Multi-Robot Negotiation: Approximating the Set of Subgame Perfect Equilibria in General-Sum Stochastic Games

Chris Murray and Geoffrey Gordon

October 2006
CMU-ML-06-114

Multi-Robot Negotiation: Approximating the Set of Subgame Perfect Equilibria in General-Sum Stochastic Games

Chris Murray and Geoff Gordon

October 2006
CMU-ML-06-114

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In real-world planning problems, we must reason not only about our own goals, but about the goals of other agents with which we may interact. Often these agents' goals are neither completely aligned with our own nor directly opposed to them. Instead there are opportunities for cooperation: by joining forces, the agents can all achieve higher utility than they could separately. But, in order to cooperate, the agents must negotiate a mutually acceptable plan from among the many possible ones, and each agent must trust that the others will follow their parts of the deal. Research in multi-agent planning has often avoided the problem of making sure that all agents have an incentive to follow a proposed joint plan. On the other hand, while game theoretic algorithms handle incentives correctly, they often don't scale to large planning problems. In this paper we attempt to bridge the gap between these two lines of research: we present an efficient game-theoretic approximate planning algorithm, along with a negotiation protocol which encourages agents to compute and agree on joint plans that are fair and optimal in a sense defined below. We demonstrate our algorithm and protocol on two simple robotic planning problems.

Keywords: Stochastic Games, subgame-perfect Nash equilibria, Q Learning.

1 INTRODUCTION

We model the multi-agent planning problem as a general-sum stochastic game with cheap talk: the agents observe the state of the world, discuss their plans with each other, and then simultaneously select their actions. The state and actions determine a one-step reward for each player and a distribution over the world's next state, and the process repeats.

While talking allows the agents to coordinate their actions, it cannot by itself solve the problem of trust: the agents might lie or make false promises. So, we are interested in planning algorithms that find *subgame-perfect Nash equilibria*. In a subgame-perfect equilibrium, every deviation from the plan is deterred by the threat of a suitable punishment, and every threatened punishment is believable. To find these equilibria, planners must reason about their own and other agents' incentives to deviate: if other agents have incentives to deviate then I can't trust them, while if I have an incentive to deviate, they can't trust me.

In a given game there may be many subgame-perfect equilibria with widely differing payoffs: some will be better for some agents, and others will be better for other agents. It is generally not feasible to compute all equilibria [1], and even if it were, there would be no obvious way to select one to implement. It does not make sense for the agents to select an equilibrium without consulting one another: there is no reason that agent A's part of one joint plan would be compatible with agent B's part of another joint plan. Instead the agents must negotiate, computing and proposing equilibria until they find one which is acceptable to all parties.

This paper describes a planning algorithm and a negotiation protocol which work together to ensure that the agents compute and select a subgame-perfect Nash equilibrium which is both approximately *Pareto-optimal* (that is, its value to any single agent cannot be improved very much without lowering the value to another another agent) and approximately *fair* (that is, near the so-called *Nash bargaining point*). Neither the algorithm nor the protocol is guaranteed to work in all games; however, they are guaranteed correct when they are applicable, and applicability is easy to check. In addition, our experiments show that they work well in some realistic situations. Together, these properties of fairness, enforceability, and Pareto optimality form a strong solution concept for a stochastic game. The use of this definition is one characteristic that distinguishes our work from previous research: ours is the first efficient algorithm that we know of to use such a strong solution concept for stochastic games.

Our planning algorithm performs dynamic programming on a set-based value function: for P players, at a state s, $V \in \mathbf{V}(s) \subset \mathbb{R}^P$ is an estimate of the value the players can achieve. We represent $\mathbf{V}(s)$ by sampling points on its convex hull. This representation is *conservative*, i.e., guarantees that we find a subset of the true $\mathbf{V}^*(s)$. Based on the sampled points we can efficiently compute one-step backups by checking which joint actions are enforceable in an equilibrium.

Our negotiation protocol is based on a multi-player version of Rubinstein's bargaining game. Players together enumerate a set of equilibria, and then take turns proposing an equilibrium from the set. Until the players agree, the protocol ends with a small probability ϵ after each step and defaults to a low-payoff equilibrium; the fear of this outcome forces players to make reasonable offers.

2 BACKGROUND

2.1 STOCHASTIC GAMES

A stochastic game represents a multi-agent planning problem in the same way that a Markov Decision Process [2] represents a single-agent planning problem. As in an MDP, transitions in a stochastic game depend on the current state and action. Unlike MDPs, the current (joint) action is a vector of individual actions, one for each player. More formally, a general-sum stochastic game G is a tuple $(S, s_{\text{start}}, P, A, T, R, \gamma)$. S is a set of states, and $s_{\text{start}} \in S$ is the start state. P is the number of players. $A = A_1 \times A_2 \times \ldots \times A_P$ is the finite set of joint actions. We deal with fully observable stochastic games with perfect monitoring, where all players can observe previous joint actions. $T : S \times A \mapsto P(S)$ is the transition function, where $P(S)$ is the set of probability distributions over S. $R : S \times A \mapsto \mathbb{R}^P$ is the reward function. We will write $R_p(s, a)$ for the pth component of $\mathbf{R}(s, a)$. $\gamma \in [0, 1)$ is the discount factor. Player p wants to maximize her *discounted total value* for the observed sequence of states and joint actions $s_1, a_1, s_2, a_2, \ldots$, $V_p = \sum_{t=1}^{\infty} \gamma^{t-1} R_p(s_t, a_t)$. A stationary policy for player p is a function $\pi_p : S \mapsto P(A_p)$. A stationary joint policy is a vector of policies $\pi = (\pi_1, \ldots, \pi_P)$, one for each player. A nonstationary policy for player p is a function $\pi_p : (\cup_{t=0}^{\infty} (S \times A)^t \times S) \mapsto P(A_p)$ which takes a history of states and joint actions and produces a distribution over player p's actions; we can define a nonstationary joint policy analogously. For any nonstationary joint policy, there is a stationary policy that achieves the same value at every state [3].

The value function $V_p^\pi : S \mapsto \mathbb{R}$ gives expected values for player p under joint policy π. The *value vector* at state s, $\mathbf{V}^\pi(s)$, is the vector with components $V_p^\pi(s)$. (For a nonstationary policy π we will define $V_p^\pi(s)$ to be the value if s were the start state, and $V_p^\pi(h)$ to be the value after observing history h.) A vector $\mathbf{V}$ is *feasible* at state s if there is a π for which $\mathbf{V}^\pi(s) = \mathbf{V}$, and we will say that π *achieves* $\mathbf{V}$.

We will assume *public randomization*: the agents can sample from a desired joint action distribution in such a way that everyone can verify the outcome. If public randomization is not directly available, there are cryptographic protocols which can simulate it [4]. This assumption means that the set of feasible value vectors is convex, since we can roll a die at the first time step to choose from a set of feasible policies.

2.2 REPEATED BATTLE OF THE SEXES

One well-known stochastic game that can illustrate many of the concepts we're presented is called *Repeated Battle of the Sexes* or *RBoS.* The shaded area in Fig. 1 illustrates the set of feasible value vectors for this game, which has one state, two players, and two actions for each player, with discount factor $\gamma = 0.99$ and reward function

$$\begin{array}{c|cc} & a_1 & a_2 \\ \hline a_1 & 3,4 & 0,0 \\ a_2 & 0,0 & 4,3 \end{array} \qquad (1)$$

In Eq. 1, the first player's action determines a row of the table and the second player's action determines a column. The corresponding entry lists the payoffs to players one and two in that order. Stochastic games with only one state, such as RBoS, are called *repeated games.*

2.3 EQUILIBRIA

While optimal policies for MDPs can be determined exactly via various algorithms such as linear programming [2], it isn't clear what it means to find an optimal policy for a general sum stochastic game. So, rather than trying to determine a unique optimal policy, we will define a set of reasonable policies: the Pareto-dominant subgame-perfect Nash equilibria.

A (possibly nonstationary) joint policy π is a *Nash equilibrium* if, for each individual player, no unilateral deviation from the policy would increase that player's expected value for playing the game. Nash equilibria can contain *incredible threats*, that is, threats which the agents have no intention of following through on. To remove this possibility, we can define the *subgame-perfect Nash equilibria.* A policy π is a subgame-perfect Nash equilibrium if it is a Nash equilibrium in every possible subgame: that is, if there is no incentive for any player to deviate after observing any history of joint actions.

Finally, consider two policies π and ϕ. If $V_p^\pi(s_{\text{start}}) \geq V_p^\phi(s_{\text{start}})$ for all players p, and if $V_p^\pi(s_{\text{start}}) > V_p^\phi(s_{\text{start}})$ for at least one p, then we will say that π *Pareto dominates* ϕ. A policy which is not Pareto dominated by any other policy is *Pareto optimal.*

RBoS has three stationary subgame-perfect Nash equilibria, whose value vectors are indicated with $\circ$ in Fig. 1.[1] The equilibrium marked with both $\circ$ and $\times$ is Pareto dominated by the other two equilibria (marked with $\circ$ only), but neither of the latter two equilibria dominates the other. The top right border of the feasible set (red where color is available) corresponds to the set of Pareto optimal policies.

[1]These equilibria are: always play a_1, a_1; always play a_2, a_2; and randomize with $P(a_1) = \frac{3}{7}$ for player 1 and $P(a_1) = \frac{4}{7}$ for player 2

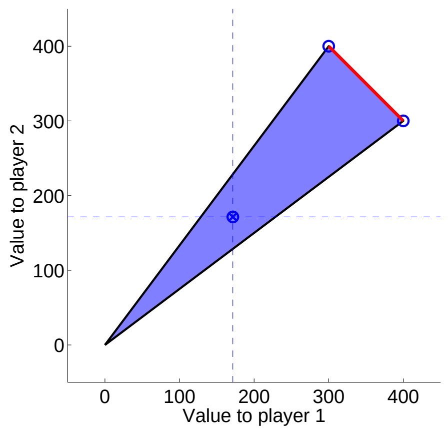

Figure 1: Illustration of feasible values, safety values, equilibria, and the folk theorem for RBoS.

2.4 RELATED WORK

Littman and Stone [5] give an algorithm for finding Nash equilibria in two-player repeated games. Hansen et al. [6] show how to eliminate very-weakly-dominated strategies in partially observable stochastic games. Doraszelski and Judd [7] show how to compute Markov perfect equilibria in continuous-time stochastic games. The above papers use solution concepts much weaker than Pareto-dominant subgame-perfect equilibrium, and do not address negotiation and coordination. Perhaps the closest work to the current paper is by Brafman and Tennenholtz [8]: they present learning algorithms which, in repeated self-play, find Pareto-dominant (but not subgame-perfect) Nash equilibria in matrix and stochastic games. By contrast, we consider a single play of our game, but allow “cheap talk” beforehand. And, our protocol encourages arbitrary algorithms to agree on Pareto-dominant equilibria, while their result depends strongly on the self-play assumption.

2.4.1 FOLK THEOREMS

In any game, each player can guarantee herself an expected discounted value regardless of what actions the other players takes. We call this value the *safety value*. Suppose that there is a stationary subgame-perfect equilibrium which achieves the safety value for both players; call this the safety equilibrium policy.

Suppose that, in a repeated game, some stationary policy π is better for both players than the safety equilibrium policy. Then we can build a subgame-perfect equilibrium with the same payoff as π: start playing π, and if someone deviates, switch to the safety equilibrium policy. So long as γ is sufficiently large, no rational player will want to deviate. This is the *folk theorem for repeated*

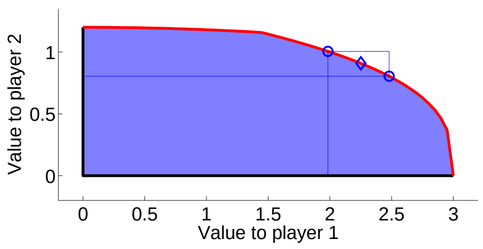

Figure 2: Equilibria of a Rubinstein game with $\gamma = 0.8$. Shaded area shows feasible value vectors $(U_1(x), U_2(x))$ for outcomes x. Right-hand circle corresponds to equilibrium when player 1 moves first, left-hand circle, when player 2 moves first. Nash point is indicated by $\diamond$.

games: any feasible value vector which is strictly better than the safety values corresponds to a subgame-perfect Nash equilibrium [9]. (The proof is slightly more complicated if there is no safety equilibrium policy, but the theorem holds for any repeated game.)

There is also a folk theorem for general stochastic games [3]. This theorem, while useful, is not strong enough for our purposes: it only covers discount factors γ which are so close to 1 that the players don't care which state they wind up in after a possible deviation. In most practical stochastic games, discount factors this high are unreasonably patient. When γ is significantly less than 1, the set of equilibrium vectors can change in strange ways as we change γ [10].

In RBoS, each player can guarantee herself an expected reward of $\min\{\frac{4}{7} \cdot 3, \frac{3}{7} \cdot 4\} = \frac{12}{7}$ on each step. This level of reward is the *safety value* (dashed lines in Fig. 1). It happens that there is a stationary subgame-perfect equilibrium which achieves the safety value for both players; this is the safety equilibrium policy for *RBoS*. This disagreement policy is for each player to play her less preferred action $\frac{4}{7}$ of the time and her more preferred action $\frac{3}{7}$ of the time.

2.4.2 RUBINSTEIN'S GAME

Rubinstein [11] considered a game where two players divide a slice of pie. The first player offers a division $x, 1-x$ to the second; the second player either accepts the division, or refuses and offers her own division $1 - y, y$. The game repeats until some player accepts an offer or until either player gives up. In the latter case neither player gets any pie. Rubinstein showed that if player p's utility for receiving a fraction x at time t is $U_p(x, t) = \gamma^t U_p(x)$ for a discount factor $0 \le \gamma < 1$ and an appropriate time-independent utility function $U_p(x) \ge 0$, then rational players will agree on a division near the so-called *Nash bargaining point.* This is the point which maximizes the product of the utilities that the players gain by cooperating, $U_1(x)U_2(1 - x)$. As $\gamma \uparrow 1$, the equilibrium will approach the Nash point. See Fig. 2 for an illustration. For three or more players, a similar result holds where agents take turns proposing multi-way divisions of the pie [12].

While the game above is restricted to two players, there is also a general multi-player version of the bargaining game. The multi-player version works as follows. Agents take turns proposing multi-way divisions of a pie. After each proposal, all agents other than the proposer decide independently whether to accept or reject. If all agents accept, the proposal is implemented. Otherwise, any agents who accepted have their shares fixed at the proposed level and are removed from further play; the next remaining agent then proposes a division of the remaining pie. As in the two-player game, the unique subgame-perfect equilibrium approaches the Nash point as $\gamma \uparrow 1$ [12].

2.5 NASH BARGAINING POINT

In a multi-player game, the Nash bargaining point is the solution maximizing the product of the excess values to each player above her safety value. That is,

$$\mathbf{V}^{\text{Nash}} = \arg\max_{\mathbf{V}} \left(\prod_{p=1}^{P} (V_p - V_p^{\text{safety}}) \right)$$

This *argmax* is taken only over values of $\mathbf{V}$ such that $V_p \geq V_p^{\text{safety}}$ for all p. The Nash bargaining point can be uniquely characterized as meeting some criteria for a "good" bargaining solution, such as symmetry and weak Pareto optimality. See [13] or [14] for more details.

3 NEGOTIATION PROTOCOL

The Rubinstein game implicitly assumes that the result of a failure to cooperate is known to all players: nobody gets any pie. The multi-player version of the game assumes in addition that giving one player a share of the pie doesn't force us to give a share to any other player. Neither of these properties holds for general sum stochastic games. They are, however, easy to check, and often hold or can be made to hold for planning domains of interest.

So, we will assume that the players have agreed beforehand on a subgame-perfect equilibrium policy π^{dis}, called the *disagreement policy*, that they will follow in the event of a negotiation failure. In addition, for games with three or more players, we will assume that each player can unilaterally reduce her own utility by any desired amount without affecting other players' utilities.[2]

[2]Our results for the multi-player problem also hold under the alternate assumption that utilities are transferable, by an argument due to Krishna and Serrano [12]. We prefer our stated assumption, since it does not require the players' utilities to be expressed in compatible units.

Given these assumptions, our protocol proceeds in two phases. In the first phase agents compute subgame-perfect equilibria and take turns revealing them. On an agent's turn she either reveals an equilibrium or passes; if all agents pass consecutively, the protocol proceeds to the second phase. When an agent states a policy π, the other agents verify that π is a subgame-perfect equilibrium and calculate its payoff vector $\mathbf{V}^{\pi}(s_{\text{start}})$; players who state non-equilibrium policies miss their turn (Such players are assigned to receive their disagreement utilities, as described below.)

At the end of the first phase, suppose the players have revealed a set Π of policies. Define

$$X_p(\pi) = V_p^{\pi}(s_{\text{start}}) - V_p^{\text{dis}}(s_{\text{start}})$$
$$\mathbf{U} = \text{convhull}\,\{\mathbf{X}(\pi) \mid \pi \in \Pi\}$$
$$\underline{\mathbf{U}} = \{\mathbf{u} \geq 0 \mid (\exists \mathbf{v} \in \mathbf{U} \mid \mathbf{u} \leq \mathbf{v})\}$$

where $\mathbf{V}^{\text{dis}}$ is the value function of π^{dis}, $X_p(\pi)$ is the excess of policy π for player p, and $\mathbf{U}$ is the set of feasible excess vectors.

In the second phase, players take turns proposing points $\mathbf{u} \in \underline{\mathbf{U}}$ along with policies or mixtures of policies in Π that achieve them. After each proposal, all agents except the proposer decide whether to accept or reject. If everyone accepts, the proposal is implemented: everyone starts executing the agreed equilibrium.

Otherwise, the players who accepted are removed from future negotiation and have their utilities fixed at the proposed levels. Fixing player p's utility at u_p means that all future proposals must give p exactly u_p. (Invalid proposals result in the proposer losing her turn.) To achieve this, the proposal may require p to voluntarily lower her own utility; this requirement is enforced by the threat that all players will revert to π^{dis} if p fails to act as required. The **choose**(i) in Figure 3 marks the place in the protocol where agent i gets to choose one of several alternatives: i picks which of the lines inside the **choose**/**end** pair will execute. The parameter ϵ is an arbitrary small positive number which determines whether we force a phase to end early; it should be small enough that there is little risk of the protocol ending before the agents want it to, but large enough that the agents feel pressure to arrive at an agreement rather than stalling forever. At the end of Phase I, the set pol contains the policies which the agents will bargain over in Phase II.

If at some point one of the remaining players declares that further negotiation is pointless, or if we hit the ϵ chance of having the current round of communication end, all remaining players are assigned their disagreement values. The players execute the last proposed policy π (or π^{dis} if there has been no valid proposal), and any player p for whom $V_p^{\pi}(s_{\text{start}})$ is greater than her assigned utility u_p voluntarily lowers her utility to the correct level. (Again, failure to do so results in all players reverting to π^{dis}.)

Under the above protocol, player's preferences are the same as in a Rubin-

```
pol ← ∅
repeat
  done ← true
  for each agent $i$
    choose($i$)
      $i$ says “pass”
      $i$ adds a policy or set of policies to pol; done ← false
    end choose
  end for
  With probability $\epsilon$, done ← true
until done
```

Figure 3: Phase I of the negotiation protocol.

stein game with utility set $\underline{\mathbf{U}}$: because we have assumed that negotiation ends with probability ϵ after each message, agreeing on $\mathbf{u}$ after t additional steps is exactly as good as agreeing on $\mathbf{u}(1-\epsilon)^{\mathrm{t}}$ now. So with ϵ sufficiently small, the Rubinstein or Krishna-Serrano results show that rational players will agree on a vector $\mathbf{u} \in \underline{\mathbf{U}}$ which is close to the Nash point $\mathrm{argmax}_{\mathrm{u} \in \underline{\mathbf{U}}} \Pi_{\mathrm{p}} \mathrm{u}_{\mathrm{p}}$. Because of this property we can give a coarse description of how the agents should play in Phase I: they should nominate policies that give themselves high payoffs to try to steer the outcome in their favor. But they will also want to nominate policies that give high payoffs to other agents, because such policies are more likely to eventually be incorporated into the plan accepted by the group as a whole.

In figures 3 and 4 we give an algorithmic description of the specific protocol followed by negotiating players.

4 COMPUTING EQUILIBRIA

In order to use the protocol of Sec. 3 for bargaining in a stochastic game, the players must be able to compute some subgame-perfect equilibria. Computing equilibria is a hard problem [15], so we cannot expect real agents to find the entire set of equilibria. Fortunately, each player will want to find the equilibria which are most advantageous to herself to influence the negotiation process in her favor. But equilibria which offer other players reasonably high reward have a higher chance of being accepted in negotiation. So, self interest will naturally distribute the computational burden among all the players.

In this section we describe an efficient dynamic-programming algorithm for computing equilibria. The algorithm takes some low-payoff equilibria as input and (usually) outputs higher-payoff equilibria. It is based on the intuition that we can use low-payoff equilibria as enforcement tools: by threatening to switch

```
for each agent $i$
  utility[$i$] $\leftarrow d_i$
  accepted[$i$] $\leftarrow$ false
end for
repeat
  for each agent $i$
    if accepted[$i$] then continue
    $i$ proposes a distribution $s$ over complete policies from pol
    done $\leftarrow$ true
    for each agent $j \neq i$
      if accepted[$j$] then continue
      $u \leftarrow$ utility of $s$ to $j$
      choose($j$)
        $j$ says "accept"; utility[$j$] $\leftarrow u$; accepted[$j$] $\leftarrow$ true
        $j$ says "reject"; done $\leftarrow$ false
      end choose
    end for
    if done then
      utility[$i$] $\leftarrow$ utility of $s$ to $i$
      return
    end if
  end for
  With probability $\epsilon$, done $\leftarrow$ true
until done
```

Figure 4: Phase II of the negotiation protocol.

```
*Initialization*
**for** $s \in S$
  $\mathbf{V}(s) \leftarrow \{\mathbf{V} \mid V_p^{\text{dis}}(s) \leq V_p \leq R_{\max}/(1-\gamma)\}$
**end**
*Repeat until converged*
**for** iteration $\leftarrow 1, 2, \ldots$
  **for** $s \in S$
    *Compute value vector set for each joint action,*
      *then throw away unenforceable vectors*
    **for** $a \in A$
      $\mathbf{Q}(s,a) \leftarrow \{\mathbf{R}(s,a)\} + \gamma \sum_{s' \in S} T(s,a)(s')\mathbf{V}(s')$
      $\mathbf{Q}(s,a) \leftarrow \{\mathbf{Q} \in \mathbf{Q}(s,a) \mid \mathbf{Q} \geq \mathbf{V}^{\text{dev}}(s,a)\}$
    **end**
    *We can now randomize among joint actions*
    $\mathbf{V}(s) \leftarrow \text{convhull} \bigcup_a \mathbf{Q}(s,a)$
  **end**
**end**
```

Figure 5: Dynamic programming using exact operations on sets of value vectors

to an equilibrium that has low value to player p, we can deter p from deviating from a cooperative policy.

In more detail, we will assume that we are given P different equilibria $\pi_1^{\text{pun}}, \ldots, \pi_P^{\text{pun}}$; we will use π_p^{pun} to punish player p if she deviates. We can set $\pi_p^{\text{pun}} = \pi^{\text{dis}}$ for all p if π^{dis} is the only equilibrium we know; or, we can use any other equilibrium policies that we happen to have discovered. The algorithm will be most effective when the value of π_p^{pun} to player p is as low as possible in all states.

We will then search for cooperative policies that we can enforce with the given threats π_p^{pun}. We will first present an algorithm which pretends that we can efficiently take direct sums and convex hulls of arbitrary sets. This algorithm is impractical, but finds all enforceable value vectors. We will then turn it into an approximate algorithm which uses finite data structures to represent the set-valued variables. As we allow more and more storage for each set, the approximate algorithm will approach the exact one; and in any case the result will be a set of equilibria which the agents can execute.

4.1 THE EXACT ALGORITHM

Our algorithm maintains a set of value vectors $\mathbf{V}(s)$ for each state s. It initializes $\mathbf{V}(s)$ to a set which we know contains the value vectors for all equilibrium policies. It then refines $\mathbf{V}$ by dynamic programming: it repeatedly attempts to improve the set of values at each state by backing up all of the joint actions, excluding joint actions from which some agent has an incentive to deviate.

In more detail, we will compute $V_p^{\mathrm{dis}}(s) \equiv V_p^{\pi_{\mathrm{dis}}}(s)$ for all s and p and use the vector $\mathbf{V}^{\mathrm{dis}}(s)$ in our initialization. (Recall that we have defined $V_p^{\pi}(s)$ for a nonstationary policy π as the value of π if s were the start state.) We also need the values of the punishment policies for their corresponding players, $V_p^{\mathrm{pun}}(s) \equiv V_p^{\pi_p^{\mathrm{pun}}}(s)$ for all p and s. Given these values, define

$$Q_p^{\mathrm{dev}}(s,a) = R_p(s,a) + \gamma \sum_{s' \in S} T(s,a)(s') V_p^{\mathrm{pun}}(s') \tag{2}$$

to be the value to player p of playing joint action a from state s and then following π_p^{pun} forever after.

From the above Q_p^{dev} values we can compute player p's value for deviating from an equilibrium which recommends action a in state s: it is $Q_p^{\mathrm{dev}}(s,a')$ for the best possible deviation a', since p will get the one-step payoff for a' but be punished by the rest of the players starting on the following time step. That is,

$$V_p^{\mathrm{dev}}(s,a) = \max_{a'_p \in A_p} Q_p^{\mathrm{dev}}(s, a_1 \times \ldots \times a'_p \times \ldots \times a_P) \tag{3}$$

$V_p^{\mathrm{dev}}(s,a)$ is the value we must achieve for player p in state s if we are planning to recommend action a and punish deviations with π_p^{pun}: if we do not achieve this value, player p would rather deviate and be punished.

Our algorithm is shown in Fig. 5. After k iterations, each vector in $\mathbf{V}(s)$ corresponds to a k-step policy in which no agent ever has an incentive to deviate. In the $k+1$st iteration, the first assignment to $\mathbf{Q}(s,a)$ computes the value of performing action a followed by any k-step policy. The second assignment throws out the pairs (a,π) for which some agent would want to deviate from a given that the agents plan to follow π in the future. And the convex hull accounts for the fact that, on reaching state s, we can select an action a and future policy π at random from the feasible pairs.[3] Proofs of convergence and correctness of the exact algorithm are in the appendix.

5 Approximate Algorithm

The exact algorithm performs operations on convex sets of value vectors. Actually storing these sets exactly may require a prohibitive amount of space, and thus a prohibitive amount of computation to perform operations on these sets. So our approximate algorithm, rather that storing $\mathbf{V}(s)$ explicitly, chooses a finite set of witness vectors $\mathbf{W} \subset \mathbb{R}^P$ and stores $\mathbf{V}(s,\mathbf{w}) = \arg\max_{\mathbf{v} \in \mathbf{V}(s)} (\mathbf{v} \cdot \mathbf{w})$ for each $\mathbf{w} \in \mathbf{W}$. $\mathbf{V}(s)$ is then approximated by the convex hull of $\{\mathbf{V}(s,\mathbf{w}) \mid$

[3]It is important for this randomization to occur *after* reaching state s to avoid introducing incentives to deviate, and it is also important for the randomization to be public.

```
*Initialization*
**for** $s \in S, w \in W$
  $V(s, \mathbf{w}) \leftarrow \mathbf{w}\, R_{\max}/(1-\gamma)$
**end**
*Repeat until converged*
**for** iteration $\leftarrow 1, 2, \ldots$
  **for** $\mathbf{w} \in W, s \in S$
    *Approximate value vector set for each joint action,*
      *then throw away unenforceable vectors*
    **for** $a \in A$
      $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a)(s')V(s', \mathbf{w})$
      **if** $Q(s,a) \not\geq V^{\text{dev}}(s,a)$
        $Q(s,a) \leftarrow V^{\text{dis}}(s)$
      **end**
    **end**
    *Approximate the convex hull*
    $V(s, \mathbf{w}) \leftarrow \arg\max_{\mathbf{q} \in \{Q(s,a) | a \in A\}} \mathbf{q} \cdot \mathbf{w}$
  **end**
**end**
```

Figure 6: Dynamic programming using approximate operations on arrays of value vectors

$\mathbf{w} \in \mathbf{W}\}$. The approximate algorithm is shown in Fig. 6. With a small $|\mathbf{W}|$ it is inaccurate but conservative: because the convex hull of $\mathbf{V}(s, \mathbf{w})$ taken over $\mathbf{w}$ is smaller than $\mathbf{V}(s)$, we can only discard vectors and not add them, so all of the returned value vectors will still correspond to vectors that the exact algorithm would have returned. However, if $\mathbf{W}$ samples the P-dimensional unit hypersphere densely enough, the maximum possible approximation error will be small. (In practice, each agent will probably want to pick $\mathbf{W}$ differently, to focus her computation on policies in the portion of the Pareto frontier where her own utility is relatively high.) As $|\mathbf{W}|$ increases, the error introduced at each step will go to zero.

6 EXPERIMENTS

We tested our value iteration algorithm and negotiation procedure on two robotic planning domains: a joint motion planning problem and a supply-chain management problem.

In our motion planning problem (Fig. 7), two players together control a two-wheeled robot, with each player picking the rotational velocity for one wheel. Each player has a list of goal landmarks which she wants to cycle through, but the two players can have different lists of goals. We discretized states based on

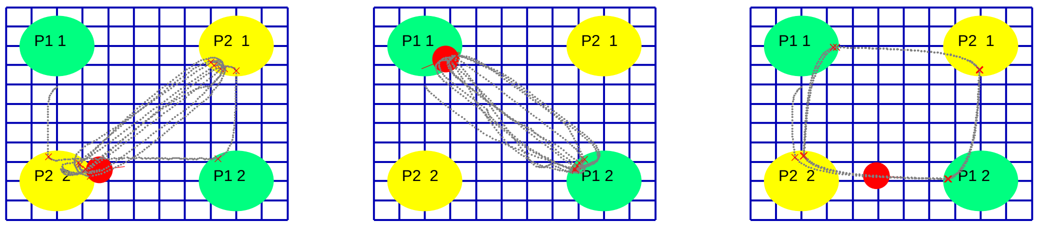

Figure 7: Execution traces for our motion planning example. Left and Center: with 2 witness vectors per state, the agents randomize between two selfish paths. Right: with 4–32 witnesses per state, the agents find a cooperative path. Steps where either player achieved a goal are marked with $\times$.

X, Y, θ and the current goals, and discretized actions into stop, slow $(0.45\frac{m}{s})$, and fast $(0.9\frac{m}{s})$, for 9 joint actions and about 25,000 states. We discretized time at $\Delta t = 1s$, and set $\gamma = 0.99$.

For both the disagreement policy and all punishment policies, we used "always stop," since by keeping her wheel stopped either player can prevent the robot from moving. Planning took a few hours of wall clock time on a desktop workstation for 32 witnesses per state.

Based on the planner's output, we ran our negotiation protocol to select an equilibrium. Fig. 7 shows the results: with limited computation the players pick two selfish paths and randomize equally between them, while with more computation they find the cooperative path.

Usually, in the first phase of the negotiation protocol, we simply had each agent reveal all the policies she knew about; this strategy is optimal if both agents know the same set of equilibria, as they do here. In the second phase, the optimal strategy is for the first player to immediately propose the Nash point and for the second to accept.[4]

We also ran experiments in the same domain, but limiting the computation of one agent and determining how that would affect the outcome of negotiation. Fig. 8 shows the results of negotiation between two players using different amounts of computation. Because the more restricted agent doesn't know about some of the best equilibria, the less restricted agent can influence negotiation by revealing only some of the equilibria that she knows about, and can alter the outcome significantly in her favor. But, revealing too few equilibria leads to an outcome that is worse for both agents.

[4]For the purpose of this experiment, we take ϵ to be so small as to make the difference between the equilibrium and the Nash point negligible. It is an interesting subject for future work to determine how large ϵ needs to be to give real agents the necessary incentive to come to an agreement expeditiously.

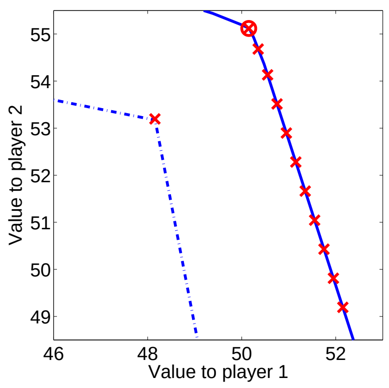

Figure 8: Negotiation between agents with different computational abilities. Solid line: Pareto frontier computed by a 32-witness agent; dash-dot line: 4-witness agent's frontier; $\times$ marks: Nash points of sets formed from the 4-witness agent's frontier and some of the 32-witness agent's frontier; $\otimes$ mark: Nash point of full set.

For our second experiment we examined a more realistic supply-chain problem. Here each player is a parts supplier competing for the business of an engine manufacturer. The manufacturer doesn't store items and will only pay for parts which can be used immediately. Each player controls a truck which moves parts from warehouses to the assembly shop; she pays for parts when she picks them up, and receives payment on delivery. Each player gets parts from different locations at different prices and neither player can individually provide all of the parts the manufacturer needs.

Each player's truck can be at six locations along a line: four warehouse locations (each of which provides a different type of part), one empty location, and the assembly shop. Building an engine requires five parts, delivered in the order $A, \{B, C\}, D, E$ (parts B and C can arrive in either order). After E, the manufacturer needs A again. Players can move left or right along the line at a small cost, or wait for free. They can also buy parts at a warehouse (dropping any previous cargo), or sell their cargo if they are at the shop and the manufacturer wants it. Each player can only carry one part at a time and only one player can make a delivery at a time. Finally, any player can retire and sell her truck; in this case the game ends and all players get the value of their truck plus any cargo. The disagreement policy is for all players to retire at all states. Fig. 9 shows the computed sets $\mathbf{V}(s_{\text{start}})$ for various numbers of witnesses. The more witnesses we use, the more accurately we represent the frontier, and the closer our final policy is to the true Nash point.

All of the policies computed are "intelligent" and "cooperative": a human observer would not see obvious ways to improve them, and in fact would say that they look similar despite their differing payoffs. Players coordinate their

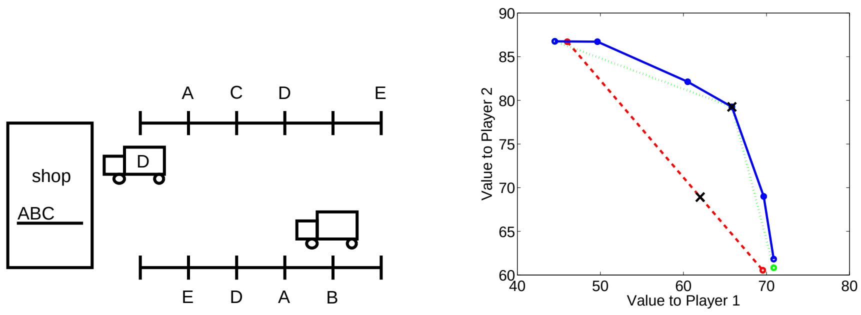

Figure 9: Supply chain management problem. In the left figure, Player 1 is about to deliver part D to the shop, while player 2 is at the warehouse which sells B. The right figure shows the tradeoff between accuracy and computation time. The solid curve is the Pareto frontier for s_{start}, as computed using 8 witnesses per state. The dashed and dotted lines were computed using 2 and 4 witnesses, respectively. Dots indicate computed value vectors; $\times$ marks indicate the Nash points.

motions, so that one player will drive out to buy part E while the other delivers part D. They sit idle only in order to delay the purchase of a part which would otherwise be delivered too soon.

7 CONCLUSION

Real-world planning problems involve negotiation among multiple agents with varying goals. To take all agents incentives into account, the agents should find and agree on Pareto-dominant subgame- perfect Nash equilibria. For this purpose, we presented efficient planning and negotiation algorithms for general-sum stochastic games, and tested them on two robotic planning problems.

Acknowledgments

We gratefully acknowledge help and comments from Ron Parr on this research. This work was supported in part by DARPA contracts HR0011-06-1-0023 (the CS2P program) and 55-00069 (the RADAR program). All opinions, conclusions, and errors are our own.

References

[1] V. Conitzer and T. Sandholm. Complexity results about Nash equilibria. Technical Report CMU-CS-02-135, School of Computer Science, Carnegie-Mellon Uni-

versity, 2002.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Massachusetts, 1995.

[3] Prajit K. Dutta. A folk theorem for stochastic games. *Journal of Economic Theory*, 66:1–32, 1995.

[4] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *Lecture Notes in Computer Science*, volume 1880, page 112. Springer, Berlin, 2000.

[5] Michael L. Littman and Peter Stone. A polynomial-time Nash equilibrium algorithm for repeated games. In *ACM Conference on Electronic Commerce*, pages 48–54. ACM, 2003.

[6] E. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

[7] Ulrich Doraszelski and Kenneth L. Judd. Avoiding the curse of dimensionality in dynamic stochastic games. *NBER Technical Working Paper No. 304*, January 2005.

[8] R. Brafman and M. Tennenholtz. Efficient learning equilibrium. *Artificial Intelligence*, 2004.

[9] D Fudenberg and E. Maskin. The folk theorem in repeated games with discounting or with incomplete information. *Econometrica*, 1986.

[10] David Levine. The castle on the hill. *Review of Economic Dynamics*, 3(2):330–337, 2000.

[11] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982.

[12] V. Krishna and R. Serrano. Multilateral bargaining. *Review of Economic Studies*, 1996.

[13] John F. Nash, Jr. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

[14] Eyal Winter Nir Dagan, Oscar Volij. A characterization of the nash bargaining solution. *Social Choice and Welfare*, 19:811–823, 2002.

[15] Vincent Conitzer and Tuomas Sandholm. Complexity results about Nash equilibria. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003.

A Proof of Convergence of Value Iteration

In the exact algorithm of figure 5 we presented a dynamic programming algorithm for computing the value vectors achievable in equilibrium in a stochastic game. In the section we provide a proof of the correctness of this algorithm. Specifically, we show that the algorithm will converge and, after convergence, will return the set of discounted value vectors achievable in subgame perfect equilibrium using the given set of punishment policies. We'll start by analyzing a simplified version of our algorithm, which omits the pruning step. (This version computes all achievable value vectors, without regard to whether they are achievable in equilibrium.) Then we will generalize the proof to apply to the full version of our algorithm.

Both versions of our algorithm can be seen as repeatedly applying a value-iteration backup operator ($\mathbf{T}$ or $\mathbf{T}_{\text{prune}}$, defined below) to an initial conservative estimate of the achievable values. In contrast to the version of value iteration for discounted MDPs, our operators are not contractions in any standard norm. Instead, our proofs rely on a monotonicity property, described in more detail below.

A.1 Definitions

We will start with some definitions that will be useful in our proof. As above, there are N states and P players, $a \in A$ is a joint action in the full set of joint actions A, and $R(s, a)$ is the one-step reward vector for state s and joint action a. $\overline{R}_a$ is an N-vector of P-vectors, telling the rewards in each state to each player of following joint action a. $P_a \in \mathbb{R}^{N \times N}$ is a transition matrix corresponding to joint action a. So $\forall i, j.\, P_{a,ij} \geq 0$ and $\forall i.\ \sum_{j=1}^{j=N} P_{a,ij} = 1$.

Write R_M for the largest absolute value of any one-step reward to any player in the game. That is, $R_M = \max_{a,s,p} |R_p(s, a)|$. Given R_M, $V_M = \frac{R_M}{1-\gamma}$ is an upper-bound on the absolute expected discounted value that any player, following any policy, can hope to achieve.

Write $\overline{\mathbf{V}}$ for the vector of sets of discounted value vectors achievable at all states in the game. That is, $\overline{\mathbf{V}}$ is a vector of length N; each component $\mathbf{V}(s)$ is a subset of $\mathbb{R}^P$ which represents a set of value vectors achievable in the game starting from state s. We introduce the overline notation to make it clear when we are referring to a vector over game states. We use the boldface notation to indicate that the structure is a set or vector of sets. $\overline{\mathbf{V}}$ is an N-vector of sets of P-vectors, where each element of a P-vector is a discounted value achievable to a player. We only introduce this complex structure (vector of sets of vectors) because it precisely captures what we want to know about the game. We need to capture all possible discounted values that can be achieved, under any policy, by all P players starting from state s: this is precisely a set of P-vectors. Since the game can start from any of the N states, we need N of these sets of P-vectors.

To measure the size of a vector of sets of vectors, we use a generalization of the infinity norm:

$$\|\overline{\mathbf{V}}\|_\infty \equiv \max_i \|\overline{\mathbf{V}}_i\|_\infty$$

That is, applying the infinity norm to a vector of sets $\overline{\mathbf{V}}$ returns the max of the infinity norm applied to each set in $\overline{\mathbf{V}}$.

$\mathbf{I} \subset \mathbb{R}^P$ is the hypercube centered at the origin with sides of length 2 in each dimension. That is, $\mathbf{I} = \{\mathbf{V} \in \mathbb{R}^P \mid \|\mathbf{V}\|_\infty \leq 1\}$. $\overline{\mathbf{I}}$ is a vector of N copies of $\mathbf{I}$. So, $\|\overline{\mathbf{I}}\|_\infty = 1$.

$\mathbf{A} + \mathbf{B}$, where $\mathbf{A}, \mathbf{B} \subset \mathbb{R}^P$, is the cross-sum operator: $\mathbf{A} + \mathbf{B} = \bigcup_{a \in \mathbf{A}, b \in \mathbf{B}} \{a + b\}$. $\mathbf{CH}_a(\mathbf{G}(a))$ is the convex closure operator: if $\mathbf{G}(a)$ is a set of vectors for each value of the dummy variable a, then $\mathbf{CH}_a(\mathbf{G}(a))$ is the set of all convex combinations of points in $\bigcup_a \mathbf{G}_a$. $\mathbf{CH}_a(\overline{\mathbf{G}}(a))$ is the componentwise generalization of $\mathbf{CH}$ to vectors of sets.

We can now define the simplified transition operator, which is the same as one iteration of the exact algorithm in figure 5 except that it omits the pruning step.

Definition

$$\mathbf{T}(\mathbf{V}) = \mathbf{CH}_{a\in A}(\overline{R}_a + \gamma P_a \overline{\mathbf{V}})$$

In this expression, applying a function P_a to a vector of sets is defined as one might expect: we use the usual expression for a matrix multiplication,

$$P_a\overline{\mathbf{V}}(s) = \sum_{s'} (P_a)_{s,s'} \overline{\mathbf{V}}(s')$$

but with cross-sum and scalar multiplication of sets rather than the usual real sum and product operations. This generalizes the usage of the transition matrix in the standard Bellman backup equations.

A.2 Convergence of the simplified algorithm

Having completed the definitions, the goal of the first set of proofs is to show that, if $\overline{\mathbf{V}}$ is initialized to a superset of the hypercube $V_M \times \mathbf{I}$, and the operation $\overline{\mathbf{V}} \leftarrow \mathbf{T}\overline{\mathbf{V}}$ is repeatedly applied, $\overline{\mathbf{V}}$ will converge.

Lemma 1 *For any* $\overline{\mathbf{V}}$*,*

$$\|P_a\overline{\mathbf{V}}\|_\infty \leq \|\overline{\mathbf{V}}\|_\infty$$

Proof: Define $V_m = \|\overline{\mathbf{V}}\|_\infty$. Then

$$\|P_a\overline{\mathbf{V}}\|_\infty = \max_i \|(P_a\overline{\mathbf{V}})_i\|_\infty = \max_i \Big\|\sum_j P_{a,ij}\overline{\mathbf{V}}_j\Big\|_\infty \leq$$

$$\max_i \Big\|\sum_j P_{a,ij} V_m\Big\|_\infty = V_m \times \max_i \sum_j P_{a,ij} = V_m$$

The first equality applies the definition of the infinity norm and the second applies the definition of matrix multiplication. The third inequality holds because all values $P_{a,ij} \geq 0$, so replacing the sets $\mathbf{V_j}$ with something at least at large makes the result no smaller. The last equality holds because V_m, a scalar, factors out, and by construction the sum of each row of any transition matrix P_a is 1. $\square$

Lemma 2

$$\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}' \Rightarrow P_a\overline{\mathbf{V}} \subseteq P_a\overline{\mathbf{V}}'$$

Proof: First note that

$$\overline{\mathbf{V}}' \equiv \overline{\mathbf{V}}' \cup \overline{\mathbf{V}}$$

Using this fact,

$$(P_a\overline{\mathbf{V}})_i = \sum_j P_{a,ij}\overline{\mathbf{V}}_j \subseteq \sum_j P_{a,ij}\overline{\mathbf{V}}'_j = P_a\overline{\mathbf{V}}'_i$$

The first equality applies the definition of matrix multiplication. The second uses the fact that a scalar times a subset of a set is a subset of that scalar times the set itself. The third equality again applies the definition of matrix multiplication. $\square$

Lemma 3 $\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}' \Rightarrow \mathbf{T}(\overline{\mathbf{V}}) \subseteq \mathbf{T}(\overline{\mathbf{V}}')$

Proof: Again note that

$$\overline{\mathbf{V}'} \equiv \overline{\mathbf{V}} \cup \overline{\mathbf{V}'} \tag{4}$$

Therefore

$$\mathbf{T}\overline{\mathbf{V}} = \mathbf{CH}_{a\in A}(\overline{R_a} + \gamma P_a(\overline{\mathbf{V}})) \subseteq \mathbf{CH}_{a\in A}(\overline{R_a} + \gamma P_a(\overline{\mathbf{V}} \cup \overline{\mathbf{V}'})) = \mathbf{T}(\overline{\mathbf{V}} \cup \overline{\mathbf{V}'})$$

The first equality is the definition of $\mathbf{T}$. The next relation follows from (4) and Lemma 2. The last equality again follows by the definition of $\mathbf{T}$. □

Lemma 3 says that $\mathbf{T}$ is monotone. So, if we start with some vector $\overline{\mathbf{V}}$ and happen to find that $\mathbf{T}\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}$, we can see by applying $\mathbf{T}$ to both sides of the relation that $\mathbf{T}^2\overline{\mathbf{V}} \subseteq \mathbf{T}\overline{\mathbf{V}}$, and in general $\mathbf{T}^k\overline{\mathbf{V}} \subseteq \mathbf{T}^{k-1}\overline{\mathbf{V}}$ for $k > 0$. That is, each application of the operator $\mathbf{T}$ gives a result that is contained in the previous iteration's $\overline{\mathbf{V}}$.

To take advantage of this fact, the next few lemmas describe an initialization that will guarantee $\mathbf{T}\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}$ for the first backup.

Lemma 4

$$P_a(k \times \bar{\mathbf{I}}) \subseteq k \times \bar{\mathbf{I}}$$

for any positive scalar k.

Proof: This follows directly from lemma 1.

By contradiction, if $P_a(k \times \bar{\mathbf{I}})$ is not a subset of $\bar{\mathbf{I}}$, then $\|P_a(k \times \bar{\mathbf{I}})\|_\infty > \|k \times \bar{\mathbf{I}}\|_\infty$, a violation of lemma 1 . □

Lemma 5

$$\|\mathbf{T}\overline{\mathbf{V}}\|_\infty \leq R_M + \gamma\|\overline{\mathbf{V}}\|_\infty$$

Proof:

$$\|\mathbf{T}\overline{\mathbf{V}}\|_\infty = \|\mathbf{CH}_{a\in A}(\overline{R_a} + \gamma P_a\overline{\mathbf{V}})\|_\infty =$$
$$\max_{a\in A}(\|\overline{R_a} + \gamma P_a\overline{\mathbf{V}}\|_\infty) \leq \max_{a\in A}\|\overline{R_a}\|_\infty + \gamma \max_{a\in A}\|P_a\overline{\mathbf{V}}\|_\infty =$$
$$R_M + \gamma\|\overline{\mathbf{V}}\|_\infty$$

The first equality applies the definition of $\mathbf{T}$. The second equality applies the definition of infinity norm on a vector of sets, and uses the fact the the convex hull operator on a set won't increase the infinity norm. The third equality uses the fact that the max of a sum is not greater than the sum of the maxes. The last equality uses the definition of R_M and lemma 1. □

Lemma 6 $\mathbf{T}(V_M\bar{\mathbf{I}}) \subseteq V_M\bar{\mathbf{I}}$

Proof:

$$\|\mathbf{T}(V_M\bar{\mathbf{I}})\|_\infty \leq R_M + \gamma \times V_M = R_M + \gamma \times \frac{R_M}{1-\gamma} = \frac{R_M}{1-\gamma} = V_M$$

Since $\|\mathbf{T}(V_M\overline{\mathbf{I}})\|_\infty \leq V_M$, it follows that $\mathbf{T}(V_M\overline{\mathbf{I}}) \subseteq V_M \times \overline{\mathbf{I}}$ by the definition of $\overline{\mathbf{I}}$. □

In fact, lemmas 6 and 3 are sufficient to show convergence. By lemma 3, as long as $\overline{\mathbf{V}}$ is initialized to $V_M \times \overline{\mathbf{I}}$, we have

$$\mathbf{T}\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}$$

But by lemma 6 we can apply $\mathbf{T}$ to both sides of this equation $k > 0$ times to get

$$\mathbf{T}^{k+1}\overline{\mathbf{V}} \subseteq \mathbf{T}^k\overline{\mathbf{V}}$$

Since $\mathbf{T}^k$ is a monotone non-increasing sequence which cannot become smaller than the empty set, it must converge.

A.3 Transition operator with pruning

To take pruning into account, we can define a new transition operator $\mathbf{T}_{\text{prune}}$. $\mathbf{T}_{\text{prune}}$ is like $\mathbf{T}$ except that it enforces incentive constraints by intersecting its backed up values with a fixed set at each state before taking the convex hull. Write $\overline{\mathbf{G}}_a$ for the vector of N components whose component $\overline{\mathbf{G}}_a(s)$ is the pruning set for state s and action a,

$$\overline{\mathbf{G}}_a(s) = \{V \mid (\forall p)\, V_p \geq V^p_{\text{dev}}(s,a)\}$$

With this definition, we can write

Definition

$$\mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}) = \mathbf{CH}_{a\in A}\Big(\overline{\mathbf{G}}_a \cap (\overline{R}_a + \gamma P_a\overline{\mathbf{V}})\Big) \tag{5}$$

where intersection between vectors of sets is defined to operate on each component separately.

Lemma 7 *If* $\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}'$, *then* $\mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}) \subseteq \mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}')$

Proof:

$$\overline{\mathbf{V}} \subseteq \overline{\mathbf{V}}' \Rightarrow \forall_a, \gamma P_a\overline{\mathbf{V}} \subseteq \gamma P_a\overline{\mathbf{V}}' \Rightarrow \forall_a, R_a + \gamma P_a\overline{\mathbf{V}} \subseteq R_a + \gamma P_a\overline{\mathbf{V}}' \Rightarrow$$

$$\forall_a, \overline{\mathbf{G}}_a \cap (R_a + \gamma P_a\overline{\mathbf{V}}) \subseteq \overline{\mathbf{G}}_a \cap (R_a + \gamma P_a\overline{\mathbf{V}}') \Rightarrow$$

$$\mathbf{CH}_{a\in A}\Big(\overline{\mathbf{G}}_a\cap(R_a+\gamma P_a\overline{\mathbf{V}})\Big) \subseteq \mathbf{CH}_{a\in A}\Big(\overline{\mathbf{G}}_a\cap(R_a+\gamma P_a\overline{\mathbf{V}}')\Big) \Rightarrow \mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}) \subseteq \mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}')$$

The first implication applies lemma 2 and the second uses the fact that multiplying by a positive scalar and adding a vector preserve containment properties. The next two implications use the fact that intersection with a fixed set and the convex hull operator preserve containment properties, and the last implication applies the definition of $\mathbf{T}_{\text{prune}}$. □

Lemma 8 $\mathbf{T}_{\text{prune}}(V_M \times \overline{\mathbf{I}}) \subseteq V_M \times \overline{\mathbf{I}}$

Proof:

$$\mathbf{T}_{\text{prune}}(V_M \times \bar{\mathbf{I}}) = \mathbf{CH}_{a \in A}\Big(\overline{\mathbf{G}}_a \cap (\overline{R_a} + \gamma P_a(V_M \times \bar{\mathbf{I}}))\Big) \subseteq$$

$$\mathbf{CH}_{a \in A}(\overline{R_a} + \gamma P_a(V_M \times \bar{\mathbf{I}})) = \mathbf{T}(V_M \times \bar{\mathbf{I}}) \subseteq V_M \times \bar{\mathbf{I}}$$

The first equality is the definition of $\mathbf{T}_{\text{prune}}$. The second inequality uses the fact that intersecting with a fixed set can't make the result any bigger. The third inequality applies the definition of $\mathbf{T}$. The last inequality is lemma 6. □

Lemma 9 *The sequence $\mathbf{T}^k_{\text{prune}}(V_M \times \bar{\mathbf{I}})$ converges as k increases.*

Proof: Lemmas 7 and 8 are sufficient for convergence of $\mathbf{T}_{\text{prune}}$, since together they make the sequence $\mathbf{T}^k_{\text{prune}}(V_M \times \bar{\mathbf{I}})$ monotone non-increasing, and a monotone non-increasing sequence which is bounded below (by the vector of empty sets) must converge. □

Definition A fixed point of $\mathbf{T}_{\text{prune}}$ is any $\overline{\mathbf{V}}$ s.t. $\mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}) = \overline{\mathbf{V}}$. A maximal fixed point is a fixed point of $\mathbf{T}_{\text{prune}}$ that is not a strict subset of any other fixed point.

Lemmas 7 and 9 imply that there will be a unique maximal fixed point: by Lemma 7 (monotonicity) our value function is bounded from below by each fixed point, and by Lemma 9 (convergence) we eventually converge to a fixed point, which must therefore contain every other fixed point. Write $\overline{\mathbf{V}}_{\text{fixed}}$ for this unique maximal fixed point.

The point of the value iteration algorithm is to find the maximal fixed point, since it tells us everything we need to know about equilibria: we show below that this fixed point contains all the value vectors that are achievable in equilibrium using the given punishment strategies. More specifically, Lemma 10 guarantees that our initial $\overline{\mathbf{V}}$ contains the maximal fixed point, which (because of monotonicity) guarantees that we cannot converge to a non-maximal fixed point. Lemma 11 will tell us a policy to achieve, in expectation, any discounted value in the fixed point, and Lemma 12 will use this policy to define a subgame-perfect equilibrium.

Lemma 10

$$\|\overline{\mathbf{V}}\|_\infty > V_M \Rightarrow \|\mathbf{T}_{\text{prune}}(\overline{\mathbf{V}})\|_\infty < \|\overline{\mathbf{V}}\|_\infty$$

So, if $\|\overline{\mathbf{V}}\|_\infty > V_M$ then $\overline{\mathbf{V}}$ cannot be a fixed set.

Proof: Let $M \equiv \|\overline{\mathbf{V}}\|_\infty$. Then

$$M > V_M \Rightarrow M > \frac{R_M}{1-\gamma} \Rightarrow (1-\gamma)M > R_M \tag{6}$$

So,

$$\|\mathbf{T}_{\text{prune}}(\overline{\mathbf{V}})\|_\infty \leq R_M + \gamma \times M < (1-\gamma)M + \gamma \times M = M$$

The first inequality comes from lemma 5, which also holds for $\mathbf{T}_{\text{prune}}$ since $\mathbf{T}_{\text{prune}}(\overline{\mathbf{V}}) \subseteq \mathbf{T}(\overline{\mathbf{V}})$ always. The second inequality holds because of equation 6: $(1-\gamma)M$ is strictly larger than R_M. □

Lemma 11 *Let $\overline{\mathbf{V}}(s)$ be a fixed point of $\mathbf{T}_{\text{prune}}$. For any value vector $V^{\text{goal}} \in \overline{\mathbf{V}}(s)$, there exists a joint policy $\pi(V^{\text{goal}}, s)$ which achieves V^{goal} if the initial state is s.*

Proof: We will begin by defining $\pi(V, s)$ for all states s and value vectors $V \in \overline{\mathbf{V}}(s)$. To motivate our definition we will pretend that, after the first step, we can achieve any value vector $V' \in \overline{\mathbf{V}}(s')$ at any state s'. We will then justify our definition by proving that our defined policy does in fact achieve the target value vector V. The proof will be by induction on the number of time steps of execution.

To define $\pi(V, s)$, we need to specify a distribution over joint actions to take from s, as well as which value vectors we will try to achieve if we wind up at another state s'. Since $V \in \overline{\mathbf{V}}(s)$ and $\overline{\mathbf{V}}$ is a fixed point, we can represent V as a convex combination of points from the sets

$$Q_a(s) = \overline{\mathbf{G}}_a(s) \cap \Big(R_a(s) + \gamma \sum_{s'} P_{a,s,s'} \overline{\mathbf{V}}(s')\Big) \tag{7}$$

for the joint actions $a \in A$. That is, there exists some set of weights w_a (with $\sum_a w_a = 1$ and $(\forall a)\, w_a \geq 0$), with each w_a corresponding to a single point q_a in $Q_a(s)$, such that

$$\sum_a w_a q_a = V \tag{8}$$

(We only need one point from each $Q_a(s)$ since $Q_a(s)$ is convex.) Now, we will choose a joint action a at random with probabilities w_a. By the definition of $Q_a(s)$, we know that

$$q_a = R_a(s) + \gamma \sum_{s'} P_{a,s,s'} V_{a,s'}$$

for some vectors $V_{a,s'} \in \overline{\mathbf{V}}(s')$. So, if the game transitions to state s', we will follow policy $\pi(V_{a,s'}, s')$ to try to achieve $V_{a,s'}$.

Now that we have defined $\pi(s, V)$ for all s and $V \in \overline{\mathbf{V}}(s)$, we will prove by induction that following $\pi(s, V)$ for k steps yields an actual expected discounted value vector $V_k^{\text{actual}}(V, s)$ which satisfies

$$\|V_k^{\text{actual}}(V, s') - V\| \leq \gamma^k V_M$$

Taking the limit as $k \to \infty$ then shows that $\pi(s, V)$ achieves V exactly.

Base Case Following any policy for 0 steps from any state s achieves discounted expected value $V_0^{\text{actual}}(V, s) = 0$. So,

$$\|V_0^{\text{actual}}(V, s) - V\| \leq V_M = \gamma^0 V_M$$

because $V \in \overline{\mathbf{V}}(s)$ and $\|\overline{\mathbf{V}}\|_\infty \leq V_M$ (Lemma 10).

Inductive Case We now know that following $\pi(V, s)$ for k steps starting from state s yields a value $V_k^{\text{actual}}(V, s)$ which satisfies

$$\|V_k^{\text{actual}}(V, s) - V\| \leq \gamma^k V_M \tag{9}$$

The expected value of following $\pi(V, s)$ for $k + 1$ steps therefore satisfies:

$$\left\|V_{k+1}^{\text{actual}}(V, s) - V\right\|_\infty =$$

$$\begin{gathered}
\Big\| V_{k+1}^{\text{actual}}(V,s) - \sum_a w_a q_a \Big\|_\infty = \\
\Big\| \sum_a w_a \Big(R_a + \gamma \sum_{s'} P_{a,s,s'} V_k^{\text{actual}}(V_{a,s'}, s') \Big) - \sum_a w_a \Big(R_a + \gamma \sum_{s'} P_{a,s,s'} V_{a,s'} \Big) \Big\|_\infty = \\
\gamma \Big\| \sum_a w_a \sum_{s'} P_{a,s,s'} (V_k^{\text{actual}}(V_{a,s'}, s') - V_{a,s'}) \Big\|_\infty \le \\
\gamma \sum_a w_a \sum_{s'} P_{a,s,s'} (\gamma^k V_M) = \\
\gamma^{k+1} V_M \sum_a w_a \sum_{s'} P_{a,s,s'} = \\
\gamma^{k+1} V_M
\end{gathered}$$

The first two equalities simply plug in the definitions of the two value vectors, where the w_a are the weights defined in equation (8). The next equality factors and cancels common terms in the sums. The inequality between the fourth and fifth lines holds because of the inductive hypothesis stated in (9). The remaining inequalities rearrange terms and use the fact that each row of the transition matrix sums to 1, as do the weights w_a. □

We've demonstrated how to construct a policy that, starting from some state s, comes arbitrarily close to any given value vector $V \in \overline{\mathbf{V}}(s)$. Now we will show that an appropriate modification of this policy is an equilibrium.

Lemma 12 *Let $\overline{\mathbf{V}}$ be a fixed point of $\mathbf{T}_{\text{prune}}$. Any value vector $V \in \overline{\mathbf{V}}(s)$ is achievable in subgame-perfect Nash equilibrium starting from state s.*

Proof: We have already showed (in Lemma 11) that a joint policy exists to achieve V from s. We can extend this policy in a simple way to make it an equilibrium: if the agents observe a deviation by player p, they will punish it by switching to the policy π_{dev}^p. (For concreteness, if two or more agents deviate simultaneously, the agents will pick at random one of the deviations to punish.) Our assumptions of public randomization and perfect monitoring mean that the agents always know what they and everyone else are supposed to do, and no agent can deviate without being caught.

All that remains is to show that, with the above threats, no agent ever wants to deviate. Since we have assumed that π_{dev}^p is a subgame-perfect equilibrium for each p, we only need to worry about the first deviation: there cannot be an incentive to deviate again in any subgame in which some agent has already deviated once.

To see whether an agent can ever have an incentive to deviate first, consider its state of knowledge immediately before acting: it knows the current state s and the joint action a which was selected by public randomization. It also knows a vector $q_a \in Q_a(s)$ and vectors $V_{a,s'} \in \overline{\mathbf{V}}(s')$ for each s'; these vectors satisfy

$$q_a = R_a(s) + \gamma \sum_{s'} P_{a,s,s'} V_{a,s'}$$

If agent p deviates, it will receive $V_{\text{dev}}^p(s,a)$ for the best possible deviation. On the other hand, if agent p does not deviate, it expects to receive q_a: it will get R_a immediately and $V_{a,s'}$ after one step, for s' chosen according to $P_{a,s,s'}$. But by the definition of $Q_a(s)$ (Eq. 7), we know $Q_a(s) \subseteq \overline{\mathbf{G}}_a(s)$. In particular, $q_a^p \ge V_{\text{dev}}^p(s,a)$, so agent p gets at least as much by following its part of action a as by deviating. □

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon