Dynamic Programming for Partially Observable Stochastic Games

Eric A. Hansen
Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
hansen@cse.msstate.edu

Daniel S. Bernstein and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{bern,shlomo}@cs.umass.edu

Abstract

We develop an exact dynamic programming algorithm for partially observable stochastic games (POSGs). The algorithm is a synthesis of dynamic programming for partially observable Markov decision processes (POMDPs) and iterated elimination of dominated strategies in normal form games. We prove that when applied to finite-horizon POSGs, the algorithm iteratively eliminates very weakly dominated strategies without first forming a normal form representation of the game. For the special case in which agents share the same payoffs, the algorithm can be used to find an optimal solution. We present preliminary empirical results and discuss ways to further exploit POMDP theory in solving POSGs.

1. Introduction

The theory of stochastic games provides a foundation for much recent work on multi-agent planning and learning [17, 3, 13, 4, 11]. A stochastic game can be viewed as an extension of a Markov decision process (MDP) in which there are multiple agents with possibly conflicting goals, and the joint actions of agents determine state transitions and rewards. Much of the literature on stochastic games assumes that agents have complete information about the state of the game; in this respect, it generalizes work on completely observable MDPs. In fact, exact dynamic programming algorithms for stochastic games closely resemble exact dynamic programming algorithms for completely observable MDPs [22, 7, 13]. Although there is considerable literature on partially observable Markov decision processes (POMDPs), corresponding results for partially observable stochastic games (POSGs) are very sparse, and no exact dynamic programming algorithm for solving POSGs has been previously described.

In this paper, we show how to generalize the dynamic programming approach to solving POMDPs in order to develop a dynamic programming algorithm for POSGs. The difficulty in developing this generalization is that agents can have different beliefs. As a result, it is not possible to solve a POSG by transforming it into a completely observable stochastic game over belief states, analogous to how a POMDP is solved by transforming it into a completely observable MDP over belief states. A different approach is needed. Our approach is related to iterative elimination of dominated strategies in normal form games, which also allows agents to have different beliefs. In fact, our approach can be viewed as a synthesis of dynamic programming for POMDPs and iterated elimination of dominated strategies in normal form games. We define a generalized notion of belief that includes uncertainty about the underlying state and uncertainty about other agent's future plans. This allows us to define a *multi-agent dynamic programming operator*. We show that the resulting dynamic programming algorithm corresponds to a type of iterated elimination of dominated strategies in the normal form representation of finite-horizon POSGs. This is the first dynamic programming algorithm for iterated strategy elimination. For the special case where all agents share the same payoff function, our dynamic programming algorithm can be used to find an optimal solution.

1.1. Related work

A finite-horizon POSG can be viewed as a type of extensive game with imperfect information [16]. Although much work has been done on such games, very little of it is from a computational perspective. This is understandable in light of the negative worst-case complexity results for POSGs [2]. A notable exception is reported in [14, 15], in which the authors take advantage of the *sequence form* representation of two-player games to find mixed strategy Nash equilibria efficiently. In contrast to their work, ours applies to any number of players. Furthermore, our algorithms are focused on eliminating dominated strategies, and do not make any assumptions about which of the remaining strategies

will be played.

For the special case of cooperative games, several algorithms have been proposed. However, previous algorithms do not guarantee optimality in general. If all agents share their private information, a cooperative POSG can be converted to a single-agent POMDP. There are also algorithms for solving cooperative POSGs with other forms of very specialized structure [10, 1]. For general cooperative POSGs, algorithms such as those of Peshkin et al. [20] and Nair et al. [18] can be used, but they are only guaranteed to converge to local optima.

2. Background

As background, we review the POSG model and two algorithms that we generalize to create a dynamic programming algorithm for POSGs: dynamic programming for POMDPs and elimination of dominated strategies in solving normal form games.

2.1. Partially observable stochastic games

A *partially observable stochastic game* (POSG) is a tuple $\langle \mathcal{I}, \mathcal{S}, \{b^0\}, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, \mathcal{P}, \{R_i\} \rangle$, where,

- $\mathcal{I}$ is a finite set of agents (or controllers) indexed $1, \ldots, n$
- $\mathcal{S}$ is a finite set of states
- $b^0 \in \Delta(\mathcal{S})$ represents the initial state distribution
- $\mathcal{A}_i$ is a finite set of actions available to agent i and $\vec{\mathcal{A}} = \times_{i \in \mathcal{I}} \mathcal{A}_i$ is the set of joint actions (i.e., action profiles), where $\vec{a} = \langle a_1, \ldots, a_n \rangle$ denotes a joint action
- $\mathcal{O}_i$ is a finite set of observations for agent i and $\vec{\mathcal{O}} = \times_{i \in \mathcal{I}} \mathcal{O}_i$ is the set of joint observations, where $\vec{o} = \langle o_1, \ldots, o_n \rangle$ denotes a joint observation
- $\mathcal{P}$ is a set of Markovian state transition and observation probabilities, where $\mathcal{P}(s', \vec{o} | s, \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state s results in a transition to state s' and joint observation $\vec{o}$
- $R_i : \mathcal{S} \times \vec{\mathcal{A}} \rightarrow \Re$ is a reward function for agent i

A game unfolds over a finite or infinite sequence of stages, where the number of stages is called the *horizon* of the game. In this paper, we consider finite-horizon POSGs; some of the challenges involved in solving the infinite-horizon case are discussed at the end of the paper. At each stage, all agents simultaneously select an action and receive a reward and observation. The objective, for each agent, is to maximize the expected sum of rewards it receives during the game.

Whether agents compete or cooperate in seeking reward depends on their reward functions. The case in which the agents share the same reward function has been called a *decentralized partially observable Markov decision process (DEC-POMDP)* [2].

2.2. Dynamic programming for POMDPs

A POSG with a single agent corresponds to a POMDP. We briefly review an exact dynamic programming algorithm for POMDPs that provides a foundation for our exact dynamic programming algorithm for POSGs. We use the same notation for POMDPs as for POSGs, but omit the subscript that indexes an agent.

The first step in solving a POMDP by dynamic programming (DP) is to convert it into a completely observable MDP with a state set $\mathcal{B} = \Delta(\mathcal{S})$ that consists of all possible beliefs about the current state. Let $b^{a,o}$ denote the belief state that results from belief state b, after action a and observation o. The DP operator can be written in the form,

$$V^{t+1}(b) = \max_{a \in \mathcal{A}} \left\{ \sum_{s \in \mathcal{S}} b(s) \left[R(s,a) + \sum_{o \in \mathcal{O}} \mathcal{P}(o|s,a) V^t(b^{a,o}) \right] \right\}, \tag{1}$$

where $\mathcal{P}(o|s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s', o|s, a)$, and the updated value function is computed for all belief states $b \in \mathcal{B}$. Exact DP algorithms for POMDPs rely on Smallwood and Sondik's [23] proof that the DP operator preserves the piecewise linearity and convexity of the value function. This means that the value function can be represented exactly by a finite set of $|\mathcal{S}|$-dimensional value vectors, denoted $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$, where

$$V(b) = \max_{1 \leq j \leq k} \sum_{s \in \mathcal{S}} b(s) v_j(s). \tag{2}$$

As elucidated by Kaelbling et al. [12], each value vector corresponds to a complete conditional plan that specifies an action for every sequence of observations. Adopting the terminology of game theory, we often refer to a complete conditional plan as a *strategy*. We use this interchangeably with *policy tree*, because a conditional plan for a finite-horizon POMDP can be viewed as a tree.

The DP operator of Equation (1) computes an updated value function, but can also be interpreted as computing an updated set of policy trees. In fact, the simplest algorithm for computing the DP update has two steps, which are described below.

In the first step, the DP operator is given a set Q^t of depth-t policy trees and a corresponding set $\mathcal{V}^t$ of value vectors representing the horizon-t value function. It computes Q^{t+1} and $\mathcal{V}^{t+1}$ in two steps. First, a set of depth $t+1$ policy trees, Q^{t+1}, is created by generating every possible depth $t+1$ policy tree that makes a transition, after an action and observation, to the root node of some depth-t policy tree in Q^t. This operation will hereafter be called an *exhaustive backup*. Note that $|Q^{t+1}| = |\mathcal{A}||Q^t|^{|\mathcal{O}|}$. For each

policy tree $q_j \in Q^{t+1}$, it is straightforward to compute a corresponding value vector, $v_j \in \mathcal{V}^{t+1}$.

The second step is to eliminate policy trees that need not be followed by a decision maker that is maximizing expected value. This is accomplished by eliminating (i.e., pruning) any policy tree when this can be done without decreasing the value of any belief state.

Formally, a policy tree $q_j \in Q_i^{t+1}$ with corresponding value vector $v_j \in \mathcal{V}_i^{t+1}$ is considered dominated if for all $b \in \mathcal{B}$ there exists a $v_k \in \mathcal{V}_i^{t+1} \setminus v_j$ such that $b \cdot v_k \geq b \cdot v_j$. This test for dominance is performed using linear programming. When q_j is removed from the set Q_i^{t+1}, its corresponding value vector v_j is also removed from $\mathcal{V}_i^{t+1}$.

The dual of this linear program can also be used as a test for dominance. In this case, a policy tree q_j with corresponding value vector v_j is dominated when there is a probability distribution p over the other policy trees, such that

$$\sum_{k \neq j} p(k) v_k(s) \geq v_j(s), \forall s \in S. \tag{3}$$

This alternative, and equivalent, test for dominance plays a role in iterated strategy elimination, as we will see in the next section, and was recently applied in the context of POMDPs [21].

2.3. Iterated elimination of dominated strategies

Techniques for eliminating dominated strategies in solving a POMDP are very closely related to techniques for eliminating dominated strategies in solving games in normal form. A game in normal form is a tuple $G = \{\mathcal{I}, \{D_i\}, \{V_i\}\}$, where $\mathcal{I}$ is a finite set of agents, D_i is a finite set of strategies available to agent i, and $V_i : \vec{D} \rightarrow \Re$ is the value (or payoff) function for agent i. Unlike a stochastic game, there are no states or state transitions in this model.

Every strategy $d_i \in D_i$ is a *pure strategy*. Let $\delta_i \in \Delta(D_i)$ denote a *mixed strategy*, that is, a probability distribution over the pure strategies available to agent i, where $\delta_i(d_i)$ denotes the probability assigned to strategy $d_i \in D_i$. Let d_{-i} denote a profile of pure strategies for the other agents (i.e., all the agents except agent i), and let δ_{-i} denote a profile of mixed strategies for the other agents. Since agents select strategies simultaneously, δ_{-i} can also represent agent i's belief about the other agents' likely strategies. If we define $V_i(d_i, \delta_{-i}) = \sum_{d_{-i}} \delta_{-i}(d_{-i}) V_i(d_i, d_{-i})$, then

$$B_i(\delta_{-i}) = \{d_i \in D_i | V_i(d_i, \delta_{-i}) \geq V_i(d_i', \delta_{-i}) \; \forall d_i' \in D_i\} \tag{4}$$

denotes the *best response function* of agent i, which is the set of strategies for agent i that maximize the value of some belief about the strategies of the other agents. Any strategy that is not a best response to some belief can be deleted.

A dominated strategy d_i is identified by using linear programming. The linear program identifies a probability distribution σ_i over the other strategies such that

$$V_i(\sigma_i, d_{-i}) > V_i(d_i, d_{-i}), \forall d_{-i} \in D_{-i}. \tag{5}$$

This test for dominance is very similar to the test for dominance used to prune strategies in solving a POMDP. It differs in using strict inequality, which is called *strict dominance*. Game theorists also use *weak dominance* to prune strategies. A strategy d_i is weakly dominated if $V_i(\sigma_i, d_{-i}) \geq V_i(d_i, d_{-i})$ for all $d_{-i} \in D_{-i}$, and $V_i(\sigma_i, d_{-i}) > V_i(d_i, d_{-i})$ for some $d_{-i} \in D_{-i}$. The test for dominance which does not require any strict inequality is sometimes called *very weak dominance*, and corresponds exactly to the test for dominance in POMDPs, as given in Equation (3). Because a strategy that is very weakly dominated but not weakly dominated must be *payoff equivalent* to a strategy that very weakly dominates it, eliminating very weakly dominated strategies may have the same effect as eliminating weakly dominated strategies in the *reduced normal form* representation of a game, where the reduced normal form representation is created by combining any set of payoff-equivalent strategies into a single strategy.

There are a couple other interesting differences between the tests for dominance in Equations (3) and (5). First, there is a difference in beliefs. In normal-form games, beliefs are about the strategies of other agents, whereas in POMDPs, beliefs are about the underlying state. Second, elimination of dominated strategies is iterative when there are multiple agents. When one agent eliminates its dominated strategies, this can affect the best-response function of other agents (assuming common knowledge of rationality). After all agents take a turn in eliminating their dominated strategies, they can consider eliminating additional strategies that may only have been best responses to strategies of other agents that have since been eliminated. The procedure of alternating between agents until no agent can eliminate another strategy is called *iterated elimination of dominated strategies*.

In solving normal-form games, iterated elimination of dominated strategies is a somewhat weak solution concept, in that it does not (usually) identify a specific strategy for an agent to play, but rather a set of possible strategies. To select a specific strategy requires additional reasoning, and introduces the concept of a Nash equilibrium, which is a profile of strategies (possibly mixed), such that $\delta_i \in B_i(\delta_{-i})$ for all agents i. Since there are often multiple equilibria, the problem of *equilibrium selection* is important. (It has a more straightforward solution for cooperative games than for general-sum games.) But in this paper, we focus on the issue of elimination of dominated strategies.

3. Dynamic programming for POSGs

In the rest of the paper, we develop a dynamic programming algorithm for POSGs that is a synthesis of dynamic programming for POMDPs and iterated elimination of dominated strategies in normal-form games. We begin by introducing the concept of a normal-form game with hidden state, which provides a way of relating the POSG and normal-form representations of a game. We describe a method for eliminating dominated strategies in such games, and then show how to generalize this method in order to develop a dynamic programming algorithm for finite-horizon POSGs.

3.1. Normal-form games with hidden state

Consider a game that takes the form of a tuple $G = \{\mathcal{I}, \mathcal{S}, \{D_i\}, \{V_i\}\}$, where $\mathcal{I}$ is a finite set of agents, $\mathcal{S}$ is a finite set of states, D_i is a finite set of strategies available to agent i, and $V_i : \mathcal{S} \times \vec{D} \rightarrow \Re$ is the value (or payoff) function for agent i. This definition resembles the definition of a POSG in that the payoff received by each agent is a function of the state of the game, as well as the joint strategies of all agents. But it resembles a normal-form game in that there is no state-transition model. In place of one-step actions and rewards, the payoff function specifies the value of a strategy, which is a complete conditional plan.

In a normal form game with hidden state, we define an agent's belief in a way that synthesizes the definition of belief for POMDPs (a distribution over possible states) and the definition of belief in iterated elimination of dominated strategies (a distribution over the possible strategies of the other agents). For each agent i, a belief is defined as a distribution over $\mathcal{S} \times D_{-i}$, where the distribution is denoted b_i. The value of a belief of agent i is defined as

$$V_i(b_i) = \max_{d_i \in D_i} \sum_{s \in \mathcal{S}, d_{-i} \in D_{-i}} b_i(s, d_{-i}) V_i(s, d_i, d_{-i}).$$

A strategy d_i for agent i is very weakly dominated if eliminating it does not decrease the value of any belief. The test for very weak dominance is a linear program that determines whether there is a mixed strategy $\sigma_i \in \Delta(D_i \setminus d_i)$ such that

$$V_i(s, \sigma_i, d_{-i}) \geq V_i(s, d_i, d_{-i}), \forall s \in \mathcal{S}, \forall d_{-i} \in D_{-i}. \quad (6)$$

These generalizations of the key concepts of belief, value of belief, and dominance play a central role in our development of a DP algorithm for POSGs in the rest of this paper.

In our definition of a normal form game with hidden state, we do not include an initial state probability distribution. As a result, each strategy profile is associated with an $|\mathcal{S}|$-dimensional vector that can be used to compute the value of this strategy profile for *any* state probability distribution. This differs from a standard normal form game in which each strategy profile is associated with a scalar value. By assuming an initial state probability distribution, we could convert our representation to a standard normal form game in which each strategy profile has a scalar value. But our representation is more in keeping with the approach taken by the DP algorithm for POMDPs, and lends itself more easily to development of a DP algorithm for POSGs. The initial state probability distribution given in the definition of a POMDP is not used by the DP algorithm for POMDPs; it is only used to select a policy after the algorithm finishes. The same holds in the DP algorithm for POSGs we develop. Like the POMDP algorithm, it computes a solution for all possible initial state probability distributions.

3.2. Normal form of finite-horizon POSGs

Disregarding the initial state probability distribution, a finite-horizon POSG can be converted to a normal-form game with hidden state. When the horizon of a POSG is one, the two representations of the game are identical, since a strategy corresponds to a single action, and the payoff functions for the normal-form game correspond to the reward functions of the POSG. When the horizon of a POSG is greater than one, the POSG representation of the game can be converted to a normal form representation with hidden state, by a recursive construction. Given the sets of strategies and the value (or payoff) functions for a horizon t game, the sets of strategies and value functions for the horizon $t+1$ game are constructed by exhaustive backup, as in the case of POMDPs. When a horizon-t POSG is represented in normal form with hidden state, the strategy sets include all depth-t policy trees, and the value function is piecewise linear and convex; each strategy profile is associated with an $|\mathcal{S}|$-vector that represents the expected t-step cumulative reward achieved for each potential start state (and so any start state distribution) by following this joint strategy.

If a finite-horizon POSG is represented this way, iterated elimination of dominated strategies can be used in solving the game, after the horizon t normal form game is constructed. The problem is that this representation can be *much* larger than the original representation of a POSG. In fact, the size of the strategy set for each agent i is greater than $|\mathcal{A}_i|^{|\mathcal{O}_i|^t}$, which is doubly exponential in the horizon t. Because of the large sizes of the strategy sets, it is usually not feasible to work directly with this representation. The dynamic programming algorithm we develop partially alleviates this problem by performing iterated elimination of dominated strategies at each stage in the construction of the normal form representation, rather than waiting until the

construction is finished.

3.3. Multi-agent dynamic programming operator

The key step of our algorithm is a *multi-agent dynamic programming operator* that generalizes the DP operator for POMDPs. As for POMDPs, the operator has two steps. The first is a backup step that creates new policy trees and vectors. The second is a pruning step.

In the backup step, the DP operator is given a set of depth-t policy trees Q_i^t for each agent i, and corresponding sets of value vectors $\mathcal{V}_i^t$ of dimension $|\mathcal{S} \times Q_{-i}^t|$.[1] Based on the action transition, observation, and reward model of the POSG, it performs an exhaustive backup on each of the sets of trees, to form Q_i^{t+1} for each agent i. It also recursively computes the value vectors in $\mathcal{V}_i^{t+1}$ for each agent i. Note that this step corresponds to recursively creating a normal form with hidden state representation of a horizon $t+1$ POSG, given a normal form with hidden state representation of the horizon t POSG.

The second step of the multi-agent DP operator consists of pruning dominated policy trees. As in the single agent case, an agent i policy tree can be pruned if its removal does not decrease the value of any belief for agent i. As with normal form games, removal of a policy tree reduces the dimensionality of the other agents' belief space, and it can be repeated until no more policy trees can be pruned from any agent's set. (Note that different agent orderings may lead to different sets of policy trees and value vectors. The question of order dependence in eliminating dominated strategies has been extensively studied in game theory, and we do not consider it here.) Pseudocode for the multi-agent DP operator is given in Table 1.

The validity of the pruning step follows from a version of the optimality principle of dynamic programming, which we prove for a single iteration of the multi-agent DP operator. By induction, it follows for any number of iterations.

Theorem 1 *Consider a set Q_i^t of depth t policy trees for agent i, and consider the set Q_i^{t+1} of depth $t+1$ policy trees created by exhaustive backup, in the first step of the multi-agent DP operator. If any policy tree $q_j \in Q_i^t$ is very weakly dominated, then any policy tree $q' \in Q_i^{t+1}$ that contains q_j as a subtree is also very weakly dominated.*

1 The value function V_i^t of agent i can be represented as a set $\mathcal{V}_i^t$ of value vectors of dimension $|\mathcal{S} \times Q_{-i}^t|$, with one for each strategy in Q_i^t, or as a set of value vectors of dimension $|\mathcal{S}|$, with one for each strategy profile in $Q_i^t \times Q_{-i}^t$. The two representations are equivalent. The latter is more useful in terms of implementation, since it means the size of vectors does not change during iterated elimination of dominated strategies; only the number of vectors changes. (Using this representation, multiple $|\mathcal{S}|$-vectors must be deleted for each strategy deleted.) The former representation is more useful in explaining the algorithm, since it entails a one-to-one correspondence between strategies and value vectors, and so we adopt it in this section.

Proof: Consider a very weakly dominated policy tree $q_j \in Q_i^t$. According to the dual formulation of the test for dominance, there exists a distribution p over policy trees in $Q_i^t \backslash q_j$ such that $\sum_{k \neq j} p(k) v_k(s, q_{-i}) \geq v_j(s, q_{-i})$ for all $s \in \mathcal{S}$ and $q_{-i} \in Q_{-i}^k$. (Recall that $v_j \in \mathcal{V}_i^t$ is the value vector corresponding to policy tree q_j.) Now consider any policy tree $q' \in Q_i^{t+1}$ with q_j as a subtree. We can replace instances of q_j in q' with the distribution p to get a *behavioral strategy*, which is a stochastic policy tree. From the test for dominance, it follows that the value of this behavioral strategy is at least as high as that of q', for any distribution over states and strategies of the other agents. Since any behavioral strategy can be represented by a distribution over pure strategies, it follows that q' is very weakly dominated. □

Thus, pruning very weakly dominated strategies from the sets Q_i^t before using the dynamic programming operator is equivalent to performing the dynamic programming operator without first pruning Q_i^t. The advantage of first pruning very weakly dominated strategies from the sets Q_i^t is that it improves the efficiency of dynamic programming by reducing the initial size of the sets Q_i^{t+1} generated by exhaustive backup.

It is possible to define a multi-agent DP operator that prunes strongly dominated strategies. However, sometimes a strategy that is not strongly dominated will have a strongly dominated subtree. This is referred to as an *incredible threat* in the literature. Thus it is an open question whether we can define a multi-agent DP operator that prunes only strongly dominated strategies. In this paper, we focus on pruning very weakly dominated strategies. As already noted, this is identical to the form of pruning used for POMDPs.

There is an important difference between this algorithm and the dynamic programming operator for single-agent POMDPs, in terms of implementation. In the single agent case, only the value vectors need to be kept in memory. At execution time, an optimal action can be extracted from the value function using one-step lookahead, at each time step. We do not currently have a way of doing this when there are multiple agents. In the multi-agent case, instead of selecting an action at each time step, each agent must select a policy tree (i.e., a complete strategy) at the beginning of the game. Thus, the policy tree sets must also be remembered. Of course, some memory savings is possible by realizing that the policy trees for an agent share subtrees.

3.4. Solving finite-horizon POSGs

As we have described, any finite-horizon POSG can be given a normal form representation. The process of computing the normal form representation is recursive. Given the definition of a POSG, we successively compute normal form games with hidden state for horizons one, two, and so on, up to horizon T. Instead of computing all possible

Input: Sets of depth-t policy trees Q_i^t and corresponding value vectors $\mathcal{V}_i^t$ for each agent i.

1. Perform exhaustive backups to get Q_i^{t+1} for each i.
2. Recursively compute $\mathcal{V}_i^{t+1}$ for each i.
3. Repeat until no more pruning is possible:
   (a) Choose an agent i, and find a policy tree $q_j \in Q_i^{t+1}$ for which the following condition is satisfied: $\forall b \in \Delta(\mathcal{S} \times Q_{-i}^{t+1})$, $\exists v_k \in \mathcal{V}_i^{t+1} \setminus v_j$ s.t. $b \cdot v_k \geq b \cdot v_j$.
   (b) $Q_i^{t+1} \leftarrow Q_i^{t+1} \setminus q_j$.
   (c) $\mathcal{V}_i^{t+1} \leftarrow \mathcal{V}_i^{t+1} \setminus v_j$.

Output: Sets of depth-$t+1$ policy trees Q_i^{t+1} and corresponding value vectors $\mathcal{V}_i^{t+1}$ for each agent i.

Table 1. The multi-agent dynamic programming operator.

strategies for each horizon, we have defined a multi-agent dynamic programming operator that performs iterated elimination of very weakly dominated strategies at each stage. This improves the efficiency of the algorithm because if a policy tree is pruned by the multi-agent DP operator at one stage, every policy tree containing it as a subtree is effectively eliminated, in the sense that it will not be created at a later stage. We now show that performing iterated elimination of very weakly dominated strategies at each stage in the construction of the normal form game is equivalent to waiting until the final stage to perform iterated elimination of very weakly dominated strategies.

Theorem 2 *Dynamic programming applied to a finite-horizon POSG corresponds to iterated elimination of very weakly dominated strategies in the normal form of the POSG.*

Proof: Let T be the horizon of the POSG. If the initial state distribution of the POSG is not fixed, then the POSG can be thought of as a normal form game with hidden state. Theorem 1 implies that each time a policy tree is pruned by the DP algorithm, every strategy containing it as a subtree is very weakly dominated in this game. And if a strategy is very weakly dominated when the initial state distribution is not fixed, then it is certainly very weakly dominated for a fixed initial state distribution. Thus, the DP algorithm can be viewed as iteratively eliminating very weakly dominated strategies in the POSG.□

In the case of cooperative games, also known as DEC-POMDPs, removing very weakly dominated strategies preserves at least one optimal strategy profile. Thus, the multi-agent DP operator can be used to solve finite-horizon DEC-POMDPs optimally. When the DP algorithm reaches step T, we can simply extract the highest-valued strategy profile for the start state distribution.

Horizon	Brute force	Dynamic programming
1	(2, 2)	(2, 2)
2	(8, 8)	(6, 6)
3	(128, 128)	(20, 20)
4	*(32768, 32768)*	(300, 300)

Table 2. Performance of both algorithms on the multi-access broadcast channel problem. Each cell displays the number of policy trees produced for each agent. The brute force algorithm could not compute iteration 4. The numbers (in italics) shown in that cell reflect how many policy trees it would need to create for each agent.

Corollary 1 *Dynamic programming applied to a finite-horizon DEC-POMDP yields an optimal strategy profile.*

For general-sum POSGs, the DP algorithm converts the POSG to a normal form representation with reduced sets of strategies in which there are no very weakly dominated strategies. Although selecting an equilibrium presents a challenging problem in the general-sum case, standard techniques for selecting an equilibrium in a normal form game can be used.

4. Example

We ran initial tests on a cooperative game involving control of a multi-access broadcast channel [19]. In this problem, nodes need to broadcast messages to each other over a channel, but only one node may broadcast at a time, otherwise a collision occurs. The nodes share the common goal of maximizing the throughput of the channel.

The process proceeds in discrete time steps. At the start of each time step, each node decides whether or not to send a message. The nodes receive a reward of 1 when a message is successfully broadcast and a reward of 0 otherwise. At the end of the time step, each node receives a noisy observation of whether or not a message got through.

The message buffer for each agent has space for only one message. If a node is unable to broadcast a message, the message remains in the buffer for the next time step. If a node i is able to send its message, the probability that its buffer will fill up on the next step is p_i. Our problem has two nodes, with $p_1 = 0.9$ and $p_2 = 0.1$. There are 4 states, 2 actions per agent, and 2 observations per agent.

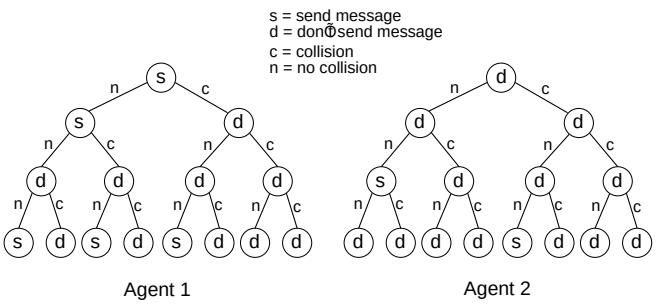

Figure 1. A pair of policy trees that is optimal for the horizon-4 problem when both message buffers start out full.

We compared our DP algorithm with a brute-force algorithm, which also builds sets of policy trees, but never prunes any of them. On a machine with 2 gigabytes of memory, the brute-force algorithm was able to complete iteration 3 before running out of memory, while the DP algorithm was able to complete iteration 4. At the end of iteration 4, the number of policy trees for the DP algorithm was less than 1% of the number that would have been produced by the brute-force algorithm, had it been able to complete the iteration. This result, shown in Table 2, indicates that the multi-agent DP operator can prune a significant number of trees. However, even with pruning, the number of policy trees grows quickly with the horizon. At the end of the fourth iteration, each agent has 300 policy trees that are not dominated. Because the piecewise linear and convex value function consists of one $|\mathcal{S}|$-vector for each pair of policy trees from the two agents, the representation of the value function requires 300^2 $|\mathcal{S}|$-vectors. In the fifth iteration, an exhaustive backup would create a value function that consists of $2 \cdot 300^4$ $|\mathcal{S}|$-vectors, or more than 16 billion $|\mathcal{S}|$-vectors, before beginning the process of pruning. This illustrates how the algorithm can run out of memory. In the next section, we discuss possible ways to avoid the explosion in size of the value function.

Figure 1 shows a pair of depth-4 policy trees constructed by the DP algorithm. In the case where the message buffers both start out full, this pair is optimal, yielding a total reward of 3.89.

5. Future work

Development of an exact dynamic programming approach to solving POSGs suggests several avenues for future research, and we briefly describe some possibilities.

5.1. Improving efficiency

A major scalability bottleneck is the fact that the number of policy trees grows rapidly with the horizon and can quickly consume a large amount of memory. There are several possible ways to address this. One technique that provides computational leverage in solving POMDPs is to prune policy trees incrementally, so that an exhaustive backup never has to be done [5]. Whether this can be extended to the multi-agent case is an open problem. Other techniques seem easier to extend. More aggressive pruning, such as pruning strategies that are *almost* very weakly dominated, can reduce the number of policy trees in exchange for bounded sub-optimality [6]. The number of policy trees may be reduced by allowing stochastic policies, as in [21]. Work on compactly represented POMDPs and value functions may be extended to the multi-agent case [9].

In addition, there exist POMDP algorithms that leverage a known start state distribution for greater efficiency. These algorithms perform a forward search from the start state and are able to avoid unreachable belief states. Whether some kind of forward search can be done in the multi-agent case is an important open problem.

5.2. Extension to infinite-horizon POSGs

It should be possible to extend our dynamic programming algorithm to infinite-horizon, discounted POSGs, and we are currently exploring this. In the infinite-horizon case, the multi-agent DP operator is applied to infinite trees. A finite set of infinite trees can be represented by a finite-state controller, and policy iteration algorithms for single-agent POMDPs have been developed based on this representation [8, 21]. We believe that they can be extended to develop a policy iteration algorithm for infinite-horizon POSGs. Because our definition of belief depends on explicit representation of a policy as a policy tree or finite-state controller, it is not obvious that a value iteration algorithm for infinite-horizon POSGs is possible.

6. Conclusion

We have presented an algorithm for solving POSGs that generalizes both dynamic programming for POMDPs and iterated elimination of dominated strategies for normal form games. It is the first exact algorithm for general POSGs, and we have shown that it can be used to find optimal solutions for cooperative POSGs. Although currently limited to solving very small problems, its development helps to clarify the relationship between POMDPs and game-theoretic models. There are many avenues for future research, in both making the algorithm more time and space efficient and extending it beyond finite-horizon POSGs.

Acknowledgments This work was supported in part by the National Science Foundation under grants IIS-0219606 and IIS-9984952, by NASA under cooperative agreement NCC 2-1311, and by the Air Force Office of Scientific Research under grant F49620-03-1-0090. Daniel Bernstein was supported by a NASA GSRP Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the NSF, NASA or AFOSR.

References

[1] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-independent decentralized Markov decision processes. In *Proceedings of the 2nd International Conference on Autonomous Agents and Multi-agent Systems*, pages 41–48, 2003.

[2] D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[3] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 478–485, 1999.

[4] R. Brafman and M. Tennenholtz. R-MAX–a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[5] A. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence*, pages 54–61, 1997.

[6] Z. Feng and E. Hansen. Approximate planning for factored POMDPs. In *Proceedings of the 6th European Conference on Planning*, 2001.

[7] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

[8] E. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, Madison, WI, 1998.

[9] E. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, pages 130–139, 2000.

[10] K. Hsu and S. I. Marcus. Decentralized control of finite state Markov processes. *IEEE Transactions on Automatic Control*, AC-27(2):426–431, April 1982.

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

[12] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[13] M. Kearns, Y. Mansour, and S. Singh. Fast planning in stochastic games. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 309–316, 2000.

[14] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 750–759, 1994.

[15] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, 1997.

[16] H. Kuhn. Extensive games and the problem of information. In H. Kuhn and A. Tucker, editors, *Contributions to the Theory of Games II*, pages 193–216. Princeton University Press, 1953.

[17] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

[18] R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 705–711, 2003.

[19] J. M. Ooi and G. W. Wornell. Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proceedings of the 35th Conference on Decision and Control*, pages 293–298, 1996.

[20] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 16th International Conference on Uncertainty in Artificial Intelligence*, pages 489–496, 2000.

[21] P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*. MIT Press, 2004.

[22] L. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39:1095–1100, 1953.

[23] R. Smallwood and E. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.