Finding Correlated Equilibria in General Sum Stochastic Games

Chris Murray and Geoff Gordon

June 2007
CMU-ML-07-113

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon

Finding Correlated Equilibria in General Sum Stochastic Games

Chris Murray and Geoff Gordon

June 2007
CMU-ML-07-113

School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Often problems arise where multiple self-interested agents with individual goals can coordinate their actions to improve their outcomes. We model these problems as general sum stochastic games. We develop a tractable approximation algorithm for computing subgame-perfect correlated equilibria in these games. Our algorithm is an extension of standard dynamic programming methods like value iteration and Q-learning. And, it is conservative: while it is not guaranteed to find all value vectors achievable in correlated equilibrium, any policy which it does find is guaranteed to be an exact equilibrium of the stochastic game (to within limits of accuracy which depend on the number of backups and not on the approximation scheme).

Our new algorithm is based on the planning algorithm of [1]. That algorithm computes subgame-perfect Nash equilibria, but assumes that it is given a set of "punishment policies" as input. Our new algorithm requires only the description of the game, an important improvement since suitable punishment policies may be difficult to come by.

Keywords: Multi-agent planning, subgame perfect correlated equilibrium, stochastic games.

1 INTRODUCTION

We model the multi-agent planning problem, where self-interested rational agents interact with each other and with the world, as a general sum stochastic game. The world state and the players' joint action determine the rewards to each player and the world's next state, and the process repeats.

Since our agents are self-interested, simply finding a policy that achieves some value for each agent will not suffice: we need to find policies where every agent has an incentive to cooperate, that is, *equilibria* of the game. An equilibrium keeps every agent in line by promising rewards for compliance or threatening punishment for deviation. More specifically, in order to give agents the maximum flexibility in jointly choosing actions, we will look for policies that are *subgame-perfect correlated equilibria*.

Unlike Nash equilibria, correlated equilibria allow agents to correlate their actions at any given point in the game. That is, agents sample from a distribution over joint actions, rather than each agent individually sampling her own action and having the joint action distribution be the product of the individual action distributions. Correlated equilibria allow a richer set of policies to be achieved, and allow agents to avoid executing unintended joint actions. In addition, as we will see below, targeting correlated equilibria allows us to develop a cleaner algorithm, since each backup operation can be viewed as approximating a convex set of payoff vectors.

Subgame perfection means that our equilibrium policies will contain no *incredible threats*: even after a deviation by one agent (which might lead to a situation which can never be observed in equilibrium play), no agent wishes to make another deviation. In other words, if our policy deters deviation with the threat of some punishment, the punishment policy is itself an equilibrium. (Of course, if we just wish to compute correlated equilibria without worrying about subgame perfection, our algorithm also provides a way to do so.)

Our focus here is on the planning problem: given a stochastic game, we want to find the set of values that can be achieved in correlated equilibrium, and policies to achieve these values. We won't focus on the problem of selecting an equilibrium from this set or executing a policy once it is chosen: we will imagine that a moderator can serve both of these functions. If a moderator is unavailable, the negotiation protocol in [1] can be used to select an equilibrium[1] and the cryptographic protocol in [2] can be used to sample from a distribution over joint actions.

[1]The negotiation protocol requires a disagreement policy as input, which we can take to be any prespecified element of $\mathbf{V}(s_{\text{start}})$. Good choices for a disagreement policy are often domain-specific, but one reasonable domain-independent choice might be that, if the players disagree, they will pick a value vector uniformly at random from the Pareto frontier of $\mathbf{V}(s_{\text{start}})$ and use it as a target.

2 STOCHASTIC GAMES

A stochastic game represents a multi-agent planning problem in the same way that a Markov Decision Process [3] 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 write a for a joint action, and α for a player's individual action. When a joint action a should be executed, but player p deviates and plays individual action α' instead of α, we write the resulting joint action as $a_{p:\alpha'}$. We deal with fully observable stochastic games with perfect monitoring, where all players can observe the true state and true joint action.

$T : S \times A \mapsto P(S)$ is the transition function, where $P(S)$ is the set of probability distributions over S. $R : S \times A \mapsto \Re^P$ is the reward function. We will write $R_p(s, a)$ for the pth component of $R(s, a)$. $\gamma \in [0, 1)$ is the discount factor. Player p wants to maximize the expectation of 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, joint) *policy* is a function $\pi_p : S \mapsto P(A)$ which tells the players how to pick their joint action at each state. A nonstationary policy is a function $\pi : (\cup_{t=0}^{\infty} (S \times A \times A)^t \times S) \mapsto P(A)$ which takes a history of states, recommended joint actions, and actual joint actions, and produces a distribution over recommended joint actions for the next time step. For any nonstationary policy, there is a stationary policy that achieves the same value at every state [4]; but, the stationary policy may not be an equilibrium even if the original nonstationary policy is.

For both kinds of policy, we imagine that there is a *moderator* who observes the current state s (or the history of states and actions h), samples a joint action a from $\pi(s)$ (or $\pi(h)$), and tells each player her recommended action a_p. We need such a moderator (or an equivalent cryptographic protocol) to make sure that no player learns another player's recommended action before choosing her own action.

The value function $V_p^\pi : S \mapsto \Re$ gives expected values for player p under joint policy π at every state. (For a nonstationary policy π we will define $V_p^\pi(h)$ to be the value after observing history h.) And, the action-value function $Q_p^\pi : S \times A \mapsto \Re$ gives expected values if we start at state s and perform action a. (If π is nonstationary, we will write $Q_p^\pi(h, a)$ for its value to p when starting at history h and performing joint action a.) The value and action-value functions for π satisfy the linear equations:

$$V_p^\pi(h) \quad = \quad \sum_a (\pi(h))(a)\, Q_p^\pi(h, a, a) \tag{1}$$

$$Q_p^\pi(h, a, a') \quad = \quad R_p(s(h), a') + \gamma \sum_{s'} P(s' \mid s(h), a')\, V_p^\pi(\langle h, a, a', s' \rangle) \qquad (2)$$

Here $s(h)$ is the state corresponding to history h (i.e., the last element of h). Note that $V_p^\pi(h)$ in Eq. 1 depends only on Q-values for *matching* recommended and actual actions, but Eq. 2 defines Q-values for both matching and non-matching cases. Also note that we have written Eqs. 1–2 for the more general case of nonstationary policies; if π is stationary, we can simplify the equations by replacing each history with its final state.

The *value vector* at state s, $V^\pi(s)$, is the vector with components $V_p^\pi(s)$ (and similarly for $V^\pi(h)$). We will write $\mathbf{V}(s)$ to represent the set of value vectors which are achievable starting from state s and following any correlated equilibrium policy (either stationary or nonstationary). This set is convex, since the moderator can randomize.

3 CORRELATED EQUILIBRIUM

A joint policy π is a correlated equilibrium if, when following π, no player ever has incentive to deviate. It's tempting to think of the correlated equilibrium condition as simply $V_p^\pi(s) \geq V_p^{\pi'}(s)$ for any policy π' which is the same as π except that player p plays some individual actions differently. However, this is not correct for two reasons. First, if player p deviates from π, the other players may react and change their actions at subsequent states to punish p. So, the immediate benefit which p achieves by deviating must be weighed against her predicted future loss from being punished.

Second, and more subtly, p may consider not only unconditional deviations of the form "ignore what I'm supposed to do and play action α instead," but also *conditional* deviations. During the execution of a policy, at some time t and state s, p first learns the individual action α which she is recommended, and then decides whether or not to follow that action. Learning α tells player p a conditional distribution on what joint action will be followed; so, player p can easily compute the conditional expectation of her future discounted value having been recommended action α. Crucially, this value depends on the recommendation α. Similarly, p's expected loss from being punished after deviating can also depend on α. So, p may wish to deviate after some recommendations and not others.

Putting these two requirements together, if a policy wants to recommend some individual action α to player p after a history h, it must promise player p more by following α than p would get from any possible deviation (and the resulting punishment). More formally, write $Q_p^\pi(h, \alpha, \alpha')$ for player p's expected value if she receives recommendation α and plays α' instead, given that the current history is h and the policy is π:

$$Q_p^\pi(h, \alpha, \alpha') = \sum_a P_\pi(a \mid \alpha, h)\, Q_p^\pi(h, a, a_{p:\alpha'}) \qquad (3)$$

For convenience, we will define $Q_p^\pi(h, \alpha, \alpha')$ to be zero if π never recommends action α to player p given history h. Note that $Q_p^\pi(h, a, a_{p:\alpha'})$ can include a penalty for player p if $\alpha' \neq \alpha$, since π can prescribe that the other players will change their behavior after observing $a_{p:\alpha'}$ instead of a.

With these definitions, the subgame-perfect correlated equilibrium condition is just

$$(\forall h, p, \alpha, \alpha')\ Q_p^\pi(h, \alpha, \alpha) \geq Q_p^\pi(h, \alpha, \alpha') \tag{4}$$

We can relax the condition of subgame perfection by requiring Eq. 4 to hold only at histories h which are reachable during on-policy play.

4 VALUE BACKUPS

The algorithm we use to find all the achievable value vectors in a stochastic game is based on dynamic programming, and is similar to (but more complicated than) value iteration or Q-learning for MDPs. The final result of the algorithm will be a P-dimensional convex set for each state, telling what vectors of values are achievable in correlated equilibrium beginning in that state. The algorithm will also return information sufficient for us to reconstruct a policy that achieves any one of those value vectors. The value backups themselves aren't that different from the normal MDP value backups, as long as the multiplication and addition operators are defined properly to work on convex sets of vectors. However, we must also include a pruning step which removes policies where some agent has an incentive to deviate.

In this section we will describe the simpler backup operator that doesn't enforce equilibrium constraints, and thus finds all value vectors achievable by *any* policy in a game, rather than only those achievable via a correlated equilibrium policy. In Section 5, we will show how to add in the incentive constraints to arrive at the complete backup operator which does enforce equilibrium constraints.

4.1 MDP backups

To derive the set-valued backup operator, we will start from the ordinary Bellman equations for Markov decision processes:

$$Q^{\text{MDP}}(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^{\text{MDP}}(s') \tag{5}$$

$$V^{\text{MDP}}(s) = \max_a Q^{\text{MDP}}(s, a) \tag{6}$$

Here $P(s' \mid s, a)$ is the probability of transitioning from state s to state s' when taking action a. The MDP backup works by treating Eqs. 5 and 6 as assignments: the operator T^{MDP} can be written

$$T^{\text{MDP}}(V)(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s' \mid s) V(s') \right]$$

or, in matrix notation,

$$T^{\mathrm{MDP}}(V) = \max_a [R_a + \gamma P_a V] \tag{7}$$

In this notation, the max operation operates componentwise.

4.2 Set-valued backups

In the multi-player generalization, $\mathbf{V}(s) \subset \Re^P$ is a set of value vectors achievable starting from state s: each $v \in \mathbf{V}(s)$ has one component for every player. But, the rules for backing up the value for a single player under a fixed joint action are exactly the same as in Eq. 5. To apply Eq. 5 to an entire set of value vectors at once, we can define addition and multiplication to work in the usual way on sets of vectors: for two sets A and B, a scalar c, and a vector d,

$$cA = \{ca \mid a \in A\} \qquad d{+}A = \{d{+}a \mid a \in A\} \qquad A{+}B = \{a{+}b \mid a \in A, b \in B\}$$

If $\mathbf{V}$ is a vector of sets and M is a matrix of scalars, the above definitions of addition and scalar multiplication also allow us to interpret the matrix multiplication $M\mathbf{V}$.

With these definitions, assuming that $\mathbf{V}^{\mathrm{noprune}}(s)$ is the set of achievable value vectors at state s, we can write

$$\mathbf{Q}^{\mathrm{noprune}}(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \mathbf{V}^{\mathrm{noprune}}(s') \tag{8}$$

for the set of all value vectors that we can achieve by starting at state s and executing action a.

Eq. 8 is one half of the Bellman equations for stochastic games without pruning. The remaining half defines $\mathbf{V}$ in terms of $\mathbf{Q}$:

$$\mathbf{V}^{\mathrm{noprune}}(s) = \mathrm{conv}_{a \in A}\, \mathbf{Q}^{\mathrm{noprune}}(s,a) \tag{9}$$

Here the conv operator first takes the union of its arguments, and then finds the convex hull of the result. The reason we need conv in Eq. 9 (instead of the max in Eq. 6) is that we want all value vectors that can be achieved from state s, not just the one which maximizes some player's payoff. (Convex combinations of achievable value vectors are achievable since the moderator can randomize among joint actions.)

By combining Equations (8) and (9), we can define the simplified transition operator, which is the same as one iteration of the exact value iteration algorithm except that it omits the pruning step.

$$\mathbf{T}^{\mathrm{noprune}}(\mathbf{V}) = \mathrm{conv}_{a \in A}\ [R_a + \gamma P_a \mathbf{V}] \tag{10}$$

If our goal were to find all value vectors achievable via *any* policy, regardless of equilibrium constraints, then repeated application of $\mathbf{T}^{\mathrm{noprune}}$ would converge

to the correct answer.[2] However, we want to prune out those values that aren't achievable by a correlated equilibrium policy. The following section details how to do so.

5 INDIVIDUAL RATIONALITY

In the full Bellman equations for discounted stochastic games (and in the corresponding backup operator), we can define $\mathbf{Q}$ exactly as we did for the no-pruning case (cf. Eq. 8):

$$\mathbf{Q}(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\mathbf{V}(s') \tag{11}$$

But now, instead of finding $\mathbf{V}(s)$ by taking the convex hull of $\mathbf{Q}(s,a)$ for all a, we need to define a new pruning operation which removes non-equilibrium action distributions so that

$$\mathbf{V}(s) = \text{prune}_a\, \mathbf{Q}(s,a) \tag{12}$$

The rest of this section defines the prune operator; Appendix A shows that our definition is correct. (That is, it shows that the unique maximal solution of Eqs. 11–12 consists of exactly the value vectors achievable in subgame-perfect correlated equilibrium.) In the expression $\mathbf{V} = \text{prune}_a\, \mathbf{Q}_a$, the set $\mathbf{V}$ and all of the sets $\mathbf{Q}_a$ are subsets of $\Re^P$.

5.1 Analyzing $V^\pi(\langle s \rangle)$

By definition, $\mathbf{V}(s)$ consists of all value vectors $V^\pi(\langle s \rangle)$ that can be achieved starting from state s under any subgame-perfect correlated equilibrium policy π. We can break π into two pieces: first, we have an immediate distribution over recommended actions, $\omega = \pi(\langle s \rangle)$. And second, we have a policy for the future: if we recommended an action a and took a possibly-different action a', then our future policy is $\pi_{s,a,a'}(h) = \pi(\langle s, a, a', h \rangle)$.

From Eqs. 1–2 and the definitions of ω and $\pi_{s,a,a'}$, we know that

$$V^\pi(\langle s \rangle) = \sum_a \omega(a) \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^{\pi_{s,a,a}}(\langle s' \rangle) \right] \tag{13}$$

Eq. 13 shows that the future policy $\pi_{s,a,a'}$ influences $V^\pi(\langle s \rangle)$ only through the value vectors $V^{\pi_{s,a,a}}(\langle s' \rangle)$ at states s' that we might reach after one step. Since we can choose our future policy arbitrarily (our future actions are not limited by how we arrived at s'), and since $\mathbf{V}(s')$ tells us what value vectors we can achieve at s', Eq. 13 means that we don't need to worry about our exact future

[2] Repeated application will converge as long as we initialize $\mathbf{V}(s)$ to a nonempty, bounded set for each s. For the full algorithm, we will in addition need to initialize $\mathbf{V}(s)$ to a set *containing* the correct answer, such as the large cube suggested in Fig. 1.

policy: for purposes of computing $\mathbf{V}(s)$, we just need to keep track of how much value our future policy will give us at each state s' having followed each possible joint action a.

In fact, by examining Eq. 11, we can see that the term in brackets in Eq. 13 is an element of $\mathbf{Q}(s, a)$. So, we have

$$V_p^\pi(\langle s \rangle) = \sum_a \omega(a) q_a \tag{14}$$

where ω is a probability distribution over actions and where $q_a \in \mathbf{Q}(s, a)$ for each a.

If we allowed all choices of ω and q_a, we would arrive at the no-pruning Bellman equations described in Sec. 4. But, not every choice of ω and q_a will correspond to an equilibrium policy. (So, $\text{prune}_a \mathbf{Q}(s, a)$ will be a subset of $\text{conv}_a \mathbf{Q}(s, a)$.) Therefore, to compute $\mathbf{V}(s)$, we still need to enforce the individual rationality constraints, Eq. 4.

5.2 Enforcing Eq. 4

Fixing $h = \langle s \rangle$ and substituting the definition of $Q_p^\pi(h, \alpha, \alpha')$ into Eq. 4, we have:

$$\sum_a P_\pi(a \mid \alpha, s)\, Q_p^\pi(\langle s \rangle, a, a) \geq \sum_a P_\pi(a \mid \alpha, s)\, Q_p^\pi(\langle s \rangle, a, a_{p:\alpha'}) \tag{15}$$

For π to be rational at s, Eq. 15 must hold for all p, α', and α with $P_\pi(\alpha \mid s) > 0$.

By Bayes' rule, $P_\pi(a \mid \alpha, s) = P_\pi(\alpha \mid a, s) P_\pi(a \mid s) / P_\pi(\alpha \mid s)$. The first term, $P_\pi(\alpha \mid a, s)$, is either 0 or 1 depending on whether α is consistent with a. The second term, $P_\pi(a \mid s)$, is given by our immediate action distribution ω. And, since the last term is positive and doesn't depend on a, we can factor it out and cancel it from both sides of Eq. 15:

$$\sum_{a|\alpha} \omega(a)\, Q_p^\pi(\langle s \rangle, a, a) \geq \sum_{a|\alpha} \omega(a)\, Q_p^\pi(\langle s \rangle, a, a_{p:\alpha'}) \tag{16}$$

for all p, α, and α'. (We can drop the qualification $P_\pi(\alpha \mid s) > 0$ since Eq. 16 is vacuous if $P_\pi(\alpha \mid s) = 0$.) The sum is over all actions a which are consistent with the recommendation α.

On the left-hand side of Eq. 16, $Q_p^\pi(\langle s \rangle, a, a)$ is just the pth element of q_a from Eq. 14 above. On the right-hand side, $Q_p^\pi(\langle s \rangle, a, a_{p:\alpha'})$ tells us how much value player p will get by deviating to α'.

Since the off-policy Q-value $Q_p^\pi(\langle s \rangle, a, a_{p:\alpha'})$ does not directly influence $V^\pi(s)$, we don't need to be concerned with its exact value except to make sure that Eq. 16 is satisfied. That is, we only need to make sure that policy π punishes deviations sufficiently severely to deter them.

By varying the future policy $\pi_{s,a,a'}$ over equilibrium policies, we can make the vector $Q^\pi(\langle s \rangle, a, a_{p:\alpha'})$ be an arbitrary element of $\mathbf{Q}(s, a_{p:\alpha'})$. So, define

$$\underline{Q}_p(s, a) = \min_{Q \in \mathbf{Q}(s,a)} Q_p$$

And, define $\underline{Q}(s,a)$ to be the vector whose pth element is $\underline{Q}_p(s,a)$.

$\underline{Q}_p(s,a)$ is the value of the harshest punishment that the other players can impose on player p (within the bounds of equilibrium) given that we start with state s and action a. So, if Eq. 16 can be satisfied at all, it will be satisfied when $Q_p^\pi(\langle s\rangle, a, a_{p:\alpha'}) = \underline{Q}_p(s, a_{p:\alpha'})$. That means that Eq. 16 reduces to

$$\sum_{a|\alpha} \omega(a) q_a \geq \sum_{a|\alpha} \omega(a) \underline{Q}(s, a_{p:\alpha'}) \tag{17}$$

for all players p, recommendations α, and deviations α'. Here ω is a probability distribution, and $q_a \in \mathbf{Q}(s,a)$ for each a. In Eq. 17, the $\geq$ operation on length-P vectors is interpreted componentwise.

If we wish to compute correlated equilibria without regard to subgame perfection, we can replace $\underline{Q}_p(s,a)$ by the minimal value that *any* feasible policy assigns to player p; since this punishment will not be visited during equilibrium play, it does not need to be an equilibrium unless we want subgame perfection. To compute the minimal feasible value for each player at each state and action, we can run the no-pruning version of our algorithm to completion.

5.3 Putting it together

Summarizing, we have that

$$\mathbf{V}(s) = \left\{ \sum_a \omega(a) q_a \right\} \tag{18}$$

where the distribution ω and the value vectors $q_a \in \mathbf{Q}(s,a)$ are constrained to satisfy Eq. 17.

Eqs. 17–18 constitute a definition of the prune operator from Eq. 12. However, to make it easier to compute $\mathbf{V}(s)$, we will rearrange this definition slightly: define

$$\bar{q}_a = \omega(a) q_a$$

And, assume that we are given a system of inequalities defining $\mathbf{Q}(s,a)$,

$$\mathbf{Q}(s,a) = \{q \mid M_a q + b_a \geq 0\}$$

(A matrix M_a and vector b_a for such a system always exist, since $\mathbf{Q}(s,a)$ is a convex set; however, M_a and b_a may be infinitely tall if $\mathbf{Q}(s,a)$ has a curved boundary.) Then $\mathbf{V}(s)$ is characterized by the linear system of inequalities

$$V = \sum_a \bar{q}_a \tag{19}$$

$$\sum_{a|\alpha} \bar{q}_a \geq \sum_{a|\alpha} \omega(a) \underline{Q}(s, a_{p:\alpha'}) \qquad \forall p, \alpha, \alpha' \tag{20}$$

$$M_a \bar{q}_a + \omega(a) b_a \geq 0 \qquad \forall a \tag{21}$$

$$\sum_a \omega(a) = 1 \tag{22}$$

$$\omega(a) \geq 0 \qquad \forall a \tag{23}$$

The constants $\underline{Q}(s,a)$ in Eq. 20 can be precomputed from $\mathbf{Q}(s,a)$. Inequality 21 ensures that $\bar{q}_a \in \omega(a)\mathbf{Q}(s,a)$, where $\omega(a)\mathbf{Q}(s,a)$ is a copy of $\mathbf{Q}(s,a)$ scaled down by $\omega(a)$.

5.4 Policy execution

If we have a solution to the Bellman equations (Eqs. 11–12), then we can use Eqs. 19–23 to find, for any target value vector $v \in \mathbf{V}(s)$, a probability distribution $\omega(a)$ and value vectors $q_a \in \mathbf{Q}(s,a)$ such that $v = \sum_a \omega(a) q_a$. (If $\bar{q}_a = 0$, then q_a is not determined, but may be chosen arbitrarily since $\omega(a) = 0$.) And, we know that each q_a satisfies

$$q_a = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v_a(s') \tag{24}$$

for some vectors $v_a(s') \in \mathbf{V}(s')$. The distribution ω and vectors $v_a(s')$ for all a and s' tell us how to achieve the target value vector v from state s:

1. Draw a joint action a according to the distribution $\omega(a)$.
2. Attempt to execute that joint action.
3. If player p deviated, switch to the policy corresponding to $\underline{Q}_p(s,a)$.[3]
4. Else, let s' be the new state.
   (a) Set the current state s to be s'.
   (b) Set the target value vector v to be $v_a(s')$.
   (c) Recompute ω, q_a, and $v_a(s)$ for all a, s according to Eqs. 19–23 and 24.
   (d) Goto 1.

6 ALGORITHM

Using the Bellman equations (Eqs. 11–12) and the linear-inequality representation of the prune operator (Eqs. 19–23), we can design a dynamic programming algorithm which computes an approximation to $\mathbf{V}(s)$ for all s. We first present a conceptual, exact algorithm that is intractable to implement; below, in Sec. 6.1, we show how to modify the algorithm so that we can implement it efficiently.

[3] A subsequent deviation by another player p' will cause us to switch to some other punishment policy, corresponding to $\underline{Q}_{p'}(s',a')$ for the state s' and action a' involved in the deviation.

Initialization
for $s \in S$
  $\mathbf{V}(s) \leftarrow \{V \mid \|V\|_\infty \leq \frac{R_{\max}}{1-\gamma}\}$
end
Repeat until converged
for iteration $\leftarrow 1, 2, \ldots$
  for $s \in S$
    Compute value vector set and punishment value for each joint action
    for $a \in A$
      $\mathbf{Q}(s,a) \leftarrow \{\mathbf{R}(s,a)\} + \gamma \sum_{s' \in S} P(s' \mid s,a)\mathbf{V}(s')$
      for $p \in \{1 \ldots P\}$, $\underline{Q}_p(s,a) \leftarrow \min_{Q \in \mathbf{Q}(s,a)} Q_p$
    end
    Do backups for enforceable one-step joint-action distributions
    $\mathbf{V}(s) \leftarrow \{V \mid (V, \omega, \bar{q}_a) \in \text{IRC}\}$
  end
end

Figure 1: Dynamic programming using exact operations on sets of value vectors. The set IRC is the intersection of the individual rationality constraints in Eqs. 19–23.

In our algorithm, the set $\mathbf{V}(s)$ for each state is initialized to a large P-dimensional hypercube, centered at the origin and extending a distance $\frac{R_{\max}}{1-\gamma}$ in each direction, where $R_{\max}$ is the absolute value of the largest reward available in the game. This initialization means that $\mathbf{V}(s)$ starts out containing all value vectors which players could ever hope to achieve.

From this initial value of $\mathbf{V}$, we compute $\mathbf{Q}$ according to Eq. 11. Then, for each state s, we intersect the linear inequalities 19–23 to find value vectors V, probability distributions ω, and scaled Q-values $\bar{q}_a$ that are consistent with individual rationality for one step into the future. We update $\mathbf{V}(s)$ to be the set of all one-step individually rational value vectors V.

We then continue in this manner, alternately recomputing $\mathbf{Q}$ and $\mathbf{V}$; each additional such backup extends the planning horizon and the individual rationality constraints another step into the future. Finally, once we have computed $\mathbf{V}$ to the desired accuracy, we can select an element of $\mathbf{V}(s_{\text{start}})$ and begin executing our policy as described in Sec. 5.4.

Write $\mathbf{V} \leftarrow T(\mathbf{V})$ for the backup operation. We show in Appendix A that $T^k(\mathbf{V})$ converges as $k \to \infty$. Unfortunately, we have not been able to show that the convergence is linear as it is for MDPs: the problem is that we could need to find a very accurate approximation to $\mathbf{V}(s')$ before we realize that some action distribution ω is irrational at s. If that action distribution was being used as a punishment to support equilibria, such a change could cause a rapid adjustment to our value sets.

6.1 Approximate backups

The algorithm given in the previous section is intractable since it operates on arbitrary convex sets. We can make a tractable algorithm by storing a finite number of points to represent each convex set $\mathbf{V}(s)$. Since the sets are convex, we need only store points on the exterior. Since the value vectors lie in $\Re^P$, we use a finite set of directions $w_1 \ldots w_K \in \Re^P$ and store only the points on the exterior of the convex sets farthest in each direction w_i.

More precisely, at step k of the algorithm we approximate each convex set $T^k(\mathbf{V}_0)(s)$ as

$$\mathbf{V}_k(s) = \text{conv}\,\{V_k^i(s) \mid i = 1 \ldots K\}$$

where $V_k^i(s)$ is the point in $T(\mathbf{V}_{k-1})(s)$ farthest in the w_i direction,

$$V_k^i(s) = \arg \max_{V \in T(\mathbf{V}_{k-1})(s)} V \cdot w_i$$

This approximation is *conservative*, since our approximate $\mathbf{V}_k(s)$ is contained in the exact $T^k(\mathbf{V}_0)(s)$. So, while our approximate algorithm might miss equilibria, it will never erroneously claim that a non-equilibrium is an equilibrium.

Using this representation, the approximate algorithm is the same as the exact algorithm in Fig. 1, except that we replace the line

$$\mathbf{V}(s) \leftarrow \{V \mid (V, \omega, \bar{q}_a) \in \text{IRC}\}$$

with

$$\textbf{for } i = 1 \ldots K, \;\; V^i(s) \leftarrow \arg\max_V V \cdot w_i \text{ s.t. } (V, \omega, \bar{q}_a) \in \text{IRC} \qquad (25)$$

The constraints and objectives for the maximizations in Eq. 25 are linear, so we can implement the approximate algorithm via calls to a standard LP solver. In particular, since the sets $\mathbf{Q}(s,a)$ are represented as the convex hull of finitely many points, there are finitely many linear constraints on the q_a vectors. We can save some time by not computing $\mathbf{Q}(s,a)$ explicitly, but instead finding the farthest point in $\mathbf{Q}(s,a)$ in the direction w_i directly from the points $V^i(s)$:

$$\begin{aligned} \arg \max_{Q \in \mathbf{Q}(s,a)} Q \cdot w_i &= R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \arg \max_{v \in \mathbf{V}(s)} V \cdot w_i \\ &= R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^i(s) \end{aligned}$$

7 EXPERIMENTS

In order to test our algorithm and our intuition, we created a simple repeated game called *three-way matching pennies*.[4] This is a three-player asymmetric game where every player holds a penny and each turn can reveal heads or tails. If players one and two reveal the same side of the coin, player three gets a

[4]Repeated games are a subset of general sum stochastic games.

Table 1: Payoff matrix to the three-way matching pennies game.

Player 1	**Player 2**	**Player 3**	
		Tails	Heads
Tails	Tails	(0,0,1)	(0,0,1)
Tails	Heads	(1,1,0)	(0,0,1)
Heads	Tails	(0,0,1)	(1,1,0)
Heads	Heads	(0,0,1)	(0,0,1)

point. If players one and two reveal different sides of their coins, they may earn a point, depending on what side player three shows. Thus the game inherently favors player three. The payoffs are shown in table 1. An important point is that players one and two will both be better off if they coordinate their joint actions: any time they both reveal the same side of the coin they will do poorly. Thus we expect that solution policies which are correlated equilibria will allow players one and two to do better than solution policies which are merely Nash equilibria.

Figure 2 shows the achievable value vectors for this game. By coordinating their actions and always playing either HT or TH, players one and two can do as well as player three and score half the time (corresponding to the point $(1, 1)$ in the figure). This is the best players one and two can do: if they play HT more than half the time (or less than half the time), player three will just play T (or H) and reduce their score.

If players one and two didn't coordinate but made their action choices independently, they would score a quarter of the time (corresponding to the point $(0.5, 1.5)$), since they would often end up playing the same side of the coin and guaranteeing player three a point. And, if players one and two anticoordinate their action choices, they can score none of the time (corresponding to the point $(0, 2)$). This unfortunate outcome is still an equilibrium: players two and three acting together can keep player one from scoring any points (and likewise players one and three can keep player two from scoring), so with the threat of such a punishment, there is no incentive for player one or two to deviate.

8 CONCLUSION

We presented a tractable approximation algorithm for finding subgame-perfect correlated equilibria in general sum stochastic games. This planning algorithm is important since it allows self-interested agents to find policies where they can jointly achieve higher payoffs by cooperating, and since subgame-perfect correlated equilibrium is a strong condition. To use this algorithm in practice, the agents would need to coordinate on a value vector in $\mathbf{V}(s_{\text{start}})$ and would need to simulate a moderator; for these purposes the agents can use the negotiation protocol in [1] and the cryptographic protocol in [2], respectively.

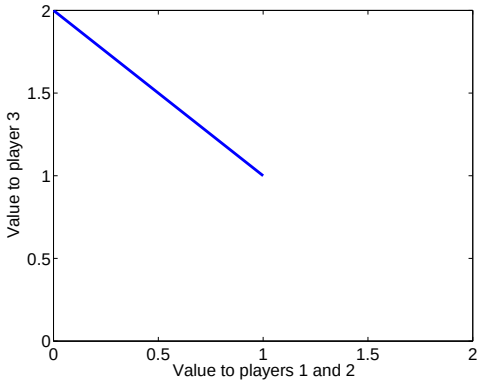

Figure 2: Achievable correlated equilibrium value vectors for the three-way matching pennies game with a discount factor of $\gamma = 0.5$. Though the true value vectors lie in three dimensions, the game's payoff structure treats players one and two the same, so the plot shows value to players one and two on the X axis and value to player three on the Y axis.

Acknowledgements

The authors would like to thank Ron Parr for helpful comments and discussion at an early stage of this work. This research was supported in part by a grant from DARPA's Computer Science Study Panel program.

References

[1] Chris Murray and Geoff Gordon. Multi-robot negotiation: Approximating the set of subgame perfect equilibria in general-sum stochastic games. In *NIPS*, 2006.

[2] 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.

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

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

A Proofs

In this appendix we will prove that the exact algorithm presented in Figure 1 is correct. We do so by making the following arguments:

Monotonicity The sets of value vectors stored by the algorithm decrease monotonically as the algorithm progresses, so if the algorithm is allowed to run for long enough they will converge to final sets of value vectors. (Since T is continuous, the final sets form a fixed point $\mathbf{V}^*$ of T, $T\mathbf{V}^* = \mathbf{V}^*$.)

Achievability After the algorithm has converged, all value vectors that it stores are achievable in subgame-perfect correlated equilibrium.

Conservative initialization The value vector sets are initialized to contain all possible value vectors that could be achieved in the game.

Conservative backups As the algorithm runs, it never throws out a value vector which is achievable in correlated equilibrium.

These properties, together, assure that the algorithm finds all value vectors achievable in correlated equilibrium.

It is interesting to note that there may be more than one solution to the Bellman equations (Eqs. 11–12). (In particular, setting $\mathbf{V}(s) = \emptyset$ for all s yields a trivial fixed point.) The achievability property means that *all* of these fixed points contain only value vectors achievable in subgame-perfect correlated equilibrium. However, because of the conservative initialization and conservative backup properties, our algorithm finds the (unique) largest fixed point, which includes all equilibrium value vectors.

The conservative initialization property is easy to show: Figure 1 initializes $\mathbf{V}(s)$ to the hypercube $[-R_{\max}/(1-\gamma), R_{\max}/(1-\gamma)]^P$, and no policy can possibly achieve more than this amount of reward. The following sections contain proofs of the remaining properties.

A.1 Monotonicity

We will show, first, that the backup operator T is monotone. That is, if $\mathbf{V}$ and $\mathbf{W}$ are two set-value functions with $\mathbf{V}(s) \subseteq \mathbf{W}(s)$ for all s (we will write this property as $\mathbf{V} \subseteq \mathbf{W}$) then

$$T(\mathbf{V}) \subseteq T(\mathbf{W}) \tag{26}$$

We will then show that, as long as we pick our initial set-value function $\mathbf{V}_0$ appropriately,

$$T(\mathbf{V}_0) \subseteq \mathbf{V}_0 \tag{27}$$

Using (27) as a base case and (26) as an inductive step, we will then have that, as claimed, our sequence of value functions decreases monotonically as our algorithm progresses.

Lemma 1 $\mathbf{T}$ *is monotone (Equation (26)).*

PROOF: Write

$$Q_a(\mathbf{V}) = R_a + \gamma P_a \mathbf{V} \tag{28}$$

Then by definition

$$T(\mathbf{V})(s) = \text{prune}_a \mathbf{Q}_a(\mathbf{V})(s) \tag{29}$$

It is easy to see that, if $\mathbf{V}(s) \subseteq \mathbf{W}(s)$ for all s, then $Q_a(\mathbf{V})(s) \subseteq Q_a(\mathbf{W})(s)$ for all s and a: linear operations on sets preserve subset relationships. So, if we can show that the pruning operator also preserves subset relationships, we will have the desired result.

The sets $Q_a(\mathbf{V})(s)$ appear in two places in Eqs. 19–23 (the definition of the pruning operator): first, they influence the feasible set for $\bar{q}_a$, and second, they influence the punishment values $\underline{Q}(s,a)$. In the first case, shrinking $Q_a(\mathbf{V})(s)$ only leads to a tighter constraint on q_a. And, in the second case, shrinking $Q_a(\mathbf{V})(s)$ only leads to a higher value for $\underline{Q}(s,a)$; since $\underline{Q}(s,a)$ appears with positive sign on the right-hand side of a $\geq$ constraint, raising $\underline{Q}(s,a)$ also results in a tighter constraint. So, the feasible set described by Eqs. 19–23 when using $Q_a(\mathbf{V})(s)$ is contained in the feasible set when using $Q_a(\mathbf{W})(s)$, which is what we wanted to prove. □

Lemma 2 *With $\mathbf{V}_0$ defined as in the initialization of Fig. 1, $T(\mathbf{V}_0) \subseteq \mathbf{V}_0$.*

PROOF: By a standard MDP argument, $Q_a(\mathbf{V}_0) \subseteq \mathbf{V}_0$: the stochastic matrix P_a maps the origin-centered cube $\mathbf{V}_0$ into itself, and the discount factor γ shrinks the cube enough that the offset R_a cannot place the resulting set outside of $\mathbf{V}_0$. But, as argued in Sec. 5, $\text{prune}_a \mathbf{Q}_a \subseteq \text{conv}_a \mathbf{Q}_a$; the desired result follows. □

Given these two lemmas, the inductive argument outlined at the beginning of the section shows that $(T^{k+1}(\mathbf{V}_0))(s) \subseteq (T^k(\mathbf{V}_0))(s)$ for each s and k. So, the sequence $(T^k(\mathbf{V}_0))(s)$ converges for each s, since it is decreasing and bounded below by the empty set. (In fact, $T^k(\mathbf{V}_0)$ is bounded below by any fixed point $\mathbf{V}$: because no policy can achieve more than $R_{\max}/(1-\gamma)$ or less than $-R_{\max}/(1-\gamma)$, we know $\mathbf{V} \subseteq \mathbf{V}_0$. By monotonicity, $T^k\mathbf{V} \subseteq T^k\mathbf{V}_0$, and by the fixed point assumption, $T^k\mathbf{V} = \mathbf{V}$. So, $T^k\mathbf{V}_0$ contains $\mathbf{V}$ for all k. Therefore, $\mathbf{V}^* = \lim_{k\to\infty} T^k\mathbf{V}_0$ contains any fixed point $\mathbf{V}$, meaning that $\mathbf{V}^*$ is the unique largest fixed point of T.)

A.2 Achievability

Lemma 3 *Let $\mathbf{V}^*$ be a fixed set of T, that is, $\mathbf{V}^* = T(\mathbf{V}^*)$. For any $v \in \mathbf{V}^*(s)$, the policy $\pi_{v,s}$ described in Sec. 5.4 achieves v in expectation starting from state s. And, no agent has an incentive to deviate from $\pi_{v,s}$ at any step.*

PROOF: The proof will be by induction. Specifically, we will show that, for all v and s, following $\pi_{v,s}$ for k steps will yield an actual expected discounted value vector $A_k(v,s)$ which satisfies

$$\|A_k(v,s) - v\| \leq \gamma^k \frac{R_{\max}}{1-\gamma} \tag{30}$$

(The norm will always be the max (infinity) norm, but we will leave off the subscript to avoid clutter.) So,

$$\lim_{k\to\infty} A_k(v,s) = v \tag{31}$$

for all s and $v \in \mathbf{V}^*(s)$. Since the $\text{prune}_a\mathbf{Q}_a$ operation enforces incentive constraints under the assumption that the $\mathbf{Q}_a$ sets correctly describe achievable values, Eq. 31 means that incentive constraints are correctly enforced. (More precisely, the incentive for any player to deviate is bounded by twice the max-norm error in achieving a target value vector; since Eq. 31 shows that this error is zero in the limit, there is no incentive to deviate.)

Base case: Following any policy for 0 steps from state s achieves $A_0(v,s) = 0$, while $v \in \mathbf{V}^*(s)$ means that $\|v\| \leq \frac{R_{\max}}{1-\gamma}$. So, $\|A_0(v,s) - v\| \leq \gamma^0 \frac{R_{\max}}{1-\gamma}$.

Inductive case: By the inductive hypothesis, we can start in any state s and, for any $v \in \mathbf{V}^*(s)$, achieve in k steps a value vector $A_k(v,s)$ satisfying $\|A_k(v,s) - v\| \leq \gamma^k \frac{R_{\max}}{1-\gamma}$.

Since $\mathbf{V}^*$ is a fixed set, for every $v \in \mathbf{V}^*(s)$ there exists a distribution ω and values $v_{a,s'}$ for all a and s' satisfying one-step incentive constraints and

$$v = \sum_a \omega(a) \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v_{a,s'} \right] \tag{32}$$

$$v_{a,s'} \in \mathbf{V}^*(s') \tag{33}$$

Our $k+1$-step policy will use the action distribution ω on the first step, and will target $v_{a,s'}$ for the next k steps to achieve

$$A_{k+1}(v,s) = \sum_a \omega(a) \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) A_k(v_{a,s'}, s') \right] \tag{34}$$

This lets us write the max-norm distance between the value achieved by the $k+1$-step policy and the target value as

$$\begin{aligned}
&\|A_{k+1}(v,s) - v\| \\
&= \left\| A_{k+1}(v,s) - \sum_a \omega(a) \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v_{a,s'} \right] \right\| && (35)\\
&= \left\| \sum_a \omega(a) \gamma \sum_{s'} P(s' \mid s,a) \left[A_k(v_{a,s'}, s') - v_{a,s'} \right] \right\| && (36)\\
&\leq \gamma \sum_a \sum_{s'} \omega(a) P(s' \mid s,a) \|A_k(v_{a,s'}, s') - v_{a,s'}\| && (37)\\
&\leq \gamma \sum_a \sum_{s'} \omega(a) P(s' \mid s,a) \gamma^K \frac{R_{\max}}{1-\gamma} && (38)\\
&= \gamma^{K+1} \frac{R_{\max}}{1-\gamma} && (39)
\end{aligned}$$

Here Eq. 35 expands v using equation 32. Eq. 36 expands $A_{k+1}(v, s)$ using equation 34. This expansion lets the reward terms $\sum_a \omega(a)R(s, a)$ cancel out. Equation 37 uses the fact that the norm of a sum of terms is not greater than the sum of the norms of the terms. Equation 38 uses the inductive hypothesis, which says that a K-step policy starting from any state s' can can achieve a value within $\gamma^K \frac{R_{\max}}{1-\gamma}$ of any value $v(s') \in \mathbf{V}^*(s')$. The last equation uses the fact that probability distributions sum to one.

Taking Eqs. 35–39 together, we have verified the inductive case and have therefore proven the lemma. □

A.3 Conservative backups

Lemma 4 *If $\mathbf{V}_0$ contains the value vectors for all subgame-perfect correlated equilibria at all states, then $T^k(\mathbf{V_0})$ also contains the value vectors for all subgame-perfect correlated equilibria at all states.*

PROOF: The proof is by induction. The base case ($k = 0$) is assumed in the lemma. For the inductive step, assume the lemma is true for T^k. The backup operator consists of two steps, defined in Eqs. 28–29. The first step throws out only value vectors that are not achievable by an arbitrary initial action followed by any equilibrium policy. The second step throws out only value vectors for which the individual rationality constraints are violated at the first step, and must therefore leave subgame-perfect correlated equilibria untouched. □

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon