Point-Based Policy Generation for Decentralized POMDPs

Feng Wu
School of Computer Science
Univ. of Sci. & Tech. of China
Hefei, Anhui 230027 China
wufeng@mail.ustc.edu.cn

Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
shlomo@cs.umass.edu

Xiaoping Chen
School of Computer Science
Univ. of Sci. & Tech. of China
Hefei, Anhui 230027 China
xpchen@ustc.edu.cn

ABSTRACT

Memory-bounded techniques have shown great promise in solving complex multi-agent planning problems modeled as DEC-POMDPs. Much of the performance gains can be attributed to pruning techniques that alleviate the complexity of the exhaustive backup step of the original MBDP algorithm. Despite these improvements, state-of-the-art algorithms can still handle a relative small pool of candidate policies, which limits the quality of the solution in some benchmark problems. We present a new algorithm, Point-Based Policy Generation, which avoids altogether searching the entire joint policy space. The key observation is that the best joint policy for each reachable belief state can be constructed directly, instead of producing first a large set of candidates. We also provide an efficient approximate implementation of this operation. The experimental results show that our solution technique improves the performance significantly in terms of both runtime and solution quality.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and coordination, Multi-agent systems*

General Terms

Algorithms, Experimentation, Performance

Keywords

Teamwork, Coordination, Multi-Agent Planning, Decision-Theoretic Planning, Decentralized POMDPs

1. INTRODUCTION

Cooperative multi-agent decision making arises naturally in many real-world applications such as cooperative robots, planetary exploration, distributed sensor networks, and disaster response. These problems are difficult or impossible to solve using centralized decision making frameworks. In the RoboCup domain, for example, a group of robots with noisy sensors and inaccurate actuators must cooperate with each other to play soccer and win the game. With only partial view of the environment, each robot must reason about the choices of the others and how they may affect the environment. There are many sources of uncertainty in this problem and the state space is very large. Developing decision-theoretic techniques that can cope with this complexity is thus an important challenge.

The Markov decision process (MDP) and its partially observable counterpart (POMDP) have proved useful in planning and learning under uncertainty. A natural extension of these models to cooperative multi-agent settings is provided by the decentralized POMDP (DEC-POMDP) framework, in which agents have different partial knowledge of the environment and other agents. The DEC-POMDP framework is very expressive and can model many practical problems including the ones mentioned above. Unfortunately, solving it optimally has been shown to be NEXP-complete [9]. Thus, optimal algorithms [5, 14, 18, 19, 26, 27] can only solve very small problems. In recent years, researches proposed several approaches to improve the ability to solve larger problems. Examples include algorithms that exploit the structure of interaction in subclasses of DEC-POMDPs such as Transition Independent DEC-MDPs (TI-DEC-MDPs) [6] and Network Distributed POMDPs (ND-POMDPs) [17]. In other efforts researchers managed to address the complexity of the general model by considering communication explicitly [13, 15, 21]. However, not all real-world problems exhibit the necessary independence conditions, and communication is often costly and sometimes unavailable in the case of robots that operate underground or on other planets.

More general algorithms that compute approximate solutions have shown great promise using either offline methods [2, 3, 10, 11, 16, 23, 24] or online techniques [12, 22, 29]. Online algorithms must often meet tight time-constraints and the solution quality highly depends on the heuristics they use. There has also been substantial work on solving approximately DEC-POMDPs with infinite horizons [1,7,8]. In this paper, we focus on approximate offline algorithms for finite-horizon DEC-POMDPs. Currently, the state-of-the-art solution techniques still suffer from limited scalability.

The approach that is closest to our work is called Memory-Bounded Dynamic Programming (MBDP) [24]. It combines top-down and bottom-up components to build and optimize policies. The top-down component is used to generate a set of reachable belief states, usually guided by some heuristics. The bottom-up dynamic programming component is then used to build a set of possible policies based on the policies of the previous step. At the end of each step, only the best policies for the reachable belief states are kept as the building blocks for the next step. Since the number of policies kept

Cite as: Point-Based Policy Generation for Decentralized POMDPs, Feng Wu, Shlomo Zilberstein, Xiaoping Chen, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

at each step is bounded by a parameter called $maxTrees$, MBDP has a linear time and space complexity with respect to the horizon. However, the exhaustive backup that MBDP uses to build a set of possible policies is very inefficient. Several successor algorithms have been developed to alleviate this problem by performing a partial backup with respect to the observations, by focusing on the most likely observations in IMBDP [23] or compressing the set of observations in MBDP-OC [10]. More recently, there have been several moderately successful attempts to further improve the exhaustive backup. One example is PBIP [11] that replaces the backup step with a branch-and-bound search in the space of joint policies. Another example is IPG [3] that can prune useless policies before actually generating them. While these ideas have produced very significant computational savings, the resulting algorithms can still handle a relative small pool of candidate policies (measured by $maxTrees$). A small pool of policies is sometimes sufficient to obtain near optimal results, but in other cases it leads to a significant loss of value. Our goal in this paper is to introduce an algorithm that can operate more efficiently with a significantly larger pool of candidate policies. The objective is to produce better quality solutions much faster and to improve the overall scalability of approximate DEC-POMDP algorithms in order to solve large problems.

We present a new algorithm called Point-Based Policy Generation for DEC-POMDPs, which combines more efficiently the top-down and bottom-up components of the MBDP family of algorithms. MBDP produces a bounded pool of policies that are optimized with respect to a set of reachable belief states. The key observation behind the development of the new algorithm is that *the best policy for each reachable belief state can be constructed directly*, instead of producing first a large set of candidates. Thus, we first construct the joint policies based on a given belief state and each joint action, then select the best one for the given belief point. Consequently, we avoid altogether performing backup for all possible policies or searching over the entire joint policy space. We prove that when performed optimally, our procedure is equivalent to what MBDP does, resulting in the same policy value. We also provide an approximate implementation of this procedure that can solve the problem efficiently–much faster than the optimal version. The experimental results show that our solution technique works well. It significantly improves the performance of existing algorithms in terms of both runtime and solution quality over several DEC-POMDP benchmark problems.

The rest of the paper is organized as follows. We first introduce the DEC-POMDP model and the MBDP family of algorithms. Then we describe the main algorithm and analyze its properties. We then present and discuss the approximation technique. Finally, we examine the performance of the algorithm on several benchmark problems and demonstrate its efficiency. We conclude with a summary of the contributions and future work.

2. DECENTRALIZED POMDPS

We adopt here the DEC-POMDP framework and notation [9], however our approach and results apply to equivalent models such as MTDP [21] and POIPSG [20].

Definition 1. A finite-horizon Decentralized Partially Observable Markov Decision Process (DEC-POMDP) is defined as a tuple $\langle I, S, \{A_i\}, \{\Omega_i\}, P, O, R, b^0 \rangle$ where

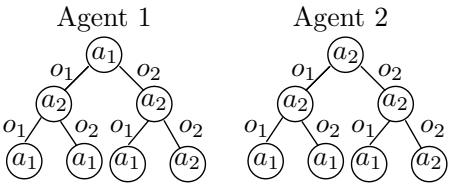

Figure 1: Example of a joint policy for 2 agents.

- I is a finite set of agents indexed $1, \cdots, n$.
- S is a finite set of system states.
- A_i is a finite set of actions available to agent i and $\vec{A} = \times_{i \in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, \cdots, a_n \rangle$ denotes a joint action.
- Ω_i is a finite set of observations available to agent i and $\vec{\Omega} = \times_{i \in I} \Omega_i$ is the set of joint observations, where $\vec{o} = \langle o_1, \cdots, o_n \rangle$ denotes a joint observation.
- P is a Markovian state transition table. $P(s'|s, \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state s results in a transition to state s'.
- O is a table of observation probabilities. $O(\vec{o}|s', \vec{a})$ denotes the probability of observing joint observation $\vec{o}$ after taking joint action $\vec{a}$ and reaching state s'.
- $R : S \times \vec{A} \rightarrow \Re$ is a reward function. $R(s, \vec{a})$ denotes the reward value obtained from taking a joint action $\vec{a}$ in state s.
- $b^0 \in \Delta(S)$ is the initial belief state distribution.

In this paper we focus on the general DEC-POMDP problem with a finite horizon T and start state distribution b^0. Solving this problem can be seen as finding policies that maximize the expected joint reward for b^0 over T. While execution is inherently distributed, planning is performed offline and can be centralized.

In DEC-POMDPs, the policy of an agent is represented as a tree and a joint policy as a vector of trees, one for each agent. As shown in Figure 1, each node of the policy tree is labeled by an action to take and each edge is labeled by an observation that may occur. This continues until the horizon T is reached at the leaf nodes. When executing a policy tree at runtime, the agent follows a path from the root to a leaf depending on the observations it receives as it performs the actions at the nodes. The value function of a joint policy $\vec{q}^{\,t+1}$ is defined recursively as follows:

$$V^{t+1}(\vec{q}^{\,t+1}, s) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V^t(\vec{q}^{\,t}_{\vec{o}}, s') \quad (1)$$

where $\vec{a}$ is the joint action at the root nodes of $\vec{q}^{\,t+1}$ and $\vec{q}^{\,t}_{\vec{o}}$ is the joint subtree of $\vec{q}^{\,t+1}$ after $\vec{o}$ is observed. The belief state used in this paper is a probability distribution over states $b \in \Delta(S)$. The value of a joint policy $\vec{q}$ for a belief state b is defined as $V(\vec{q}, b) = \sum_{s \in S} b(s) V(\vec{q}, s)$. A survey of the DEC-POMDP model and algorithms is available in [25].

3. MBDP AND ITS SUCCESSORS

Memory-Bounded Dynamic Programming (MBDP) [24] was the first algorithm to combine top-down and bottom-up solution techniques for DEC-POMDPs. In this section we describe previous work on MBDP and its successors, which

Algorithm 1: The MBDP Algorithm

$\vec{Q}^1 \leftarrow$ initialize all 1-step policy trees
for $t = 1$ **to** $T - 1$ **do**
  $\vec{Q'}^{t+1} \leftarrow$ perform full backup on $\vec{Q}^t$
  $\vec{Q'}^{t+1} \leftarrow$ prune dominated policies in $\vec{Q'}^{t+1}$
  $\vec{Q}^{t+1} \leftarrow \{\}$
  for $k = 1$ **to** $maxTrees$ **do**
    $b \leftarrow$ generate a belief using a heuristic portfolio
    $\vec{q} \leftarrow$ select the best joint policy in $\vec{Q'}^{t+1}$ for b
    $\vec{Q}^{t+1} \leftarrow \vec{Q}^{t+1} \cup \{\vec{q}\}$
return the best joint policy in $\vec{Q}^T$ for b^0

are the best existing solution techniques for finite-horizon DEC-POMDPs. In MBDP, policies are constructed incrementally using bottom-up dynamic programming. A parameter, $maxTrees$, is chosen to ensure that a full backup with this number of policies for the current step does not exceed the available memory. At each iteration, a full backup of the policies from the last iteration is performed. Then, top-down heuristics are selected from the portfolio to compute a set of reachable belief states. Finally, the best joint policies for these belief states are added to the new sets of policies. After the Tth backup, the best joint policy for the initial belief state is returned. The *best* joint policy is a joint policy with the highest value for a certain belief point. The main procedure is shown in Algorithm 1.

3.1 Backup Operations

The original MBDP uses *exhaustive backups* (or full backups) to construct policy trees. This operation generates every possible depth-t+1 policy tree for each action and each possible observation to the root node of some depth-t policy tree. If an agent has $|Q_i|$ depth-t policy trees, $|A_i|$ actions, and $|\Omega_i|$ observations, there will be $|A_i||Q_i|^{|\Omega_i|}$ depth-$t+1$ policy trees. This is inefficient because the number of possible depth-t+1 policy trees is still exponential in the size of the observation space. Thus, an improved version of MBDP (IMBDP) [23] was introduced to use *partial backups*. It first identifies the set of most likely observations for every agent bounded by a predefined number of observations $maxObs$, and then performs a backup with only these observations for each agent. The missing observation branches are filled up by using local search. Another approach to this problem is implemented in MBDP with *observation compression* (MBDP-OC) [10]. Instead of ignoring observations that are less probable, MBDP-OC merges certain sets of observations guided by the value lost. It seeks to reduce the exponential generation of new policies by forcing different branches of the new root policies to contain the same subtrees. The observation branches are merged so as to minimize the loss of value. Although the methods above can alleviate the complexity of the one-step backup operation, they are still time-consuming. We show in this paper that there is much to be gained if the bottom-up policy construction considers the reachable belief states generated by the top-down heuristics *from the very beginning*. The reason is that only the best policy trees for the reachable belief states are kept at the end of each iteration and most of them are useless.

3.2 Pruning Techniques

In order to reduce the the number of policy trees, the MBDP algorithm does pruning by using *iterated elimination of dominated policies* after each backup. A policy tree is dominated if for every possible belief state there is at least one other policy tree which is as good as or better than it. This test for dominance is performed using a linear program. Removing a dominated policy tree does not reduce the value of the optimal joint policy. Unfortunately, even with this pruning technique, the number of policy trees still grows quickly. To alleviate this problem, a new approach called *point-based incremental pruning* (PBIP) [11] was proposed, which uses branch-and-bound search in the space of joint policy trees instead. PBIP computes upper and lower bounds on the partial depth-t+1 joint policy trees using heuristics, and prunes dominated trees at earlier construction stages. The bounds are calculated by considering the belief states and the depth-t policy trees. PBIP prunes depth-t+1 policy trees that are outside the upper and lower bounds, but it does not exploit the reachability of policies.

3.3 Reachability Analysis

Recently, a new method called *incremental policy generation* (IPG) [3] was proposed to generate policies based on a state space reachability analysis. Intuitively, the action taken and observation seen may limit the possible next states no matter what actions the other agents perform. This allows only policies that are useful for some possible states to be retained. This approach may generate a smaller number of policies without losing value. It first generates all possible sets of depth-t trees for each observation with a fixed action, one for each agent. Then, it creates all possible depth-t+1 trees that begin with the fixed action followed by choosing any trees from the depth-t set after an observation is obtained. Once all depth-$t+1$ trees for each action are generated, it takes the union of the sets and produces the set of depth-t+1 trees. This approach can be incorporated with any DEC-POMDP algorithm that performs dynamic programming backups. While it exploits the state space reachability, this approach does not consider the reachable belief states generated by the top-down heuristics.

4. POINT-BASED POLICY GENERATION

As mentioned above, a better way to perform the bottom-up dynamic programming step is to construct the best joint policy for each belief state only–avoiding either full or partial backups. In this section, we propose a new method which generates directly the best joint policy for each belief state, namely *Point-Based Policy Generation* (PBPG).

4.1 Problem Formulation

Given a belief state b, the basic idea is as follow: for every joint action $\vec{a}$, we first find the best sub-policy trees for every possible observation branch after taking action $\vec{a}$; then, we choose the best joint action for b and build the best joint policy. Formally, this problem can be defined as follow:

Definition 2. Given a belief state b, a joint action $\vec{a}$, depth-t policy tree sets $\vec{Q}^t = \langle Q_1^t, Q_2^t, \cdots, Q_n^t \rangle$ and a value function $V^t : \vec{Q}^t \times S \rightarrow \Re$, find mappings $\delta_i : \Omega_i \rightarrow Q_i^t, \forall i \in I$ which maximize the depth-t+1 value function

$$V^{t+1}(\vec{a}, b) = R(\vec{a}, b) + \sum_{s', \vec{o}} Pr(\vec{o}, s' | \vec{a}, b) V^t(\vec{\delta}(\vec{o}), s') \quad (2)$$

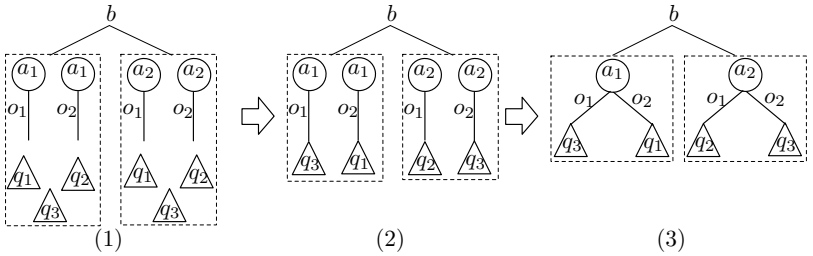

Figure 2: Example of the policy-tree construction process for two agents with two observations: (1) shows the input, which includes a belief state b, a joint action $\langle a_1, a_2 \rangle$, and sets of depth-t policy trees represented by the triangles; (2) shows the best mappings for the given belief state and joint action from each observation to a depth-t policy tree; (3) shows a joint policy tree built using the mappings.

where $\vec{\delta}(\vec{o}) = \langle \delta_1(o_1), \delta_2(o_2), \cdots, \delta_n(o_n) \rangle = \langle q_1^t, q_2^t, \cdots, q_n^t \rangle$, the probability $Pr(\vec{o}, s'|\vec{a}, b) = O(\vec{o}|s', \vec{a}) \sum_s P(s'|s, \vec{a}) b(s)$, and the reward function $R(\vec{a}, b) = \sum_s b(s) R(s, \vec{a})$.

This problem is essentially a subtree selection problem, where the goal is to choose depth-t subtrees in response to observations to maximize the depth-$t+1$ value function (Equation 2). However, the subtrees must be chosen in a decentralized manner. That is, subtree q_i^t is chosen based only on an observation of o_i for agent i. Our goal is to choose mappings or selection rules $\delta_i : \Omega_i \rightarrow Q_i^t, \forall i \in I$ to maximize the depth-$t+1$ value function and build the best depth-$t+1$ policy trees *given* the belief b and the joint action $\vec{a}$.

With the joint action $\vec{a}$ and the best mappings $\delta_i, \forall i \in I$, it is quite straightforward to construct the depth-$t+1$ policy trees. For agent i, the depth-$t+1$ policy tree can be built by using a_i as the root node and assigning the sub-policies for each observation branch based on δ_i. Figure 2 shows an example of the construction process with the resulting mappings $\delta_1 : o_1 \rightarrow q_3, o_2 \rightarrow q_1$ for agent 1 and $\delta_2 : o_1 \rightarrow q_2, o_2 \rightarrow q_3$ for agent 2. After the joint policy trees for b and every joint action are generated, we can find the best joint policy for the belief state b by choosing the one with the root nodes of $\vec{a}\,* = \langle a_1^*, \cdots, a_n^* \rangle$ computed as follows:

$$\vec{a}\,* = \arg\max_{\vec{a} \in \vec{A}} V^{t+1}(\vec{a}, b) \tag{3}$$

PROPOSITION 1. *For a given belief state b, the joint policy chosen by the method mentioned above yields the same value as the one selected by the MBDP algorithm.*

PROOF. Note that the depth-1 policy trees for both methods are the same with a single node of every possible action. Assume that the depth-t policy trees for both methods are also the same. For a belief state b, the MBDP algorithm first generates all possible depth-$t+1$ policy trees by exhaustive backup of the depth-t policies and then selects the joint policy $\vec{q}^{*t+1}$ which maximizes the value function

$$V^{t+1}(\vec{q}^{*t+1}, b) = \sum\nolimits_{s \in S} b(s) V^{t+1}(\vec{q}^{*t+1}, s) \tag{4}$$

where $V^{t+1}(\vec{q}^{*t+1}, s)$ is computed by Equation 1. The method described above first constructs a set of joint policy trees which maximize Equation 2 for every possible joint action and then chooses the joint policy with the joint action computed by Equation 3. Generally, we have

$$\begin{aligned} V^{t+1}(\vec{q}^{*t+1}, b) =& \sum_s b(s)[R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) \cdot \\ & O(\vec{o}|s', \vec{a}) V^t(\vec{q}_{\vec{o}}^{\,t}, s')] \qquad Eq.4 \\ =& \sum_s b(s) R(s, \vec{a}) + \sum_{s', \vec{o}} [O(\vec{o}|s', \vec{a}) \cdot \\ & \sum_s P(s'|s, \vec{a}) b(s)] V^t(\vec{q}_{\vec{o}}^{\,t}, s') \\ =& R(\vec{a}, b) + \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) V^t(\vec{q}_{\vec{o}}^{\,t}, s') \quad Eq.2 \\ =& V^{t+1}(\vec{a}, b) \end{aligned}$$

where $\vec{a}$ is the root nodes of $\vec{q}^{*t+1}$ and $\vec{q}_{\vec{o}}^{\,t}$ is the sub-policy trees of $\vec{q}^{*t+1}$ for observation branches $\vec{o}$.

Thus, the joint policies selected by both methods yield the same value for belief state b at depth-$t+1$. Therefore, the proposition holds for every depth by induction. □

4.2 Approximate Solution

The key question is how to compute the best mappings $\delta_i, \forall i \in I$. Note that the number of possible mappings is $(|Q_i^t|^{|\Omega_i|})^{|I|}$ for a fixed joint action $\vec{a}$. Therefore, the straightforward way to enumerate all possible mappings is very inefficient. Actually, this problem is equivalent to the *decentralized decision making* problem studied by Tsitsiklis and Athans, which has been proved to be NP-hard even for two agents [28]. In this paper, we propose an efficient approximate method which solves the problem using a linear program and produces suboptimal solutions.

The approximation technique uses stochastic mappings instead of deterministic ones. They are defined as follows:

$$\pi_i : \Omega_i \times Q_i^t \rightarrow \Re, \forall i \in I.$$

That is, $\pi_i(q_i^t|o_i)$ is a probability distribution of subtrees q_i, given observation o_i for agent i. Similar to $\vec{\delta}$, we denote the joint stochastic mapping $\vec{\pi} = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$. Note that given b and $\vec{a}$, $R(\vec{a}, b)$ is a constant in Equation 2. Therefore, maximizing Equation 2 is equivalent to maximizing the following function:

$$V^{t+1}(\vec{\delta}, b) = \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) V^t(\vec{\delta}(\vec{o}), s') \tag{5}$$

With the stochastic mappings, Equation 5 can be rewritten as follows:

$$V^{t+1}(\vec{\pi}, b) = \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) \sum_{\vec{q}^t} \prod_i \pi_i(q_i^t|o_i) V^t(\vec{q}^{\,t}, s') \tag{6}$$

Table 1: The linear program for optimizing π_i

Variables: $\varepsilon, \pi_i'(q_i^t \mid o_i)$
Objective: maximize ε
Subject to:
Improvement constraint:
$V^{t+1}(\vec{\pi}, b) + \varepsilon \leq \sum_{s',\vec{o}} Pr(\vec{o}, s' \mid \vec{a}, b) \sum_{\vec{q}} \pi_i'(q_i^t \mid o_i) \cdot \pi_{-i}(q_{-i}^t \mid o_{-i}) V^t(\vec{q}^{\,t}, s')$
Probability constraints:
$\forall o_i \in \Omega_i, \sum_{q_i^t \in Q_i^t} \pi_i'(q_i^t \mid o_i) = 1;$
$\forall o_i \in \Omega_i, q_i^t \in Q_i^t, \pi_i'(q_i^t \mid o_i) \geq 0.$

There are multiple ways to solve Equation 6 and calculate the solution $\pi_i, \forall i \in I$. Our approximate method computes a suboptimal solution by choosing initial parameters for $\pi_i, \forall i \in I$ and iteratively optimizing the parameters of one agent while leaving the parameters of the other agents fixed, until no improvement is obtained. This process always terminates after a finite number of iterations if a threshold of minimum improvement is set. The suboptimal solution computed in this way can be considered analogous to a Nash equilibrium where no agent can benefit unilaterally.

To start, each local stochastic mapping $\pi_i, i \in I$ is initialized to be deterministic, by selecting a random $q_i^t \in Q_i^t$ with a uniform distribution. Then, each agent is selected in turn and its policy is improved while keeping the other agents' policies fixed. This is done for agent i by finding the best parameters $\pi_i'(q_i^t \mid o_i)$ satisfying the following inequality:

$$V^{t+1}(\vec{\pi}, b) \leq \sum_{s',\vec{o},\vec{q}^{\,t}} Pr(\vec{o}, s' \mid \vec{a}, b) \pi_i'(q_i^t \mid o_i) \pi_{-i}(q_{-i}^t \mid o_{-i}) V^t(\vec{q}^{\,t}, s')$$

where $\pi_{-i}(q_{-i}^t \mid o_{-i}) = \prod_{k \neq i} \pi_k(q_k^t \mid o_k)$.

The linear program shown in Table 1 is used to find the new parameters. The procedure terminates and returns $\vec{\pi}$ when ε becomes sufficiently small for all agents. Random restarts are used to avoid local maxima.

4.3 The PBPG Algorithm

Once the stochastic mapping π_i is computed for agent i, q_i^t is selected for observation branch o_i according to the distribution $\pi_i(q_i^t \mid o_i)$. The main steps of PBPG are shown in Algorithm 2. The number of joint policy trees generated at each iteration is bounded by $(|\vec{A}| maxTrees)$, much less than $(|A_i| maxTrees^{|\Omega_i|})^{|I|}$ produced by the full backup in the original MBDP. It is worth pointing out that the policy evaluation is very time-consuming and can easily run out of memory, especially for large problems. It computes the value for every *joint policy* at every state as shown in Equation 1. Note that the number of joint policies actually evaluated here is $(|\vec{A}| maxTrees + maxTrees^{|I|})$, not the total enumeration $(|A_i| maxTrees)^{|I|}$. We evaluate the joint policy for each joint action and prune the dominated ones at an early stage. This makes the algorithm more efficient. Efficiency can also be improved by exploiting the sparsity of the transition matrix, observation matrix and belief vector, which often have many zero elements. This property can be used to solve the necessary equations more quickly.

We also use a heuristic portfolio to generate the belief b at the beginning of each iteration, just as MBDP does. A heuristic portfolio is a set of heuristics which can be used to compute a set of belief states. Each heuristic is used to select a subset of the policy trees. In our implementation, we use two types of heuristics: the MDP heuristic and the random heuristic. The MDP heuristic is based on solving the underlying MDP in a centralized manner and executing the resulting policy to create sample belief states. The random heuristic produces random reachable belief points using random policies. Sometimes, several belief points select the same policy tree because the sampled beliefs are quite close to each other. When that happens, we re-sample to generate different belief points. Unlike MBDP, it is not necessary to run our algorithm *recursively* to obtain good results. The experimental results show significant improvement over all the existing algorithms simply by using a portfolio containing the above two simple heuristics in our algorithm.

Algorithm 2: Point-Based Policy Generation

```
T ← horizon of the DEC-POMDP model
maxTrees ← max number of trees at each step
Q⃗^1 ← initialize and evaluate all 1-step policy trees
for t = 1 to T − 1 do
    Q⃗^{t+1} ← {}
    for k = 1 to maxTrees do
        b ← generate a belief using a heuristic portfolio
        ν* ← −∞
        for a⃗ ∈ A⃗ do
            π⃗* ← compute the best mappings with b, a⃗
            q⃗ ← build a joint policy tree based on a⃗, π⃗*
            ν ← evaluate q⃗ by given the belief state b
            if ν > ν* then q⃗* ← q⃗, ν* ← ν
        Q⃗^{t+1} ← Q⃗^{t+1} ∪ {q⃗*}
    evaluate every joint policy in Q⃗^{t+1} with Equation 1
q⃗*^T ← select the best joint policy from Q⃗^T for b^0
return q⃗*^T
```

4.4 Summary and Discussion

To summarize, MBDP does an exhaustive backup for all depth-t policy trees before selecting a joint policy for each belief state. IMBDP ignores less-likely observation branches and MBDP-OC merges observations while minimizing the loss of value. PBIP prunes depth-$t+1$ policy trees using upper and lower bounds at the early stage, and IPG limits the number of depth-t policies using state reachability.

Unlike these existing algorithms, which try to improve the performance of the *backup* operation by limiting the number of observations or policies, we completely replace the backup step with an efficient policy generation method. Instead of generating a large set of policy trees, our approach constructs only a small set of possible candidates for each belief state using an efficient linear program. The number of policy trees generated for each belief state is bounded by the number of joint actions.

It is possible to incorporate previously developed methods with our algorithm, particularly observation compression and incremental policy generation, to further improve its performance. Merging equivalent observations and limiting the size of possible subtrees can immediately reduce the number of variables in the linear program and improve its scalability. Implementing these improvements, however, is beyond the scope of the paper and is left for future work.

5. EXPERIMENTAL EVALUATION

We tested our algorithm on the hardest existing benchmark problems. PBPG shares the same linear time and

space properties of MBDP with respect to the horizon. Therefore, our experiments focus mainly on scalability with respect to $maxTrees$, and the value gains that result from increasing $maxTrees$. We compared our results mainly to PBIP-IPG [3] – the latest algorithm, which has outperformed the other existing algorithms such as MBDP, IMBDP, MBDP-OC and PBIP. The purpose of these experiments is to show that our algorithm can solve problems with a much larger number of $maxTrees$ with good runtime and solution quality. Actually, the $maxTrees$ parameter presents a good way to tradeoff between runtime and solution quality, and it makes it possible to design a *contract* anytime algorithm for DEC-POMDPs [30]. The experiments with different values of $maxTrees$ illustrate the advantage of our algorithm as a contract algorithm.

5.1 Experimental Setting

In the experiments, we use two types of heuristics – the random policy heuristic and the MDP policy heuristic. With the random policy heuristic, the agents act randomly top-down to the current step and sample a set of belief states. With the MDP policy heuristic, the agents act according to a pre-computed MDP policy of the model. For the fairness of comparison, we used the same heuristic portfolio as the first recursion of other MBDP-based algorithms (MDP: 45%, Random: 55%). They run recursively using the complete solution of the previous recursion as a new part of the heuristic portfolio for the next run. However, as we demonstrate below, the quality of results produced by our algorithm could be further improved by using a better heuristic portfolio.

We performed re-sampling up to 10 times if the sizes of both agents' policies were less than $maxTrees$. Due to the randomness of the sampling, we ran the algorithm 10 times per problem and reported the average runtime and value. All timing results are CPU times with a resolution of 0.01 second. Many of the parameter settings we used are too large for PBIP-IPG to be able to produce an answer with a reasonable amount of time. An "x" in Table 2 means that the algorithm cannot solve the problem within 12 hours. As a reference, we also provide for each test problem an upper bound on the value based on solving the underlying full-observable MDP (V_{MDP}). Notice that this is a loose bound [18]. When the results we obtain are close to this value, we can be sure that they are near optimal. But when there is a big difference, it is not possible to know how close to optimal we are. PBPG was implemented in Java 1.5 and ran on a Mac OSX machine with 2.8GHz Quad-Core Intel Xeon CPU and 2GB of RAM available for JVM. Linear programs were solved using lp_solve 5.5 [1].

5.2 Test Problems and Results

The Meeting in a 3×3 Grid domain [8] is a classical benchmark for DEC-POMDP algorithms. In this domain, two robots navigate on a grid world and try to stay as much time as possible in the same grid location. We adopted the version of the problem used by Amato *et al.* [3], which has 81 states, 5 actions and 9 observations per robot. As shown in Table 2, our algorithm took much less time than PBIP-IPG with the same number of $maxTrees$ and got competitive values. Even with a relatively large number of $maxTrees$, e.g. $maxTrees$=20, our algorithm still ran faster than PBIP-IPG with $maxTrees$=3 and produced better value as expected.

[1] http://lpsolve.sourceforge.net/5.5/

Table 2: Experimental Results (10 trials)

$maxTrees$	PBIP-IPG		PBPG	
	Time	Value	Time	Value
Meeting in a 3×3 Grid, $\|S\| = 81,\ \|O\| = 9,\ T = 100$				
3	3084s	92.12	27.21s	87.01
10	x	x	201.50s	93.46
20	x	x	799.90s	93.90
50	x	x	8345.13s	94.79
100	x	x	34820.67s	95.42
$V_{MDP} = 96.08$				
Cooperative Box Pushing, $\|S\| = 100,\ \|O\| = 5,\ T = 100$				
3	181s	598.40	11.34s	552.79
10	x	x	69.12s	715.95
20	x	x	287.42s	815.72
50	x	x	2935.24s	931.43
100	x	x	19945.56s	995.50
$V_{MDP} = 1658.25$				
Stochastic Mars Rover, $\|S\| = 256,\ \|O\| = 8,\ T = 20$				
3	14947s	37.81	12.47s	41.28
10	x	x	59.97s	44.30
20	x	x	199.45s	45.48
50	x	x	987.13s	47.15
100	x	x	5830.07s	48.41
$V_{MDP} = 65.11$				
Grid Soccer 2×3, $\|S\| = 3843,\ \|O\| = 11,\ T = 20$				
3	x	x	10986.79s	386.53
$V_{MDP} = 388.65$				

PBPG could solve this problem with $maxTrees$=100 while PBIP-IPG ran out of time with $maxTrees \geq 10$. Note that this problem has 9 observations, which makes it more difficult to solve than the following two benchmark problems. The result for $maxTrees$=100 was actually near-optimal considering the loose upper bound V_{MDP}.

The Cooperative Box Pushing problem [23] is another common benchmark problem for DEC-POMDPs. In this domain, two agents are pushing three boxes (1 large and 2 small) in a 3×4 grid. The agents will get a very high reward if they cooperatively push the large box into the goal area together. This domain has 100 states and each agent has 4 actions and 5 observations. The results have been shown in Table 2. With $maxTrees$=3, our algorithm ran ten times faster than PBIP-IPG but got competitive value. As the number of $maxTrees$ increases, the solution became better as expected. In the experiments, we observed that with $maxTrees$=100, 99% of the iterations would do re-sampling about 10 times. It means that the heuristic portfolio has reached its limit to construct different policy trees when $maxTrees$=100. The frequent re-sampling in this case contributed to an increase in the runtime of PBPG. It is quite likely that 999.53, the highest value we got with $maxTrees$=100 in these experiments, is quite close to the optimal value for this problem with horizon 100.

The Stochastic Mars Rover problem [4] is a larger domain with 256 states, 6 actions and 8 observations for each agent. The runtime of PBIP-IPG is substantially larger than in the previous benchmarks due to the larger state space. We used in these experiments T=20 because it takes too much time for PBIP-IPG to solve problems with T=100. The horizon we used is the same as in the original PBIP-IPG paper [4]. In contrast, our algorithm scales better over the state space as well as the action and observation spaces. Surprisingly, our

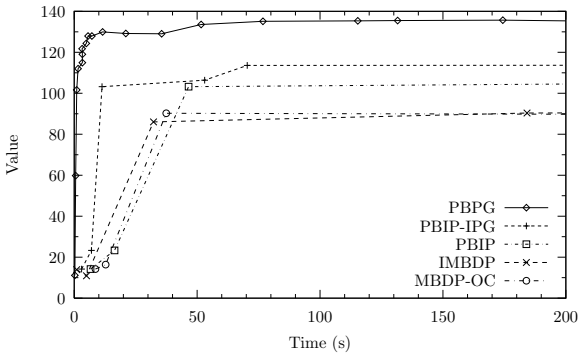

Figure 3: Value vs. Runtime for Cooperative Box Pushing with T=10. In these experiments, different algorithms used different numbers of $maxTrees$.

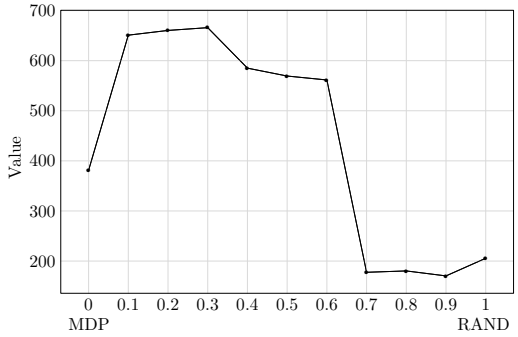

Figure 4: Values with different heuristic portfolios for the Cooperative Box Pushing problem with T=100, $maxTrees$=3. The composition of the portfolio changes gradually from the MDP heuristic only (left) to the random heuristic only (right).

algorithm can solve this problem rather quickly. Compared with the results in the Cooperative Box Pushing domain, runtime did not grow up substantially despite the fact that there are twice as many states. The main reason is that the transition and observation matrixes of this problem are quite sparse. Our algorithm with $maxTrees$=100 is still faster than PBIP-IPG with $maxTrees$=3. Again, the larger number of $maxTrees$ helps produce better solutions.

To test scalability, we also tried a more challenging problem introduced in [29], namely Grid Soccer 2×3. This problem has 3843 states, 6 actions and 11 observations. Currently, this problem can only be solved approximately by online algorithms. This is the first time that a problem of this size is solved successfully by an offline algorithm. Although online algorithms are generally very fast, offline algorithms can often provide higher solution quality. Besides, online algorithms often require the ability to communicate at runtime, which may be very costly or impossible in some domains. Offline algorithms have the added advantage of producing a coordinated joint policy prior to execution, reducing the need to communicate. In fact, our algorithm could get a value of 386.53 without any communication, which is near optimal. In comparison, the leading online algorithm called MAOP-COMM can only produce a value of 290.6 (without considering the cost of communication) while using communication 14.8% of the time on average [29].

The parameter $maxTrees$ does not only limit the usage of memory but also provides a tradeoff between the solution quality and runtime. Intuitively, an algorithm with larger $maxTrees$ will produce better value but also take longer time to execute. In our experiments, we tried different $maxTrees$ and recoded the runtime and value of each algorithm for the Cooperative Box Pushing domain with T=10. As shown in Figure 3, our algorithm got better value by given the same amount of time. Notice that the MDP upper bound [18] (V_{MDP}) of this problem with T=10 is 175.13. Thus, our algorithm performs quite well with respect to the value of the solutions. It is worth pointing out that all the algorithms we compared have linear time and space complexity with respect to the horizon. We chose a short horizon (T=10) here to make it possible for the other algorithms to solve the problem with different $maxTrees$ (1 to 8 as we tested, runtime >200s is not shown) in a reasonable amount of time; otherwise, they would run out of time very quickly.

In Figure 4 we show the results for the Cooperative Box Pushing problem with different heuristic portfolios. The x-coordinate indicates the frequency in which the random heuristic is used. The leftmost point (0) indicates that the random heuristic is not used at all and the MDP heuristic is used all the time. The rightmost point (1) indicates that the random heuristic is used all the time and the MDP heuristic is not used at all. We can see that the best heuristic portfolio for this domain is obtained at x=0.3. That is, the best results are obtained when 30% of the policies are selected using the random heuristic, and 70% using the MDP heuristic. The default portfolio used in the main experiments is (MDP: 45%, Random: 55%), which is also the portfolio used by the first recursion of other MBDP-based algorithms. Therefore, we could actually further improve the performance in Table 2 by using a better heuristic portfolio.

6. CONCLUSIONS

We present the point-based policy generation algorithm for finite-horizon DEC-POMDPs. Similar to previous MBDP-based algorithms, it also combines top-down heuristics and bottom-up dynamic programming to construct joint policy trees for an initial belief state. Our approach also uses the parameter $maxTrees$ to limit the usage of memory, thus it shares the same linear time and space complexity with respect to the horizon. The main contribution is a new point-based policy generation technique that builds the joint policies directly at each iteration, instead of performing a complex backup operation. By using this technique, many more policy trees can be kept as building blocks for the next iteration compared to the state-of-the-art algorithms. With a larger number of candidate subtrees, the solution quality can be further improved. Even when used with the same number of $maxTrees$, our algorithm runs orders of magnitude faster and produces competitive values in all the domains we tested. One important characteristic of the new algorithm is that it scales better over the state space and it can solve larger problems than currently possible with existing offline techniques. The experimental results show that the new PBPG algorithm significantly outperforms all the state-of-the-art algorithms in all the domains we tested.

In future work, we plan to incorporate other general methods such as observation compression and state reachability analysis into our algorithm to solve even larger problems. Currently, one limitation of our algorithm is that ev-

ery joint observation must be considered in order to model the problem as several linear programs. For some problems with large observation sets, additional techniques will have to be employed to further improve scalability. We are also investigating ways to learn the best heuristic portfolio automatically for different domains. The algorithm proposed in this paper eliminates much of the time- and space-consuming backup operation and opens up new research directions for developing effective approximation algorithms for multi-agent planning under uncertainty.

Acknowledgments

We thank Christopher Amato for providing the code of PBIP-IPG and the anonymous reviewers for their helpful comments. This work was supported in part by the China Scholarship Council, the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0181, the National Science Foundation under Grant No. IIS-0812149, the Natural Science Foundations of China under Grant No. 60745002, and the National Hi-Tech Project of China under Grant No. 2008AA01Z150.

7. REFERENCES

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 2009.

[2] C. Amato, A. Carlin, and S. Zilberstein. Bounded Dynamic Programming for Decetralized POMDPs. In *AAMAS 2007 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2007.

[3] C. Amato, J. S. Dibangoye, and S. Zilberstein. Incremental Policy Generation for Finite-Horizon DEC-POMDPs. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling*, pages 2–9, 2009.

[4] C. Amato and S. Zilberstein. Achieving goals in decentralized POMDPs. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 593–600, 2009.

[5] R. Aras, A. Dutech, and F. Charpillet. Mixed Integer Linear Programming for Exact Finite-Horizon Planning in Decentralized Pomdps. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling*, pages 18–25, 2007.

[6] R. Becker, S. Zilberstein, V. R. Lesser, and C. V. Goldman. Transition-independent Decentralized Markov Decision Processes. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 41–48, 2003.

[7] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.

[8] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded Policy Iteration for Decentralized POMDPs. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence*, pages 1287–1292, 2005.

[9] D. S. Bernstein, S. Zilberstein, and N. Immerman. The Complexity of Decentralized Control of Markov Decision Processes. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, pages 32–37, 2000.

[10] A. Carlin and S. Zilberstein. Value-based observation compression for DEC-POMDPs. In *Proc. of the 7th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 501–508, 2008.

[11] J. S. Dibangoye, A. Mouaddib, and B. Chaib-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 569–576, 2009.

[12] R. Emery-Montemerlo, G. J. Gordon, J. G. Schneider, and S. Thrun. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 136–143, 2004.

[13] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 137–144. ACM, 2003.

[14] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. In *Proc. of the 19th National Conf. on Artificial Intelligence*, pages 709–715, 2004.

[15] R. Nair, M. Tambe, M. Roth, and M. Yokoo. Communication for Improving Policy Computation in Distributed POMDPs. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 1098–1105, 2004.

[16] R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence*, pages 705–711, 2003.

[17] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In *Proc. of the 20st National Conf. on Artificial Intelligence*, pages 133–139, 2005.

[18] F. A. Oliehoek and N. Vlassis. Q-value functions for decentralized POMDPs. In *Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 833–840, 2007.

[19] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 577–584, 2009.

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

[21] D. V. Pynadath and M. Tambe. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

[22] M. Roth, R. G. Simmons, and M. M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proc. of the 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 786–793. ACM, 2005.

[23] S. Seuken and S. Zilberstein. Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. In *Proc. of the 23rd Conf. in Uncertainty in Artificial Intelligence*, 2007.

[24] S. Seuken and S. Zilberstein. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *Proc. of the 20th Int.l Joint Conf. on Artificial Intelligence*, pages 2009–2015, 2007.

[25] S. Seuken and S. Zilberstein. Formal Models and Algorithms for Decentralized Decision Making under Uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[26] D. Szer and F. Charpillet. Point-based Dynamic Programming for DEC-POMDPs. In *Proc. of the 21st National Conf. on Artificial Intelligence*, pages 1233–1238, 2006.

[27] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proc. of the 21st Conf. on Uncertainty in Artificial Intelligence*, pages 576–590, 2005.

[28] J. Tsitsiklis and M. Athans. On the complexity of decentralized decision making and detection problems. *IEEE Transaction on Automatic Control*, 30:440–446, 1985.

[29] F. Wu, S. Zilberstein, and X. Chen. Multi-Agent Online Planning with Communication. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling*, pages 321–328, 2009.

[30] S. Zilberstein. Optimizing Decision Quality with Contract Algorithms. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1576–1582, 1995.