a tuple $PO = \langle N, A, S, O, P, R, b^{(0)} \rangle$.

- N is the set of players. $n = |N|$
- $A = \prod_{i=1}^{n} A_i$ is the set of joint-actions
- S is the set of states
- $O = \prod_{i=1}^{n} O_i$ is the set of joint-observations
- $P : S \times A \to \Delta(S \times O)$ is the probability transition function with $P(s', \vec{o}|s, \vec{a})$ being the probability of ending up in state s' with observations $\vec{o}$ after taking joint-action $\vec{a}$ in state s
- $R : S \times A \to \mathbb{R}^n$ is the joint-reward function
- $b^{(0)} \in \Delta(S)$ is the initial state distribution

For POMDPs a sufficient (for optimal behavior) type space is the set of probability distributions over possible states. This is not the case for POSGs. Not only do agents have to worry about which state they are in, but also must worry about other player's beliefs. Worse, players must reason about the beliefs that players hold about each others beliefs. This meta-reasoning continues indefinitely (reasoning about beliefs of beliefs of beliefs etc...) and makes the problem significantly more conceptually and computationally complex. An agents' observations may include private information about other player's beliefs, and in the worst case it might be impossible to losslessly reduce a player's type beyond the full history.

The POSG framework is incredibly general and thus very difficult to solve. The authors know of no tractable attempt to compute optimal (or even reasonable) policies for general POSGs. Much work has been done on limited subsets of POSGs. Game theory has dealt heavily with Bayesian games (which can be thought of as finite horizon POSGs), but has typically had a descriptive focus instead of prescriptive algorithms that compute policies. Game theory has also dealt with stochastic games but very few game theoretic results exist at the intersection of stochastic and incomplete information games. Those results that do exists almost exclusively deal with two player zero-sum repeated games with incomplete information [26]. The multi-agent learning community has also studied the two player zero-sum case with many success (particularly for Poker [76]), and is an active area of research. However, these results make extensive use of properties unique to zero-sum games and don't extend to the general case. Likewise successes have been made in MARL for games of common payoff [78], but these also do not generalize.

2.2.6 POSGs as a Sequence of Bayesian Games

A stochastic game can be thought of as a model of the world where agents play a sequence of normal-form games where the next game in the sequence depends on the previous game and the actions taken. A stochastic game can then be solved by solving each normal-form stage game augmented to include expected utility. It would be nice to use the same process to solve a POSG by turning it into a series of Bayesian games. Unfortunately unlike how a stochastic game can easy be seen as a sequence of normal-form games, a POSG does not naturally correspond to a sequence of Bayesian games. However, a non-trivial transformation can be applied to a POSG to compute successive Bayesian games which have the same decision theoretic properties as the original POSG. Emery-Montemerlo et.al [[23]] presented a method for achieving this transformation. The method keeps track of two distributions and updates them each time-step: the probably of seeing a particular observation history $\theta^{(t)}$ (which are defined as the player types) and the probability that the underlying state is $s^{(t)}$ given $\theta^{(t)}$.

After $t+1$ time steps the current Bayesian game, that is equivalent to the decision

problem faced by players in the POSG, can be computed as follows: N and A are the same as in the POSG. The player types, $\Theta^{(t)}$, are all possible observation histories. The common knowledge distribution over types, $\tau^{(t+1)}$, can be computed recursively given the sequence of states and actions as follows. The probability that joint-type $\theta^{(t+1)} = \langle \vec{o}, \theta^{(t)} \rangle$ at time $t+1$ is given by:

$$\tau^{(t+1)}(\langle \vec{o}, \theta^{(t)} \rangle) = \tau^{(t)}(\theta^{(t)}) Pr[\vec{o}|\theta^{(t)}] \quad \text{where:} \tag{4}$$

$$Pr[\vec{o}|\theta^{(t)}] = \sum_{s^{(t+1)}} P(s^{(t+1)}, \vec{o}|s^{(t)}, \vec{a}^{(t)}) Pr[s^{(t)}|\theta^{(t)}] \tag{5}$$

Because $\theta^{(t)}$ is the history of joint observations we can compute $P(s^{(t)}|\theta^{(t)})$ by treating the POSG as a hidden Markov model and performing filtering with observations $\theta^{(t)}$ (recall that $\theta^{(t)}$ includes observed joint-actions). We can compute the one step reward function:

$$R^{(t+1)}(\vec{a}, \theta^{(t)}) \quad = \textstyle\sum_{s^{(t)}} Pr[s^{(t)}|\theta^{(t)}] \sum_{s'} P(s'|s^{(t)}, \vec{a}^{(t)}) R(s', \vec{a}^{(t)}) \tag{6}$$

This reward function provides the short-term stage-game reward, however agents actually want to maximize their long term rewards (utility). However, knowing the exact utility is tantamount to solving the problem. Instead, a standard trick is to use estimated utilities of each successor state, $V(\langle \vec{o}, \theta^{(t)} \rangle)$. Using V the expected utility for each joint-action $\vec{a}$ can be calculated resulting in the augmented reward $R(\vec{a}, \theta) = \sum_s P(s|\theta) R(s, \vec{a}) + \sum_{\vec{o}} V(\langle \vec{o}, \theta^{(t)} \rangle) P(\vec{o}|\theta)$.

The challenge then lies in computing the state utility estimations $V(\langle \vec{o}, \theta^{(t)} \rangle)$. A common tactic when the world is fully observable is to apply dynamic programming and iteratively compute V based on the previous estimation of V (*e.g.* value iteration). Unfortunately the state-space of Bayesian games is unbounded as it includes all possible histories, making it impossible to iterate over all states and thus infeasible to compute V in this way. If we did have a V we would have a Bayesian game and could compute equilibria of this game. Emery-Montemerlo et.al [23] constructed an approximation of V by removing all types with low probability combined with a heuristic to guess V. While this provides a tractable algorithm that might perform reasonably on some problems if the heuristic is good and types don't become too diluted, it likely will produce arbitrarily poor solutions to general POSG problems.

Our solution is to define a new model, Markov games of incomplete information (MaGII), that by definition will produce a bounded type-space for the Bayesian stage-games (Chapter 4). This will allow V to be estimated using dynamic programming (Chapter 5).

2.2.7 Decentralized partially observable Markov decision processes

When agents in a POSG share a reward signal the problem is significantly easier. This problem is known as a decentralized partially observable Markov decision processes (DecPOMDP). Because agents share rewards, they cooperate with each other and don't have to worry about individual rationality, unilateral deviation, threats, or trade-offs between maximizing on agent's utility over another. However, because the world is still partially observable, agents still need to reason over other agent's possible observation histories.

We define a DecPOMDP formally as the tuple $\langle N, A, S, O, P, R, b^{(0)} \rangle$ where:

- N is the set of players. $n = |N|$
- $A = \prod_{i=1}^{n} A_i$ is the set of joint-actions
- S is the set of states
- $O = \prod_{i=1}^{n} O_i$ is the set of joint-observations
- $P : S \times A \to \Delta(S \times O)$ is the probability transition function with $P(s', \vec{o}|s, \vec{a})$ being the probability of ending up in state s' with observations $\vec{o}$ after taking joint-action $\vec{a}$ in state s
- $R : S \times A \to \mathbb{R}$ is the shared reward function
- $b^{(0)} \in \Delta(S)$ is the initial state distribution

2.3 Factored representations

The dynamics of many environments can be more efficiently defined by breaking apart the state into a set of factors (*a.k.a.* features). Such a representation allows us to more naturally define many scenarios along with being able take advantage of independencies. For example, a state in a grid-world might include a factor for the agent's 'x' location along with another

factor for the 'y' location. Moving left/right or up/down would then only depend on one of the factors and not the other.

When there are multiple agents, the use of factors grants even more power. Each agent often has state specific to that agent (such as their location, observations, or actions), leading naturally to a factored representation. Furthermore, an agent's actions typically only have a local effect thus a representation using local factors can be decomposed granting even more power to factored representation [59]. Such a representation is a powerful tool for making complex domains easier to solve. In chapters 4 and 5 we make extensive use of factored representations by folding the belief of each agent into the underlying state using a factored representation. Each single-shot and sequential model described in the previous sections can make use of a factored representation. However in this thesis we will primarily use a factored representation for POSGs and DecPOMDPs.

We define a factored POSG as the tuple $\langle N, A, X, S, O, P, R, b^{(0)} \rangle$ where:

- N is the set of players. $n = |N|$
- $A = \prod_{i=1}^{n} A_i$ is the set of joint-actions
- $X = \{X_1, \cdots, X_{|X|}\}$ is the set of factors, each of which is a set.
- $S = \prod_{i=1}^{|X|} X_i$ is the set of states
- $O = \prod_{i=1}^{n} O_i$ is the set of joint-observations
- $P : S \times A \to \Delta(S \times O)$ is the probability transition function with $P(s', \vec{o}|s, \vec{a})$ being the probability of ending up in state s' with observations $\vec{o}$ after taking joint-action $\vec{a}$ in state s
- $R : S \times A \to \mathbb{R}^n$ is the joint-reward function
- $b^{(0)} \in \Delta(S)$ is the initial state distribution

The important advantage of this representation is that the transition and reward functions do not have to enumerate the full joint probability distributions across factors (*i.e.,*

these functions need not be tabular). Instead the functions can be defined using any factorization scheme, such as an algebraic decision diagram [30], Bayesian network [11], a Markov random field [93], or as any other graphical model [35]. Other factored decision models are defined similarly. For example, a factored DecPOMDP is identical to a factored POSG except with a single shared reward $R : S \times A \to \mathbb{R}$.

CHAPTER III

THE ACHIEVABLE-SET METHOD FOR STOCHASTIC GAMES

This chapter addresses the problem of efficiently solving infinite-horizon general-sum stochastic games of complete information. We aim to find all policies in correlated equilibrium. We take a value based approach and extend the Bellman heritage to multiple independent agents. Importantly, throughout this approach we are careful to make strong guarantees about the quality of our solution. We assume the reader is familiar with single agent value-iteration as well as basic linear programming.

In reinforcement learning, Bellmans dynamic programming equation is typically viewed as a method for determining the value function the maximum achievable utility at each state. Instead, we can view the Bellman equation as a method of determining all possible achievable utilities. In the single-agent case we care only about the maximum utility, but for multiple agents it is rare to be able to simultaneously maximize all agents utilities. Instead of computing a single optimal value, this chapter presents methods for computing the set of all achievable joint-utilities (a vector of utilities, one for each player). We call this set the achievable-set. Given this goal, we can reconstruct a proper multi-agent equivalent to the Bellman equation that operates on achievable-sets for each state instead of values.

This chapter develops this value-based dynamic programming approximation approach for solving general-sum stochastic games. We create an algorithm with polynomial efficiency that has strong guarantees about the error induced. It is extensible to a variety of different solution concepts and provides good empirical results.

3.1 Multi-agent value methods

The goal of any agent (no matter which decision problem they face) is to maximize their utility (sum of long term discounted reward). An agent does not care which policy they ultimately execute, only the utility that the policy achieves. For sequential problems, this utility can be broken down into the immediate stage-game reward followed by the utility

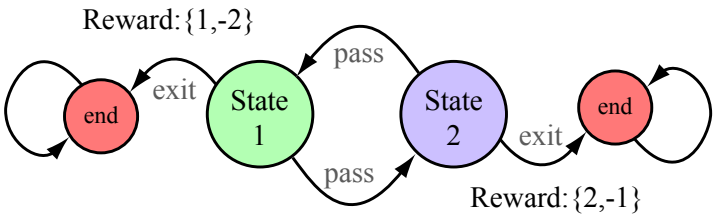

Figure 2: A graphical depiction of the breakup game. The Breakup Game demonstrates the limitation of traditional value-function based approaches. Each circle is a state and arcs are transitions for the corresponding action. At most one player has a choice of action in each state.

achieved in each possible successor state (or belief-state). This decomposition is immensely useful, allowing the knowledge of state utilities to be used to create policies. The estimation of a state's utility is known as that state's *value*. This knowledge - a mapping from state (or belief-state) to estimated utility - is known as a *value function*.

For single agent problems, the most basic and wildly used method for computing the value function is value iteration (Section refsec:mdps) using the Bellman equation (Equation refeq:bellman). Many attempts have been made to extend the Bellman equation to domains with multiple agents. Most of these attempts have focused on retaining the idea of a value function $V : S \rightarrow \mathbb{R}$ as the memoized solution to sub-problems in Bellman's dynamic programming approach [28], [49], [94]. These approaches are very similar to single agent value iteration. They involve replacing the maximization over actions in the Bellman equation with an equilibrium computation (a maximization over the joint-actions such that agents don't want to unilaterally change their individual actions). This has lead to a few successes particularly in the zero-sum case where the same guarantees as standard reinforcement learning have been achieved [49]. Unfortunately, more general convergence results have not been achieved. In fact, a negative result has shown that any value function based approach cannot solve the general multi-agent scenario [94]. Consider the breakup game, where we can imagine player 1 is in a relationship with player 2 and wishes to breakup but would rather not be the one to actually end things (Figure 2).

This game has four states with two terminal states. In the two middle states play alternates between the two players until one of the players decides to exit the game. In this game the only equilibria are mixed (*e.g.,* the randomized policy of each player passing

and exiting with probability $\frac{1}{2}$). In each state only one of the agents takes an action, so an algorithm that depends only on a value function will myopically choose to deterministically take the best action, and never converge to the stochastic equilibrium. This result exposed the inadequacy of value functions to capture cyclic equilibrium (where the equilibrium policy may revisit a state).

In the wake of this negative result, a new line of research has emerged that replaces the value-function with a multi-dimensional equivalent referred to as the *achievable-set* (Section 3.2). When there is only one agent (or agents share reward), then there is exactly one optimal utility that can be achieved. This is not typically the case for general sum games. It may not be possible to simultaneously maximize all agent's utilities. Instead of viewing a value function as describing the optimal utility for every belief-state, we advocate that value functions should not only describe the optimal utility but the set of all achievable utilities (from the best to the worst). It so happens that for single agents this achievable-set can be specified with only two points (the best and the worst). This is because for single agents, the set of achievable utilities is convex (interpolated policies produce interpolated utilities). When we think of value functions in this way they transition well to general sum multi-agent problems. In Section 3.2 we explore this transition and the complexities therein.

While the achievable-set approach was rediscovered recently in the AI community, very similar approaches have been used to solving stochastic games in game theory and operations research, mostly focused on two-player, zero sum games or repeated games. One line of work uses a recursive "self-generating set" approach that has a very similar flavor to achievable-sets [1, 19, 38, 31]. A recent example of this work is that of Burkov and Chaib-draa [13] where the set of sub-game perfect Nash equilibria is found via a repeatedly finer grain tiling of the set. Unfortunately, none of these approaches has appealing error or computational guarantees able to generalize to stochastic games.

Murray and Gordon [55] made a significant advance towards using achievable-sets with stochastic games and derived an exact but intractable algorithm for calculating the achievable-set based on the Bellman equation and proved correctness and convergence. While this exact

algorithm is intractable, they also provided an approximation algorithm. However, their approximation algorithm is not generally guaranteed to converge with bounded error, even with infinite time.

Several other complaints have been leveled against the motivation behind MAL research following the Bellman heritage. One such complaint is that value function based algorithms inherently target only stage-game equilibria and not full-game equilibria potentially ignoring much better solutions [81]. Our approach solves this problem and allows a full-game equilibrium to be reached. Another complaint goes even further, challenging the desire to even target equilibria [83]. Game theorists have shown us that equilibrium solutions are correct when agents are rational (infinitely intelligent), so the argument against targeting equilibria boils down to either assuming other agents are not infinitely intelligent (which is reasonable) or that finding equilibria is not computationally tractable (which we tackle here). We believe that although MAL is primarily concerned with the case when agents are not fully rational, first assuming agents are rational and subsequently relaxing this assumption will prove to be an effective approach.

3.2 Achievable-sets

The core idea of the algorithms presented in this chapter is to compute an *achievable-set function* $V : S \rightarrow \{\mathbb{R}^n\}$, the multi-agent analogy to the single agent value-function. In MDPs, the value-function represents the optimal achievable utility for every state. When there are multiple agents it is not always possible to simultaneously maximize all agent's utilities, so the multi-agent achievable-set-function must represent all possible combinations of values. As a group of n agents follow a joint-policy, each player i receives rewards. The discounted sum of these rewards is that player's *utility*, u_i. The vector $\vec{u} \in \mathbb{R}^n$ containing these utilities is known as the *joint-utility*, or value-vector. Thus, a joint-policy yields a joint-utility in $\mathbb{R}^n$. If we examine all mixed joint-policies starting from state s, discard those not in equilibrium, and compute all the joint-utilities of the remaining policies we will have a set of points in $\mathbb{R}^n$: the *achievable-set*.

Formally, let Π_{eq} be the set of policies in equilibrium and $V^{\pi}(s)$ be the joint-value of

policy π starting in state s, then we define the achievable-set function (over states s) as the function:

$$V(s) = \{x \in \mathbb{R}^n : \exists \pi \in \Pi_{eq} \text{ s.t. } x = V^{\pi}(s)\}$$

3.2.1 Examples

Our first example is a normal-form game given in Figure 3-A. This game has three pure equilibria corresponding to the players playing (up, right), (down, left), and (down, right). Any convex combination of these three policies is a valid correlated equilibrium. More precisely, any probability distribution over these three joint-actions will satisfy the linear inequalities defining the set of correlated equilibria (Equation 1). These are the only correlated equilibria ((up, left) can never be played). The value of these policies will correspond to the convex hull of the corresponding rewards given by the pure strategies: (2, 7), (7, 2), and (3, 3). A graphical depiction of this hull is shown in Figure 3-B.

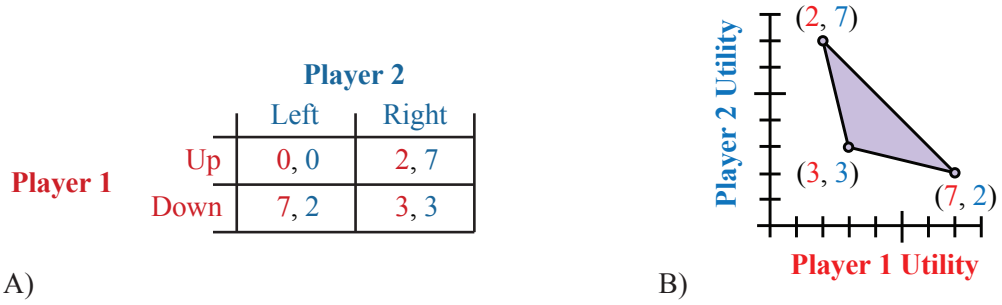

Figure 3: An illustration of an achievable set for a simple normal-form game. A) A two-player two-action normal-form game. B) The game's achievable-set using correlated equilibria.

Our second example is the normal form game of chicken (Figure 4). Unlike the previous example, this game's achievable set has two extreme points (out of four) that each correspond to a mixed correlated equilibrium. One of these mixed correlated equilibrium is also better than any Nash equilibrium (in that it has a greater sum of rewards).

Our third example is a repeated version of the game of chicken with discount $\gamma = 0.9$ (Figure 5). We also add an extra action for player 1 which allows player 1 to punish player 2 with a revenge action. Note that this action is dominated and has outcomes worse for all players than any other action. If we insist that only credible threats are used (Section 2.2.4),

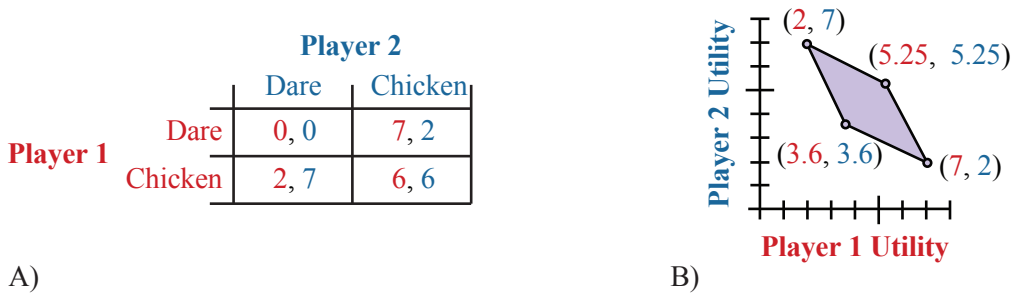

Figure 4: An example achievable-set. A) The game of chicken. B) The achievable-set of correlated equilibria for chicken. For each joint-utility within the region there exists a correlated equilibrium yielding the given rewards for each player. For example the point (5.25, 5.25) is achieved by the agents playing (chicken, chicken) with probability $\frac{1}{2}$ while playing (dare, chicken), and (chicken, dare) each with probability $\frac{1}{4}$.

then this action will never be played and can be ignored (Figure 5-B). Even ignoring this action and just using credible threats, the joint-strategy (chicken, chicken) is now viable because the threat of switching to a policy where the other agent always dares (which is an equilibrium) prevent agents from switching to dare. More formally, the value of defecting from the always chicken policy is $7 + 2/(1-\gamma)$ while the value of sticking with the policy is $6/(1-\gamma)$. With credible threats, no player can receive less than the worst equilibrium for them. However, if we allow all threats then player 1 can force player 2 to get accept a joint-policy which yields less than any stage-game equilibrium (Figure 5-C).

Our final example, the breakup game (Figure 6) is a multi-state sequential game. Unlike the game of chicken, the breakup game has multiple states and thus multiple achievable sets (one for each state). Each state has its own achievable set representing the values starting from that state. Once again we show the achievable sets for correlated equilibria which are therefore closed convex hulls. For example, when in state 1 (where player 1 has a chance to breakup) the achievable set has extreme points $(1, -0.5)$, $(1, -2)$, and $(1.8, -0.9)$. This achievable-set depicts the fact that when starting in player 1's state any full game equilibria will result in a joint-utility that is some weighted average of these three points. For example the players can achieve $(1, -0.5)$ by having player 1 always pass and player 2 exit with probability 0.55. If player 2 tries to cheat by passing when its supposed to exit, player 1 will immediate exit in retaliation (recall that history is implicitly included in

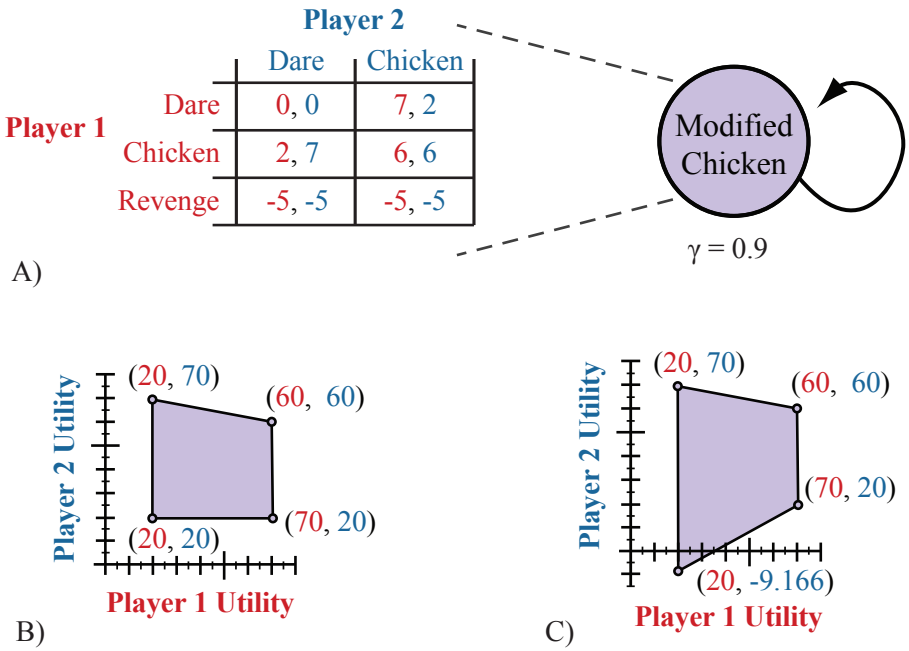

Figure 5: An illustration of an achievable set for a simple sequential game. A) A visual depiction of the repeated game. There is only one state which corresponds to a modified chicken game with an extra revenge action for the row player. B) The achievable set when only credible threats are allowed. C) The achievable set when we allow non-credible threat.

state). This threat is credible and prevents player 2 from always passing which enforces the equilibrium.

In summary, an achievable-set contains all possible joint-utilities that players can receive using policies in equilibrium. Each dimension represents the utility for a player. This definition is valid for any equilibrium solution concept, but in this thesis we only compute achievable-sets representable as convex polytopes, such as those from correlated equilibria. Since this set contains all attainable joint-utilities, it will contain the optimal joint-utility for *any* definition of "optimal." From this achievable-set, an optimal joint-utility can be chosen using a bargaining solution [56] (Section 3.2.2). We note that within game theory this set is typically refereed to as the feasible set. However this term is overloaded with linear programming (which we use in this dissertation) so to avoid confusion we call it an achievable set.

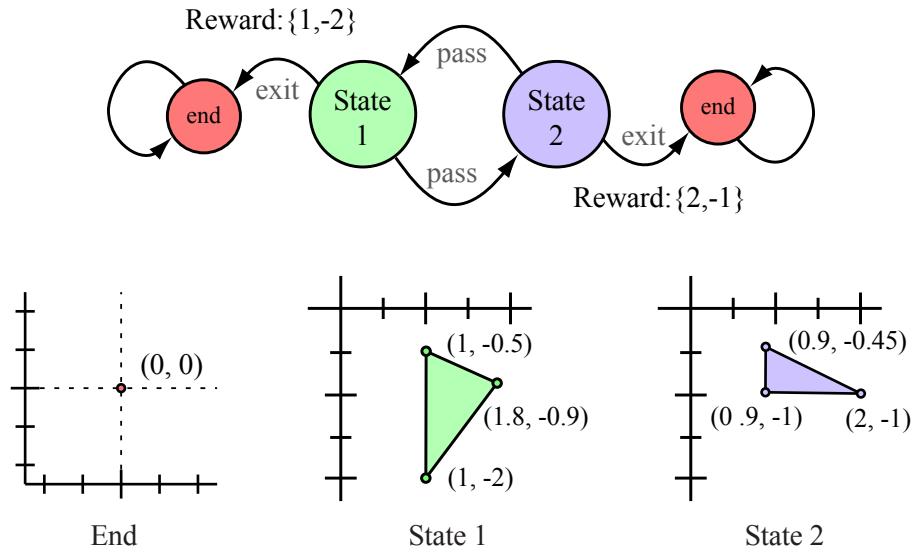

Figure 6: The Breakup Game (above) along with each state's achievable set (below). Circles represent states, outgoing arrows represent deterministic actions. Unspecified rewards are zero. The achievable-set for each state is shown for correlated equilibria using grim trigger threats ($\gamma = 0.9$).

3.2.2 Equilibrium selection

Achievable-sets describe all possible joint-utilities that players could potential receive, but when it comes to selecting a policy, agents must somehow coordinate and agree on a particular policy that achieves a particular joint-utility that they will target. Each policy benefits each agent to a different degree. Therefore, agents have competing incentives to argue for or choose one joint-policy over another. For example in the game of chicken both players

Table 1: Tabular reward and transitions for the breakup game. Each row is a possible action of Player 1, while each pair of columns is an action of player 2. All transitions are deterministic.

State 1

	NOOP	
	Joint-Reward	Transition
Pass	$(0, 0)$	State 2
Exit	$(1, -2)$	End

End State

	NOOP	
	Reward	Transition
NOOP	$(0, 0)$	End

State 2

	Pass		Exit	
	Reward	Transition	Reward	Transition
NOOP	$(0, 0)$	State 1	(2, -1)	End

would want the equilibrium followed to be them choosing "dare" and the opponent choosing "chicken". However if both players assumed this equilibrium they will both end up playing "dare" to disastrous consequences. If all agents are rational then they should be able to predict and agree on a joint-policy to follow even without communication. Despite this conclusion it is not clear how rational agents would go about agreeing on an equilibrium.

One approach is to view equilibrium selection from an evolutionary point of view. Agents not in equilibrium achieve lower utility and thus don't reproduce as well. This leads to populations converging on locally optimal equilibria. Local populations that target superior policies will out compete other local populations that have converged on less efficient policies. In this way global populations will converge on Pareto-efficient equilibria that maximizes social utility. This is one argument for how agents can converge on one particular policy. However, differences in starting conditions can lead to different equilibria or depending on the conditions could even lead to cyclic behavior changes.

Another approach to equilibrium selection takes the view that the correct equilibrium is the one that would arise if agents were to sit down and negotiate. Attempting to agree on a such a utility target is hard problem in its own right, and known within game theory as a *bargaining problem*[36]. Bargaining problems take either an axiomatic or strategic view of the problem. Axiomatic solutions postulate that rational agents will agree on fundamental properties of a solution (for example, that the solution should be fair to both agents and give each equal gains). If such axioms are defined carefully, exactly one joint-utility will satisfy the axioms and thus be the natural choice. Unfortunately, several axiomatic systems have been proposed and there is no clear system of axioms that seems obviously correct. Each system of axioms leads to a different solution such as the Nash bargaining solution [36], the Kalai-Smorodinsky bargaining solution [41], or the Egalitarian bargaining solution [40] (Figure 7).

Unlike the axiomatic view, the strategic view models the bargaining process as an explicit sequential game where there is only a single equilibrium. The best example for this is the two-player alternating offer game [74] where one player suggests an equilibrium and the other player must accept or reject it. If the offer is rejected the other player get to propose

a solution or the bargain terminates with some small probability. If no bargain is reached the agents receive a disagreement value (which, for example, can be each agent's safety value). It was shown that the only equilibrium of such a situation is for the first player to immediately offer the Nash bargaining point of the game. However, if the dynamics of this game change slightly (for example if the next player to propose an offer is chosen randomly instead of alternating, or players have incomplete information [75]) then the resulting equilibrium changes. For more information on bargaining, see [56].

Clearly, equilibrium selection is a hard problem. Hundreds of papers have been written on the subject and the field is still active. While equilibrium selection is essential for eventually choosing a policy, it can be viewed as a separate problem from equilibrium computation. In order to engage in equilibrium selection, agents must know the set of possible equilibria that they can choose from. Also, often times just knowing the set of possible equilibria is sufficient (for example when faced with the problem of mechanism design). This dissertation focuses on equilibria computation and assumes that agents have some unspecified means of equilibrium selection.

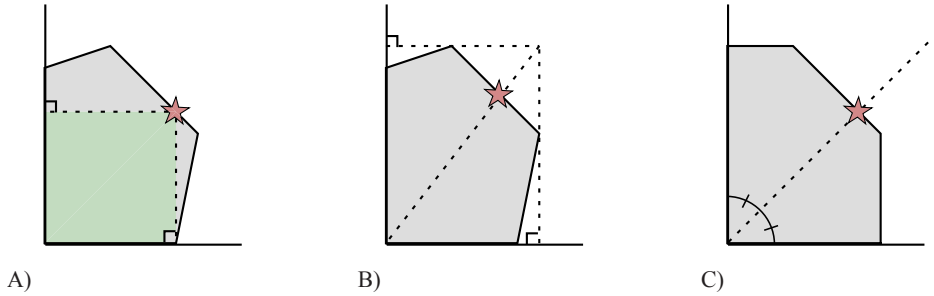

Figure 7: Graphical interpretation of three bargaining solutions with the achievable set (shaded) translated so the disagreement point is at the origin. a) The Nash bargaining solution (maximum product of gains). b) The Kalai-Smorodinsky bargaining solution (maximum gains proportional to individually ideal gains). c) The egalitarian bargaining solution with a comprehensive achievable set (maximum equal gains).

3.2.3 From achievable-sets to policies

Once the agents in the game agree on the joint-utility (through equilibrium selection), a policy can be constructed in a greedy manner for each player that achieves the targeted

utilities [55]. The construction of a policy is similar to the procedure in which a value function can be used to construct a policy in single-agent reinforcement learning. Like for a single agent, after a payoff has been chosen the actors in the world can dynamically achieve the chosen payoff without needing to plan their entire policy. however, unlike a single agent, a targeted utility-vector must be specified for each successor state and a distribution over joint actions determined such that the expectation over joint actions and successor utility-vectors achieves the desired payoff. After arriving in a new state the targeted payoff is updated based on what was agreed upon in the previous state. Repeating this process will yield a joint-policy that achieves the targeted utility. As agents care only about the utility they achieve and not about the specific policy they use, computing the achievable-set for each state is sufficient for optimal play in the stochastic game.

As an example consider once again the breakup game (Figure 6). Let us assume that before the game starts agents do not know which state (1 or 2) they will first find themselves and that there is an even probability of ending up in either state. If the players want to achieve the value of $(1, -0.5)$ the two players would agree to switch to a target value of $(0, 0)$ if the top state was reached or to a target value of $(2, -1)$ if the bottom state was reached. This agreement is maintained because it itself is an equilibrium and each choice of target value for each successor state is also an equilibrium (these points are in successor achievable sets).

3.2.4 Representing achievable-sets

A key difference between various achievable-set based approaches is in their representation of achievable-sets. These have been represented in a number of different ways: convex combinations of vertices [51, 27], polytopes of arbitrary complexity [38], and as unions of hypercubes [13]. In the algorithms presented in this chapter we focus of problems where the achievable set is closed and convex (typically this means we focus on correlated equilibria instead of Nash equilibria). This allows us to treat the achievable set as a polytope and represent it in a standard way using a set of halfspaces. This means we have a set of normals A_i and a set of offsets b_i where the achievable set is all points x where $Ax \leq b$. In order

to make tractable algorithms we need to bound the number of halfspaces used and thus we utilize a regular polytope approximation (as explained in Section 3.4.2).

3.3 Exact achievable-set computation

Achievable-set based approaches replace the value-function, $V(s)$, in Bellman's dynamic program (Section 3) with an achievable-set function – a mapping from state to achievable-set. We update the achievable-set function over a series of iterations, akin to value iteration. Each iteration produces an improved estimate of the achievable-set function using the previous iteration's estimate. Murray and Gordon's technical report [54] presented the details of an exact algorithm that follows this process, including a formal treatment of the backup, various game theoretic issues, and convergence proofs. Before the first iteration, each achievable-set is initialized to some large over-estimate and the algorithm shrinks the sets during each iteration until insufficient progress is made, at which point the current estimate is returned (Figure 8). Under full observability we can make guarantees that these final sets are very close to the true achievable-sets.

3.3.1 Set-valued backups

An exact dynamic programing solution falls out naturally after replacing the value-function in Bellman's dynamic program with a achievable-set function. However there are two main complications which must be worked through. The first is that achievable-sets are multi dimensional, not just a single scalar value. This change in variable dimension complicates the backup. In the bellman backup: $V^t(s) = \max_a\{R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^{t-1}(s')$ each operation (additions and multiplications) transforms into a related set operation. We can break the bellman update apart by first computing the state-action achievable-sets $Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^{t-1}(s')$ and then computing the state achievable-sets $V(s) = \max_a Q(s,a)$. Computing the state achievable-sets for multi-agent problems is the second

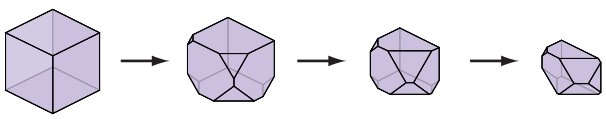

Figure 8: An example achievable-set contraction.

major complication. Instead of a simple maximization (or even a set union, which would be the natural set equivalent), we must now worry about equilibria filtering.

An illustration of the modified set backup for state-action achievable-sets is shown in Figure 9, where steps $A \rightarrow D$ solve for the action-achievable-set ($Q(s, \vec{a})$), and steps $D \rightarrow E$ solve for $V(s)$ given $Q(s, \vec{a})$. Figure 9 does not depict how we are performing equilibria filtering. This process is dependant on the particular equilibrium solution concept targeted.

The Bellman equation states that the value of a state is the expected value of the maximal action. When there are multiple agents we can't simply maximize the joint-action, we also have to worry about agent's selfishly defecting (*i.e.,* going against the prescribed policy). This means we can only maximize over joint-actions in equilibrium. Eliminating policies not in equilibrium (and thereby eliminating unachievable values) is the process of *equilibria filtering* (Figure 9-D→E). The mechanism of filtering depends on the particular equilibrium solution concept used so for generality we assume a filter function $F_{eq} : [A \rightarrow \{\mathbb{R}^n\}] \rightarrow \{\mathbb{R}^n\}$ is provided to the algorithm, which is applied to eliminate non-equilibrium policies. This function takes in an expected achievable-set for each joint-action and returns the resulting achievable set for the current stage game. If we don't care about the policy being an equilibrium (or we allow for binding contracts) than the filter function performs no filtering and simply returns the union over its inputs (all joint-action values). Details of a correlated equilibrium filter function are given in Section 3.5.4.

3.3.2 Set-valued dynamic programming

The exact achievable-set dynamic program starts by initializing each achievable-set to be some large over-estimate (a hypercube of the maximum and minimum utilities possible for each player). Each iteration of the backup then contracts the achievable-sets, eliminating unachievable utility-vectors. Eventually the algorithm converges and only achievable joint-utilities remain. The invariant of achievable-sets always overestimating is crucial for guaranteeing correctness, and is a point of great concern for the algorithms given later in this chapter.

The algorithm for this exact solution is given as Algorithm 1 where the threat function

F_{th} (the outcome of defection - see Section 2.2.4) is folded into F_{eq}. An example of the backup step is given as Figure 9. An example of the result of this backup in the dynamic program applied to the breakup game is given as Figure 9. Step

Algorithm 1 An Exact Achievable-set Algorithm

Inputs	stochastic game $(S, s^{(0)}, P, A, R)$, discount factor γ
	equilibrium filter function F_{eq} using threats F_{th}
Output	state achievable-sets V
	state-action achievable-sets Q

1: **for all** $s, \vec{a}$ **do**
2: $V(s) \Leftarrow$ hypercube with max and min utilities: $\frac{R^p_{min}}{1-\gamma}$, $\frac{R^p_{max}}{1-\gamma}$ for each agent p
3: **repeat**
4: **for all** $s, \vec{a}$ **do**
5: **for all** s' **do**
6: $Q_(s, \vec{a}, s') \Leftarrow P(s'|s, \vec{a})\dot{V}(s')$
7: $Q(s, \vec{a}) \Leftarrow \sum_{s'} Q_(s, \vec{a}, s')$
8: $Q(s, \vec{a}) \Leftarrow \gamma Q(s, \vec{a})$
9: $Q(s, \vec{a}) \Leftarrow R(s, a) + Q(s, \vec{a})$
10: **for all** s **do**
11: $V(s) \Leftarrow F_{eq}(Q(s, \vec{a}))$
12: **until** Converged
13: **return** V, Q

We have broken the algorithm up into several steps operating on the state achievable-sets $V(s)$ and state-action achievable-sets $Q(s, \vec{a})$. After initializing each achievable-set to an overestimate (line refi:init) we repeat backups (lines 6-11) until convergence. Lines 6-11 backup each state-action achievable set and line 11 computes the final state achievable-sets. The backup is very similar to a Bellman backup except operating on sets instead of a single value. This changes each basic scalar operation to a set operation. In line 6, we scale each state achievable-set by the probability of reaching that state. Scaling a set B by scalar a is defined as: $a\dot{B} = x|x/a \in B$. In line 7 we compute the possible expected value of taking action $\vec{a}$ in state s. This achievable-set depends on which utility is targeted in the next state. We therefore have to allow for all combinations of values in successor states $Q_(s, \vec{a}, s')$. This combinatorial sum of sets $\sum_1^k Q_i = Q_1 \oplus Q_2 \oplus ... \oplus Q_k$ is known as a Minkowski sum [77] and is defined as: $A \oplus B = \{a + b \mid a \in A, b \in B\}$. Line refl:gamscale is once again a set scaling, while line refl:trans is a translation defined as: $a + B = x|x - a \in B$.

The final backup step (line refl:eqop) is the equilibrium contraction step and can be

quite complex. If we didn't worry about defection, this operation would be a set union. However, because each agent is individually rational we much ensure that the joint policy chosen is an equilibrium. Let $\mathbb{G}(Q(s,\vec{a})\})$ be the normal form game defined across $Q(s,\vec{a})$ such that if players take joint action $\vec{a}$, player i receives reward $Q(s,\vec{a})\}_i$. We define the equilibrium filtering function formally as:

$$F_{eq}(Q_{\vec{a}}) = \{x | \exists \pi \in \Delta(A) \text{ s.t. } x = \sum_{\vec{a}} [\pi(\vec{a}) Q_{\vec{a}}] \text{ and } \pi \text{ is an equilibrium of } \mathbb{G}(Q(s,\vec{a}))\}$$

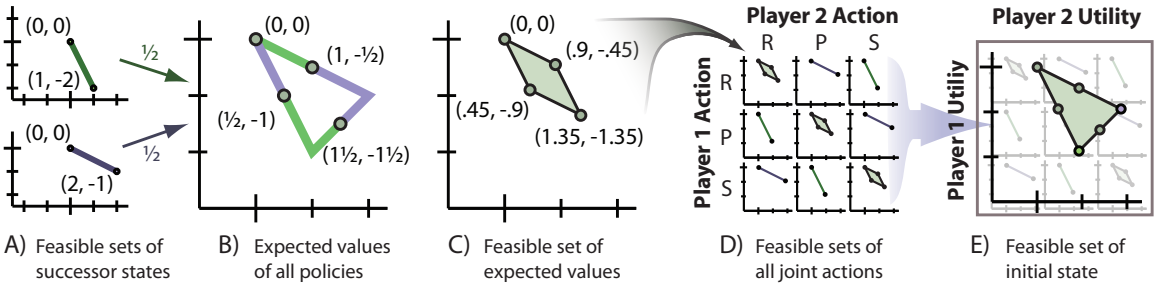

Figure 9: An example of the backup step (one iteration of our modified Bellman equation). The state shown being calculated is an initial rock-paper-scissors game played to decide who goes first in the breakup game from Figure 6. A tie results in a random winner. The backup shown depicts the 2nd iteration of the dynamic program when achievable-sets are initialized to (0,0) and binding contracts are allowed (F_{eq} = set union). In step A the achievable-set of the two successor states are shown graphically. For each combination of points from each successor state the expected value is found (in this case 1/2 of the bottom and 1/2 of the top). These points are shown in step B as circles. Next in step C, the minimum encircling polygon is found. This feasibility region is then scaled by the discount factor and translated by the immediate reward. This is the feasibility-set of a particular joint action from our original state. The process is repeated for each joint action in step D. Finally, in step E, the feasible outcomes of all joint actions are fed into F_{eq} to yield the updated achievable-set of our state.

3.4 A generic approximation algorithm for convex solution concepts

There are a few serious computational bottlenecks in the exact algorithm. The first problem is that the size of the game itself is exponential in the number of agents because joint actions are exponential in the number of players. This problem is unavoidable unless we approximate the game which is outside the scope of this thesis. The second problem is that although the exact algorithm always converges, it is not guaranteed to converge in finite

time (during the equilibrium backup, an arbitrarily small update can lead to a drastically large change in the resulting contracted set). A third big problem is that maintaining an exact representation of an achievable-set becomes unwieldy (the number of faces of the exact polytope may grow very large).

We make two important modifications to the exact algorithm in order to make the algorithm tractable: approximating the achievable-sets with a bounded number of vertices, and adding a stopping criterion. Our general approach for approximating each achievable-set is to approximate the set at the end of each iteration after first calculating it exactly (in the next section we present an algorithm that is able to skip the intermediate exact computation step). The degree of approximation is captured by two user-specified parameter: $\epsilon_1$and ϵ_2. These parameters represent the error induced by the stopping criterion and the set approximation respectively. It is useful to refer to the sum of these errors as $\epsilon_{(1+2)}$ which is the total tolerated error introduced by both the set approximation and the stopping criterion.

3.4.1 Consequences of a stopping criterion

When performing our set based dynamic programing, we must decided when to stop improving each achievable-set. While we would like to continue iterating until the sets stop improving, the sets may never converge. Instead, we add a criterion to stop when all achievable-sets contract by less than ϵ_1 (in terms of Hausdorff distance). This is added to ensure that the algorithm makes ϵ_1 progress during the exact backup of each iteration. The set approximation introduces no more than ϵ_2 error so the algorithm with contract each set at least $\epsilon_1 - \epsilon_2$ each iteration and thus will take no more than $O((R_max - R_min)/(\epsilon_1 - \epsilon_2))$ iterations to converge.

After our stopping criterion is triggered the total error present in any state is no more than $\epsilon_1/(1-\gamma)$ (*i.e.* if agents followed a prescribed policy they would find their actual rewards to be no less than $\epsilon_1/(1-\gamma)$ promised). Therefore, each value in the achievable-sets corresponds to a $\epsilon_1/(1-\gamma)$-equilibrium. Recall that our approach is to first perform an exact backup followed by an approximation of each set. We can therefor check the stopping criterion before approximating thereby preventing the set approximation error from effecting

our final error.

By stopping early we yield a solution that is an $\epsilon_1/(1-\gamma)$-equilibrium of the full game while guaranteeing there exists no exact equilibrium that Pareto-dominates the solution's utility. This means that despite not being able to calculate the true utilities at each stage game, if other players did know the true utilities they would gain no more than $\epsilon_1/(1-\gamma)$ by defecting. Moreover our approximate solution is as good or better than any true equilibrium. By targeting an $\epsilon_1/(1-\gamma)$-equilibrium we do not mean that the backup's equilibrium filter function F_{eq} is an ϵ-equilibrium (it could be, although making it such would not alleviate the convergence problem, but instead would guarantee that our solution is better than any ϵ-equilibrium and is an $(\epsilon+\epsilon_1)$-equilibrium). We do not change the filter function to guarantee convergence. Instead, we apply the standard filter function but stop if no achievable-set has changed by more than ϵ_1. In other words, because we are only checking for a stopping condition, and not explicitly targeting the $\epsilon_1/(1-\gamma)$-equilibrium in the backup we can't guarantee that the algorithm will terminate with the best $\epsilon_1/(1-\gamma)$-equilibrium. Instead we can guarantee that when we do terminate we know that our achievable-sets contain all equilibria satisfying our original equilibrium filter and no equilibria with incentive greater than an $\epsilon_1/(1-\gamma)$ to deviate.

3.4.2 Approximating achievable-sets

Bounding the complexity of each achievable-set is crucial for achieving a tractable algorithm. At the end of each iteration we can replace each state achievable-set ($V(s)$) with an N point approximation. The computational geometry literature is rich with techniques for approximating convex hulls. However, we want to insure that our achievable estimation is always an over estimation and not an under estimation, otherwise the equilibrium contraction step may erroneously eliminate valid policies. Also, we need the technique to work in arbitrary dimensions and guarantee a bounded number of vertices for a given error bound. A number of recent algorithms meet these conditions and provide efficient running times and optimal worse-case performance [50], [16], [18].

Despite the nice theoretical performance and error guarantees of these algorithms they

admit a potential problem. The approximation step is controlled by a parameter $\epsilon_2 (0 < \epsilon_2 < \epsilon_1)$ determining the maximum tolerated error induced by the approximation. This error results in an expansion of the achievable-set by at most ϵ_2. On the other hand by targeting ϵ_1-equilibrium we can terminate if the backups fail to make ϵ_1 progress. Unfortunately this ϵ_1 progress is not uniform and may not affect much of the achievable-set. If this is the case, the approximation expansion could potentially expand past the original achievable-set (thus violating our need for progress to be made every iteration, see Figure 10-A). Essentially our approximation scheme must also insure that it is a subset of the previous step's approximation. With this additional constraint in mind we develop the following approximation inspired by [17]:

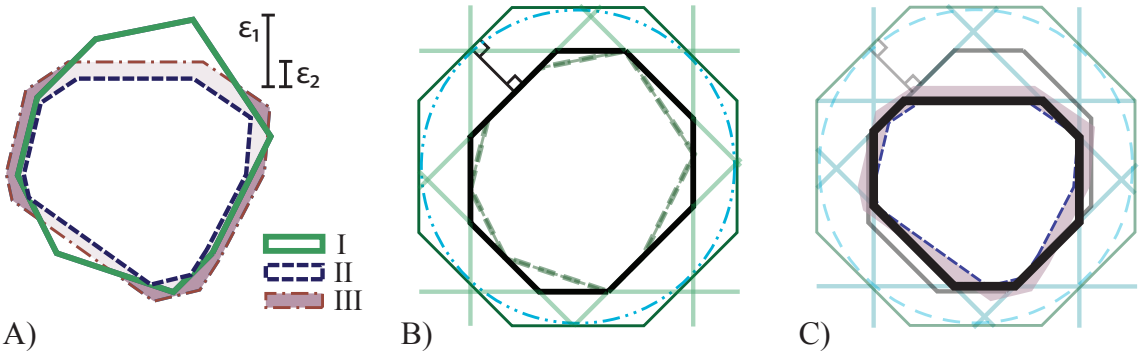

Figure 10: A) (I) achievable hull from previous iteration. (II) achievable hull after equilibrium contraction. The set contracts at least ϵ_1. (III) achievable hull after a poor approximation scheme. The set expands at most ϵ_2, but might sabotage progress. B) The hull from A-I is approximated using halfspaces from a given regular approximation of a Euclidean ball. C) Subsequent approximations using the same set of halfspaces will not backtrack.

We take a fixed set of hyperplanes which form a regular approximation of a Euclidean ball such that the hyperplane's normals form an angle of at most θ with their neighbors (*E.G.* an optimal Delaunay triangulation). We then project these halfspaces onto the polytope we wish to approximate (*i.e.,* retain each hyperplanes' normals but reduce their offsets until they touch the given polytope). After removing redundant hyperplanes the resulting polytope is returned as the approximation (Figure 10-B). To insure a maximum error of ϵ_2 with n players: $\theta \leq 2 \arccos[(r/(\epsilon_2 + r))^{1/n}]$ where $r = R_{max}/(1-\gamma)$.

The scheme trivially uses a bounded number of halfspaces (only those from the predetermined set). Also, by using a fixed set of approximating hyperplanes successive approximations will strictly be subsets of each other - no hyperplane will move farther away when the set its projecting onto shrinks (Figure 10-C). After both the ϵ_1-equilibrium contraction step and the ϵ_2 approximation step we can guarantee at least $\epsilon_1 - \epsilon_2$ progress is made. Although the final error depends only on ϵ_1 and not ϵ_2, the rate of convergence and the speed of each iteration is heavily influenced by ϵ_2. Our experiments suggest that the theoretical requirement of $\epsilon_2 < \epsilon_1$ is far too conservative.

3.4.3 Computing expected achievable-sets

Another difficulty occurs during the backup of $Q(s, \vec{a})$. Finding the expectation over achievable-sets involves a Minkowski sum (step B in fig 9 and line 7 in Algorithm 1), which naively requires an exponential looping over all possible combinations of taking one point from the achievable-set of each successor state. We can help the problem by applying the Minkowski sum on an initial two sets and fold subsequent sets into the result. This leads to polynomial performance, but to an uncomfortably high-degree. Instead we can describe the problem as the following multi-objective linear program (MOLP):

$$
\begin{aligned}
&\textbf{Simultaneously maximize} \\
&\qquad \text{player rewards:} \quad \text{foreach player } i \text{ from 1 to } n\text{:} \\
&\qquad\qquad \sum_{s'} \sum_{\vec{v} \in V(s')} v_i x_{s'\vec{v}} \\
&\textbf{Subject to:} \\
&\qquad \text{continuation utility:} \quad \text{for every state } s' \\
&\qquad\qquad \sum_{\vec{v} \in V(s')} x_{s'\vec{v}} = P(s'|s, \vec{a})
\end{aligned}
\tag{7}
$$

where we maximize over variables $x_{s'\vec{v}}$ (one for each $\vec{v} \in V(s')$ for all s') and $\vec{v}$ is a vertex in the achievable-set $V(s')$ and v_i is the value of that vertex to player i. This returns only the Pareto frontier. An optimized version of the algorithm described in this paper would only need the frontier, not the full set as calculating the frontier depends only on the

frontier (unless the threat function needs the entire set). For the full achievable-set 2^n such MOLPs are needed, one for each orthant.

Like our modified view that the Bellman equation is trying to find the entire set of achievable policy payoffs so too can we view linear programming as trying to find the entire set of achievable values of the objective function. When there is a single objective function this is simply a maximum and minimum value. When there is more than one objective function the solution then becomes a multidimensional convex set of achievable vectors. This problem is known as multi-objective linear programming and has been previously studied by a small community of operation researchers under the umbrella subject of multi-objective optimization [21]. MOLP is formally defined as a technique to find the Pareto frontier of a set of linear objective functions subject to linear inequality constraints. The most prominent exact method for MOLP is the Evans-Steuer algorithm [12].

3.4.4 Computing correlated equilibria of sets

Our generalized algorithm requires an equilibrium-filter function F_{eq}. Formally this is a monotonic function $F_{eq} : \mathcal{P}(R^n) \times \ldots \times \mathcal{P}(R^n)) \to \mathcal{P}(R^n)$ which outputs a closed convex subset of the smallest convex set containing the union of the input sets. Here $\mathcal{P}$ denotes the powerset, so F_{eq} takes a list of subsets in R^n and returns another subset. It is monotonic so $x \subseteq y \Rightarrow F_{eq}(x) \subseteq F_{eq}(y)$. The inputs x and y are ordered lists of sets, so when we say $x \subseteq y$ we mean that corresponding sets are subsets (i.e. $\forall i, x_i \subseteq y_i$). The th reat function F_{th} is also passed to F_{eq}. Note than requiring F_{eq} to return a closed convex set disqualifies Nash equilibria and its refinements. Due to the availability of cheap talk, reasonable choices for F_{eq} include correlated equilibria (CE), ϵ-CE, or a coalition resistant variant of CE. Filtering non-equilibrium policies takes place when the various action achievable-sets (Q) are merged together as shown in step E of Figure 9. Constructing F_{eq} is more complicated than computing the equilibria for a stage game so we describe below how to target CE.

For a normal-form game the set of correlated equilibria can be determined by taking the intersection of a set of halfspaces (linear inequality constraints) [28]. Each variable of these halfspaces represents the probability that a particular joint action is chosen (via

a shared random variable) and each halfspace represents a rationality constraint that a player being told to take one action would not want to switch to another action. There are $\sum_1^n |A_i|(|A_i|-1)$ such rationality constraints (where $|A_i|$ is the number of actions player i can take).

Unlike in a normal-form game, the rewards for following the correlation device or defecting (switching actions) are not directly given in our dynamic program. Instead we have a achievable-set of possible outcomes for each joint action $Q(s,\vec{a})$ and a threat function F_{th}. Recall that when following a policy to achieve a desired payoff, not only must a joint action be given, but also subsequent payoffs for each successor state. Thus the halfspace variables must not only specify probabilities over joint actions but also the subsequent payoffs (a probability distribution over the extreme points of each successor achievable-set). Luckily, a mixture of probability distributions is still a probability distribution so our final halfspaces now have $\sum_{\vec{a}} |Q(s,\vec{a})|$ variables (we still have the same number of halfspaces with the same meaning as before).

At the end of the day we do not want feasible probabilities over successor states, we want the utility-vectors afforded by them. To achieve this without having to explicitly construct the polytope described above (which can be exponential in the number of halfspaces) we can describe the problem as the following MOLP (given $Q(s,\vec{a})$ and F_{th}):

Simultaneously maximize:

player rewards: foreach player i from 1 to n:

$$\sum_{\vec{a}\vec{u}} u_i x_{\vec{a}\vec{u}}$$

Subject to: (8)

probability constraints: $\sum x_{\vec{a}\vec{u}} = 1$ and $x_{\vec{a}\vec{u}} \geq 0$

rationality constraints: foreach player i, actions a_1,$a_2 \in A_i$, $(a_2 \neq a_1)$

$$\sum_{\vec{a}\vec{u}|a_i=a_1} u_i x_{\vec{a}\vec{u}} \geq \sum_{\vec{a}\vec{u}|a_i=a_2} F_{th}(s,\vec{a}) x_{\vec{a}\vec{u}}$$

where variables $x_{\vec{a}\vec{u}}$ represent the probability of choosing joint action $\vec{a}$ and subsequent

payoff $\vec{u} \in Q(s, \vec{a})$ in state s and u_i is the utility to player i.

Despite scaling linearly with the number of states, this multiobjective linear program for computing the equilibrium hull scales very poorly. The MOLP remains tractable only up to about 15 joint actions (which results in a few hundred variables and a few dozen constraints, depending on achievable-set size). In the next section we show how to replace this MOLP with a series of linear programs. This dramatically increases the number of joint-actions we can handle, up to a few hundred.

3.4.5 Proof of correctness

Murray and Gordon [54] proved correctness and convergence for the exact algorithm by proving four properties: 1) Monotonicity (achievable-sets only shrink), 2) Achievability (after convergence, achievable-sets contain only achievable joint-utilities), 3) Conservative initialization (initialization is an over-estimate), and 4) Conservative backups (backups don't discard valid joint-utilities). We show that our approximation algorithm maintains these properties.

1) Our achievable-set approximation scheme was carefully constructed so that it would not permit backtracking, maintaining monotonicity (all other steps of the backup are exact). 2) We have broadened the definition of achievability to permit $\epsilon_1/(1-\gamma)$ error. After all achievable-sets shrink by less than ϵ_1 we could modify the game by giving a bonus reward less than ϵ_1 to each player in each state (equal to that state's shrinkage). This modified game would then have converged exactly (and thus would have a perfectly achievable achievable-set as proved by Murray and Gordon). Any joint-policy of the modified game will yield at most $\epsilon_1/(1-\gamma)$ more than the same joint-policy of our original game thus all utilities of our original game are off by at most $\epsilon_1/(1-\gamma)$. 3) Conservative initialization is identical to the exact solution (start with a huge hyperrectangle with sides $R^i_{max}/(1-\gamma)$). 4) Backups remain conservative as our approximation scheme never underestimates (as shown in Section 3.4.2) and our equilibrium filter function F_{eq} is required to be monotonic and thus will never underestimate if operating on overestimates (this is why we require monotonicity of

F_{eq}). CE over sets as presented in Section 3.4.4 is monotonic. Thus our algorithm maintains the four crucial properties and terminates with all exact equilibria (as per conservative backups) while containing no equilibrium with error greater than $\epsilon_1/(1-\gamma)$.

3.4.6 Approximate set dynamic programming

We have now presented all the pieces needed to preform approximate value iteration on achievable sets (Algorithm 2). We first start with over estimates for each achievable set. When then perform an exact backup using Equation 3.4.3 to improve the efficiency of our weighted set sums. After backing up a state, we approximate it (while guaranteeing no contraction occurs). This process is repeated until convergence to a given threshold.

Algorithm 2 An Approximate Achievable-set Algorithm

Inputs	stochastic game $(S, s^{(0)}, P, A, R)$, discount factor γ
	equilibrium filter function F_{eq} using threats F_{th}
	approximation parameters ϵ_1 and ϵ_2 a set of approximation hyperplanes H
Output	state achievable-sets V
	state-action achievable-sets Q

for all $s, \vec{a}$ **do**
  $V(s) \Leftarrow$ hypercube with max and min utilities: $\frac{R^p_{min}}{1-\gamma}$, $\frac{R^p_{max}}{1-\gamma}$ for each agent p
repeat
  for all $s, \vec{a}$ **do**
    $Q(s, \vec{a}) \Leftarrow \sum_{s'} P(s'|s, \vec{a})\dot{V}(s')$ using Equation 7
    $Q(s, \vec{a}) \Leftarrow R(s, a) + \gamma Q(s, \vec{a})$
  for all s **do**
    $V(s) \Leftarrow$ result of Equation 8
    $V(s) \Leftarrow$ approximation of $V(s)$ (*e.g.,* using the method from Section 3.4.2)
until Converged to within ϵ_1
return V, Q

This approximate algorithm, using Section 3.4.2's set-approximation, yields a solution that is an $\epsilon_1/(1-\gamma)$-equilibrium of the full game while guaranteeing there exists no exact equilibrium that Pareto-dominates the solution's utility. This means that despite not being able to calculate the true utilities at each stage game, if other players did know the true utilities they would gain no more than $\epsilon_1/(1-\gamma)$ by defecting. Moreover our approximate solution is as good or better than any true equilibrium. By targeting an $\epsilon_1/(1-\gamma)$-equilibrium we do not mean that the backup's equilibrium filter function F_{eq} is

an ϵ-equilibrium (it could be, although making it such would not alleviate the convergence problem, but instead would guarantee that our solution is better than any ϵ-equilibrium and is an $(\epsilon + \epsilon_1)$-equilibrium). We do not change the filter function to guarantee convergence. Instead, we apply the standard filter function but stop if no achievable-set has changed by more than ϵ_1.

The primary bottleneck with this algorithm is that mutli-objective linear programming can be costly to perform exactly. Even over a single iteration, the number of resulting vertices can be very large, and the number of iterations required to compute these vertices can be prohibitive. It is inefficient to first compute an exact backup and then approximate. In the next section we show how to combine these two steps so that the backup and the approximation can be combined into a set of linear programs.

3.5 The quick polytope approximation algorithm for all correlated equilibria (QPACE)

The generic achievable-set based approximation algorithm given above is severely limited in its scalability by the complexity of multi-objective linear programs (MOLP). Both the computation of Minkowski sums as well as the set-based equilibrium filtering each require one MOLP per state per iteration. In this section we change the representation of achievable-sets from a context combination of vertices to an intersection of half-spaces. This change allows us to efficiently approximate each step of the backup while arriving at a final approximation no worse than the previous algorithm. Most importantly, we are now able to extremely efficiently compute approximate Minkowski sums as well as compute the set-based equilibrium filtering as a series of linear programs instead of a MOLP. First we discuss the new achievable-set representation. We then make more precise our view that a stochastic game is a sequence of augmented normal-form games. We then present how to improve computation of action achievable sets (which includes the Minkowski-sum) as well as how to compute the set of correlated equilibria (*a.k.a.* filtering). Finally, we give a few algorithmic optimizations and prove that this new algorithm is still correct.

3.5.1 Representing achievable-sets

QPACE represents each achievable-set as a set of m linear inequalities with fixed coefficients. Recall that the achievable-set is closed and convex, and can be thought of as an n-dimensional polytope (one dimension for each player's utility). Polytopes can be defined as an intersection of halfspaces. Each of the m halfspaces $j \in H$ consists of a normal H_j and an offset b_j such that the achievable joint-utility x is restricted by the linear inequality $H_j x \leq b_j$. The halfspace normals form a matrix $H = [H_1, \cdots, H_m]$, and the offsets a vector $b = \langle b_1, \cdots, b_m \rangle$. For example, the polytope depicted in Figure 11c can be represented by the equation $Hx \leq b$ where H and b are as shown.

While any achievable-set may be represented by an intersection of halfspaces, it may require an unbounded number. Such a growth in the complexity of achievable-sets does indeed occur in practice; therefore, we compute an approximation using halfspaces sharing the same fixed set of normals. Each of our polytopes differ only in their offsets, so QPACE must only store the b vector for each $V(s)$ and $Q(s, \vec{a})$. The normals H are chosen at initialization to enable a regular approximation of a Euclidean ball (Figure 11a).

To approximate an achievable-set, we find the minimum offsets such that our approximation completely contains the original set; that is, we retain each hyperplane's normal but contract each offset until it barely touches the given polytope. This process is shown for the game of chicken in Figure 11. We call this the *regular polytope approximation (RPA)* of polytope P using normals H where $RPA(H, P)_j = \max_{x \in P} [H_j \cdot x]$ for each offset j. We say that H *permits error* ϵ if $\|RPA(H, P) - P\| \leq \epsilon$. RPA was first presented in Mac Dermed and Isbell [51].

3.5.2 Augmented stage games and threats

Agents choosing which action to execute consider not only their immediate reward but also the long term utility of the next state in which they may arrive. The expected joint-utility for each joint-action is known as the *continuation utility*, $\overrightarrow{cu_{\vec{a}}}$. Because the agents consider both their immediate reward and their continuation utility they are in essence playing an *augmented stage game* with payoffs $R(s, \vec{a}) + \overrightarrow{cu_{\vec{a}}}$.

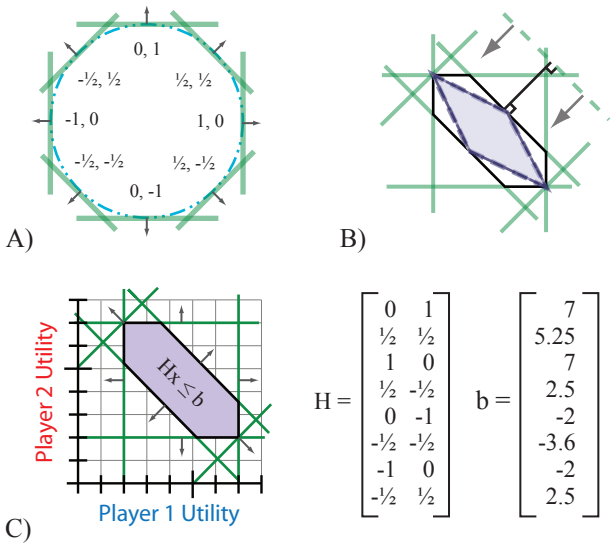

Figure 11: The QPACE representation. A) A Euclidean ball surrounded by halfspaces suitable for a regular polytope approximation (RPA). The normals of each halfspace are illustrated. Note that the halfspaces approximate the inscribed dotted circle. B) The achievable-set from Figure 4b being approximated using the RPA from (A). Each halfspace of the Euclidean ball is moved such that it touches the achievable-set at one point. C) The resulting polytope approximation whose normals and offsets are specified in H and b.

Assuming that the players cooperate with each other, they choose to jointly play any equilibrium policy in the successor state s'. Thus, the continuation utility $\overrightarrow{cu}^{s'}$ of each successor state may be any point in the achievable-set $V(s')$ of that state. We call the set of possible continuation utilities, $Q(s, \vec{a})$, the *action achievable-set function*. When there is only one successor, *i.e.*, $P(s'|s, \vec{a}) = 1$, then $Q(s, \vec{a}) = V(s')$. When there is more than one successor state, the continuation utility is the expectation over these utilities, *i.e.*,

$$Q^*(s, \vec{a}) = \left\{ \sum_{s'} P(s'|s, \vec{a}) \overrightarrow{cu}_{\vec{a}s'} \mid \overrightarrow{cu}_{\vec{a}s'} \in V^*(s') \right\} \tag{9}$$

It is advantageous for players to punish other players who deviate from the agreed upon joint-policy. Given rationality, players will not deviate from the chosen equilibrium and thus players will never be forced to enact their threats. This allows threats to be any policy not necessarily in equilibrium (so called incredible threats). The harshest threat possible is a global *grim trigger strategy*, where all other players cooperate in choosing policies minimizing the defecting player's rewards for the rest of the game. We employ the grim trigger strategy, as this maximizes the size of the achievable-set. The grim trigger

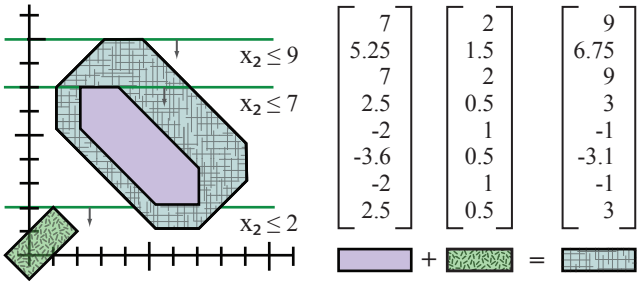

Figure 12: An example approximate Minkowski sum of RPA polytopes. Our method is exact in two dimensions and permits only ϵ error in higher dimensions.

threat-point ($\overrightarrow{gt_{\vec{a}i}}$) is a continuation utility that players choose for each joint-action $\vec{a}$ in the event that player i defects. This utility can be calculated as a zero-sum game using Nash-Q learning [33] independently at initialization.

3.5.3 Computing action achievable-sets

At the beginning of each iteration, QPACE calculates the action-achievable-set $Q(s, \vec{a})$ for each state s and each joint action $\vec{a}$ in the stochastic game. We do so by rewriting Equation 9 using Minkowski addition. Given two sets A and B, their Minkowski sum $A + B$ is defined as $A + B = \{a + b \mid a \in A, b \in B\}$. Efficient computation of Minkowski addition of two convex polytopes of arbitrary dimension d is an open and active problem, with complexity $O(|A|^d)$ where $|A|$ is the number of halfspaces; however we can approximate a Minkowski sum efficiently using our representation as the sum of the offsets (Figure 12). We prove this result:

Lemma 3.5.0.1. *Suppose that A and B represent polytopes ($Hx \leq b^A$) and ($Hx \leq b^B$) and H permits only ϵ error as discussed in Section 3.5.1. Then the weighted Minkowski sum $\alpha A + \beta B$ for constants $\alpha, \beta \in \mathbb{R}^+$ and $\alpha + \beta = 1$ is approximated by the polytope $Hx \leq \alpha \cdot b^A + \beta \cdot b^B$ with at most ϵ relative error.*

Proof. Noting $RPA(H, A) \leq \epsilon$ and $RPA(H, B) \leq \epsilon$,

$$\begin{aligned} RPA(H, \alpha A + \beta B) &= \max_{x \in A} \max_{y \in B} [\alpha(H_i \cdot x) + \beta(H_j \cdot y)] \\ &= \alpha \cdot \max_{x \in A} [H_j \cdot x] + \beta \cdot \max_{y \in B} [H_j \cdot y] \\ &= \alpha \cdot RPA(H, A) + \beta \cdot RPA(H, B) \\ &\leq \epsilon \cdot (\alpha + \beta) \leq \epsilon, \end{aligned}$$

□

Note that multiple additions do not increase relative error. Equation 9 rewriten as a weighted Minkowski sum becomes: $Q(s, \vec{a}) = \sum_{s'} P(s'|s, \vec{a})V(s)$. To improve the performance of subsequent steps we scale the set of continuation utilities by the discount factor γ and add the immediate reward (translating the polytope). The final offsets for the action-achievable-sets may be computed for each $j \in H$ as:

$$\boxed{Q(s, \vec{a})_j = R(s, \vec{a}) \cdot H_j + \gamma \sum\nolimits_{s?} P(s?|s, \vec{a})V(s?)_j} \tag{10}$$

3.5.4 Defining the set of correlated equilibria

Calculating the set of CE in a normal form stage game (*e.g.* Figure 4b) is straightforward. A probability distribution X over joint-actions (with $x_{\vec{a}}$ being the probability of choosing joint-action $\vec{a}$) is in equilibrium if and only if the reward of following the prescribed action is no worse than taking any other action. More formally, X is a CE if and only if in state s, for each player i, for distinct actions α, $\beta \in A_i$, where $\vec{a}^{(\alpha)}$ means joint-action $\vec{a}$ with player i taking action α:

$$\sum\nolimits_{\vec{a}} x_{\vec{a}^{(\alpha)}} R(s, \vec{a}^{(\alpha)})_i \;\geq\; \sum\nolimits_{\vec{a}} x_{\vec{a}^{(\alpha)}} R(s, \vec{a}^{(\beta)})_i \tag{11}$$

These rationality constraints are linear inequalities and together with the probability constraints $\sum x_{\vec{a}} = 1$ and $x_{\vec{a}} \geq 0$ define a polytope in $\mathbb{R}^n$. Any point in this polytope represents a CE which yields a value-vector $\sum_{\vec{a}} x_{\vec{a}} R(s, \vec{a})$. The union of all such value-vectors (the polytope projected into value-vector space) is the achievable-set of state s when agents do not consider utilities gained from future states.

Agents do not play a single normal form game. Instead they play the augmented game where in order for an agent to make a decision in the current state, they must know which utility among the many possible in $Q(s, \vec{a})$ they will receive in each successor state. Therefore, a CE of the augmented stage game consists of both a probability $x_{\vec{a}}$ for each joint-action and an expected continuation utility $\overrightarrow{cu}_{\vec{a}i}$ for each player and joint-action, resulting in $(n+1)|A|^n$ variables. Given a state s, the set of possible CEs over the variables $x_{\vec{a}}$ and $\overrightarrow{cu}_{\vec{a}i}$ of the augmented game becomes:

$$\text{For each player } i, \text{ distinct actions } \alpha, \beta \in A_i,$$

$$\sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} \overrightarrow{cu_{\vec{a}^{(\alpha)}}}_i \geq \sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} [\overrightarrow{gt_{\vec{a}^{(\beta)}}}_i + R(s, \vec{a}^{(\beta)})_i] \tag{12}$$

$$\text{For each joint-action } \vec{a} \in A^n, \text{ and } j \in H\text{:}$$

$$\overrightarrow{cu}_{\vec{a}} \in Q(s, \vec{a}) \qquad (i.e., \quad H_j \, \overrightarrow{cu}_{\vec{a}} \leq Q(s, \vec{a})_j \quad) \tag{13}$$

The rationality constraints (12) are quadratic because we have a variable for the continuation utility ($\overrightarrow{cu}_{\vec{a}i}$) which must be scaled by our variable for the probability of that joint-action occurring ($x_{\vec{a}}$). We can eliminate this multiplication by scaling the action-achievable-set $Q(s, \vec{a})$ in inequality 13 by $x_{\vec{a}}$ making $\overrightarrow{cu}_{\vec{a}i}$ dependent on $x_{\vec{a}}$. This gives us our polytope in $\mathbb{R}^{(n+1)|A|^n}$ over variables $\overrightarrow{cu}_{\vec{a}i}$ and $x_{\vec{a}}$ of achievable correlated equilibria:

$$\boxed{\begin{array}{c} \text{For each player } i, \text{ distinct actions } a_1, a_2 \in A_i, \\ \sum_{\vec{a} \in A^n} \overrightarrow{cu_{\vec{a}^{(\alpha)}}}_i \geq \sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} [\overrightarrow{gt_{\vec{a}^{(\beta)}}}_i + R(s, \vec{a}^{(\beta)})_i] \\ \sum_{\vec{a} \in A^n} x_{\vec{a}} = 1 \text{ and } \forall \vec{a} \in A^n, \; x_{\vec{a}} \geq 0 \\ \text{For each joint-action } \vec{a} \in A^n, \\ \overrightarrow{cu}_{\vec{a}} \in x_{\vec{a}} \, Q(s, \vec{a}) \quad (i.e., \quad H_j \, \overrightarrow{cu}_{\vec{a}} \leq x_{\vec{a}} \, Q(s, \vec{a})_j \;) \end{array}} \tag{14}$$

3.5.5 Computing the achievable-set function

The polytope defined above in (14) is the set of CE in joint-action-probability ($x_{\vec{a}}$) and continuation-utility ($\overrightarrow{cu}_{\vec{a}i}$) space; however, we do not ultimately want the set of CEs only

the set of resulting value-vectors $V(s)$. The value that a particular correlated equilibrium presents to a player i is $\sum_{\vec{a}} \overrightarrow{cu}_{\vec{a}i}$. Recall that we are approximating our achievable-sets with a fixed set of halfspaces, where each halfspace touches a vertex of the exact polytope (Figure 11b). This vertex will be the point in the exact achievable polytope that maximizes the dot product with the halfspace's normal. We can then find this vertex using a linear program where the achievable-set is as defined in (14) and the objective function is a dot product of the value to each player and the halfspace's normal. In other words, we wish to find the CE that maximizes a weighted average of the agent's utilities where the weights are given by the halfspace's normal. We compute one linear program for each approximating halfspace and update its offset to the optimal value found. For each halfspace j with normal H_j we compute offset b_j as the maximum objective function value to the following LP:

$$V(s)_j = \begin{cases} \max & \sum_{\vec{a}} \sum_{i \in I} H_{j,i} \cdot \overrightarrow{cu}_{\vec{a}i} \\ \text{subject to} & \text{inequalities in (14)} \end{cases} \tag{15}$$

3.5.6 Linear program solution caching

Every iteration, QPACE solves one LP (Equation 15) for each halfspace in every state. Depending on the value of ϵ, the number of halfspaces and hence the number of LPs can become very large. Solving these LPs is the main bottleneck in QPACE. Fortunately, we can dramatically improve the performance of these LPs by taking advantage of the fact that the solutions of many of the LPs do not tend to "change" significantly from iteration to iteration. More precisely, the maximum offset difference for a given achievable-set in consecutive iterations is small. For a fixed state, we can therefore expect the solution corresponding to the LP of one iteration to be quite close to that of the previous iteration. LPs typically spend the bulk of their running time searching over a space of *basic achievable solutions* (BFS), so choosing an initial BFS close to the optimal one dramatically improves performance. QPACE caches the optimal BFS of each state/halfspace calculation and uses the solution as the starting BFS in the next iteration. Empirical tests confirm that after the first few iterations of QPACE the LPs will find an optimal BFS very near the cached

Algorithm 3 The QPACE Algorithm.

Inputs	stochastic game $(S, s^{(0)}, P, A, R)$, discount factor γ
	equilibrium filter function F_{eq} using threats F_{th}
	approximation parameters ϵ_1 and ϵ_2 a set of approximation hyperplanes H
Outputs	state achievable-sets V
	state-action achievable-sets Q

for all $s \in S$, $j \in H$ **do**
  $V(s)_j \Leftarrow \max_{s',\vec{a}}(R(s', \vec{a}) \cdot H_j)/\gamma$
repeat
  for all $s, \vec{a}$, $j \in H$ **do**
    $Q(s, \vec{a})_j =$ equation (10)
  for all s, $j \in H$ **do**
    $V(s)_j \Leftarrow$ LP (15) starting at BFS-cache(s, j)
    BFS-cache$(s, j) \Leftarrow$ optimal BFS of the LP
until $\forall s : V(s)_j$ changed less than $\epsilon/2$
return $\{V\}$

BFS (Figure 16f). In fact, a majority of cached BFS are actually already optimal for the new LP. Note that even if the optimal BFS from one iteration is exactly the optimal BFS of the next, the optimal values may be different as the offsets change.

This optimization only applies if a BFS from one iteration, x^*, is a valid BFS in the next. Every LP has identical variables, objectives, and constraints, with subsequent iterations differing only in the offsets b; therefore, x^* is a basic solution for the new LP. While x^* is basic, it may not be feasible for the new LP. However, LP sensitivity analysis guarantees that an optimal BFS from one iteration is a BFS of the dual problem in the next iteration when only b changes. Therefore, we alternate each iteration between solving the primal and dual problems.

3.5.7 Proof of correctness

Murray and Gordon [27] proved correctness and convergence for the exact algorithm by proving four properties: 1) Monotonicity (achievable-sets only shrink), 2) Achievability (after convergence, achievable-sets contain only achievable joint-utilities), 3) Conservative initialization (initialization is an over-estimate), and 4) Conservative backups (backups don't discard valid joint-utilities). We show that our approximation algorithm maintains these properties.

1) Our achievable-set approximation scheme was carefully constructed so that it would not permit backtracking, maintaining monotonicity (all other steps of the backup are exact). 2) We have broadened the definition of achievability to permit $\epsilon_1/(1-\gamma)$ error. After all achievable-sets shrink by less than ϵ_1 we could modify the game by giving a bonus reward less than ϵ_1 to each player in each state (equal to that state's shrinkage). This modified game would then have converged exactly (and thus would have a perfectly achievable achievable-set as proved by Murray and Gordon). Any joint-policy of the modified game will yield at most $\epsilon_1/(1-\gamma)$ more than the same joint-policy of our original game thus all utilities of our original game are off by at most $\epsilon_1/(1-\gamma)$. 3) Conservative initialization is identical to the exact solution (start with a huge hyperrectangle with sides $R^i_{max}/(1-\gamma)$). 4) Backups remain conservative as our approximation scheme never underestimates (as shown in Section 3.4.2) and our equilibrium filter function F_{eq} is required to be monotonic and thus will never underestimate if operating on overestimates (this is why we require monotonicity of F_{eq}). CE over sets as presented in Section 3.5.4 is monotonic. Thus our algorithm maintains the four crucial properties and terminates with all exact equilibria (as per conservative backups) while containing no equilibrium with error greater than $\epsilon_1/(1-\gamma)$.

3.6 Empirical results

3.6.1 Breakup game

We implemented a version of our algorithm targeting exact correlated equilibrium using grim trigger threats (defection is punished to the maximum degree possible by all other players, even at one's own expense). The grim trigger threat reduces to a 2 person zero sum game where the defector receives their normal reward and all other players receive the opposite reward. Because the other players receive the same reward in this game they can be viewed as a single entity. Zero sum 2-player stochastic games can be quickly solved using FFQ-Learning [49]. Note that grim trigger threats can be computed separately before the main algorithm is run. When computing the threats for each joint action, we use the GNU Linear Programming Kit (GLPK) to solve the zero-sum stage games. Within the main algorithm itself we use ADBASE [85] to solve our various MOLPs. Finally we use QHull

[8] to compute the convex hull of our achievable-sets and to determine the normals of the set's facets. We use these normals to compute the approximation. To improve performance our implementation does not compute the entire achievable hull, only those points on the Pareto frontier. A final policy will exclusively choose targets from the frontier (using F_{bs}) (as will the computed intermediate equilibria) so we lose nothing by ignoring the rest of the achievable-set (unless the threat function requires other sections of the achievable-set, for instance in the case of credible threats). In other words, when computing the Pareto frontier during the backup the algorithm relies on no points except those of the previous step's Pareto frontier. Thus computing only the Pareto frontier at each iteration is not an approximation, but an exact simplification.

We tested our algorithm on a number of problems with known closed form solutions, including the breakup game (Figure 14). We also tested the algorithm on a suite of random games varying across the number of states, number of players, number of actions, number of successor states (stochasticity of the game), coarseness of approximation, and density of rewards. All rewards were chosen at random between 1 and -1, and γ was always set to 0.9. A typical run is shown in Figure 13. Note that unlike single agent value iteration, the convergence rate is very erratic (recall that a small change in utility estimation can lead to a large change in the equilibrium).

An important empirical question is what degree of approximation should be adopted. Our testing (Figure 15) suggests that the theoretical requirement of $\epsilon_2 < \epsilon_1$ is overly conservative. While the bound on ϵ_2 is theoretically proportional to $R_{max}/(1-\gamma)$ (the worst case scale of the achievable-set) a more practical choice for ϵ_2 would be in scale with the final achievable-sets (as should a choice for ϵ_1).

3.6.2 Random games

We ran tests to determine the scalability of QPACE across a number of different dimensions: precision (Figure 16a), number of states (Figure 16b), number of joint-actions (Figure 16c), and number of players (Figure 16d). We ran the algorithms over generated games, random in the state transition and reward functions. Unless otherwise indicated, the games were

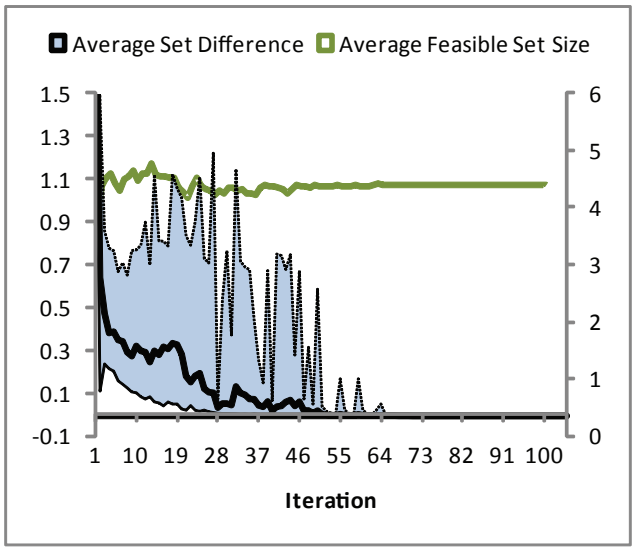

Figure 13: A representative run of the generic approximation algorithm on a random game. Shaded blue area represents the minimum and maximum change in feasible sets between iterations.

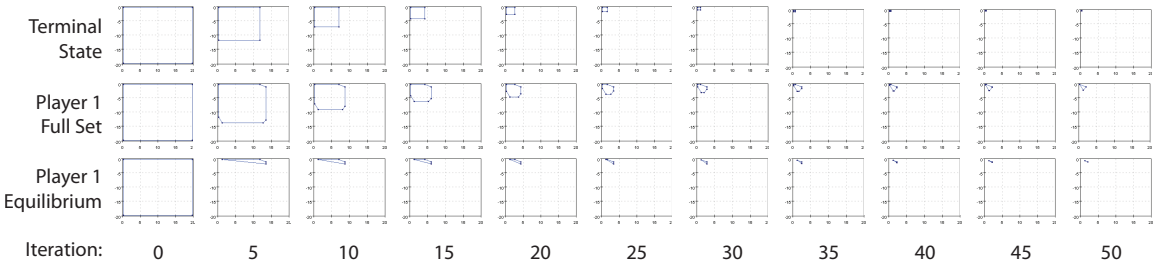

Figure 14: A visualization of achievable-sets for the terminal state and player 1's state of the breakup game at various iterations of the dynamic program. By the 50th iteration the sets have converged.

run with 10 states, 2 players, 2 actions each, and with $\epsilon = 0.05$ and $\gamma = 0.9$. We used the GLPK library as our linear programming solver, and used the FFQ-Learning algorithm [49] to compute grim trigger threats.

As users specify more precision, the number of hyperplanes in our approximate Euclidean ball increases exponentially. So, wall clock time increases exponentially as ϵ decreases as Figures 16a confirms. The QPACE algorithm begins to do worse than Mac Dermed and Isbell (MI09) for very low values of ϵ, especially in QPACE without caching (Figure 16a). To see why, note that if an achievable-set can be represented by only a few vertices or halfspaces (*e.g.* a hypercube) than MI09 will only use those vertices necessary while QPACE

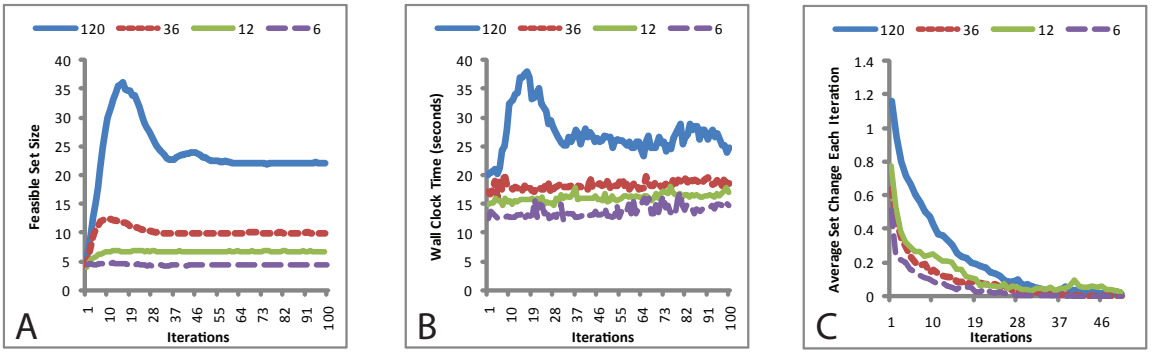

Figure 15: Statistics from a random game (100 states, 2 players, 2 actions each, with $\epsilon_1 = 0.02$) run with different levels of approximation. The numbers shown (120, 36, 12, and 6) represent the number of predetermined hyperplanes used to approximate each Pareto frontier. A) The better approximations only use a fraction of the hyperplanes available to them. B) Wall clock time is directly proportional to the size of the achievable-sets. C) Better approximations converge more each iteration (the coarser approximations have a longer tail), however due to the additional computational costs the 12 hyperplane approximation converged quickest (in total wall time). The 6, 12, and 36 hyperplane approximations are insufficient to guarantee convergence ($\epsilon_2 = 0.7$, 0.3, 0.1 respectively) yet only the 6-face approximation occasionally failed to converge.

will always use all halfspaces in H.

Both MI09 and QPACE scale linearly with the number of states independent of other parameters (Figure 16b), however QPACE is about 240 times faster than MI09 with the default parameters. The number of variables used in each LP is linear with respect to the number of joint-actions. LP's in turn are polynomial with respect to the number of variables. Therefore as expected, both algorithms are polynomial with respect to joint-actions (Figure 16c). However, QPACE uses significantly fewer variables per joint-action and therefore has a lower rate of growth.

QPACE scales much better than MI09 with respect to the number of players. The number of joint-actions grows exponentially with the number of players, so both algorithms have exponential growth in the number of players; however, the number of vertices needed to represent a polytope also grows exponentially while the number of halfspaces only grows polynomially. Thus, QPACE is able to handle many more players than MI09.

A natural question to ask is how much each of QPACE's new aspects contribute to the overall efficiency. The change of representation permits three key improvements: fewer

variables in the linear programs, an elimination of the need to calculate vertices of the polytopes, fast Minkowski addition, and LP solution caching. By comparing QPACE without caching to MI09 in Figures 16a-c we observe the impact of the first two improvements. We now examine Figures 16e&f to determine the contribution of the last two improvements.

Both QPACE and MI09 employ Minkowski sums when computing the expected continuation utility over future states (when transitions are deterministic, neither algorithm computes Minkowski sums). As the number of possible successor states increases, both algorithms perform more Minkowski additions. Figure 16e graphs the effect of additional successor states for both MI09 and QPACE and shows that while MI09 suffers from a 10% increases in time when the number of successors is large, QPACE is unaffected.

From Figure 16f, we observe that as the number of iterations progresses, the average number of LP iterations involved decreases quickly for QPACE with caching. The algorithm initially takes around 100 LP iterations, and drops to less than 3 LP iterations by the sixth iteration. On the other hand, QPACE without caching starts at around 20 LP iterations and plateaus to a consistent 70 LP iterations. The graphs demonstrate that the LP caching does in fact make a significant difference in running time: QPACE with caching consistently takes around 0.2 seconds per iteration after the tenth iteration, but QPACE without caching takes over 1.5 seconds per iteration. In the long run, LP caching contributes an order of magnitude speed boost.

3.7 Extensions

The achievable-set based dynamic programming algorithms, specifically QPACE, presented in this chapter can easily be augmented to target a variety of solution concepts. The easiest means to accomplish this is to change the threat function and/or the equilibrium filter function.

For example, QPACE can easily be modified to produce sub-game perfect equilibria by making the threat points $\overrightarrow{gt_{\vec{a}i}}$ variables in (14) and constraining them in a similar way to the $\overrightarrow{cu_{\vec{a}i}}$; we then employ the same trick described in Section 3.5.4 to remove the resulting quadratic constraints. As another example, we explore below the case of changing the

equilibrium filter function to target commitment equilibrium instead of correlated.

QPACE can also be extended to work in games of imperfect monitoring and imperfect recall. In these games player's don't observe actions so they can't use threats and can't choose different continuation points for each joint-action. Thus $\overrightarrow{gt_{\vec{a}i}} = \overrightarrow{cu_{\vec{a}i}}$ and the values of $Q(s, \vec{a})_j$ are interdependent across actions, forcing equations (9) and (14) to become one large set of inequalities.

3.7.1 Computing Commitment Equilibria

The QPACE algorithm (3) as described so far computes the set of correlated equilibria. However, the algorithm can be easily modified to solve for other solution concepts as well, as long as the concept has a convex achievable region. One popular solution concept is a commitment equilibrium (*a.k.a.* Stackelberg strategies), where one player in a two-player game is able to select their strategy first (the leader), and reveal this locked-in strategy to the other player (follower). This ability gives the leader an advantage for two reasons: One, the leader can unilateral narrow down the choices of equilibrium, often times outright selecting the equilibrium. This avoids any issues with equilibrium selection (Section 3.2.2). Two, the leader is no longer constrained by rationality constraints. This improves the set of possible strategies. Computing optimal mixed commitment strategies has proved valuable for several real security problems, including airport security [69, 72], assigning Federal Air Marshals to flights [88], and Coast Guard patrols [80].

The set of commitment equilibria is the same as the set of correlated equilibria except without rationality constraints for the leader. Therefore commitment equilibria generalize correlated equilibria, meaning that commitment equilibria can be better than any correlated equilibria. Because the leader gets to narrow down the choice of possible equilibria by selecting a strategy first, the advantage of being a leader is strictly beneficial (never decreases utility) [90].

The following well known example (Figure 3.7.1) illustrates how advantageous commitment can be. When players move simultaneously (a standard game without a leader), the unique Nash equilibrium is (U, L) because U strictly dominates D and the game is solvable

by iterated elimination of dominated policies. In this unique Nash equilibrium, Player 1 (the row player) receives a payoff of 1. However, when player 1 becomes the leader, then she can commit to playing D. Player 2 will now rationally play R resulting in player 1 getting a payoff of 2. If player 1 is the leader, player 1 can do even better if she uses the mixed strategy of $(.5 - \epsilon, .5 + \epsilon)$ which still leads player 2 to choose R and player 1 gets a payoff of $2.5 - \epsilon$. While it can be advantageous for the leader to commit to a mixed strategy, the follow will never do better than his pure strategies (player 2's decision becomes a single agent problem).

Most of the work on computing mixed Stackelberg strategies has focused on normal-form games (albeit combinatorialy large normal-form games) where the leader must commit to a strategy over her entire lifetime without being able to react to the actions of the other agent. While there is one exception that works towards computing Stackelberg strategies in extensive-form games [46], there has been no work towards computing Stackelberg strategies for infinite horizon sequential problems. In such problems, we can also decide to what degree an agent can commit – perhaps a leader can only commit for short time spans and then must necessarily open themselves up to reprisal. We can modify the QPACE algorithm to solve this gap [47].

Every iteration of QPACE performs a backup which can be broken down into three steps: (1) Calculate the action achievable-sets $Q(s, \vec{a})$, giving us the set of possible continuation utilities. (2) Construct a set of inequalities that defines the set of equilibria. (3) Approximately project this achievable-set into value-vector space by solving a linear program for each hyperplane $V(s)_j$. Step two is the only step that needs to be changed to compute achievable commitment policies instead of correlated equilibria. We modify Equation 15 to not include rationality constraints for the leader. The resulting set of inequalities defining the set of commitment equilibria is shown in Equation 16. This gives us our polytope in $\mathbb{R}^{(n+1)|A|^n}$ over variables $\overrightarrow{cu}_{\vec{a}i}$ and $x_{\vec{a}}$ of achievable correlated equilibria:

$$
\boxed{\begin{array}{c}
\text{For each player } i \text{ who cannot commit,} \\
\text{for each pair of distinct actions } \alpha,\, \beta \in A_i \\
\sum_{\vec{a} \in A^n} \overrightarrow{cu_{\vec{a}^{(\alpha)}\, i}} \geq \sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} [\overrightarrow{gt_{\vec{a}^{(\beta)}\, i}} + R(s, \vec{a}^{(\beta)})_i] \\
\sum\nolimits_{\vec{a} \in A^n} x_{\vec{a}} = 1 \text{ and } \forall \vec{a} \in A^n,\ x_{\vec{a}} \geq 0 \\
\text{For each joint-action } \vec{a} \in A^n \text{ and halfspace } j \\
H_j\ \overrightarrow{cu_{\vec{a}}} \leq x_{\vec{a}}\ Q(s, \vec{a})_j
\end{array}} \tag{16}
$$

The rest of QPACE remains unchanged. Our modification to QPACE is minor and leaves the strong theoretical properties of the original algorithm intact. Most importantly, the algorithm converges to within ϵ in polynomial time and returns a set of commitment equilibria which is guaranteed to include all exact equilibria with additional solutions being no worse than ϵ-equilibria, where ϵ is the approximation parameter.

3.7.1.1 Commitment experiments on random games

We ran suites of experiments over sets of random games. These random games varied over the number of states, actions, stochasticity (the number of successor states with non-zero transition probability), and discount (γ). Unless otherwise noted, games were run with four joint-actions, five states, two successor states, a γ of 0.9 and an ϵ approximation error of 0.01. Results are averaged over 1000 games, which allows our utility results to be accurate within 0.02 with 99% confidence. A random game is generated using the following procedure: for each state joint-action pair k successor states are chosen at random. The simplex over these k states represents all possible probability distributions over these states. A transition probability distribution is chosen uniformly at random from this simplex. Each state joint-action pair is also assigned a reward for each player drawn uniformally at random. Finally, these rewards are normalized such that each player's rewards range between 0 and 1.

Our first set of experiments examines the scalability of the algorithm. Despite our algorithm having fewer constraints for each linear program than the original QPACE algorithm, we found our algorithm to have running time nearly identical to the original. Because

QPACE starts each linear program at the solution of the previous iteration's linear program, the total number of basis changes over the course of the entire algorithm is relatively small. Thus, fewer constraints reduces the overall running time by an insignificant amount. Our algorithm appears to scale linearly in the number of states, joint-actions, and $1/\epsilon$ (the first two of these are shown in Figure 18).

Our second set of experiments focuses on determining the importance of commitment vs. equilibrium selection. One of the more powerful aspects of committing is being able to dictate which particular equilibrium of the many possible will be chosen. Without commitment, players are faced with a bargaining problem to determine the which equilibrium will be chosen. This may result in significantly less utility for a potential leader. It is important to differentiate between the benefit of being able to commit to sub-rational policies and the benefit of equilibrium selection. Towards this end we compute both the Kalai-Smorodinsky bargaining solution [41], which favors equal gains to all parties, and the optimal selfish equilibrium for the potential leader with and without commitment (Figure 19).

The results show that as γ increases, the importance of committing, over and above just being able to select the equilibrium, decreases. This relationship is caused by threats becoming more powerful as the horizon increases. Strong threats act as a binding mechanism, permitting a wider array of possible equilibria by allowing players to punish each other for deviations without the need of someone violating her rationality constraint (as per folk-theorems). On the other hand, the benefit of equilibrium selection remains important regardless of the discount factor. The effect of equilibrium selection on the follower is even more dramatic than it is for the leader (Figure 19). With a small discount factor, the set of possible equilibria is small and thus likely to provide both players with similar utilities, even when the leader selects selfishly. As γ increases, a leader has more options for forcing the follower down a path more preferable to the leader at the expense of the follower.

Our third set of experiments examines how the number of actions affects the value of committing. We tested random games with a γ of both 0.0 and 0.4 across varying numbers of actions per player (Figure 20). We observe that as the number of actions increases, the

relative commitment gain decreases slightly because additional actions increase the probability that a correlated equilibrium without commitment will approach the unrestricted optimum, leaving less room for improvement by committing. While more actions decreases the importance of committing, the importance of being able to select the equilibrium (as opposed to having to bargain) remains high. This effect is stronger for larger values of γ. For games with very high discount factors, increasing the number of actions tends to increase the total value of the game, but not the relative importance of committing.

3.7.2 Dynamic halfspaces

Each polytope in QPACE is approximated by a regular set of halfspaces (Figure 11A). Using this regular scheme, the number of halfspaces grows proportionally to ϵ^{-n} where n is the number of players. Typically (from empirical observations) only a few halfspaces are necessary for high accuracy approximations – the rest are overkill. Determining which halfspaces are needed in which contexts is non-trivial but avoiding unnecessary halfspace calculations could lead to dramatic computational gains, particularly when the number of players is large.

One simple approach to alleviate this halfspace problem is to first solve the problem with a few regularly spaces halfspaces. We can then add halfspaces to our set H (keeping the originals) and solve the problem again. The old halfspaces with keep their offsets, while the new halfspaces will be initialized to a large over estimate. This will keep the solution of the previous run as the starting point for the next run. One difficulty with this approach is maintaining regularly spaces half-spaces each run. Minimizing the distance between halfspace normals is crucial for our error guarantees, however if we are running the algorithm multiple times, only the final normal distances will matter. This leads to the idea of selecting halfspaces for the most refined run and selecting subsets for the preceding runs. Another approach is to start with a hypercube for the first run, and then each additional run add a halfspace normal for each vertex of the previous set of normals with unit offset. Figure 21 shows the result of this process. Having each run use a regular set of halfspaces improves the accuracy of that iteration and places less demand on subsequent runs.

Another approach is to be selective about which halfspaces to add. By performing sensitivity analysis on each linear program we can determine which variables in the linear program are important to the result. These variables correspond to vertices in successor achievable-sets. Therefore if a variable contributes to a linear program's solution then the corresponding successor halfspaces that bound that particular vertex needs to be accurate. If they are not, then a new halfspace should be added to bound that particular vertex. On the flip side, if a halfspace bounds no important vertices, it no longer needs to be updated. We believe such a scheme will lead to substantial performance increases. Beyond general dynamic halfspace allocation, we believe the same approach can be used to efficiently target a specific achievable vector (*e.g.,* the point that maximizes equal division of the reward).

3.7.3 Limitations

Our approach is overkill when the achievable-sets are one dimensional (line segments) (as when the game is zero-sum, or agents share a reward function), because CE-Q learning [28] will converge to the correct solution without additional overhead. When there are no cycles in the state-transition graph (or one does not wish to consider cyclic equilibria) traditional game-theory approaches suffice. In more general cases, our algorithm brings significant advantages, however because we operate on vanilla Stochastic games with tabular transition and reward functions the problem still scales quadratically with the number of actions and exponentially with the number of agents. This prevents our algorithms from running efficiently on all but the simplest problems with more than four agents. Another large limitation is the assumption that the world is fully observable. Most multi-agent problems have some degree of uncertainty either about the state of the world or the goals of other agents. Neither case can be modeled by stochastic games. In the next chapter we begin to explore game of incomplete information and how to extend ideas from this chapter to partially observable stochastic games.

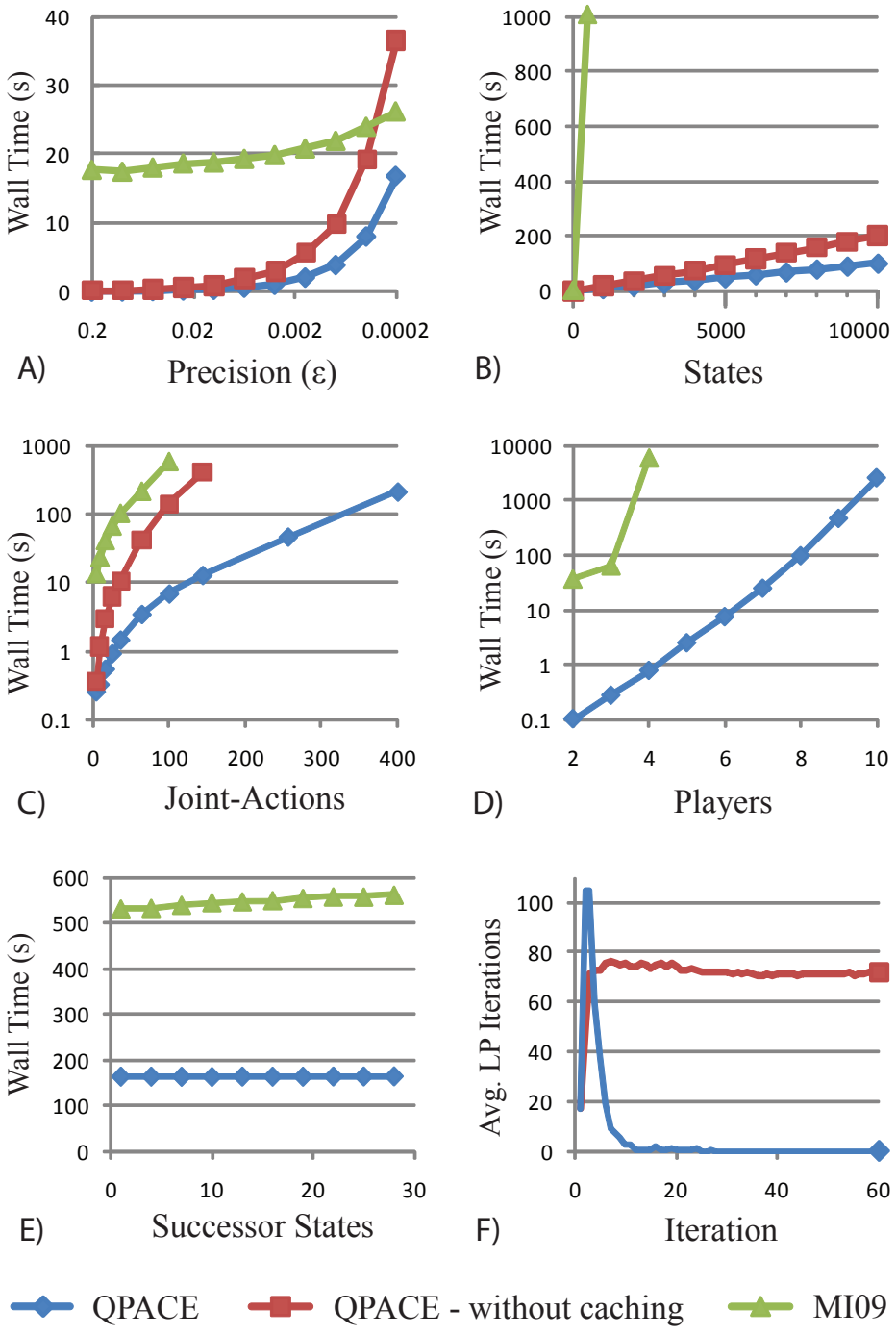

Figure 16: Performance evaluation. A-E) An illustration of how our algorithm scales along a number of variables. Note that some axis are logarithmic. F) Average number of iterations each LP runs for with and without caching.

	L	R
U	(1,1)	(3,0)
D	(0,0)	(2,1)

Figure 17: A Normal-form game demonstrating the value of commitment.

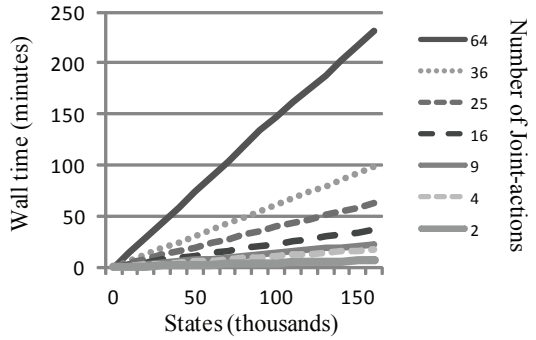

Figure 18: The running time of our algorithm is linear in the number of states and joint-actions.

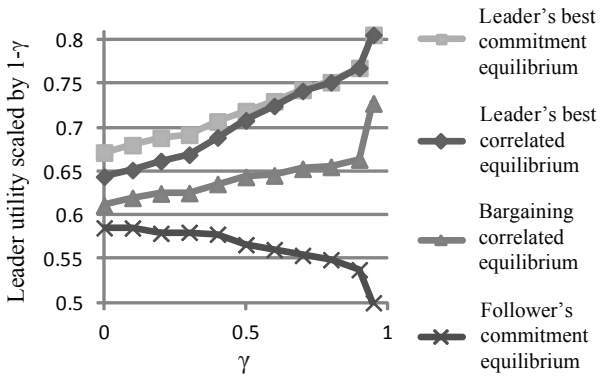

Figure 19: The average utility of being able to select a correlated or commitment equilibrium selfishly as opposed to the leader's Kalai-Smorodinsky bargaining solution or the follower's utility when the leader selected a commitment equilibrium.

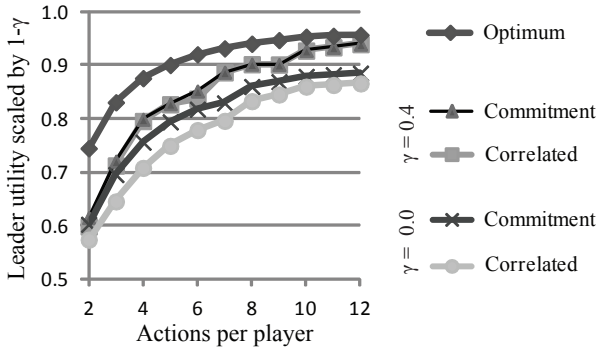

Figure 20: The best utility achievable by the leader using either correlated or commitment equilibria compared to the unrestricted optimum. Result are shown for $\gamma = 0.0$ and 0.4.

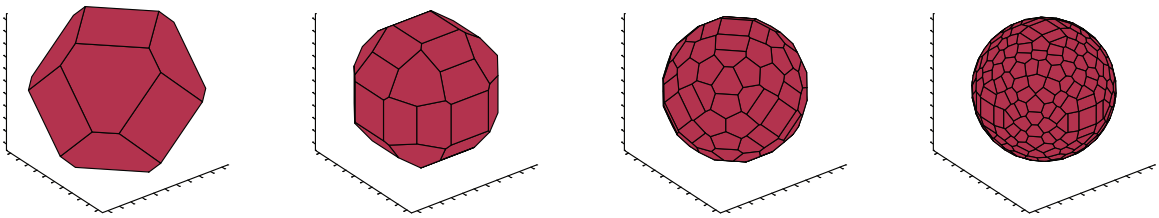

Figure 21: The sequence of unit polytopes (unit offsets) resulting from starting with a cube and adding a normal for each vertex of the previous polytope. The set of resulting halfspaces are fairly regular. These can be used for successive runs of increasingly accurate approximations.

CHAPTER IV

MARKOV GAMES OF INCOMPLETE INFORMATION

Multi-agent reinforcement learning (MARL) is challenging when the world is fully observable. When the world is only partially observable (as is the case for many problems), the problem is overwhelming. Previous research has modeled these scenarios as partially observable stochastic games (POSG) (Section 2.2.5). POSGs are extremely complicated and attempts to solve POSGs have all met large computational barriers. In this chapter we argue that POSGs are too general. One of the major reasons why POSGs are so hard to solve is that beliefs grow double-exponentially with the planning horizon. This makes reasoning over all beliefs intractable. We believe that the POSG model should be slightly weakened by assuming that there are a bounded number of beliefs for each agent. This assumption dramatically reduces the complexity of solving POSGs and leads to a novel model – Markov games of incomplete information (MaGII, pronounced "Ma-jai"). MaGIIs can be converted into belief-stochastic games, allowing them to be solved (much like POMDPs being converted into belief-MDPs). MaGIIs can approximate POSGs and exactly model many interesting problems while being (relatively) easier to solve.

4.1 Motivation

Agents living in a POSG view their world as lacking the Markov property; agents must utilize past observations to make optimal decisions. Even if an agent returns to the same underlying Markovian state of the world, the dynamics of the world may appear to change because other agent's may have different beliefs, and thus take different actions. Naively an agent can retain the history of all observations received. However, such retention is impossible in the long run. In order to understand how to efficiently solve games of incomplete information we need to understand how to compactly represent the useful information of an agent's history for decision making. We call an agent's current compact representation of history that agent's *belief*. The set of possible beliefs that an agent might hold is referred

to as the agent's *belief space*.

For POMDPs a sufficient (for optimal behavior) belief space is the set of probability distributions over possible states. This is not the case for POSGs. Not only do agents have to worry about which state they are in, but they must also worry about other player's beliefs. Worse, players must reason about the beliefs that players hold about each other's beliefs. These levels of meta-reasoning are called *nested beliefs*, and can potentially continue indefinitely (reasoning about beliefs of beliefs of beliefs etc...). This additional level of meta-reasoning makes the problem significantly more conceptually and computationally complex. An agent's observations may include private information about other player's beliefs, and in the worst case it might be impossible to losslessly reduce a player's knowledge beyond their full observation history. We refer to an agent's belief about the states-of-nature as that agent's *level-zero belief* (a probability distribution over states).

One way to compactly define the infinite nesting of beliefs is to utilize the notion of common-knowledge. If a fact is common-knowledge then all players not only know that fact, they know that other player's know it and that all such nested knowledge is known. Most game theoretic results assume prior common-knowledge of the game, as well as common-knowledge of each agent's policy (derived from the assumption of rationality). We also make these assumptions. Thus at the first time-step there is only one infinitely nested belief for each player defined by the common-knowledge of the model. Unfortunately, in a typical POSG, every time-step pushes the common-knowledge back a level. For example, after 3 time-steps only level-3 beliefs are commonly known (given the previous levels) and agents must reason about all possible level-0 through level-2 beliefs. In a single agent POMDP, the number of beliefs about the true world state grows exponentially with time. However, with multiple agents a belief should capture not just what has been seen, but also what **could** have been seen by other agents. This product of possible histories for each agent has doubly-exponential growth. This problem is known as "the curse of history" [70].

There have been previous approaches that attempt to operate directly on the infinitely nested belief structure [22] , but these must necessarily be approximations of unknown accuracy (if we stop at the n^{th} nested belief the $n^{th}+1$ could dramatically change the

outcome). These results have gotten reasonable empirical results in a few limited domains but it seems unlikely these methods will eventually generalize to general-sum games or scale with the complexity of the game. The POSG model typically assumes that the model itself along with the initial state distribution $b^{(0)}$ is common-knowledge at the beginning of the game. At this initial point, players only have level-zero beliefs. However, for every transition of the game the belief levels increase by one. Every possible history could potentially lead to a different set of nested-beliefs, and thus the belief-space grows combinatorially. Our solution, MaGIIs, arise from assuming that this explosion of beliefs does not occur.

4.2 MaGIIs without public observations

Markov games of incomplete information (MaGII) arise from making a single simple assumption: that the number of beliefs at any time-step is bounded for each agent. This means that at any point in time there are only a fixed finite set of possible beliefs for each agent, and that the maximum number of possible beliefs is known in advanced. Note that we are not assuming that the total number of beliefs over all time is bounded, only at any particular point in time. In particular, beliefs at every time-step, while bounded, may be distinct from all previous beliefs. While this assumption seems straightforward, the consequences are not.

First, recall that beliefs are sufficient statistics for history, meaning they capture all relevant information previously seen. This means that an agent does not need to remember any information or history beyond their current belief in order to make optimal decisions. In other words, an agent's belief is Markovian. If the number of beliefs is bounded, then because beliefs are Markovian we can transform a POSG into a new decision equivalent POSG where each agent's belief is included in the underlying (unobserved) state, and each observation corresponds to a belief. This new factored model is a MaGII and it will have a state-space only polynomially larger (with respect to the number of beliefs) than the original model. Such a transformation can be done for any POSG, but if there are an unbounded number of beliefs, the resulting state-space will also be unbounded.

There is a difference between a belief that is a true sufficient statistic (no worse than

knowing the full history) and a compressed-belief which may correspond to multiple histories. When we say that beliefs are bounded, we mean so if the first sense. We therefore know a priori the mapping from action-observation-histories to beliefs (because we know which histories are information-equivalent). Using induction, (with a base case of the first observation), if we assume the previous observation was the previous belief, we can replace each agents ⟩previous-belief, observation⟨ pair with a new observation that corresponds to the new belief. There will be no more distinct observations than possible beliefs, which is bounded. Because we know this transformation a priori, we can fold it into the transition function. Therefore, in a MaGII, beliefs and observations are synonymous.

We define a MaGII formally as the tuple $\langle N, A, O, \Omega, S, P, R, b^{(0)} \rangle$ where:

- N is the set of players. $n = |N|$
- $A = \prod_{i=1}^{n} A_i$ is the set of joint-actions
- $O = \prod_{i=1}^{n} O_i$ is the set of observations (*a.k.a.* types, corresponding to beliefs).
- Ω = is the set of states-of-nature (underlying states) where $\omega \in \Omega$ is the current state-of-nature.
- $S = \Omega \bigtimes O$ is the set of states. This is a factored state representation with $s = \langle \omega, \vec{o} \rangle = \langle \omega, o_1, \cdots, o_n \rangle$ where $\vec{o}$ is the joint-observation/belief.
- $P : S \times A \to \Delta(S)$ is the probability transition function.

  P must have a Markov property on observations (making them beliefs): For each agent, the current state is independent from previous observations given the previous state, joint-action, and current observation. Formally:

  $\forall\ s^t, s^{t+1}, \vec{a},$ players i :

$$P(s^{t+1}|s^t, \vec{a}, o_i^{t+1}, o_i^t) = P(s^{t+1}|s^t, \vec{a}, o_i^t)$$

- $R : S \times A \to \mathbb{R}^n$ is the joint-reward function
- $b^{(0)} \in \Delta(S)$ is the initial state distribution

There is no explicit observation function defined for a MaGII. Instead each agent always observes their corresponding belief factor. This belief captures any observations the agent may have made. A subtle but important aspect of the model is that the agent knows how many time-steps have transpired. This means that the same observation does not necessarily correspond to the same belief. This is very important because we allow an agent's policy to change over time, in other words a policy is a mapping from observation and time-step to action. The current time-step is common knowledge so does not increase the number of beliefs. Because policies can be time dependent, the exact belief that an observation maps to may change over time as well.

The key to MaGIIs is that the beliefs for each agent is explicitly included as a factor in the underlying states. This means that an agent needs to only reason about level-0 beliefs. Each state dictates not only the environment dynamics but also the other agent's dynamics (who's policies are based on beliefs which are included in the state). It might seem like the problem has been reduced to a POMDP, but there are still two major differences. 1) We must simultaneously compute policies for all agents (which may include equilibrium analysis for general-sum games). 2) Agent's still need to reason about possible beliefs of other agents (not just the true belief). In fact, beliefs in a MaGIIs include infinitely nested reasoning, but there are a bounded number of such beliefs. Both of these differences must be addressed. Nested reasoning is captured by accounting for a common-knowledge distribution over beliefs and treating the decision problem as a Bayesian game.

4.2.1 MaGIIs as a sequence of Bayesian games

Previously, in Section 2.2.6 we showed how an agent's decision problem in a POSG can be viewed as a sequence of Bayesian games. However there was a serious problem with effectively utilizing this transformation; the number of types in each Bayesian game is equal to the number of beliefs which typically grows exponentially. This is not a problem for MaGIIs. In fact, the definition of a MaGII makes it even easier than a POSG to define the succession of Bayesian games.

At each time-step an agent receives an observation (recall observations and beliefs are

synonymous in a MaGII) which is that agent's type in the current Bayesian game. Each agent then takes an action and receives a reward based on the joint-type. The only piece missing is a commonly known distribution across observations (types). But we can derive this using induction. At time-step 0 the initial distribution is commonly known. Also, the policies for each player are commonly known (again, due to rationality). The successor state is completely determined by the previous state and the policies for each agent. Policies in turn are mappings from belief (which is included in state) to distributions over actions. So, for time-step $t+1$ a common-knowledge distribution across states can be computed using the common-knowledge distribution from time-step t. In particular, with joint-policy π:

$$p^{t+1}(s') = \sum_{s} p^t(s) \sum_{\vec{a}} \pi(\vec{a}) P(s'|s, \vec{a}) \tag{17}$$

This common-knowledge probability distribution across underlying states specifies the complete infinite nesting of beliefs. For example, once an agent sees their belief, they know the posterior probability of other agents beliefs along with those agents posterior beliefs about other agents posterior beliefs, and so forth. In a MaGII, all observations are private so the only common-knowledge is the model itself (including starting state) and the policy each agent takes. Thus at each stage the common-knowledge distribution is independent of all observations seen and actions taken.

We now have all the pieces needed to define the Bayesian game at each time-step. However, there is one minor complication we need to address. Bayesian games don't explicitly deal with uncertainty over unobservable states, only uncertainty over player types. Game theory has typically handled this by including an extra player, called *nature* who has no choice of action but has the unobserved state as their type. This is why we call the underlying unobservable state the *state of nature*. We follow this convention here.

After $t+1$ time-steps the current Bayesian game, that is equivalent to the decision problem faced by players in the MaGII, can be computed as follows: N and A are the same as in the MaGII. The player types are equal to the private observations for each agent, including the unobserved states for nature ($\Theta^{(t)} = S$). The common knowledge distribution over types, $\tau^{(t)} = p^t$ can be computed recursively as given by Equation 17. Note that

because nature has only one action, she does not affect this distribution. The rewards are the same as in the MaGII ($R'(\theta, a) = R(s, a)$).

This is equivalent to the Bayesian game that results from a POSG when we simplify by utilize the Markov property on private observations.

4.2.2 Beliefs

MaGIIs are based on assuming there are a bounded number of beliefs. These beliefs constitute the state of a MaGII and is the space over which agents must reason. However, we have yet to describe what the actual beliefs are. Recall that the agent knows both their current observation and the number of time-steps since the beginning of the game. This information uniquely defines a belief, but is not the belief itself (it does not account for nested reasoning). In the previous section (4.2.1) we stated that the common-knowledge distribution across states defines the infinite nesting of beliefs. This common-knowledge distribution can be derived from the time-step dynamically using Equation 17. This distribution along with a players private observation constitutes a belief. We prove this fact:

Lemma 4.2.0.2. *In a MaGII, a common knowledge distribution τ across states combined with an agent's private observation is a sufficient statistic for history (the agent's belief).*

Proof. The proof is by induction on time-step. At time-step 0, agents trivially have the same common knowledge distribution $\tau = o^{(0)}$ as given by the model and combined with their (known) type represents the players' sum of useful information (their belief). Now assume that τ is common knowledge at time t, and that combined with player's private signals, $\vec{o}^{(t)}$ is a sufficient statistic for history. At time $t+1$, player's beliefs will include the new observations and players must reason about which observation every other player received (level-one beliefs).

Let us look from the perspective of agent i. At time-step $t+1$ after observation $o_i^{(t+1)}$ using a known joint-policy π^t agent i wishes to determine the probability of being in state s'.

$Pr[\vec{o}^{(t+1)}|s', s, o_i^{(t+1)}, \vec{a}^{(t)}] = \sum_{\vec{o}^{(t)}} P(s', \vec{o}^{(t+1)}|s, \vec{o}^{(t)}, \vec{a}^{(t)}) Pr[\vec{o}^{(t)}|o_i^{(t+1)}]$. However, note that because of the Markov property $Pr[\vec{o}^{(t)}|o_i^{(t+1)}] = Pr[\vec{o}^{(t)}] = \tau^{(t)}(\vec{o}^{(t)})$. Therefore

$\tau^{(t+1)}(\vec{o}^{(t+1)}) = \sum_{\vec{o}^{(t)}} \tau^{(t)}(\vec{o}^{(t)}) P(s', \vec{o}^{(t+1)} | s, \vec{o}^{(t)}, \vec{a}^{(t)})$ which is independent of any private information and is thus common knowledge and identical for all players. What we have neglected in the preceding equations is that any joint-observation inconsistent with a players observed type will have probably zero for that player, however this information is already captured by using the posterior of the joint-type distribution τ. Thus a player's belief will consist of a common knowledge type distribution and their private type. □

Lemma 4.2.0.2 shows that MGIIs permit an efficient belief representation. This fact is the key difference between POSGs and MaGIIs and allows us to efficiently solve MaGIIs.

4.2.3 Policies

Ultimately our goal is to figure out what actions agents should take. We are therefore interested in computing policies. We have shown that a lossless belief in a MaGII can take the form of a common-knowledge distribution combined with the current observation. Therefore, an optimal policy can be a mapping from common-knowledge distribution and observation to action.

It is more convenient to view this as a sequence of *decision rules* $\delta^t : O \to \Delta(A)$ that specify the actions (or distribution across actions) each agent should take at time t. This decision rule takes the form of a probability distribution across joint-actions for each joint-observation. Based on the solution concept targeted, this distribution will have additional constraints. In the case when we wish to compute Nash-equilibria or when the payoffs are linearly related (*e.g.,* zero-sum or common-payoff) this will mean that each agent's action probabilities must be independent (and such a decision rule can be broken down into n individual decision rules, 1 per agent). However, in Section 2.1.2 we argue that an agent normal form correlated equilibrium is a better solution concept for Bayesian games. This is much like how we argued that correlated equilibria are a better solution concept for stochastic games than Nash-equilibria (Section 2.1.1). We therefore leave decision rules as general probability distributions across joint-actions.

Each decision rule $\delta^t : O \to \Delta(A)$ corresponds to a solution of the current Bayesian stage game. A policy $\pi = (\delta^0, \delta^1, \cdots, \delta^h)$ is a (possibly infinite) sequence of such decision

rules. We use the notation $\pi^t(\vec{o}, \vec{a})$ to be the probability that agents will take joint action $\vec{a}$ after joint-observation $\vec{o}$. Again, not all decision rules are valid. They must adhere to the conditions of the solution concept being targeted (such as maintaining that no private information is communicated).

Given such a policy, we can compute the utility that it achieves.

$$V_\pi(b^t) = R(b^t, \pi) + \gamma V_\pi(b^{t+1})$$

where

$$b^{t+1}(s) = \sum_{s' \in S} b^t(s') \sum_{\vec{a} \in A} \pi^t(s'_{\vec{o}}, \vec{a}) P(s|s', \vec{a})$$

and

$$R(b^t, \pi) = \sum_{s \in S} b^t(s) \sum_{\vec{a} \in A} \pi^t(s_{\vec{o}}, \vec{a}) R(s, \vec{a})$$

is the expected immediate reward.

This utility computation follows the Bayesian stage game view. At each stage game the agents receive rewards $R(b^t, \pi)$ (this is a vector of rewards, one for each agent) and progress to a new Bayesian game with type distribution b^{t+1}. Notice that the progression of common-knowledge beliefs b is independent of the actual observations as these are all private. This means that the sequence of decision rules and common-knowledge beliefs is fixed ahead of time. Therefore the value of a policy starting at a distribution across states will be equal to the linear combination of the value of that policy for each each state separately (*i.e.,* $V_\pi(b^t) = \sum_{s \in S} b^t(s) V_\pi^t(s)$). This allows us to compute the value only at individual states. We can then simplify the value of a policy to:

$$V_\pi^t(s) = \sum_{\vec{a} \in A} \pi^t(s_{\vec{o}}, \vec{a}) \left[R(s, \vec{a}) + \gamma \sum_{s' \in S} P(s|s', \vec{a}) V_\pi^{t+1}(s') \right] \tag{18}$$

This trick - that the value of policy for a distribution across states is a linear combination of the that policy for each state separately - is the same property that allows a POMDPs value function to be represented as a set of hyperplanes supported by a policy that achieves that value across beliefs. We use this fact when solving common-payoff MaGIIs (Chapter 5). However, just because the policy can achieve a value does not mean the policy is acceptable;

the same policy may be an equilibrium for some common-knowledge beliefs but not for others.

4.2.4 Policies in equilibrium

A solution to a MaGII (or POSG) takes the form of a policy tree (*a.k.a.* strategy) for each agent. A policy tree is a function mapping every sequence of observations (including any implicit observation of one's own action) to a distribution across actions. Note that policy trees need not and can not be represented explicitly. Assuming agents are rational dictates that a solution must be an equilibrium. While we have described what an equilibrium of a Bayesian stage-game looks like, an equilibrium for the full MaGII does not directly follow. Given fixed policy trees for each player the world becomes a hidden Markov model and the utility for each player can be calculated (or at least sampled to arbitrary precision). We can therefore decide if it is beneficial (for any sequence of observations) for an agent to change one of their actions in the tree if all other actions stay the same. If no player can benefit by changing an action, then the policy-trees are in equilibrium.

A fundamental result from single agent RL tells us that when determining if a policy is optimal it is sufficient to only worry about changing a single action, and not all actions at once [86] (the policy must be defined and evaluated on all states, even those off-policy). This is true for both fully observable and partially observable domains. If we fix other agents' policies (as we do when determining if policies are in equilibrium) then the problem looks like a single agent problem. Therefore, to determine if a set of policies are in equilibrium we need only make sure that no agent wishes to change a single action in any stage-game.

This hinges on insuring that when agents choose which action to execute in a stage-game they consider not only their immediate reward but also the long term utility of the result of their action given the other player's policy-trees. The expected joint-utility for each joint-action and joint-type is known as the *continuation utility*, $\overrightarrow{cu_{\vec{o},\vec{a}}}$. Because the agents consider both their immediate reward and their continuation utility they are in essence playing an *augmented Bayesian stage game* with payoffs $R(s,\vec{o},\vec{a}) + \overrightarrow{cu_{\vec{o},\vec{a}}}$. If each of these augmented stage games are in equilibrium, and the continuation utility equals the

true utility of following the policy trees than the policy trees are an equilibrium of the full MaGII.

4.2.5 The relationship between POSGs and MaGIIs

A MaGII is defined with factored state and without an explicit observation function. This makes them easy to view as a sequence of Bayesian games. However, despite these differences the important distinction between MaGIIs and POSGs lies in the Markov property enforced on observations. To demonstrate this point we compare MaGIIs without the Markov property to POSGs and show that they are equivalent models.

Lemma 4.2.0.3. *MaGIIs without the Markov property on signals have equivalent representational power as POSGs.*

Proof. We show that MaGIIs without the Markov property and POSGs can be converted into each other using polynomially similar space. To convert a MaGII into a POSG all we need to do is to remove the factored representation by enumerating states and creating an observation function that gives each agent their respective observation (as included in the original MaGII's state). The remaining variables are trivially similar.

To convert a POSG into MaGII we first create the factored state representation which includes the state of the POSG along with the observations that each agent receives. There is no observation function. The transition function P falls out naturally but becomes quite large in the constructed MaGII requiring $O(|S||A||O|^n)$ entries while the POSG only requires $O(|S||A||O|)$ entries (assuming the number of successor states is small). The same is true for the reward function which now requires $O(|S||O|^n)$ entries. Players in this new MaGII will receive the same observations and rewards as in the original POSG. □

The POSG representation is more efficient because it assumes observations are conditionally independent of previous observations given the true-state, while the MaGII representation of the POSG transitions based on all the players' observations, not just the underlying state. The transition and reward functions could be defined in terms of just ω (the state-of-nature) in which case this inefficiency would not exist. We have defined MaGIIs to explicitly look like a sequence of Bayesian games. In Bayesian games, not only

the state-of-nature but also the joint-type determines rewards. We follow this precedence and allow joint-types to also changes the dynamics.

While there are problems that can't be exactly modeled by MaGIIs (Figure 24), MaGIIs can exactly represent some problems and do a surprisingly efficient job of approximating many other POSGs. For example, any finite horizon POSG can be represented exactly as a MaGII where beliefs are all possible histories. While the number of beliefs will be exponential in the horizon, there will still be a finite number of such beliefs and thus representable as a MaGII. Even for infinite horizon problems MaGIIs can approximate the problem by making observations include a finite history of relevant observations which could include much if not all of the history information an agent needs. This can be done more efficiently by choosing a good compact statistic for history. Surprisingly, in Section 5.3.4 we show how we can create a MaGIIs which allows each agent to choose for themselves an optimal compact statistic for history and thus achieve exceptional results using very few beliefs at each time-step. This is probably the most compelling evidence that MaGIIs can act as very good approximations for POSGs.

4.3 MaGIIs with public observations (PP-MaGIIs)

The number of distinct beliefs grow in a POSG because agent's must reason about other agent's private information. However, information that is common-knowledge does not increase the number of beliefs it only transforms the nature of those beliefs. This allows us to add observations which are common-knowledge (*a.k.a.* public observations) to the MaGII model. This improves the expressibility of MaGIIs without hurting the tractability of the model. While public observations ultimately improve efficiency and the power of the model, they do make algorithms and their analysis significantly more complicated. Also, not all problems can take advantage of public observations. This is why we do not include public observations in the original definition of MaGIIs.

We call MaGIIs with public observations PP-MaGIIs (public/private-MaGIIs). The definition of PP-MaGIIs is very similar to vanilla MaGIIs, except with the addition of a public signal. Previously, when analyzing MaGIIs and viewing them as a sequence of Bayesian

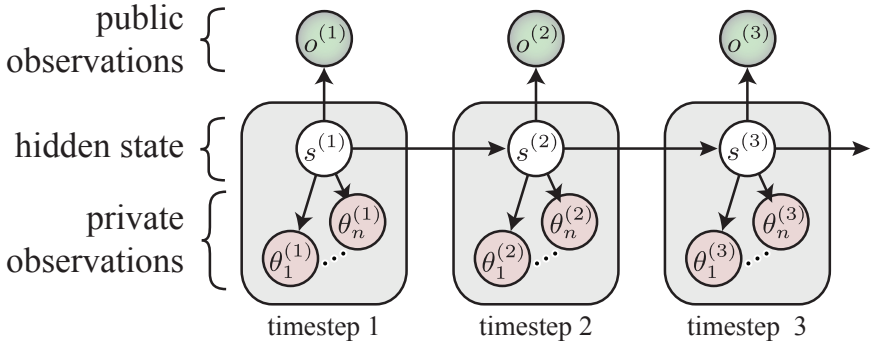

Figure 22: A PP-MaGII visualized as a Bayesian network.

games we were able to assume a single common-knowledge belief trajectory (there was only one possible). This is no long the case for PP-MaGIIs which is the main complicating factor of their analysis.

Definition 4.3.1. *A PP-MaGII consists of a tuple* $\langle N, A, S, P, R, O, s^{(0)}, o^{(0)} \rangle$ *where:*

- N *is the set of players.* $n = |N|$.
- $A = \Pi_{i=1}^{n} A_i$ *is the set of actions.*
- O *is the set of public signals, where* $\vec{o}^{\,(t)}$ *is the history of public signals at time* t.
- $\Theta = \Pi_{i=1}^{n} \Theta_i$ *is the set of private signals. (each signal* $\theta_i \in \Theta_i$ *corresponds to a type for player* i*)*
- Ω = *is the set of states-of-nature (unobserved underlying states) where* $\omega \in \Omega$ *is the current state-of-nature.*
- $S = \Omega \bigtimes \Theta$ *is the set of states. This is a factored state representation with* $s = \langle \omega, o_1, \cdots, o_n \rangle$.
- $P : S \times A \to \Delta(S \times O)$ *is the probability transition function with* $P(s', o'|s, \vec{a})$ *being the probability of ending up in state* s' *with public signal* o' *after joint-action* $\vec{a}$ *in state* s.

  P *must have the following Markov property:*

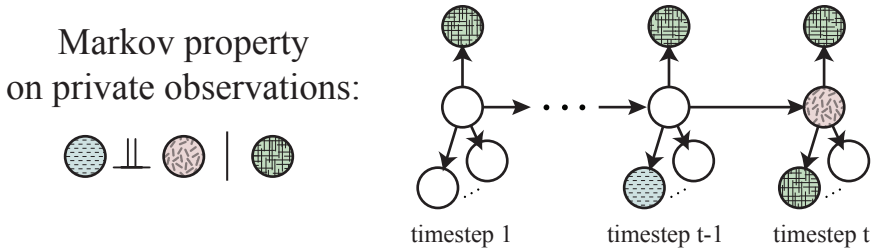

Figure 23: A visual depiction of the Markov property on private observations which must be satisfied in a PP-MaGII.

$\forall$ *public histories* $\vec{o}^{(t)}$, *players* i *and types* $\theta_i^{(t)}$

$$(\theta_i^{(t-1)} \perp\!\!\!\perp s^{(t)}) \mid \vec{o}^{(t)},\ \theta_i^{(t)}$$

i.e., the state of the world is independent of a player's previous signal given what is common knowledge and the player's new signal (Figure 23).

- $R : S \times A \to \mathbb{R}^n$ *is the reward function.*
- $s^{(0)} \in S$ *is the initial state with known types* $\theta^{(0)} \in \Theta$

While there are still problems that can't be efficiently modeled even by PP-MaGIIs (Figure 24), PP-MaGIIs can represent a larger set of problems than MaGIIs. Unlike MaGIIs, PP-MaGIIs can represent all POMDPs by making all observations public (with only one agent, the distinction between public and private is meaningless). Like MaGIIs, PP-MaGIIs can approximate any POSG by folding the history of relevant observations into each signal. Also there are many interesting problems which can be modeled exactly by PP-MaGIIs. For example, if players have initial private information followed by purely public observations (*e.g.,* poker, or opponent modeling) then observations do not contain information about the other player's beliefs. In fact, any POSG that always has a bounded number of distinct nested beliefs can be defined exactly as a PP-MaGII. Thus the PP-MaGII model is equivalent to the class of POSGs with a bounded number of beliefs. We prove this fact, but first need the following definition of what it means to have a bounded number of distinct nested beliefs:

Definition 4.3.2. *The set of* possible *beliefs is equal to the smallest set of joint-histories that includes the actual joint history along with any joint-history* $h1$ *for which there exists*

an agent i with observations $h\vec{2}_i$ from another history $h2$ in the set such that there is a non-zero probability of $h1$ occurring given $h\vec{2}_i$.

This definition is constructive in that it describes the set recursively: first take the actual joint-history. Then iterate over every joint-history $h1$ in the set and every player i, and add every history occurring with non-zero probability given what player i observes in history $h1$. This process closely follows the infinitely nested belief structure. Initially the set captures all possible level-0 beliefs. After the first iteration all possible histories involved in level-1 beliefs are added. Each subsequent iteration adds new histories involved in level-n beliefs which were not involved in level-(n-1) beliefs. This process continues until no additional histories are added.

Definition 4.3.3. *The set of* distinct *beliefs is equal to the largest subset of possible beliefs such that there does not exist two joint-histories such that the probability of every other joint-history occurring is the same for both joint-histories from all player's perspectives (i.e., $\nexists$ histories $h1, h2$ s.t. $\forall$ players i and joint-histories $h3 \neq h1$ or $h2$, $h3|h1_i = h3|h2_i$)*

The set of distinct beliefs eliminates histories that lead to the same nested beliefs. These definitions allow us to make the following claim.

Lemma 4.3.0.4. *The set of possible beliefs is common knowledge.*

Proof. It follows from our recursive definition of the possible belief set that in order for a history to not be included in the set of distinct beliefs, all players must believe the probability of that history occurring is zero given their observations. Furthermore, they must also believe that all other players believe the history has zero probability of occurring. Likewise they must believe that all players believe that all players believe the history has zero probability of occurring, and so forth. If at any level a player believes the history could have occurred then it would have been added to our set in the constructive process. This infinite sequence of believing that the history has zero probability of occurring is exactly the definition of common knowledge. If the histories *not* in the set of possible beliefs is common knowledge then the complement set will also be common knowledge. □

Less formally, we could have proved lemma 4.3.0.4 by noticing that any agent can construct the set of possible beliefs by initializing it with all possible joint-histories that includes what that agent observed. This will include the true history along with all other possible histories for other agents because each other agent's initial belief will be added after the first iteration of the constructive process, given the true history.

Corollary 4.3.0.1. *The set of distinct beliefs is common knowledge.*

Proof. This follows directly from the above theorem because the set of distinct beliefs is independent from a player's information given the set of possible beliefs. This independency exists because the definition of "distinct beliefs" is from all player's perspectives.

□

This corollary allows us to prove that the class of problems PP-MaGIIs can represent is exactly the class of POSG problems with bounded beliefs.

Theorem 4.3.1. *A POSG where the set of distinct beliefs is bounded in size can be expressed as a decision theoretically equivalent PP-MaGII.*

Proof. Because the set of distinct beliefs is always common knowledge, this information can be conveyed using only the public observations. Because this set is bounded in size we can have the set of private observations enumerate the possible distinct private observations for each player in the histories included in the distinct belief set. The number of such observations can never be larger that the set of all histories and thus will also be bounded.

□

The Markov restriction on private observations can be viewed as requiring that private observations only contain information about the current state of the world, and not private information about other players' future mental states. This is not to say that players don't receive persistent information about other player's beliefs – they do – but only if such information is contained in either the public signal or explicitly persists in the current private signal. Thus, all information about other player's current beliefs are either part of the current private signal or are common knowledge. Players do not need to reason about

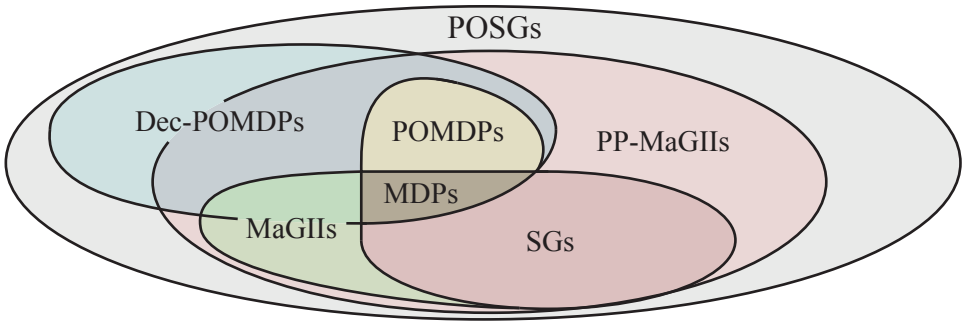

Figure 24: The relationship between various sequential decision models. POSGs generalize all models, while all models generalize MDPs. Vanilla MaGIIs do not generalize POMDPs because any POMDP with bounded belief can be converted to a finite state MDP. PP-MaGIIs generalize stochastic games and POMDPs (all observations are public for these models) but can't efficiently represent continuous private beliefs and thus don't generalize decentralized models or POSGs.

possible histories of past private observations, they only need to reason about possible current private signals. The entire history of public observations may still be relevant, but because this history is commonly known it does not add to the *number* of possible histories (all possible histories include the same public observations). Furthermore, while the public history may be infinite, its possible to compactly represent this information.

Lemma 4.3.1.1. *In a PP-MaGII, a common knowledge distribution τ across states combined with an agent's private observation is a sufficient statistic for history (the agent's belief).*

Proof. While this lemma is analogous to Lemma 4.2.0.2 about vanilla MaGII, the inclusion of public observations makes the proof more involved. Without loss of generality let us assume that the hidden state includes the private observations (the boxed region in Figure 22). Suppose we add an additional agent to our PP-MaGII that never receives any private information and has no choice of actions. This new agent's belief will exactly correspond to the sum of common knowledge. Our Markov property on private information insures that past private information does not affect current state transitions, meaning that policies will be a function of public observations and the current state only. Thus, our hidden state is a true Markov state. From the new agent's perspective, the PP-MaGII is a hidden Markov model with a sequence of hidden states and public observations. A sufficient statistic for history in

a hidden Markov model is a distribution over states [6]. Because our states include private observations, a compact sufficient statistic for the history of public information comes in the form of a probability distribution across states and private observations. Our Markov property on private information insures that an agent's belief only consist of the sum of public information, which we have shown to be a common knowledge distribution across possible joint-types, along with the current privately observed type. □

Lemma 4.3.1.1 shows that PP-MaGIIs permit an efficient belief representation. This fact is the key difference between POSGs and PP-MaGIIs and allows us to convert PP-MaGIIs into stochastic games of complete information – a model which has both an intractable exact solution [54] and a tractable solution with bounded error [52].

4.3.1 Ensuring the Markov Property

While PP-MaGII's require the Markov property on private observations, it is not immediately obvious how to define models that maintain this property. Instead of the canonical representation of a PP-MaGII given above (Figure 22), we can instead define a more complex Bayes net that guarantees that the private-observation Markov property will hold. Two examples of such models are given in Figure 25 and Figure 26. Once we have a Bayes net we can use a Bayesian inference algorithm such as Bayes-ball [79] to determine conditional independencies and prove that the private-observation Markov property is satisfied. Although Bayes nets provide a means of modeling many problems, there may exist PP-MaGIIs for which no Bayesian net can capture.

4.3.2 Converting a PP-MaGII into a belief-stochastic game

The standard approach for solving a POMDP is to convert the problem into a continuous belief-space MDP. We follow this same process and convert a PP-MaGII into a continuous belief-space stochastic game. We show that a solution to the resulting stochastic game will be a solution to the original PP-MaGII. While no continuous state stochastic game algorithms exist, this continuous belief-space stochastic game can then be discretized and solved using existing stochastic game algorithms such as QPACE [52]. While we don't

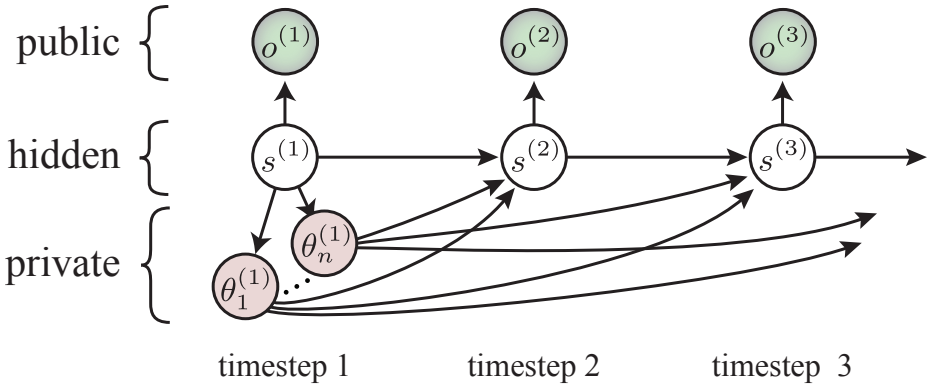

Figure 25: A Bayes net depicting a PP-MaGII with initial private information, followed only by public observations. It can readily be observed that this alternate representation for a PP-MaGII guarantees that the private-observation Markov property is satisfied.

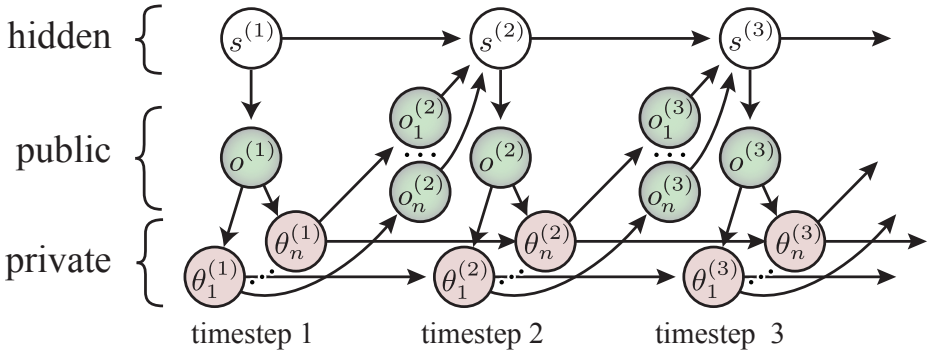

Figure 26: A Bayes net depicting an alternate representation for a PP-MaGII. While this network is complicated the private-observation Markov property can be proven to hold.

provide an exact algorithm to solve the resulting continuous belief-space stochastic game, we believe algorithms can be developed in a similar way to current POMDP algorithms that take advantage of the belief structure to operate in the continuous state space.

Given a PP-MaGII we define the following continuous belief-space stochastic game:

- $N' = \bigcup_{i \in N} O_i$ one player for each player type. Recall that out]r solution concept is agent-normal-form which operates as if each agent type acts independently.
- $A' = \Pi_{i=1}^{n} \Delta(A_i)^{|O_i|}$ each player type submits a probability distribution across actions in the PP-MaGII as their action in the stochastic game. Unlike in fully observable domains, in games of incomplete information the distribution of actions not taken by players matters, as the relative ratio of actions across player types informs other players as to the actual type of the player. This is true even if action are unobservable.

- $S' = S \times \Delta(\Theta)$ states consist of the original state along with a common knowledge probability distribution, τ, across types Θ.

- $P' = S' \times A' \to \Delta(S')$:

$$P'(s', \tau' | s, \tau, \vec{A}) = \sum_{\vec{a}} Pr[\vec{a}|\vec{A}] \sum_{\theta} \tau(\theta) \sum_{\theta'} P(s', \theta' | s, \vec{a}_\theta) Pr[\tau' | s, s', \tau, \vec{a}, \vec{A}]$$

  where: $Pr[\vec{a}|\vec{A}]$ is the probability of joint-action $\vec{a}$ given actions distributions $\vec{A}$, $\vec{a}_\theta \subseteq \vec{a}$ is the vector of actions taken just by agent-types θ, and $\tau(\theta)$ is the current probability mass assigned to joint-types θ.

  τ transitions deterministically given the public observations $s, s', \tau, \vec{a}, \vec{A}$. $\vec{A}$ is public because agents are rational, and thus able to predict the behavior of other agents.

  $Pr[\tau' | s, s', \tau, \vec{a}, \vec{A}] = 1$ if: $\forall \theta', Pr[\tau'_{\theta'} | s, s', \tau, \vec{a}, \vec{A}]$

$$\begin{aligned} &= \sum_{\theta 1} Pr[\theta 1 | \vec{A}, \vec{a}] Pr[\theta' | \theta 1] \\ &= \sum_{\theta 1} \sum_{\theta 2} \frac{Pr[\vec{a}_\theta 1 | \vec{A}]}{Pr[\vec{a}_{\theta 1} = \vec{a}_{\theta 2} | \vec{A}]} Pr[\theta' | \theta 1] \\ &= \sum_{\theta 1} \sum_{\theta 2} \frac{Pr[\vec{a}_\theta 1 | \vec{A}]}{Pr[\vec{a}_{\theta 1} = \vec{a}_{\theta 2} | \vec{A}]} P(s', \theta' | s, \theta 1, \vec{a}_\theta) \end{aligned}$$

  $Pr[\tau' | s, s', \tau, \vec{a}, \vec{A}] = 0$ Otherwise.

- $R' = R'(s, \tau, \vec{A})_{\theta_i \in N'} =$

$$\sum_{\theta 1 \in \Delta\Theta s.t. \theta 1_i = \theta_i} \tau(\theta 1) \sum_{\vec{a} \in \vec{A}} Pr[\vec{a}|\vec{A}] R(s, \theta, \vec{a}_{\theta 1})$$

- $s'^{(0)} = \{s^{(0)}, \tau'^{(0)}\}$ where $\tau'^{(0)}(\theta^{(0)}) = 1$ and 0 elsewhere.

We now argue that an equilibrium of this stochastic game (lets call it $\mathbb{E}$) maps to an equilibrium of the original PP-MaGII. $\mathbb{E}$ takes the form of a joint-policy mapping states to joint-action distributions. Every observation trace of the original PP-MaGII corresponds to a state in the stochastic game, and thus maps to a joint-action distribution of $\mathbb{E}$. In particular the transition function P' of the stochastic game insures that the common-knowledge

probability distribution across types $\tau \in \Delta(\Theta)$ of the successor states matches Equation 4 when $\tau^{(t)}(\theta^{(t)})$ is given by the current state's common knowledge distribution. In other words, for a PP-MaGII the sequence of Bayesian games that a POSG follows as laid out by Emery-Montemerlo et.al [23] corresponds to the stochastic game's states. The reward in the stochastic game for each player type is defined to be the expected reward given the probability distribution across other player types. This is the same expected immediate reward in equations 1 and in the corresponding Bayesian game. Because the sequence of Bayesian games is the same for both the PP-MaGII and the stochastic game, the continuation utility of each augmented stage-game in the stochastic game for each player-type will be the same as the continuation utilities in the PP-MaGII. This mapping allows us to compare the equilibrium constraints of the stochastic game with the PP-MaGII Bayesian stage games. The reward R' given in the stochastic game is equal to the expected reward in both sides of the Bayesian rationality constraints (Equation 1), thus the fully observable rationality constraints $\sum_{\vec{a}|a_i=a_1} R'(\vec{a})Q_{\vec{a}} \geq \sum_{\vec{a}|a_i=a_2} R'(\vec{a})Q_{\vec{a}}$ equals Equation 1). Thus policies in equilibrium for the stochastic game will be in equilibrium for the PP-MaGII. Finally, the stochastic game is designed so that all player-types are playing independently and thus can not collaborate with each other (as per an agent-normal form equilibrium). This also insures that the true type of players can not be disclosed, since no true type exists in the stochastic game. This satisfies the non-disclosure constraints (Equation 2). Therefore each state corresponds to an augmented stage-game of the original PP-MaGII and an equilibrium of the stochastic game $\mathbb{E}$ satisfies the equilibrium inequalities for all possible Bayesian-games of the PP-MaGII. Therefore a correlated equilibrium of the stochastic game corresponds to an agent-normal form correlated equilibrium of the original PP-MaGII.

4.3.3 Imposing the Markov property on beliefs

One of the primary purposes for developing MaGIIs and PP-MaGIIs is that we believe that they can be used to approximate POSGs. We accomplish this approximation by using *belief-compression* whereby multiple different beliefs are combined into a single belief. We expect agents with the same belief to have no incentive to distinguish between themselves.

For example, if two different observation-histories ($\vec{h}^1$ and $\vec{h}^2$) are compressed into the same belief θ, then we should expect an agent with belief θ to think and act identically to an agent despite which history ($\vec{h}^1$ or $\vec{h}^2$) the agent actually observed. If a rational agent has an incentive to remember the difference between $\vec{h}^1$ and $\vec{h}^2$ then θ is not a true belief. If we use belief-compression to approximate POSGs then we will lose information in the compression and agents will want to remember past beliefs, breaking the Markov property on beliefs. This may seem like a problem, but it is not.

In a MaGII, even if agents can gain information by remembering past beliefs (the MaGII lacks the Markov property on beliefs), there exists a true MaGII (with the Markov property) that is equivalent to the original MaGII when assuming that the Markov property holds (agent behavior is only dictated by their current belief). In other words, we can assume that while agents can gain by remembering past beliefs, they simply don't act on that knowledge. Using this assumption does not violate our assumption of rationality or the spirit of MaGIIs. It only means that agents are acting rationally (optimally) with regards to a different MaGII.

By forcibly imposing the Markov property on a MaGII we are in essence guaranteeing that current Bayesian game's factored state is drawn from the common knowledge type-distribution. Because the type-distribution itself is commonly-known the only additional information an agent gets is its current observation (their type). Therefore, to prevent past beliefs from being informative we only need to make sure the game samples factored-states independently at each time-step. To accomplish this, we can define a new MaGII where the state-of-nature *is* the common-knowledge distribution. This will allow types to depend only on the current state-of-nature and be redrawn at each time-step. This new MaGII will have a continuous state-space, but the continuous state-of-nature will be fully observable (as it is commonly-known) and thus will not increase the complexity of the created belief-SG. In fact, the belief-SG of these two games will be identical. This allows us to successfully impose the Markov property and make both the assumption that agents are rational and that approximated beliefs will be acted upon as though they were true beliefs.

4.3.4 Extended Example:

The single agent tiger problem [14] has been used often to test and explain POMDP algorithms. In the problem there are two doors. One holds a tiger (-100 reward), the other some money (+10 reward) with even probability of which door holds which. A contestant must choose the correct door, but can wait and listen for growling noises (which are reported with 85% accuracy). There have been a number of multi-agent versions of this game [22, 23, 92]. However, we choose to construct yet another version because the previous multi-agent versions mostly hinge on continuous belief spaces which PP-MaGIIs can't model exactly. Also, our version has two agents with rewards that are not linearly related (*i.e.,* the rewards are not shared or zero-sum).

In our multi-agent version, the tiger is an additional agent which is actively trying to prevent the human from opening a door. The dynamics of the world are similar to the single agent version except the two rooms have only a low wall that permits the tiger to attempt to jump over and reach the other room. Jumping the wall is tough for the tiger and only succeeds 10% of the time. Also, the act of jumping causes extra noise and thus regardless of outcome the jump causes the contestant to correctly hear the location of the tiger's growl. However the contestant does not observe the tiger's action and thus does not know if this observation was due to a jump or not. Importantly, which observation the contestant receives (the door from which the growl was heard) is also known to the tiger. The tiger is a second agent which wishes to prevent the human from opening either door, receiving a reward of -1 whenever the game resets. Note that this game is not zero sum. The tiger does not wish to eat the human, only prevent her from opening a door. The tiger must decide on when to risk a jump based on the contestant's confidence of the tiger's location. Figure 27 depicts this game represented as a PP-MaGII.

Because the tiger's private observation along with the public observation represents the true state, the tiger's previous private observations will be independent of the current state given the tiger's current observation. Thus the private-information Markov property is satisfied. Figure 28 depicts this game represented as a sequence of Bayesian games.

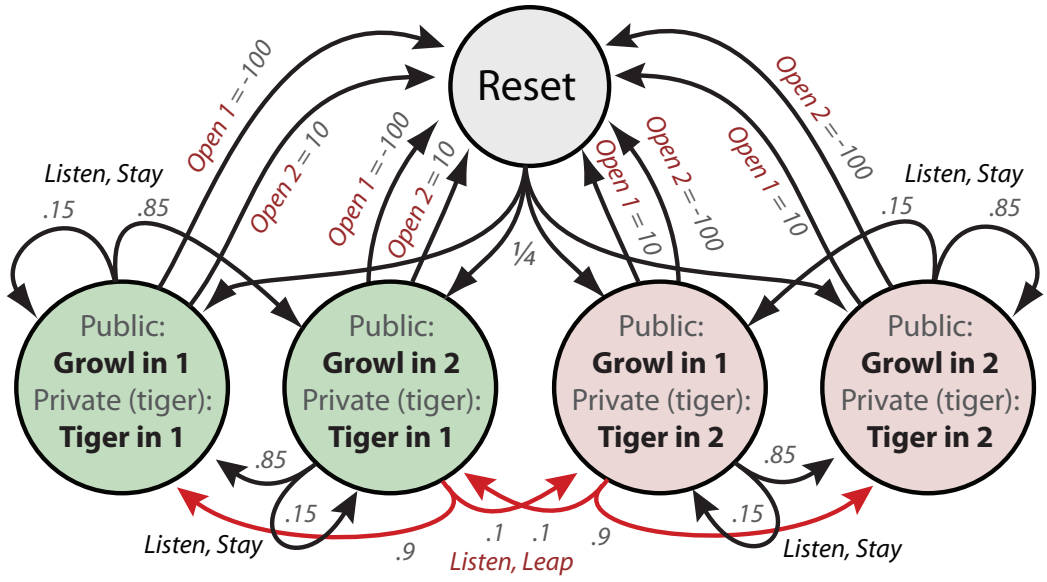

Figure 27: A PP- representing the human vs. tiger problem. There are four states representing the set of possible observations (both private and public). There are no unobserved states of nature. The outcome and rewards of joint-actions are given by the arrows for each state.

The game has two non-trivial states representing the two common knowledge observations of a growl behind door one, and a growl behind door two. We could have easily chosen to have only one state, but then we would have had to include the contestants observation with the tiger's type. It is easier (both conceptually and computationally) to have two simpler Bayesian games than one complex game. In each of the state, the tiger has two possible types corresponding to which door it is behind, while the contestant has only one type (as the contestant has no private knowledge). A key feature of this representation is the fact that while the contestant's belief about the probability of which door holds the tiger may lie in a continuous range, the particular distribution at any time step is common knowledge, and thus agents don't need to reason about nested beliefs.

4.3.5 From private to public observations

A PP-MaGII has explicit public observations, but just because an observation is private (as all observations are in normal MaGIIs) doesn't mean it isn't also common-knowledge. For example, a game might be defined to always give each agent the same private signal. Or more commonly, an agent could receive a private "reset" signal that is only given if its

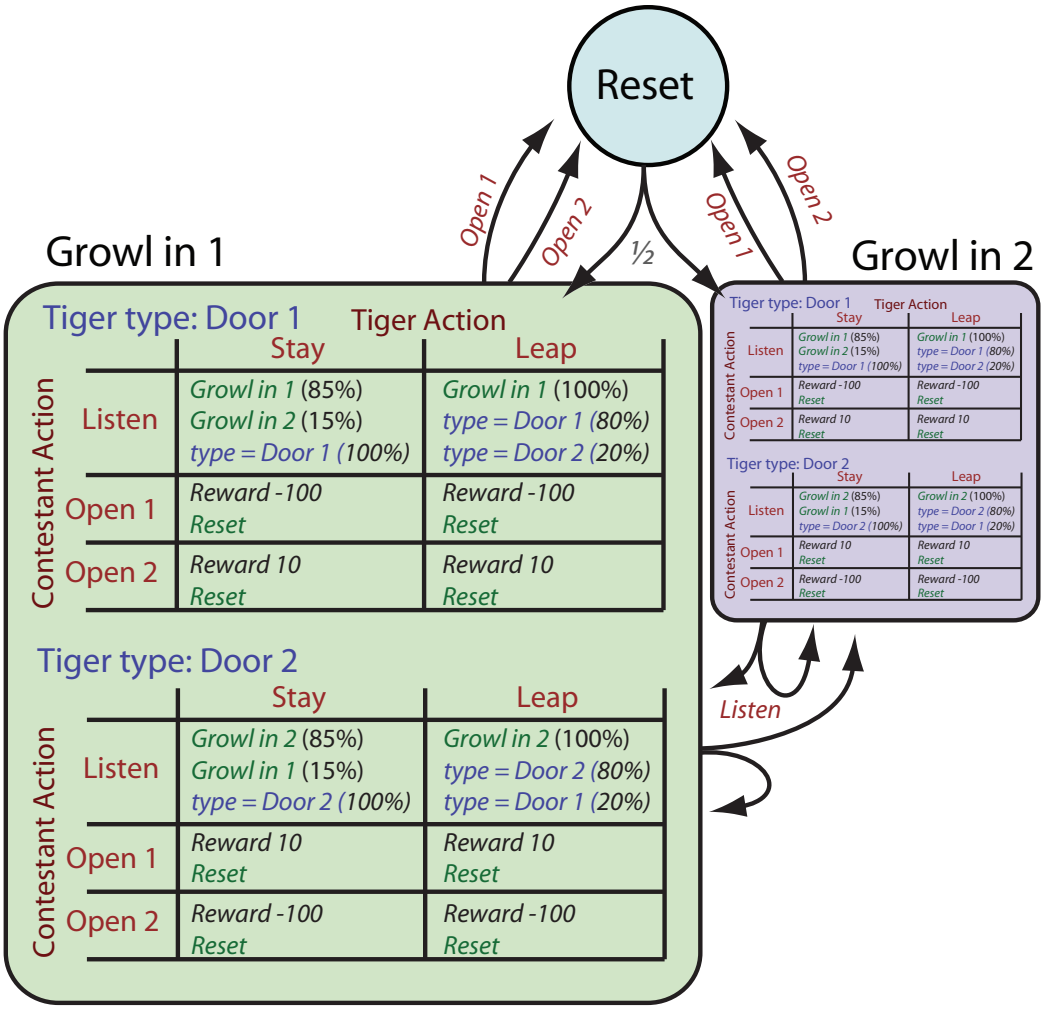

Figure 28: The Bayesian game representation of the human vs. tiger problem. There are two states which are identical Bayesian games.

given to all agents. Agents receiving this signal could reason the reset is common knowledge. Such a reset signal is particularly important to identify as public as it reduces the number of beliefs to one (which can't be done unless the signal is public). A number of researchers have incorporated reset signals into their belief compression schemes, however there has been no formal treatment of public signals in general.

Information is common-knowledge (*a.k.a.* public) if all players know it and know all player know it ad infinitum. In other words, there is zero probability that the world is in a state (including beliefs) where any agent doesn't know the information. What does this mean in terms of our PP-MaGII beliefs? First, let us be clear what information is. The sum of a player's information is their posterior probability over states (after knowing

their private information). This sum of information is equivalent to their type. Public knowledge is therefore the common-knowledge that certain types have zero probability. In order for a type θ_i to be commonly known to have zero probability, then from every player's perspective (after learning their type) all infinitely nested beliefs of each player must assign θ_i zero probability (as per the definition of common-knowledge). This reasoning should look familiar, it is the same definition that we used for possible beliefs (definition 4.3.2).

Identifying private knowledge that is actually public is the search for beliefs that never have positive probability at any level of the nested belief structure for any player. Because an agent reasons about other agent's reasoning as part of their nested beliefs, It is sufficient to only perform this search for a single agent (each agent can perform this search for themselves as end up with the same results). The recursive definition of possible beliefs (definition 4.3.2) gives us an algorithmic way to identify private information that is actually public. Our goal is to find the set of possible beliefs B. We start with the agent's current belief ($B = \{\theta_i\}$). We then find all states that have positive probabilities of occurring given the beliefs in B ($S = \{s : \forall \theta \in B, p(s|\theta) > 0\}$). We then add all beliefs that result from the states in S to our belief set ($B = \{\theta : \forall s \in S, \forall i \in N, S_{\theta_i}\}$). This process iterates until no more beliefs are added to B. This is not the most efficient way to perform this search. However, we can actually view the problem as a connected component graph search.

This above algorithm is equivalent to a connected component search on the following undirected graph $G = \langle V, E \rangle$ for a given common knowledge belief distribution τ for PP-MaGII $\langle N, A, S, P, R, O, s^{(0)}, o^{(0)} \rangle$ where each state is a vertex and there exists an edge between states if an agent in one state can believe they are actually in the other state (*i.e.,* both states give an agent the same private observation and have positive probability in τ):

$$V = \{s \in S : \tau(s) > 0\} \tag{19}$$

$$E = \{\langle s', s'' \rangle \in V^2 : \exists i \in N | s'_{\theta_i} = s''_{\theta_i}\} \tag{20}$$

This graph is undirected because the definition of an edge is symmetric; an agent can only be unsure about which state they are in (positive probability on two states) if they are

unable to distinguish between those states (they both give the same private observation).

Each connected component of this graph represented a distinct public observation. This information is public (commonly known) because every possible nested belief only has positive probability on states within a single connected component. Therefore, once an agent knows which component the current state is in, that agent also knows that all other agents know that the current state is in that component as well. This knowledge is recursive, and the knowledge of the current state's component is public.

From a single agent's perspective, an agent's level-0 belief represents a probability distribution over vertices (states). All of these vertices must be in the same connected component (we defined the graph so that each of these states will have an edge between them). Therefore, the agent knows that the true state lies within a given connected component. The agent also knows that each agent's level-0 belief must contain the current state, so they must also know the true belief lies in that same component. This argument can be used inductively to show that all agents knows know that all other agents know, ad infinitum, which component the current state is in.

4.3.6 The advantage of public observations

Realizing that a piece of private information is public does not add any additional information. However, it does reduce the current set of possible beliefs - once agent's know which connected component of states they are in, they only need to reason over states in that component. This reduction can happen at every time-step, and can thus lead to an exponentially smaller belief-space. For example, consider a multi-agent tiger problem (Section 5.4.1.1) where the each agent receives the same growl signal. An agent can reason, that once they here which door the growl sounds that the growl is public. Agents therefore don't need to reason about possible observation histories, they know there is exactly one possible joint-observation history. This then makes only one belief possible for each agent, compressing what otherwise would have been 2^t possible beliefs down to just one.

Public observations are very common in multi-agent problems, however most research to date does not utilize this information. In fact, we know of know previous formal treatment

of pubic vs private observations. By identifying and reasoning about public observations, we can potentially solve larger problems with more complicated belief spaces.

4.4 Conclusion

We presented a new model for multi-agent sequential decision problems with hidden state: Markov games of incomplete information (MaGIIs). MaGIIs enforce a Markov property on private information such that the only useful information which is not common knowledge is a player's most recent private signal (a player's type). We proved that the class of problems representable as MaGIIs is exactly the class of problems representable as POSGs with a bounded number of distinct beliefs. This in itself has interesting ramifications for determining the class of problems we should hope to solve efficiently.

In the next two chapters we address how to efficiently solve MaGIIs. This work builds off of transforming a MaGII into a sequence of Bayesian games where all common knowledge in the MaGII is compactly represented as a probability distribution across types (distinct beliefs). This feature permits MaGIIs to be converted into a continuous but bounded belief-space stochastic game, much as how POMDPs can be converted into continuous belief-space MDPs.

CHAPTER V

VALUE ITERATION FOR COMMON-PAYOFF MAGIIS

Decentralized partially observable Markov decision processes (DecPOMDPs) are a popular model for cooperative multi-agent decision problems. However, they are very difficult to solve (NEXP-complete [68]). One of the primary causes of this complexity is that DecPOMDPs, unlike single agent POMDPs, suffer from a doubly-exponential curse of history [70] – not only do agents have to reason about the observations they see, but also about the *possible* observations of other agents. This causes agents to view their world as lacking the Markov property because even if an agent returns to the same underlying Markovian state of the world, the dynamics of the world may appear to change due to other agent's holding different beliefs and taking different actions then the last time the state was visited. For POMDPs a sufficient (for optimal behavior) belief space is the set of probability distributions over possible states. This is not the case for DecPOMDPs where an agent must reason about the beliefs of other agents (who are recursively reasoning about beliefs as well). This leads to *nested beliefs* which can make it impossible to losslessly reduce a player's knowledge beyond their full observation history.

This lack of a compact belief-space has prevented value based dynamic programming (*e.g.,* value iteration) methods from being used to solve DecPOMDPs. While value methods have been quite successful at solving POMDPs, all current DecPOMDP approaches are policy based methods, where policies are sequentially improved and evaluated at each iteration. Even using policy methods, the curse of history is still a big problem, and current methods deal with it in a number of different ways. [23] simply removed beliefs with low probability. Some use heuristics to prune (or never explore) particular belief regions [87, 73, 61, 60]. Other approaches merge beliefs together (*a.k.a.* belief compression) [23, 24]. This can sometimes be done losslessly [62], but such methods have limited applicability and still usually result in an exponential explosion of beliefs. There have also been approaches

that attempt to operate directly on the infinitely nested belief structure [22], but these are approximations of unknown accuracy (if we stop at the n^{th} nested belief the $n^{th}+1$ could dramatically change the outcome). All of these approaches have gotten reasonable empirical results in a few limited domains but ultimately scale and generalize poorly.

Despite their computational complexity, DecPOMDPs are still much easier to analyze and solve than POSGs - their general-sum counterparts (Section 2.2.5). This distinction is immediately evident when looking at research on POSGs vs. DecPOMDPs (which are common-payoff POSGs). There has been a great bulk of research into solving DecPOMDPs while there have been almost no attempts to solve general POSGs. There even exists a suite of public benchmark problems for DecPOMDPs while none exist for POSGs. DecPOMDPs are a strict subset of POSGs so if we believe MaGIIs are a good approximation for POSGs we should believe that they would be a good approximation for DecPOMDPs as well.

This chapter shows how to approximate DecPOMDP problems using a MaGII (with shared reward) and use the framework we presented in the previous chapter to develop a point based value iteration (PBVI) algorithm which efficiently solves the MaGII. The algorithm developed is the first value-based (as opposed to policy-based) dynamic program for general DecPOMDPs and outperforms all previous algorithms on standard benchmarks.

5.1 Common-payoff MaGIIs

When all agents share the same reward function the game is called common-payoff. Common-payoff MaGIIs are thus MaGIIs where each agent receives the same reward. Just like for DecPOMDPs, because agents share rewards they cooperate with each other and don't have to worry about individual rationality, unilateral deviation, threats, or trade-offs between maximizing one agent's utility over another. However, because the world is still partially observable, agents still need to reason over other agent's beliefs. This is where MaGIIs become a boon over DecPOMDPs. As explained in Section 4.2.2, agents in a MaGII only need to reason about level-0 beliefs (a distribution over states). One requirement for MaGIIs is that agents know the policies of other agents. This requirement is satisfied in general by assuming agents are rational (which is a dubious assumption). However, common-knowledge

of policies is easily justified for common-payoff games where policies are likely created together (*e.g.,* a designer dictating a policy to each agent), or agreed on in advance (to mutually improve utility).

5.1.1 Common-payoff MaGIIs as POMDPs

In this section we show that a common-payoff MaGII can be converted into a decision theoretic equivalent POMDP. The corresponding POMDP has exponential actions, but this equivalence allows us to use many theoretical POMDP results, and allows us to adapt POMDP solutions techniques.

Given a MaGII $\langle N', A', O', \Omega', S', P', R', s'^{(0)} \rangle$, let us define the following factored POMDP (which we will call the belief-POMDP of the MaGII):

- $S = S'$
- $O = \{\}$ (no observations)
- $A = \prod_{i=1}^{n} \prod_{j=1}^{|O'_i|} A'_i$ is the set of actions (one action for each agent for each belief) where $a_{i,o \in O'_i} \in A'_i$
- $P(s'|s,a), = P'(s'|s, \langle a_{o_1}, \cdots, a_{o_n} \rangle)$
- $R(s,a) = R'(s, \langle a_{o_1}, \cdots, a_{o_n} \rangle)$
- $s^{(0)} = s'^{(0)}$ is the initial state distribution

An action in this belief-POMDP is essentially a strategy for each agent (*a.k.a.* specifies what each agent should do for every belief they might have). In other words, it is a mapping from observation to action ($A = \{O' \to A'\}$). The action space thus has size $\prod_i |A'_i|^{|O'_i|}$ which is exponentially more actions than the number of joint-actions in the original MaGII. Both the transition and reward functions use the modified joint-action $\langle a_{o_1}, \cdots, a_{o_n} \rangle$. This joint-action consists of one action per agent. Each agent i's action a_{o_n} is the action prescribed to belief o_i where o_i is the state factor specifying which belief agent i has. In other words, $\langle a_{o_1}, \cdots, a_{o_n} \rangle$ is the action that would be taken once agents see their beliefs and follow action $a \in A$.

The agent in the belief-POMDP acts like a mediator for the sequence of Bayesian games induced by the MaGII. The agent is playing a collaborative Bayesian game. At every time-step the agent must give a strategy to each agent (a solution to the current Bayesian game). The states of the belief-POMDP still contain the belief for each of the decentralized agents of the MaGII. The belief-POMDP uses these underlying beliefs to transition and give rewards based on if the agents with those beliefs took the actions that the mediator (the agent of the belief-POMDP) prescribed. The mediator only knows what is commonly known and thus receives no observations.

Every policy of the MaGII maps (one-to-one) to a policy in the belief-POMDP in a natural way. Recall that in Section 4.2.2 we define a belief in a MaGII as a common-knowledge distribution along with the current observation. Thus a policy of the MaGII is a mapping from common-knowledge distribution and current observation to action ($\pi' : \Delta(S') \times O' \to A'$). A belief in the POMDP defined above is the common-knowledge distribution. Actions of the belief-POMDP are mappings from observations to actions. Thus a policy of the belief-POMDP is a mapping from common-knowledge distribution to a mapping from observations to actions ($\pi : \Delta(S) \to O' \to A'$). This composite function has the same domain and range as a policy of the MaGII and thus the sets of possible functions have a one-to-one correspondence where $\pi(s^{(t)})_{i,o} = \pi'_i(s^{(t)}, o)$

Not only is there a correspondence between policies of the two models, but these policies result in the same expected utility starting from any common-knowledge distribution over states. Note that the initial starting distribution s^0 is such a common-knowledge distribution over states. We prove this utility equivalence:

Lemma 5.1.0.2. *Let $\pi' :: \Delta(S') \times O' \to A'$ be a policy of a common-payoff MaGII $\langle N', A', O', \Omega', S', P', R', s'^{(0)} \rangle$ and $\pi : \Delta(S) \to \{O' \to A'\}$ is a policy of the belief-POMDP with $\pi(s^{(t)})_{i,o} = \pi'_i(s^{(t)}, o)$, then $V_{\pi'}(s'^{(0)}) = V_\pi(s^{(0)})$ (the expected utility of the two policies are equal).*

Proof. Using Equation 18 and the fact that for common-payoff MaGIIs, stage game strategies are pure (actions will be deterministic), the value of a joint-policy in a MaGII is:

$$^{\text{MaGII}}V_{\pi}^{t}(s) = R(s,\vec{a}) + \gamma \sum_{s' \in S} P(s|s',\vec{a}) V_{\pi}^{t+1}(s')$$

The value of a policy-tree in a POMDP is:

$$V_{\pi}(\vec{O},s) = R(s,a) + \gamma \sum_{s' \in S} \sum_{o' \in O} P(s',o'|s,a) V_{\pi}(\left\langle \vec{O},o' \right\rangle, s')$$

However we can simplify this because there are no observations. At every time-step there is only one possible history of observations, and thus the policy tree is only a policy chain which we can index by the time-step. Removing uncertainty over observations we get:

$$^{\text{POMDP}}V_{\pi}(t,s) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{\pi}(t+1,s')$$

While the notation of these two value computations is slightly different, the policy tree progression is identical yielding analogous actions and thus identical rewards. The two models will receive the same expected reward at each time-step and therefore end up receiving the same utility.

□

We have shown that common-payoff MaGIIs can be turned into POMDPs but this does not mean that we can easily solve these POMDPs using existing methods. The action-space of the belief-POMDP is exponential with respect to the number of observations of the MaGII. Existing POMDP algorithms assume that actions can be enumerated efficiently, which isn't possible beyond the simplest belief-POMDPs. Luckily, belief-POMDPs exhibit a great deal of structure which can be exploited. In Section 5.2 we develop an efficient point-based value iteration algorithm to solve the belief-POMDP.

5.1.2 Value functions for common-payoff MaGIIs

For single or decentralized agents, a value function is a mapping from belief to value (the maximum expected utility that the agents can achieve). In an MDP, beliefs correspond to states so this becomes a mapping from state to utility. A belief in a POMDP is a probability

distribution over states, so the utility function is over the continuous $|S|$-dimensional simplex of probability distributions over states. Luckily this function is concave and locally linear. A DecPOMDP's value function is not so lucky. As Oliehoek [59] showed in his thesis, an optimal belief for a DecPOMDP requires the entire action-observation history. This makes the domain of any optimal DecPOMDP value function grow exponentially with history.

Common-payoff MaGIIs do not face this problem. As shown in Section 5.1.1, common-payoff MaGIIs can be viewed as belief-POMDPs. While belief-POMDPs have exponential action-spaces, their state-space is the same as the MaGII. We also showed that every policy of a belief-POMDP corresponds to a policy of the MaGII which achieves the same value. Therefore the value function of the belief-POMDP can double as the value function for the common-payoff MaGII. This value function is a mapping from only the common-knowledge distribution to utility. Thus this value represents the ex-ante utility (before private information is known) for an agent. The ex-post utility (after private information) can be computed using this ex-ante value function.w

An optimal belief for a MaGII consists of a common-knowledge distribution over states combined with the current observation. This means that the value function for a common-payoff MaGII can look much like a POMDP's value function - a mapping from state-distribution to action.

We represented such a value function for a common-payoff MaGII in the same way that is typical for belief-MDPs, by using a set of hyperplanes $\Gamma = \{\alpha_1, \alpha_2, \cdots, \alpha_m\}$ where $\alpha_i \in \mathbb{R}^{|S|}$. These hyperplanes each represent the value of a particular policy across beliefs. The value function is then the maximum over all hyperplanes. For a belief $b \in \mathbb{R}^{|S|}$ its value as given by the value function Γ is:

$$V_\Gamma(b) = \max_{\alpha \in \Gamma} \alpha \cdot b$$

Such a representation acts as both a value function and a policy. While each α vector corresponds to the value achieved by following a specific policy, we only need to specify the immediate action to take. An agent can take the prescribed action, compute the resulting

belief state and find the new optimal α vector which will direct the agent in the next time-step. The ex-post value can be computed in the same way with the only difference being that the belief b should be the posterior probability distribution across beliefs after an agent receives their observation.

5.2 Point-based value iteration for common-payoff MaGIIs

In this section we present an algorithm for solving for the optimal value function of a common-payoff MaGII. Our algorithm very closely resembles the PERSEUS algorithm [84] for POMDPs which is a specific version of point-based value iteration [71]. We start with a poor approximation of the value function and at each stage of the value iteration we improve it by performing a one-step backup. We keep improving the value function using the previous value function estimate until the value function no longer improves meaningfully.

Recall that the difficulty with belief-POMDPs is that they have an exponential action-space which can not be enumerated explicitly. Spaan and Vlassis [84] showed that PERSEUS can be extended to very large action spaces by randomly sampling actions. While this may work for some domains, often for decentralized problems only one particular strategy among the many will prove effective. This will stymie a randomized approach.

Our approach is to overcome the exponential action space by defining an integer linear program which computes the optimal joint-strategy for a particular starting belief. This integer program is typically very fast because it's linear relaxation is usually integral. First, we review the standard point-based value iteration algorithm and then explain how to perform our special backup.

5.2.1 Point-based value iteration

The high-level outline of our point-based algorithm is the same as PERSEUS. First, we sample common-knowledge beliefs and collect them into a belief set B. This is done by taking random actions from a given starting belief and recording the resulting belief state. This sampling is given as Algorithm 4.

We then initialize our starting value function Γ^0 to have a single low conservative value vector (such as $R_{\min}/\gamma$). Every iteration then attempts to improve the value function at

Algorithm 4 Belief sampling

Inputs: MaGII $\langle N, A, O, \Omega, S, P, R, s^{(0)} \rangle$
number of samples c
sampling depth d
Output: set of beliefs $B : \{b \in \mathbb{R}^{|S|}\}$

1: $B \Leftarrow \emptyset$
2: **for** $1 \rightarrow c$ **do**
3: $b \Leftarrow s^{(0)}$
4: **for** $1 \rightarrow d$ **do**
5: **for all** $i \in |N|,\ o \in O_i$ **do**
6: $a_{i,o} \Leftarrow \text{Random}(a \in A_i)$
7: **for all** $s \in S$ **do**
8: $b'(s) \Leftarrow \sum_{s' \in S} b(s')P(s'|s, a)$
9: $B \Leftarrow B \bigcup b'$
10: $b \Leftarrow b'$
11: **return** B

each belief in our belief set B. A random common-knowledge belief $b \in B$ is selected and we compute an improved policy for that belief by performing a one step backup. This backup involves finding the best immediate strategy-profile (an action for each observation of each agent) at belief b along with the best continuation policy from Γ^t. We then compute the value of the resulting strategy + continuation policy (which is itself a policy) and inserts this new α-vector into Γ^{t+1}. Any belief that is improved by α (including b) is removed from B. We then select a new common-knowledge belief and iterate until every belief in B has been improved. We give this as Algorithm 5. The details of the one step backup (line 9) are explained in the next section.

5.2.2 One step belief backup

A value iteration backup improves the value-function by computing the best immediate action using the current value-function as the value estimate for the next time-step. In this way we push back the uncertainty in our value estimate by one time-step and let the discount factor diminish our error. For a POMDP, this backup is quite simple. We can enumerate all actions and take the best in a very similar way to the Bellman backup for MDPs. Unfortunately we can't do this for belief-POMDPs because the action space is too large. Instead we define an integer linear program, one for each belief and successor policy.

Algorithm 5 A point-based value iteration algorithm.

Inputs: common-payoff MaGII $\langle N, A, O, \Omega, S, P, R, s^{(0)} \rangle$
belief set $B^{\forall}$
stopping criterion ϵ_Γ
Output: value function Γ

$\Gamma' \Leftarrow \{\langle R_{\min}, \cdots, R_{\min} \rangle\}$
repeat
  $B \Leftarrow B^{\forall}$
  $\Gamma \Leftarrow \Gamma'$
  $\Gamma' \Leftarrow \emptyset$
  while $B \neq \emptyset$ **do**
    $b \Leftarrow \text{Random}(b \in B)$
    $\alpha \Leftarrow \Gamma(b)$
    $\alpha'_v \Leftarrow \max_{a \in A^O} \max_{\alpha' \in \Gamma} V_{a,\alpha'}(b)$ {where $A^O = \prod_{i=1}^{n} \prod_{j=1}^{|O'_i|} A'_i$}
    if $\alpha'_v > \alpha_v$ **then**
      $\alpha_v \Leftarrow \alpha'_v$
      $\alpha_\pi \Leftarrow \arg\max_{a \in A^O} \max_{\alpha' \in \Gamma} V_{a,\alpha'}(b)$
    $\Gamma' \Leftarrow \Gamma' \bigcup \alpha$
    for all $b \in B$ **do**
      if $\Gamma'(b)_v > \Gamma(b)_v$ **then**
        $B \Leftarrow B/b$
until $\Gamma' - \Gamma < \epsilon_\Gamma$
return Γ

The integer program finds the optimal pure joint strategy starting from a given belief $b \in B$ and following a given continuation policy with value $\alpha \in \Gamma$. It optimizes over variables $x_{\vec{a},s}$, one for each joint action for each state. Each variable represents the probability of joint-action $\vec{a}$ being taken if the state is in fact s. Remember that a state is not directly observed and includes the private beliefs of each agent. Our program is integral because we restrict each variable to be either zero or one. This is equivalent to only searching for a pure strategy. The common-payoff nature of the game allows agents to cooperate completely and guarantees that the best strategy will be a pure strategy. We must force our variables to be integral because otherwise agents could implicitly share private information through the coordination mechanism.

Constraints must be imposed on the program's variables $x_{\vec{a},s}$ to insure that they form a proper probability distribution and that the distribution provides no additional private information to an agent beyond what they already know. More formally, the probability that an agent takes an action is independent from any other agent's private observation. This is

also part of the restriction of an agent normal form correlated equilibria, although because agents share a reward we need not worry about deviation. From an agent's perspective, their action is conditionally independent of the other agents' observations.

These restrictions can be expressed by the following linear constraints (Equation 21):

For each agent i, and for each partial joint-actions of other agents $\vec{a}_{-i} \in A_{-i}$

For each state $s = \langle \omega, o_1, \cdots, o_n \rangle$: (where $s/o_i = \langle \omega, o_1, \cdots, o_{i-1}, o_{i+1}, \cdots, o_n \rangle$)

$$\sum_{a_i \in A_i} x_{\vec{a},s} \;=\; x_{\langle \vec{a}_{-i}, s/o_i \rangle} \tag{21a}$$

For each unobserved factor $\omega \in \Omega$ and for each joint-action $\vec{a} \in A$

For each state $s = \langle \omega, o_1, \cdots, o_n \rangle$ (where $s/\omega = \langle o_1, \cdots, o_n \rangle$)

$$x_{\vec{a},s} \;=\; x_{\langle \vec{a}, s/\omega \rangle} \tag{21b}$$

for each $s \in S$:

$$\sum_{\vec{a} \in A} x_{\vec{a},s} = 1 \tag{21c}$$

for each $s \in S$ and $\vec{a} \in A$:

$$x_{\vec{a},s} \geq 0 \tag{21d}$$

In order to make the description of the conditional independence constraints (21a) more concise, we use the additional variables $x_{\langle \vec{a}_{-i}, s/o_i \rangle}$. These represent the posterior probability that agent i thinks the unobserved state is s after receiving observation o_i. These constraints enforce that an agent's posterior probabilities are unaffected by other agent's observations. However, we also need to enforce that agent's posterior probabilities are unaffected by the unobserved state factors ω. Constraints 21b handle this is the same way as constraints 21a. If we viewed nature as a separate agent who observes the true state but takes no actions then (21b) could be included as part of (21a). Note that all these new variables can be immediately substituted for and thus do not increase the number of free variables. Constraints 21b and 21c make sure that for each state, each set of variables over joint-actions is a probability distribution.

Any feasible assignment of variables $x_{\vec{a},s}$ represents a valid strategy for the agents. It also corresponds to an action of the belief-POMDP. We have defined these constraints to be independent of the current belief or continuation policy. This allows us to use this same feasible set for all linear programs, improving performance.

In order to backup a particular belief point we must maximize the utility of a strategy x. When using a continuation policy that has value vector α a resulting belief b' will achieve value $\sum_s b'(s)\alpha(s)$. The resulting belief b' after taking action $\vec{a}$ from belief b is $b'(s') = \sum_s b(s)P(s'|s,\vec{a})$. Putting these together, along with the probabilities $x_{\vec{a},s}$ of taking action $\vec{a}$ in state s we get the value of a strategy x from belief b followed by α:

$$V_{x,\alpha}(b) = \sum_{s \in S} b(s) \sum_{\vec{a} \in A} \sum_{s'} P(s'|s,\vec{a})\alpha(s')x_{\vec{a},s} \tag{22}$$

This is the quantity that we wish to maximize, and can combine with constraints 21 to form a linear program which returns the best action for each agent for each observation (strategy) given a continuation policy α. To find the best strategy/continuation policy pair, we can perform this search over all continuation vectors in Γ:

Maximize:	$\sum_{s \in S} b(s) \sum_{\vec{a} \in A} \sum_{s'} P(s'|s,\vec{a})\alpha(s')x_{\vec{a},s}$
Over:	$x \in \{0,1\}^{\lvert S\rvert\lvert A\rvert}$, $\alpha \in \Gamma$
Subject to:	inequalities 21

(23)

Each integer program has $|S||A|$ variables but there are also a great number of equality constraints. Most of these equality constraints are linearly independent from the other constraints and thus reduce the number of free variables by one each. However, these equality constraints grow at a slower rate than the number of variables. There is one important exception to this which are the constraints 21b. There are exactly $|\Omega||A|$ of these constraints - one for each unobserved factor and joint-action. Therefore the number of unobserved states does not increase the number of free variables. Taking this into account, the number of free variables is $\mathbb{O}(|O||A|)$.

The optimization problem given above requires searching over all $\alpha \in \Gamma$ to find the best

continuation value-vector. We could solve this problem as one large linear program with a different set of variables for each α, however each set of variables would be independent and thus can be solved much faster as separate individual problems.

5.2.3 The basic algorithm

A full point-based value iteration algorithm for common-payoff MaGIIs starts with collecting beliefs using Algorithm 4. We then modify Algorithm 5 by using the one step backup operation given in Equation 23 as step 9. This algorithm will iteratively improve the value function at all beliefs. The algorithm stops when the value function improves less than the stopping criterion ϵ_Γ. Therefore, at every iteration at least one of the beliefs must improve by at least ϵ_Γ. Because the value function at every belief is bounded above by $R_{\max}/\gamma$ we can guarantee that the algorithm will take fewer than $(|B|R_{\max})/(\gamma \cdot \epsilon_\Gamma)$ iterations.

There are a few subtleties with this algorithm that one should be careful about. One such subtlety is that successive value functions may not monotonically increase. It is true that the next iteration's value function will be greater or equal at all sampled belief points, however in-between belief points there are no guarantees. Two α-vectors may have pivoted on their sampled belief axis and lowered the value at their intersection. This can cause problems when computing the backup - an optimal value of the backup may be lower than the previous iteration's computed value (because it relies on an in-between successor belief). When this happens, it does not mean that the previously computed value is invalid. It only means that the α-vector previously used is no longer supported by the sampled beliefs, and has thus been removed from the value function. Such cases are rare, but when they happen we can simply use the previous α-vector. To make this concrete, line 9 of Algorithm 5 should read:

$$\alpha_b \Leftarrow \textbf{maximum of: } \max_{\alpha \in \Gamma} b \cdot \alpha \textbf{ and } \max_{a \in A^O} \max_{\alpha' \in \Gamma} V_{a,\alpha'}(b)$$

5.2.4 Extracting a policy from the value function

Algorithm 5 returns a value function. Ultimately we want a policy. Using the value function a policy can be constructed in a greedy manner for each player. This is accomplished using a very similar procedure to how we construct a policy greedily in the fully observable case. Every time-step the actors in the world can dynamically compute their next action without needing to plan their entire policy.

Agents start with a known common-knowledge distribution over states. At each time-step agents use their value function to find the value-vector $\alpha \in \Gamma$ that has the maximum utility for their current common-knowledge belief. This value-vector has an associated strategy (a joint-action for each state) α_π. Because of the conditional independence constraints used to create this strategy, each agent-type will have the same action across all states consistent with that type (in other words, the strategy will be equivalent to each agent-type being prescribed an action). This joint strategy is common-knowledge so when each agent follows their own individual part of the strategy, they know how to update the common-knowledge belief using Equation 17: $p^{t+1}(s') = \sum_s p^t(s) \sum_{\vec{a}} \alpha_\pi(\vec{a}) \; P(s'|s, \vec{a})$. In the next state agents repeat this process of finding the best support-vector and using the prescribed strategy followed by updating their common-knowledge belief.

5.2.5 Improved PBVI for MaGIIs

Each backup requires solving a linear program which is significantly more computationally intensive than a traditional PERSEUS backup. This places much greater demand on the algorithm to more judiciously utilize backups. We introduce several improvements that help mitigate the number of heavy backups needed. First, we improve the use of randomness in the algorithm. Second, we show how to more effectively search over the space of successor value-vectors $\alpha \in \Gamma$. Finally, we introduce light backups that do not require integer or linear programming. These light backups are not guaranteed to be optimal, but often produce improved value regardless.

5.2.5.1 *Improved belief sampling*

The basic algorithm is a randomized algorithm in two regards. First, the algorithm is highly dependant on the initial selection of beliefs for the belief set. The value function will not be backed-up anywhere except these points. The value function will still be valid and useful for all belief points, but will likely decrease in accuracy the farther away it is from a sampled belief. Ultimately, we only need one belief point per optimal value vector. We also only need beliefs that are reachable by the optimal policy. Recall that the value function is always an under estimate. By omitting beliefs we are potentially lowering our value estimate in that region. If we would never want to enter that belief region anyway, then a lower belief will only help to keep us out. We also don't want to spend unnecessary computation improving suboptimal policies. It is therefore preferable for the sampled points to be clustered along the optimal belief trajectory, while sparsely sampled elsewhere to generate coverage.

In order to make sure we improve the value function every iteration it is important to use all the sampled belief points from one iteration in the next iteration. The basic algorithm makes this simple by fixing the belief set ahead of the main algorithm. However, there is no reason why we can't add beliefs over time. This makes sense for two reasons. First, when starting the algorithm it is easy to improve the value function with only a few value vectors so a large number of beliefs are not needed. And second, in the process of improving the value function and associated policies we learn about the value space and where good choice for beliefs may lay. We use this intuition to propose a simple new method for selecting belief points.

Our improved belief sampling (Algorithm 6) starts in the same way as the original, albeit with fewer samples (*e.g.,* 1/10th what would normally be used). We then run the algorithm until convergence with a greater ϵ_Γ value (*e.g.,* 10 times). When the algorithm converges we add a batch of new beliefs to our sampled belief set. However, this time we do not choose random actions. Instead, we use an ϵ_π-greedy policy (*e.g.,* with $\epsilon_\pi = 0.1$). In this way we add points closer to the optimal trajectory, while still allowing for exploration. We can also change the number of samples, the depth of samples, ϵ_Γ and ϵ_π over time to gradually improve the accuracy of the value function. Experimentally our improved belief

Algorithm 6 Improved belief sampling

Inputs: MaGII $\langle N, A, O, \Omega, S, P, R, s^{(0)} \rangle$
number of samples c
sampling depth d
greedy policy π
random exploration probability ϵ_π
Output: set of beliefs $B : \{b \in \mathbb{R}^{|S|}\}$

$B \Leftarrow \emptyset$
for $1 \rightarrow c$ **do**
  $b \Leftarrow s^{(0)}$
  for $1 \rightarrow d$ **do**
    for all $i \in |N|,\ o \in O_i$ **do**
      if Random($[0, 1]$) $< \epsilon_\pi$ **then**
        $a_{i,o} \Leftarrow$ Random($a \in A_i$)
      else
        $a_{i,o} \Leftarrow \pi(i, o)$
    for all $s \in S$ **do**
      $b'(s) \Leftarrow \sum_{s' \in S} b(s') P(s'|s, a)$
    $B \Leftarrow B \bigcup b'$
    $b \Leftarrow b'$
return B

sampling improves performance by an order of magnitude, allowing both faster convergence and higher accuracy value functions.

5.2.5.2 Belief symmetry

If a MaGII has transition and reward probabilities based only upon the unobserved factors Ω then types have no inherent meaning beyond inducing a prior on the unobserved state factors. For these problems, the distribution is what matters not the particular label of an agent's type. Therefore when two common-knowledge beliefs are equal up to a change of type labels then they will have equal value. When problems have this feature, we can use this fact to reduce our belief sample space by sorting the labels of sampled beliefs into a canonical representation where the most likely belief for each agent is ordered first.

Problems that transition independently from type are actually quite common. For example, any MaGII where types only represent internal beliefs and not features of the world have this property. More importantly, all of the methods to approximate DecPOMDPs with MaGIIs presented below (Section 5.3) have this structure. We can therefore use this trick

to reduce the number of samples needed to cover our belief space, and in turn reduce the number of value-vectors needed to represent our value function.

5.2.5.3 Improved continuation policy search

Each heavy backup (as given by Equation 23) requires performing a linear program for each continuation value $\alpha \in \Gamma$. Every backup does not need to search all continuation values every backup; we only need to find a single one that improves the backup belief. The set of value-vectors Γ may be large. All of these vectors are optimal for some set of beliefs, however the beliefs for which they are optimal may not be reachable from the belief being backed up. Any value-vector that can not be reached will never be the optimal successor. Even if a value-vector can be reached, if it has much lower value compared to other reachable value-vectors then it is also a poor candidates for succession.

Using these insights, we try to rank the possible continuation value-vectors by likelihood of being optimal. When performing a backup we search by greatest likelihood first and stop as soon as we find a continuation policy that improves our belief. For each belief we keep track of the values achieved of any successor value-vector that we previously tried. We then estimate the value of all the other value-vectors by picking the most recent successor belief and evaluating the value-vectors at that point. This gives a rough estimate to how well we believe all the value-vectors will fair allowing us to prioritize the order in which we search value-vectors.

The value-function Γ changes quickly, so the exact same value-vectors will not stick around for many iterations. However, each value-vector is supported by a witness belief (in B), and for a particular belief the vector it supports does not typically change drastically over time. We therefore don't keep ranks on value-vectors, but on the witnesses. This allows us to rank a new value-vector using a previously computed optimal strategy based on the new value-vector's witness. In this way continuation value-vectors that are close to successor beliefs are more likely to have a higher rank than value-vectors far away from successor beliefs.

5.2.5.4 *Light backups*

Backing up a belief point in the basic algorithm requires solving $|\Gamma|$ separate linear programs. We showed in Section 5.2.5.3 that not all of these linear programs need to be run - we only need to find a single one that improves the belief's value. However, even a single linear program is costly. There are $\mathbb{O}(|O||A|)$ variables and linear programs typically have running time quadratic in the number of variables. We introduce light backups as a means to prevent solving unnecessary linear programs.

The value at a belief increases during a backup for two reasons. Either a better immediate strategy is found, or the value at the successor belief improves (the same strategy from the same starting belief will always end up at the same successor belief). The second case is common. In fact, anytime a belief improves it is often the case that other beliefs' values are dependent on it. We can take advantage of this fact by propagating belief updates without searching for better strategies.

Backing up a belief for a known strategy and continuation value is very quick and does not require solving a linear program. All we have to do is compute the value using Equation 22 which is just a weighted sum. This weighted sum is equivalent to the value of the linear program 23 at the previous best basic feasible solution. After every round of heavy backups we perform multiple rounds of light backups until the values converge. These light backups dramatically lower the number of heavy backups needed.

5.2.5.5 *Improved algorithm*

We have introduced a number of new parameters to the original algorithm. These control how we sample and how quickly we add new samples. We can also change these parameters over time. For the final improved algorithm we pass functions over time for these parameters. While finding new sample beliefs (using Algorithm 6) we check to make sure the new belief is not equal to a belief we previously sampled. If we find a collision we simply throw out the belief and re-sample. While this doesn't effect performance greatly, it does make the algorithm more consistent and predictable. The final improved algorithm is given as Algorithm 7.

5.3 Approximating DecPOMDPs

We have shown how to compute the value function for common-payoff MaGIIs. While there are problems that can be exactly represented as a common-payoff MaGII, a key motivation for developing MaGIIs in the first place was to solve general problems. In this section we explore a number of different methods for approximating DecPOMDPs with common-payoff MaGIIs. This section can be viewed independently from the solutions to solve MaGIIs. There are likely more powerful algorithms for solving MaGIIs but regardless of solution technique the transformation from DecPOMDP to MaGII is an important modeling step. A bad transformation has the potential to produce very poor results, even if the optimal policies for the resulting MaGII are found. Here we present a number of different possible approximations. The first two have been implicitly used in previous research while the third is novel. Unlike the previous approximations, our approximation scheme has optimality guarantees.

In all approximations the goal is to reduce the possibly infinite number of beliefs of a DecPOMDP to a small finite set of beliefs. This *belief compression* should maintain an agent's ability to produce high quality policies as much as possible. Note that it is not enough to specify a mapping from histories to compressed beliefs. We must also describe a transition function that operates only on the small belief set and does not rely on observation histories. In other words, a successor belief can only depend on the most recent belief not any previous belief. Recall from Section 4.3.3 that it is acceptable if the MaGII we create does not satisfy the Markov property on observations, we can assume it does. It is this assumption (that beliefs in the MaGII are sufficient so agents dont need to remember history) that is the true point of approximation.

In all the approximation methods presented here the states of the DecPOMDP are used as the unobserved states in the MaGII. The transition function of the new MaGII mirrors the DecPOMDPs for the unobserved states. The differences are in the beliefs (*a.k.a.* observations) given to each agent. This makes each of these approximations equivalent to belief compression or observation compression.

5.3.1 Example

As an example we attempt to construct a MaGII for the multi-agent tiger problem (Section 5.4.1.1). An agent's observation history consists of a series of right/left growls. During a single episode the order of these observations does matter, only how many of each type of growl have been heard (*e.g.,* hearing R,L,L contains the same information as hearing L,R,L). Furthermore, it is only the difference of growls that matters, not the absolute number (*e.g.,* hearing R,L,L,L contains the same information as hearing L,L). Also, the greater the difference between growl locations, the less additional growls effects an agent's belief (*e.g.,* hearing a growl left seven times is only slightly more informative than hearing the growl left six times). This reasoning could lead us to propose a MaGII approximation of the tiger problem with 5 beliefs corresponding to the difference of growls heard (+2, +1, 0, -1, -2). Indeed, this turns out to be a good choice and achieves rewards close to optimal, however our reasoning for constructing this approximation does not generalize to other models. We want to develop approximation methods that work well across all problems.

5.3.2 Truncated history

An easy and straight forward approximation method is to simply truncate the history of observations - in other words, assume that agents have long term amnesia and can't remember what they observed more than h time-steps ago. This results in $\sum_{k=1}^{h} O_i^k$ possible beliefs for each agent. There are two ways to define these beliefs depending on whether or not agents know the current time-step. If agents can't remember how much time has passed then each of these beliefs will be identical at ever time-step. Thus the policy will be equivalent to a finite state controller where each belief is a state in the controller (with a corresponding action to take) and each observation corresponds to a transition. The problem of optimizing each agent's finite state controller has been tackled by a number of researchers. A major problem of these approaches is that optimizing finite state controllers is not a convex optimization problem and thus suffers from local optima.

5.3.3 Probability Threshold

Another method for bounding the number of histories is to remove or merge any history with low probability of occurring. If we set a threshold of ϵ_h, then we can guarantee that no more than $1/\epsilon_h$ histories will be present at any given time-step. However, after a few time-steps individual histories will become more and more unlikely. It is also unclear how to handle pruned histories. One solution is to always keep the top $1/\epsilon_h$ histories and re-normalize perceived transition probabilities. In other words, we could pretend that pruned histories just don't exist. The problem with this approach and other probability threshold based approximations is that histories with lower probability may be just as important to reason about as more likely histories. There has been no theoretic accounting of when it might be reasonable to ignore low probability trajectories and no algorithm evidence to suggest doing so is generally acceptable.

5.3.4 Dynamic belief compression

We present here a novel approximation method based around letting agents themselves have the ability to decide how they want to merge their beliefs. This pushes the onus of belief compression onto the MaGII solver instead of the approximation method. We give agents this ability by creating a new modified MaGII where we interleave each normal time-step (where we fully expand each belief) with a belief-compression time-step (where the agents must explicitly decide how to best merge beliefs). We call these two phases belief expansion, and belief compression respectively.

In our compressed model, the type/observation given to each agent has two factors. The first factor is the compressed belief. The second factor is the most recent observation (or $\emptyset$ if agents just compressed their belief). This second factor is used to designate which of the two alternating phases the game is currently in (where $\emptyset$ designates belief expansion).

The first phase acts like the original DecPOMDP without any belief compression - the observation given to each agent is their previous belief along with the DecPOMDP's observation. No information is lost during this phase; each observation for each agent-type results in a distinct belief. This belief expansion occurs with the same transitions and

rewards as the original DecPOMDP.

The second phase is belief contraction. The dynamics of this phase are unrelated to the DecPOMDP. Instead an agent's actions are decisions about how to compress their belief. In this phase, each agent-type must choose their next belief but they only have a fixed number of beliefs to choose from (the number of beliefs t_i is a free parameter). All agent-types that choose the same belief will be unable to distinguish themselves in the next time-step; the belief label in the next time-step will equal the action index they take in the belief contraction phase. All rewards are zero. This second phase can be seen as a purely mental phase and does not effect the environment beyond changes to beliefs.

The number of contracted beliefs for each agent t_i are free parameters of the approximation scheme. At the beginning of the expansion phase each agent has possible observations (*a.k.a.* types) $\theta_1, \theta_2, \cdots, \theta_{t_i}$. There is no meaning to these beliefs beyond their index and how agents themselves define the beliefs. During the belief expansion phase the set of possible observations is $\{\theta_1, \theta_2, \cdots, \theta_{t_i}\} \times |O_i|$ which increases the number of possible observations from t_i to $t_i|O_i|$. During the belief contraction phase, each agent has t_i possible actions. Whichever action index a_i an agent takes will deterministically transition that agent into the expansion phase with belief θ_{a_i}. The second phase can be seen as a purely mental phase and thus does not effect the environment beyond the changes of beliefs. For this reason we also square-root the discount factor (or only discount during belief expansion, which is equivalent because all rewards during belief contraction are zero).

5.3.4.1 Formal approximation

Given a DecPOMDP $\langle N, A, S, O, P, R, b^{(0)} \rangle$ we formally define the new MaGII approximation model $\langle N', A', O', \Omega', S', P', R', s'^{(0)} \rangle$ with parameters $t_1, \cdots, t_n$ as:

- $N' = |N|$
- $A'_i = \max(A_i, t_i)$
- $O'_i = (O_i \bigcup \{\emptyset\} \times \{\theta_1, \theta_2, \cdots, \theta_{t_i}\}$
- $\Omega' = S$

- $S' = \Omega' \bigtimes O'$ with $s' = \langle \omega', o'_1, \theta'_1, \cdots, o'_n, \theta'_n \rangle$.
- $P'(s'|s,a) = \begin{cases} P(\omega', \langle o'_1, \cdots, o'_n \rangle | \omega, a) & \text{if } \forall i : o_i = \emptyset \text{ and } \theta'_i = \theta_i \\ 1 & \text{if } \forall i : o_i \neq \emptyset \text{ and } \omega' = \omega, o'_i = \emptyset, \theta'_i = \theta_{(a_i \mod t_i)} \\ 0 & \text{otherwise} \end{cases}$
- $R'(s,a) = \begin{cases} R(\omega, a) & \text{if } \forall i : o_i = \emptyset \\ 0 & \text{otherwise} \end{cases}$
- $s'^{(0)} = \left\langle b^{(0)}, \emptyset_1, 1, \cdots, \emptyset_n, 1 \right\rangle$ is the initial state distribution

We have constructed the MaGII such that at each time-step agents receive two observations: an observation factor $o_i \in O_i \bigcup \{\emptyset\}$, and their belief factor (type) $\theta \in \{\theta_1, \theta_2, \cdots, \theta_{t_i}\}$. The observation factor is $\emptyset$ at the expansion phase and the most recent observation, as given by the DecPOMDP, when starting the compression phase. The observation factor therefore distinguishes which phase the model is currently in. Agents should either all have their observation factor be equal to the empty set ($o_i = \emptyset$) or none of them. The probability of transitioning to a state where some agents have the empty set observation while others don't is always zero. Note that transitions during the contraction phase are deterministic (probability one) and the underlying state ω does not change. The action set sizes may be different in the two phases, however we can easily get around this problem by mapping any action outside of the designated actions to an equivalent one inside the designated action. We described this process above as ($a_i \mod t_i$). The new BB-DecPOMDP's state size is $|S'| = |S|(|O|+1)^n t^n$.

The agent specified beliefs have no intrinsic meaning - they are just labels. Their meaning is what agents give them. As an example, note that during belief compression an agent could rotate their actions for each agent-type (*e.g.,* instead of action 1 they would take action 2, and instead of action 2 they would take action 3 *etc.*). This would result in a relabeling of beliefs in the next time-step but would not affect utility. These compressed beliefs have meaning in two important ways. First, they encode level-0 beliefs by inducing a distribution over unobserved factors Ω. Second, because strategies are fully-observable (due to rationality), the belief compression choices of agents are common knowledge and

Table 2: Dimensions of seven common DecPOMP benchmark problems. S' are the factored states of the optimal approximation scheme given by the compression model in Section 5.3.4.

					$\|S'\|$ with $t_i =$				
	$\|N\|$	$\|S\|$	$\|A_i\|$	$\|O_i\|$	1	2	3	4	5
Dec-Tiger	2	2	2	2	18	72	162	288	450
Broadcast	2	4	2	2	36	144	324	576	900
Recycling	2	4	3	2	36	144	324	576	900
Grid small	2	16	5	2	144	576	1296	2304	3600
Box Pushing	2	100	4	5	3600	14400	32400	57600	90000
Wireless	2	64	2	6	3136	12544	28224	50176	78400
Mars Rover	2	256	2	6	12544	50176	112896	200704	313600

thus change the common knowledge distribution over time. This in turn causes agents to reach different belief-points in the belief-POMDP.

5.4 Experiments

This section gives experimental results for our our PBVI for DecPOMDP algorithm on six well known DecPOMDP benchmark problems. On all of these problems we met or exceeded the current best solution. This is particularly impressive considering that some of the algorithms were designed to take advantage of specific problem structure, while our algorithm is general.

5.4.1 Benchmark problems

We tested our PBVI for DecPOMDP algorithm on six well known DecPOMDP benchmark problems: DecTiger, Broadcast, Grid-small, Cooperative Box Pushing, Recycling Robots, and Wireless Network. An overview of the sizes of each problem is given in Figure 5.4.1. We also attempted to solve the Mars Rovers problem, except its PP-MaGII transition model was too large for our 8GB memory limit. Because Mars Rover is also a common benchmark problem, we list its properties here.

5.4.1.1 Decentralized tiger

The single agent tiger problem [14] has been used often to test and explain POMDP algorithms. In the problem there are two doors. One holds a tiger (-100 reward), the other some money (+10 reward) with even probability of which door holds which. A contestant must

choose the correct door, but can wait and listen for growling noises (which are reported with 85% accuracy). Waiting results in a penalty of one (-1 reward). Figure 29 depicts this game. There have been a number of multi-agent versions of this game [22, 23, 92]. Here we present the standard DecPOMDP version.

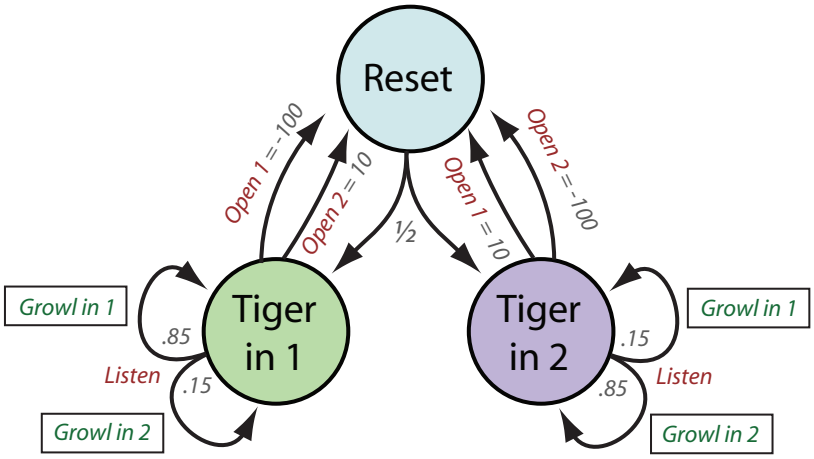

Figure 29: The single agent tiger problem. Each circle is an underlying unobserved state.

The decentralized tiger problem [57] is similar to the single agent tiger problem, except two agents must work together to open the correct door. At every time-step the two agents must decide to open a door or listen. The transitions are the same as in the single agent version where if either agent opens a door (regardless of what the other agent does) then the game resets. The rewards are mostly similar with a few exceptions. One, if one of the agents opens the wrong door (with the tiger behind it) then the pair receives a reward of -50, -100, or -101 depending respectively on if the other agent opened the same door, the opposite door, or listened. Otherwise both agents receive the sum of rewards as given by the single agent reward structure. A tabular representation of the game is given by Table 5.4.1.1.

This game heavily favors coordination between the agents; they are penalized if they don't choose the same action at every stage. However, neither agent ever receives any information about the actions of the other player, or even their own reward. They don't know when the game resets unless they themselves reset the game (by opening a door). In order to solve this problem the agents must reason about the other agent's mental states

Table 3: Tabular reward, transition and observation functions for the decentralized tiger problem.

State: Tiger Left

	Open-left			Open-right			Listen		
	Reward	Trans	Obs	Reward	Trans	Obs	Reward	Trans	Obs
Open-left	-50	.5/.5	.5/.5	-100	.5/.5	.5/.5	-101	.5/.5	.5/.5
Open-right	-100	.5/.5	.5/.5	20	.5/.5	.5/.5	9	.5/.5	.5/.5
Listen	-101	.5/.5	.5/.5	9	.5/.5	.5/.5	-2	1.0/0	.85/.15

State: Tiger Right

	Open-left			Open-right			Listen		
	Reward	Trans	Obs	Reward	Trans	Obs	Reward	Trans	Obs
Open-left	20	.5/.5	.5/.5	-100	.5/.5	.5/.5	9	.5/.5	.5/.5
Open-right	-100	.5/.5	.5/.5	-50	.5/.5	.5/.5	-101	.5/.5	.5/.5
Listen	9	.5/.5	.5/.5	-101	.5/.5	.5/.5	-2	0/1.0	.15/.85

and not just about the underlying state of the world. This makes it a suitable DecPOMDP toy problem.

5.4.1.2 Broadcast channel

The second benchmark problem models a multi-access broadcast channel [29, 63]. There is a single channel on which both agents wish to send messages. If agents attempt to send a message at the same time there is a collision and neither message is sent. Each agent has a message buffer capable of holding a single message. When there is a message in the buffer the agent can attempt to send it. The message remains in the buffer until sent. When any agent successfully sends their message all agents receive a reward of +1 and the sending agent's message buffer becomes empty. If their buffer is empty, each agent has probability p_i that their buffer receives a new message. Any other action besides successfully sending a message has reward 0. Every time-step an agent gets a noisy observation (accurate with probability 0.9) of if there was a collision on the network or not. This is the only observation an agent gets. The standard broadcast channel problem has two agents with $p_1 = 0.9$ and $p_2 = 0.1$. This means there are four states corresponding to the four possible configurations of the two buffers.

5.4.1.3 Meeting on a grid

The meeting on a grid problem [10] models agents who live in a grid world, move stochasticly, and wish to meet each other on the grid. A small version of this problem with a 2x2 grid and only two observations was proposed by C. Amato, D. S. Bernstein, and S. Zilberstein [2]. This version is known as the grid-small problem and has been used as a common benchmark by DecPOMDP researchers.

The state of the world consists of each agent's position on the grid. Therefore, with two agents and a 2x2 grid there are 16 states. There are typically no obstacles in the grid, and agents deterministically start in the top left corner. Each agent can move in any of the four cardinal directions or choose to stay still, but only successfully accomplish this action with probability 0.6 (otherwise the agent is moved randomly). If an agent moves off the grid (intentional or otherwise), than the agent will remain in the same square. To model the desire the meet, each agent receives a reward of 1 when they both are on the same grid square, and a reward of 0 otherwise. Agents deterministically observe if there is a wall to their left or right. These observations are independent of the other agent's position. While in general there are 4 observations possible per agent, in a 2x2 grid there are only 2 possible observations.

5.4.1.4 Recycling robots

Problems involving robots moving around an office attempting to pick up trash have a long history in robotics and AI [86]. A decentralized multi-agent version was proposed by C. Amato, D. S. Bernstein, and S. Zilberstein [3]. In this version a team of two robots attempt to clean up an office which is in a constant state of mess (there is always more trash to find). The office has two types of trash: small cans and large cans. The large cans require the two robots to work together in order to pick it up. Over time the robots lose power and must recharge, or else risk loosing all power and forcing an operator to manual recharge the robot (incurring a large negative reward).

This decision problem does not worry about low-level actions (such as moving). Instead, each agent must decide between three high-level actions: search for a small can, search for

a big can, or recharge. In order to successfully clean a big can both agents must be looking for it. If either agent chooses to pick up a small can, then the team gets a reward of 2 (and a reward of 4 if both agents seek small cans). If both agents cooperate and search for a big can then they receive a reward of 5. Agents can be in one of two power states, either low or high power. If agents don't recharge they have a small probability of decreasing their power. If this happens when the agent is already in low power, then they transition to high power and lose reward (someone had to recharge them).

5.4.1.5 Cooperative Box-Pushing

This problem is a simplified version of a common robotics problems [44]. Operating in a grid world, robots attempt to push a set of boxes into a goal region. There are two different sizes of boxes (large and small). Small robots can be moved by a single robot pushing against it, while large boxes require at least two robots pushing on it to move. Each robot can only observe what is immediately in front of it (empty, wall, robot, small-box, big-box). Each agent has four stochastic actions which succeed with probability 0.9 and stay put with probability 0.1: turn left, turn right, move forward and stay put. The simplest version of this problem takes place on a 4x3 grid world with two agents and three boxes. Agents attempt to move all three boxes up into the top row. When they succeed, a reward of 10 is given for small boxes and 100 for the big box. Every time-step agents also get a penalty of -0.1 along with a larger penalty of -5 if agents bump into a wall or a box they can't move.

5.4.1.6 Wireless Network

The wireless network problem [65] simulates two (or more) wireless devices attempting to use the same frequency to communicate on. Each device has a queue of packets waiting to be sent. An agent for each device attempts to minimize the number packets currently queued. Each time-step agents receive a negative reward equal to the sum of queue lengths. Agents can take two actions: send or not-send. Each device has a set of neighbors and if that device sends a packet and no neighbors send, that agent's queue is reduced by one. Otherwise no packets are sent. Each agent's queue is controlled by a Markov process with two states. When in the first "packet" state, packets are added each time-step. The second

"idle" state adds no packets. This Markov process switches from the packet to the idle state with probability 0.0741 and from the idle state to the packet state with probability 0.0470. These values were obtained from observing a physical wireless device. Agents observe their queue size and if their previous packet was sent or not. The wireless network benchmark problem used here and elsewhere has only 2 agents.

5.4.2 Methodology

We implemented the improved MaGII PBVI algorithm in Java using the GNU Linear Programming Kit to solve our linear programs. We ran the algorithm on all four benchmark problems using the dynamic belief compression approximation scheme to convert each of the DecPOMDP problems into MaGIIs. For each problem we converted them into a MaGII with one, two, thee, four, and five dynamic types.

We used the following fixed parameters while running the PBVI algorithm: $\epsilon_\Gamma = 0.0005$, $\epsilon_\pi = 0.1$, $c = 100$, $d = 36$, $k = 30$. Because of our choices for c and k we terminated the search after exploring approximately 3000 sampled beliefs. There was some small variance here due to throwing out beliefs that equaled previously sampled beliefs. All of the problems except Wireless were solved using a discount factor γ of 0.9 for the original problem and $\sqrt{0.9}$ for our dynamic approximation (recall that an agent traverses two states for every one of the original problem). Wireless has a discount rate of 0.99. Our empirical evaluations were run on a six-core 3.20 GHz Phenom processor with 8GB of memory. We terminated the algorithm and used the current value if it ran longer than a day (only Box Pushing and Wireless took longer than five hours).

5.4.3 Results

Our algorithms performed very well on all benchmark problems (Table 5.4.3). Surprisingly, most of the benchmark problems only require two approximating types in order to beat the previously best known solution. Only Dec-Tiger required three types to perform well. None of the problems benefited substantially from using four or five types. Only grid-small continued to improve slightly when given more types. This lack of improvement with extra types is strong evidence that our BB-DecPOMDP approximation is quite powerful and that

Table 4: Utility achieved by our PBVI-BB-DecPOMDP algorithm compared to the previously best known policies on a series of standard benchmarks. Higher is better. Our algorithm beats all previous results except on Dec-Tiger where we believe an optimal policy has already been found.

	$\|S\|$	$\|A_i\|$	$\|O_i\|$	Previous Best Utility	1-Type		2-Types	
					Utility	$\|\Gamma\|$	Utility	$\|\Gamma\|$
Dec-Tiger	2	2	2	13.4486 [66]	-20.000	2	4.6161	187
Broadcast	4	2	2	9.1 [3]	9.2710	36	9.2710	44
Recycling	4	3	2	31.92865 [3]	26.3158	8	31.9291	13
Grid small	16	5	2	6.89 [66]	5.2716	168	6.8423	206
Box Pushing	100	4	5	149.854 [4]	127.1572	258	223.8674	357
Wireless	64	2	6	-175.40 [45]	-208.0437	99	-167.1025	374

	3-Types		4-Types		5-Types	
	Utility	$\|\Gamma\|$	Utility	$\|\Gamma\|$	Utility	$\|\Gamma\|$
Dec-Tiger	13.4486	231	13.4486	801	13.4486	809
Broadcast	9.2710	75	9.2710	33	9.2710	123
Recycling	31.9291	37	31.9291	498	31.9291	850
Grid small	6.9826	276	6.9896	358	6.9958	693
Box Pushing	224.1387	305	-	-	-	-

the policies found are near optimal. It also suggests that these problems do not have terribly complicated optimal policies and new benchmark problems should be proposed that require a richer belief set.

The belief-POMDP state-space size is the primary bottleneck of our algorithm. Recall that this state-space is factored causing its size to be $O(|S||O|^n|t|^n)$. This number can easily become intractably large for problems with a moderate number of states and observations, such as the Mars Rovers problem. Taking advantage of sparsity can mitigate this problem (our implementation uses sparse vectors), however value-vectors tend to be dense and thus sparsity is only a partial solution. A large state-space also requires a greater number of belief samples to adequately cover and represent the value-function; with more states it becomes increasingly likely that a random walk will fail to traverse a desirable region of the state-space. This problem is not nearly as bad as it would be for a normal POMDP because much of the belief-space is unreachable and a belief-POMDP's value function has a great deal of symmetry due to the label invariance of beliefs (a relabeling of beliefs will still have the same utility).

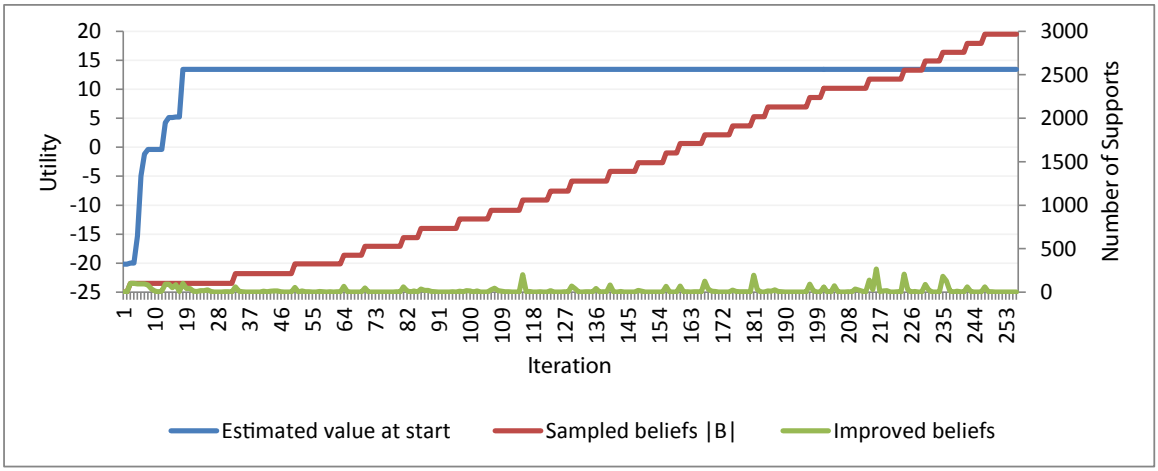

Figure 30: The starting belief value over for the Dec-Tiger problem with 3-types. The algorithm converges quickly for the optimal policy, however off-policy beliefs keep getting added and updated over time, growing the number of supports needed to represent the value function.

5.5 Limitations

A Common-payoff MaGII is an approximation of a DecPOMDP. Solving a DecPOMDP exactly is NP-complete [9]. Despite DecPOMDPs being hard to solve exactly, they are very amenable to approximation. However, while our belief compression scheme in Section 5.3.4 is optimal, it is still possible for the number of beliefs to be insufficient to produce a quality policy. The number of distinct policies used at any time-step can be no larger than the number of beliefs. This means that for problems where there is a large difference between using an optimal and sub-optimal policy at any given time-step there will be a significant loss of utility if an agent uses too few beliefs.

For example, consider a problem where an agent's action must be to repeat back an observed password. If the number of beliefs (m) is less than the number of observations, then an agent will be forced to merge observations and will not be able to distinguish between them. Without being able to distinguish between observations, the agent will not be able to take different actions, and will therefore only be able to correctly remember up to m passwords. This problem in not limited to a single time-step. There could be delays between when the password is given, and when a response is expected and the number of distinct responses will still be m. If the agent is expected to react to the environment in

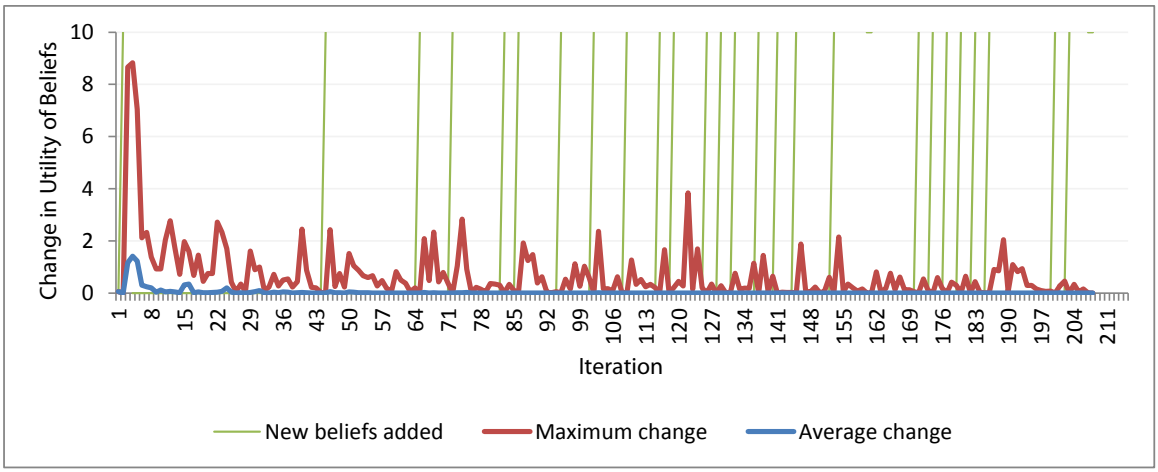

Figure 31: The rate of belief improvement over time for the the Dec-Tiger problem with 3-types. Both the maximum value improvement (over all sampled beliefs) and average value improvement are shown. New beliefs are added whenever the maximum change dips below the ϵ_Γ threshold. After the initial convergence (Figure 30) there is little average change to the beliefs.

the meantime, then the agent will have to forget even more passwords in order use those beliefs to react instead of distinguishing between passwords. In fact, if an agent wishes to remember k_1 passwords but requires k_2 beliefs to operate between observing the password and recalling it, the agent will need $k_1 \times k_2$ beliefs.

Loss of utility due to an insufficient number of beliefs can be mitigated in problems with a reward structure beyond simply getting the password correct or not, for example perhaps the agent could respond with "I don't know" and be penalized far less. The agent would then be able to group all cases where he forgets the password into a single belief and take this safety action. This dramatically lowers the cost of having too few beliefs.

The potential need for a greater distinctions between histories becomes even greater the more possible underlying states there are in the environment. Actions can be arbitrarily bad if the environment is in one state, while arbitrarily good in another state. While the common-knowledge distribution across states can help mitigate this issue for some problems, other problems may not give any common-knowledge information. In general, problems where it is crucial to take exactly the right action (among many) in exactly the right state (among many) will need a larger number of beliefs.

5.6 Conclusion

This chapter used common-payoff Markov games of incomplete information (MaGII) to tackle the problem of solving DecPOMDPs. We presented several contributions which when combined produced a value iteration algorithm for DecPOMDPs that outperforms all existing algorithms on a suite of infinite horizon benchmark problems. The contributions of this chapter are made possible by utilizing the theory of MaGII presented in the previous chapter. We first define a common-payoff MaGII and show that they can be converted into POMDPs. Given this insight we present how an existing point-based value iteration algorithm for POMDPs can be easily modified to solve common-payoff MaGIIs.

One of the major accomplishments presented in this chapter is the ability to convert a DecPOMDP into a common-payoff MaGII using optimal dynamic belief compression. This compression is achieved by modifying the game itself thereby placing the onus of belief compression on our point-based value iteration algorithm instead of a separate belief compression algorithm. Along with the ability to efficiently approximate any DecPOMDP as a MaGII, we also present several algorithmic improvements to the point-based value iteration algorithm due to the special nature of the multi-agent problem. These include: identifying significant symmetry in the value function of the approximated DecPOMDP, improving the method of belief sampling, and reducing the number full backups needed each iteration. While we described and tested point-based value iteration, we believe the advancements of this chapter opens the door towards a large and fruitful line of research into modifying and adapting existing value-based POMDP algorithms towards the specific difficulties of belief-POMDPs.

Algorithm 7 Improved MaGII PBVI algorithm

Inputs: common-payoff MaGII $M = \langle N, A, O, \Omega, S, P, R, s^{(0)} \rangle$
stopping criterion function $\epsilon_\Gamma : \mathbb{N} \to \mathbb{R}$
random exploration probability function $\epsilon_\pi : \mathbb{N} \to [0, 1]$
number of samples function $c : \mathbb{N} \to \mathbb{N}$
sampling depth function $d : \mathbb{N} \to \mathbb{N}$
sampling iterations $k_{\max}$
Output: value function Γ

$\alpha_v^0 \Leftarrow \langle R_{\min}, \cdots, R_{\min} \rangle$
$\alpha_\pi^0 \Leftarrow$ random strategy
$\Gamma' \Leftarrow \{\alpha^0\}$
$B^\forall \Leftarrow \emptyset$
for $k = 1 \to k_{\max}$ **do**
  $B^+ \Leftarrow$ `Improved_belief_sampling`$(M, c, d(k), \Gamma, \epsilon_\pi(k))$ (Algorithm 6)
  $B^\forall \Leftarrow B^\forall \bigcup B^+$
  repeat
    $B \Leftarrow B^\forall$
    $\Gamma \Leftarrow \Gamma'$
    $\Gamma' \Leftarrow \emptyset$
    while $B \neq \emptyset$ **do**
      $b \Leftarrow \text{Random}(b \in B)$
      $\alpha \Leftarrow \Gamma(b)$
      for $\alpha' \in \Gamma$ sorted in descending order by: $\alpha \cdot \Gamma(b)_\pi(b)$ {the successor belief} **do**
        $\alpha_v'^b \Leftarrow$ value of linear program 23
        if $\alpha_v'^b > \alpha_v$ **then**
          $\alpha_v \Leftarrow \alpha_v'^b$
          $\alpha_\pi \Leftarrow$ optimal point of linear program 23
          break
      $\Gamma' \Leftarrow \Gamma' \bigcup \alpha$
      for all $b \in B$ **do**
        if $\Gamma'(b)_v > \Gamma(b)_v$ **then**
          $B \Leftarrow B/b$
    repeat
      $\Gamma'' \Leftarrow \Gamma'$
      for all $\alpha \in \Gamma'$ **do**
        $\alpha_v = \max_{\alpha' \in \Gamma} V_{\alpha_\pi, \alpha'}(b)$ {light backup using previous best action}
    until $\Gamma'' - \Gamma' < \epsilon_\Gamma(k)$
  until $\Gamma' - \Gamma < \epsilon_\Gamma(k)$
return Γ

REFERENCES

[1] Abreu, D., "On the theory of infinitely repeated games with discounting," *Econometrica*, vol. 56, no. 2, pp. 383–96, 1988.

[2] Amato, C., Bernstein, D. S., and Zilberstein, S., "Optimal fixed-size controllers for decentralized POMDPs," in *Proceedings of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, (Hakodate, Japan), pp. 61–71, 2006.

[3] Amato, C., Bernstein, D. S., and Zilberstein, S., "Optimizing memory-bounded controllers for decentralized POMDPs," in *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, (Vancouver, British Columbia), pp. 1–8, 2007.

[4] Amato, C. and Zilberstein, S., "Achieving goals in decentralized pomdps," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 593–600, International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[5] AMIR, R., "Stochastic games in economics and related fields: an overview," CORE Discussion Papers 2001060, Dec. 2001.

[6] Astrm, K., "Optimal control of markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[7] Aumann, R. J., "Correlated equilibrium as an expression of bayesian rationality," *Econometrica*, vol. 55, pp. 1–18, January 1987.

[8] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H., "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, vol. 22, pp. 469–483, 1995.

[9] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S., "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[10] Bernstein, D. S., Hansen, E. A., and Zilberstein, S., "Bounded policy iteration for decentralized POMDPs," in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, (Edinburgh, Scotland), pp. 1287–1292, 2005.

[11] Boutilier, C. and Poole, D., "Computing optimal policies for partially observable decision processes using compact representations," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 1168–1175, 1996.

[12] Branke, J., Deb, K., Miettinen, K., and Steuer, R. E., eds., *Practical Approaches to Multi-Objective Optimization, 7.-12. November 2004*, vol. 04461 of *Dagstuhl Seminar Proceedings*, Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany IBFI, Schloss Dagstuhl, Germany, 2005.

[13] Burkov, A. and Chaib-draa, B., "An approximate subgame-perfect equilibrium computation technique for repeated games," *CoRR*, vol. abs/1002.1718, 2010.

[14] Cassandra, A. R., Kaelbling, L. P., and Littman, M. L., "Acting optimally in partially observable stochastic domains," tech. rep., Providence, RI, USA, 1994.

[15] Cassandra, A. R., *Exact and approximate algorithms for partially observable markov decision processes*. PhD thesis, Providence, RI, USA, 1998. AAI9830418.

[16] Chan, T. M., "Faster core-set constructions and data stream algorithms in fixed dimensions," in *Comput. Geom. Theory Appl*, pp. 152–159, 2003.

[17] Chen, L., "New analysis of the sphere covering problems and optimal polytope approximation of convex bodies," *J. Approx. Theory*, vol. 133, no. 1, pp. 134–145, 2005.

[18] Clarkson, K. L., "Algorithms for polytope covering and approximation, and for approximate closest-point queries," 1993.

[19] Cronshaw, M. B., "Algorithms for finding repeated game equilibria," *Computational Economics*, vol. 10, pp. 139–68, May 1997.

[20] Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H., "The complexity of computing a nash equilibrium," in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pp. 71–78, ACM, 2006.

[21] Deb, K., "Multi-objective optimization," *Multi-objective optimization using evolutionary algorithms*, pp. 13–46, 2001.

[22] Doshi, P. and Gmytrasiewicz, P., "Monte Carlo sampling methods for approximating interactive POMDPs," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, pp. 297–337, 2009.

[23] Emery-Montemerlo, R., Gordon, G., Schneider, J., and Thrun, S., "Approximate solutions for partially observable stochastic games with common payoffs," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 136–143, IEEE Computer Society, 2004.

[24] Emery-Montemerlo, R., Gordon, G., Schneider, J., and Thrun, S., "Game theoretic control for robot teams," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 1163–1169, IEEE, 2005.

[25] Forges, F., "Correlated equilibrium in games with incomplete information revisited," *Theory and decision*, vol. 61, no. 4, pp. 329–344, 2006.

[26] Gilpin, A. and Sandholm, T., "Solving two-person zero-sum repeated games of incomplete information," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 903–910, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[27] Gordon, G. J., "Agendas for multi-agent learning," *Artificial Intelligence*, vol. 171, no. 7, pp. 392 – 401, 2007.

[28] Greenwald, A. and Hall, K., "Correlated-q learning," in *Proc. 20th International Conference on Machine Learning (ICML)*, pp. 242–249, 2003.

[29] Hansen, E. A., Bernstein, D. S., and Zilberstein, S., "Dynamic programming for partially observable stochastic games," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, (San Jose, California), pp. 709–715, 2004.

[30] Hansen, E. A. and Feng, Z., "Dynamic programming for pomdps using a factored state representation.," in *AIPS*, pp. 130–139, 2000.

[31] Horner, J., Sugaya, T., Takahashi, S., and Vieille, N., "Recursivemethods in discounted stochastic games: an algorithm for $\delta \to 1$ and a folk theorem," *Econometrica*, 2010.

[32] Houba, H. and Bolt, W., *Credible Threats in Negotiations*, vol. 32. Springer, 2002.

[33] Hu, J. and Wellman, M., "Nash q-learning for general-sum stochastic games," in *Journal of Machine Learning Research 4:1039-1069.*, 2003.

[34] Hu, J. and Wellman, M. P., "Multiagent reinforcement learning: theoretical framework and an algorithm," in *Proc. 15th International Conf. on Machine Learning (ICML)*, pp. 242–250, 1998.

[35] Jiang, A. X., Leyton-Brown, K., and Bhat, N. A., "Action-graph games," *Games and Economic Behavior*, vol. 71, no. 1, pp. 141–173, 2011.

[36] John F. Nash, J., "The bargaining problem," *Econometrica*, vol. 18, no. 2, pp. 155–162, 1950.

[37] Jr., J. F. N., "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 36, pp. 48–49, January 1950.

[38] Judd, K. L., Yeltekin, S., and Conklin, J., "Computing supergame equilibria," *Econometrica*, vol. 71, no. 4, pp. 1239–1254, 2003.

[39] Kaelbling, L., Littman, M., and Moore, A., "Reinforcement learning: A survey," *Arxiv preprint cs/9605103*, 1996.

[40] Kalai, E., "Proportional solutions to bargaining situations : Interpersonal utility comparison," in *Econometrica: Journal of the Econometric Society*, vol. 45, pp. 1623–1630, 1977.

[41] Kalai, E. and Smorodinsky, M., "Other solutions to nash's bargaining problem," *Econometrica: Journal of the Econometric Society*, pp. 513–518, 1975.

[42] Kar, A., Ray, I., and Serrano, R., "A difficulty in implementing correlated equilibrium distributions," *Games and Economic Behavior*, vol. 69, no. 1, pp. 189–193, 2010.

[43] Kolter, J. and Ng, A., "Near-Bayesian exploration in polynomial time," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 513–520, ACM, 2009.

[44] KUBE, C. R. and ZHANG, H., "Task modelling in collective robotics," *Autonomous Robots*, vol. 4, pp. 53–72, 1997.

[45] Kumar, A. and Zilberstein, S., "Anytime planning for decentralized pomdps using expectation maximization,"

[46] Letchford, J. and Conitzer, V., "Computing optimal strategies to commit to in extensive-form games," in *Proceedings of the 11th ACM conference on Electronic commerce*, pp. 83–92, ACM, 2010.

[47] Letchford, J., MacDermed, L., Conitzer, V., Parr, R., and Isbell, C. L., "Computing optimal strategies to commit to in stochastic games.," in *AAAI*, 2012.

[48] Littman, M. L., "Markov games as a framework for multi-agent reinforcement learning," in *Proc. 11th International Conf. on Machine Learning (ICML)*, pp. 157–163, Morgan Kaufmann, 1994.

[49] Littman, M. L., "Friend-or-foe Q-learning in general-sum games," in *Proc. 18th International Conf. on Machine Learning (ICML)*, pp. 322–328, 2001.

[50] Lopez, M. A. and Reisner, S., "Linear time approximation of 3d convex polytopes," *Comput. Geom. Theory Appl.*, vol. 23, no. 3, pp. 291–301, 2002.

[51] Mac Dermed, L. and Isbell, C., "Solving stochastic games," in *Advances in Neural Information Processing Systems 22*, pp. 1186–1194, 2009.

[52] MacDermed, L., Narayan, K. S., Isbell, C. L., and Weiss, L., "Quick polytope approximation of all correlated equilibria in stochastic games," in *Twenty Fifth AAAI Conference on Artificial Intelligence (AAAI '11)*, 2010.

[53] Monahan, G. E., "A survey of partially observable markov decision processes: theory, models, and algorithms," *Manage. Sci.*, vol. 28, no. 1, pp. 1–16, 1982.

[54] Murray, C. and Gordon, G., "Finding correlated equilibria in general sum stochastic games," tech. rep., School of Computer Science, Carnegie Mellon University, 2007.

[55] Murray, C. and Gordon, G. J., "Multi-robot negotiation: Approximating the set of subgame perfect equilibria in general-sum stochastic games," in *Advances in Neural Information Processing Systems 19*, pp. 1001–1008, 2007.

[56] Myerson, R. B., *Game Theory: Analysis of Conflict.* Harvard University Press, Cambridge, 1991.

[57] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., and Marsella, S., "Taming decentralized pomdps: towards efficient policy computation for multiagent settings," in *Proceedings of the 18th international joint conference on Artificial intelligence*, IJCAI'03, (San Francisco, CA, USA), pp. 705–711, Morgan Kaufmann Publishers Inc., 2003.

[58] Nguyen, K. C., Alpcan, T., and Basar, T., "Stochastic games for security in networks with interdependent nodes," *CoRR*, vol. abs/1003.2440, 2010.

[59] Oliehoek, F., *Value-based planning for teams of agents in stochastic partially observable environments.* Vossiupers UvA, 2010.

[60] Oliehoek, F. A., Spaan, M. T., and Vlassis, N., "Optimal and approximate q-value functions for decentralized pomdps," *Journal of Artificial Intelligence Research*, vol. 32, no. 1, pp. 289–353, 2008.

[61] Oliehoek, F. A. and Vlassis, N., "Q-value functions for decentralized pomdps," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, p. 220, ACM, 2007.

[62] Oliehoek, F. A., Whiteson, S., and Spaan, M. T., "Lossless clustering of histories in decentralized pomdps," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 577–584, International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[63] Ooi, J. M. and Wornell, G. W., "Decentralized control of a multiple access broadcast channel: Performance bounds," in *Decision and Control, 1996., Proceedings of the 35th IEEE*, vol. 1, pp. 293–298, IEEE, 1996.

[64] Pajarinen, J. and others, "Planning under uncertainty for large-scale problems with applications to wireless networking," 2013.

[65] Pajarinen, J. and Peltonen, J., "Efficient planning for factored infinite-horizon dec-pomdps," in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pp. 325–331, AAAI Press, 2011.

[66] Pajarinen, J. and Peltonen, J., "Periodic finite state controllers for efficient pomdp and dec-pomdp planning," in *Proc. of the 25th Annual Conf. on Neural Information Processing Systems*, 2011.

[67] Panait, L. and Luke, S., "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005.

[68] Papadimitriou, C. H. and Tsitsiklis, J., "On the complexity of designing distributed protocols," *Information and Control*, vol. 53, no. 3, pp. 211–218, 1982.

[69] Paruchuri, P., Pearce, J. P., Marecki, J., Tambe, M., Ordonez, F., and Kraus, S., "Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 895–902, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[70] Pineau, J., Gordon, G., and Thrun, S., "Point-based value iteration: An anytime algorithm for pomdps," in *International joint conference on artificial intelligence*, vol. 18, pp. 1025–1032, 2003.

[71] Pineau, J., Gordon, G., and Thrun, S., "Point-based value iteration: an anytime algorithm for pomdps," in *Proceedings of the 18th international joint conference on Artificial intelligence*, IJCAI'03, (San Francisco, CA, USA), pp. 1025–1030, Morgan Kaufmann Publishers Inc., 2003.

[72] Pita, J., Jain, M., Ordónez, F., Portway, C., Tambe, M., Western, C., Paruchuri, P., and Kraus, S., "Using game theory for los angeles airport security," *AI Magazine*, vol. 30, no. 1, p. 43, 2009.

[73] Roth, M., Simmons, R., and Veloso, M., "Reasoning about joint beliefs for execution-time communication decisions," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 786–793, ACM, 2005.

[74] Rubinstein, A., "Perfect equilibrium in a bargaining model," *Econometrica*, vol. 50, no. 1, pp. 97–109, 1982.

[75] Rubinstein, A., "A bargaining model with incomplete information about time preferences," *Econometrica: Journal of the Econometric Society*, pp. 1151–1172, 1985.

[76] Sandholm, T., "The State of Solving Large Incomplete-Information Games, and Application to Poker," *AI Magazine*, vol. 31, no. 4, pp. 13–32, 2010.

[77] Schneider, R., *Convex bodies: the Brunn-Minkowski theory*, vol. 44. Cambridge University Press, 1993.

[78] Seuken, S. and Zilberstein, S., "Memory-bounded dynamic programming for dec-pomdps," in *Proceedings of the 20th international joint conference on Artifical intelligence*, (San Francisco, CA, USA), pp. 2009–2015, Morgan Kaufmann Publishers Inc., 2007.

[79] Shachter, R. D., "Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams," in *In Uncertainty in Artificial Intelligence*, pp. 480–487, Morgan Kaufmann, 1998.

[80] Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., and Meyer, G., "Protect: A deployed game theoretic system to protect the ports of the united states," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 13–20, International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[81] Shoham, Yoav, P. and Grenager, "If multi-agent learning is the answer, what is the question?," in *Artificial Intelligence*, 2006.

[82] Shoham, Y. and Leyton-Brown, K., *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2009.

[83] Shoham, Y., Powers, R., and Grenager, T., "Multi-agent reinforcement learning: a critical survey," tech. rep., 2003.

[84] Spaan, M. T. and Vlassis, N., "Perseus: Randomized point-based value iteration for pomdps," *Journal of artificial intelligence research*, vol. 24, no. 1, pp. 195–220, 2005.

[85] Steuer, R. E., "Adbase: A multiple objective linear programming solver for efficient extreme points and unbounded efficient edges," 2006.

[86] Sutton, R. S. and Barto, A. G., *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.

[87] Szer, D., Charpillet, F., Zilberstein, S., and others, "Maa*: A heuristic search algorithm for solving decentralized pomdps," in *21st Conference on Uncertainty in Artificial Intelligence-UAI'2005*, 2005.

[88] Tsai, J., Kiekintveld, C., Ordonez, F., Tambe, M., and Rathi, S., "Iris-a tool for strategic security allocation in transportation networks," 2009.

[89] Von Neumann, J. and Morgenstern, O., "The theory of games and economic behavior," 1947.

[90] Von Stengel, B. and Zamir, S., "Leadership games with convex strategy sets," *Games and Economic Behavior*, vol. 69, no. 2, pp. 446–457, 2010.

[91] Wolf, D. M. and Arkin, A. P., " Motifs, modules and games in bacteria.," *Current Opinion in Microbiology*, vol. 6, p. 125134, 2003.

[92] Zettlemoyer, L. S., Milch, B., and Kaelbling, L. P., "Multi-agent filtering with infinitely nested beliefs," *Advances in Neural Information Processing Systems 22*, 2009.

[93] Zhang, X., Aberdeen, D., and Vishwanathan, S., "Conditional random fields for multi-agent reinforcement learning," in *Proceedings of the 24th international conference on Machine learning*, pp. 1143–1150, ACM, 2007.

[94] Zinkevich, M., Greenwald, A., and Littman, M. L., "Cyclic equilibria in markov games," in *Proceedings of Neural Information Processing Systems*, (Vancouver, BC, Canada), 2005.