Incremental Policy Generation for DEC-POMDPs

Christopher Amato and **Shlomo Zilberstein**
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{camato,shlomo}@cs.umass.edu

Abstract

Developing optimal algorithms for DEC-POMDPs is an important challenge. Recently, an ϵ-optimal algorithm for the infinite-horizon case was presented, but this approach is very resource intensive. Consequently, the algorithm cannot find high-quality solutions within the available amount of time and memory. As an alternative, we have developed a more efficient algorithm that is able to limit the search process based on the structure of the problem. This method is more scalable than the previous approach, while maintaining the optimality guarantee. We demonstrate that this improved scalability allows higher-valued solutions to be found for a range of problems. To further improve scalability, we also provide a heuristic extension which can solve larger problems or produce better solution quality with the given resources.

1 Introduction

Determining decentralized solutions to sequential decision making problems is a fundamental research area. A general way to model these problems is the decentralized partially observable Markov decision process (DEC-POMDP). In a DEC-POMDP, the reward function and state of the problem depend on the actions of all agents. Also, each agent may receive uncertain information about the state of the system at each step. Then, using only their own local information, the agents must choose actions. The goal is to maximize the collective value accumulated over the length of the problem.

There have been several algorithms developed recently for solving the infinite-horizon DEC-POMDP, which continues for an infinite number of steps. These include approximate approaches [Amato *et al.*, 2007; Bernstein *et al.*, 2005; Szer and Charpillet, 2005] and an ϵ-optimal approach [Bernstein *et al.*, 2008]. The approximate approaches can perform well on the chosen test problems, but offer no bound on optimality. Bernstein et al.'s policy iteration approach will converge within ϵ of optimal in a finite number of steps, but due to very high resource requirements, it often exhausts resources many steps before this bound is achieved.

To improve the efficiency of ϵ-optimal dynamic programming for infinite-horizon DEC-POMDPs, a better way of generating policies for the set of agents is needed. The current approach exhaustively generates all possibilities for a given step and removes those that have lower value for all states of the problem and all policies of the other agents. Because the value of a policy depends on the other agent policies, it is not obvious how to eliminate policies without exhaustive generation. To combat this problem, we propose generating policies for each agent based on those that are useful for a given action and observation. The action taken and observation seen may limit the possible next states of the system no matter what actions the other agents choose, allowing only policies that are useful for these possible states to be retained. This approach may allow a smaller number of policies to be generated, while maintaining ϵ-optimality. Because solutions are built up for each action and observation, we call our method *incremental policy generation*. The incremental nature of our algorithm allows a larger number of steps to be completed and thus can provide higher solution quality.

In order to further improve scalability, we have also developed a heuristic version of our approach which makes use of start state information to further reduce the number of policies considered for each agent. While the heuristic algorithm can no longer guarantee optimality, it is able produce higher values on our set of test problems. It is worth noting that any DEC-POMDP algorithm that uses dynamic programming backups, such as [Hansen *et al.*, 2004; Seuken and Zilberstein, 2007], can be made more efficient by implementing our approach. In general, an incremental backup can be conducted with much less time and memory than an exhaustive one. This could reduce the resource usage of these algorithms, allowing larger problems to be solved and improving the solution quality on other problems.

The remainder of the paper is organized as follows. We first provide background on the DEC-POMDP model and Bernstein et al.'s policy iteration algorithm. We then present the incremental policy generation approach and proof that it maintains the optimality guarantee. Our heuristic extension is described in the next section. Finally, experimental results showing the improved efficiency of the optimal and heuristic algorithms are then provided. These results suggest that a step has been taken to reduce a major bottleneck in dynamic programming algorithms for DEC-POMDPS.

2 Background

We first discuss the DEC-POMDP model and representing solutions as finite-state controllers. We then describe the infinite-horizon policy iteration algorithm.

2.1 DEC-POMDPs

A DEC-POMDP can be defined with the tuple $\langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O, T\rangle$ with: I, a finite set of agents, S, a finite set of states with designated initial state distribution b_0, A_i, a finite set of actions for each agent, i, P, a set of state transition probabilities: $P(s'|s, \vec{a})$, the probability of transitioning from state s to s' when the set of actions $\vec{a}$ are taken by the agents, R, a reward function: $R(s, \vec{a})$, the immediate reward for being in state s and taking the set of actions $\vec{a}$, Ω_i, a finite set of observations for each agent, i, and O, a set of observation probabilities: $O(\vec{o}|s', \vec{a})$, the probability of seeing the set of observations $\vec{o}$ given the set of actions $\vec{a}$ was taken which results in state s'.

A DEC-POMDP involves multiple agents that operate under uncertainty based on different streams of observations. An infinite-horizon DEC-POMDP unfolds over an infinite sequence of stages. At each stage, every agent chooses an action based purely on its local observations, resulting in an immediate reward for the set of agents and an observation for each individual agent. Because the state is not directly observed, it may be beneficial for each agent to remember its observation history. A *local policy* for an agent is a mapping from local observation histories to actions while a *joint policy* is a set of policies, one for each agent in the problem. The goal is to maximize the infinite-horizon total cumulative reward, beginning at some initial distribution over states b_0. A discount factor, γ, is used to maintain finite sums.

As an elegant way to represent infinite-horizon solutions, stochastic finite-state controllers can be used. Controllers are natural infinite-horizon representations and stochasticity allows better use of finite memory. Each controller can formally be defined by the tuple $\langle Q, \psi, \eta\rangle$, where Q is the finite set of controller nodes. The parameter $\psi : Q \to \Delta A$ is the action selection model for each node, which defines a probability that an action will be chosen in a given node. The parameter $\eta : Q \times A \times O \to \Delta Q$ represents the node transition model for each node. This provides the probability of transitioning from one node to another after an action has been taken and an observation has been seen. For n agents, the value for being at nodes $\vec{q}$ and state s is

$$V(\vec{q}, s) = \sum_{\vec{a}} \prod_{i}^{n} P(a_i|q_i) \Bigg[R(s, \vec{a}) + \gamma \sum_{s'} P(s'|\vec{a}, s) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q'}} \prod_{i}^{n} P(q_i'|q_i, a_i, o_i) V(\vec{q'}, s') \Bigg]$$

These values can be calculated offline in order to find controllers that can be executed online for distributed control.

2.2 Policy iteration for DEC-POMDPs

Recently, an ϵ-optimal algorithm for solving infinite-horizon DEC-POMDPs has been developed [Bernstein *et al.*, 2008]. This approach is an extension of policy iteration for POMDPs [Hansen, 1998] to the multiagent case. The algorithm alternates between improving the controllers of a set of agents and evaluating them. Each agent begins with an arbitrary one node controller. In the improvement phase, an *exhaustive backup* adds new nodes to each agent's controller that represent all possible initial actions and after each observation is seen, deterministic transitions into the current controller. Evaluation is then conducted for each possible state of the system and each possible choice of initial nodes for the set of agents. Nodes that have lower value for all states of the problem and initial nodes of the other agents are then pruned.

The linear program used to determine dominated nodes maximizes the variable ϵ given:

$$\forall s, q_{-i} \quad V(s, q_i, q_{-i}) + \epsilon \leq \sum_{\hat{q}_i} x(\hat{q}_i) V(s, \hat{q}_i, q_{-i})$$

where the variable $x(\hat{q}_i)$ represents the probability of beginning in node $\hat{q}_i$ rather than q_i for agent i. Constraints are also added to ensure $x(\hat{q}_i)$ is a proper probability. Thus, for a node q_i if ϵ is nonnegative, then q_i can be replaced with the distribution of nodes given by x and value is at least maintained for each state and initial node of the other agents. After transitions to the pruned nodes are redirected to the dominating distribution, the updated controller is evaluated and pruning continues until no agent can remove further nodes. Backups and pruning are conducted until the the value no longer changes due to to discounting. This assures the process will converge in a finite number of steps to a solution within ϵ of optimal for any initial state.

3 Incremental policy generation

One of the major bottlenecks of policy iteration is the large memory use due to exhaustive backups. If policies for the agents can be generated without first exhaustively generating all next step possibilities, the algorithm would become more scalable. Methods such as incremental pruning [Cassandra *et al.*, 1997] have been developed for POMDPs to combat this problem, but they cannot be directly extended to DEC-POMDPs. This is due to the fact that the state of the problem and the value of an agent's policy depend on the policies of all agents, requiring all next step policies for the other agents to be known before incremental pruning could be applied.

As an alternative, we build up the policies for each agent by using a one-step reachability analysis. After an agent chooses an action and sees an observation, the agent may not know the state of the world, but it can often determine which states are possible given the current information. For instance, assume an agent has an observation in which a wall is perfectly observed when it is to the left or right. If the agent sees the wall on the right, it may not know the exact state it is in, but it can limit the possibilities to those states with walls on the right. Likewise, in the commonly used two agent tiger domain [Nair *et al.*, 2003], after an agent opens a door, the problem transitions back to the start state of the tiger equally likely to be located behind either door. Thus, after an open action is performed, the agent knows the exact state of the world.

So, how can we use this information? One way is to limit the policies that are generated during policy iteration. By using the state information provided by taking an action and

Algorithm 1: Incremental policy generation

input : A set controllers for all agents, C and an agent index, i
output: A backed up controller for the given agent, C_i'
begin
  for *each action,* a **do**
    for *each observation,* o **do**
      $S' \leftarrow$ *possibleStates*(a, o)
      $C_i^{a,o} \leftarrow$ *usefulTrans*(S', C)
    $C_i^a \leftarrow \oplus_o C_i^{a,o}$
  $C_i' \leftarrow \cup_a C_i^a$
  return C_i'
end

seeing an observation, these polices do not need to be built up exhaustively. Instead, we can determine which states are possible after an agent chooses an action and sees an observation. Only subpolicies that are useful for the resulting states need to be considered for that action and observation pair.

We first discuss determining which states are possible after an action is taken and an observation seen. We then describe the incremental policy generation algorithm in more detail, including proof that it will provide an ϵ-optimal solution.

3.1 Limiting the state

Any agent can calculate the possible states that result from taking an action and seeing an observation. To determine this next state exactly, the agent would generally need to know the probability of the previous state and of the agents selecting distributions of actions. Since we do not assume the agents possess this information, the exact state can only be known in some circumstances, but the set of possible next states can often be limited.

For instance, given probability of the current state and that other agents will choose actions, $P(\vec{a}_{-i}, s|a_i)$, then the probability of a resulting state s' after agent i chooses action a_i and observes o_i is determined by,

$$P(s'|a_i, o_i) = \frac{\sum_{\vec{a}_{-i}, \vec{o}_{-i}, s} P(\vec{o}|s, \vec{a}, s') P(s'|s, \vec{a}) P(\vec{a}_{-i}, s|a_i)}{P(o_i|a_i)} \tag{1}$$

where the normalizing factor is:

$$P(o_i|a_i) = \sum_{\vec{a}_{-i}, \vec{o}_{-i}, s, s'} P(\vec{o}|s, \vec{a}, s') P(s'|s, \vec{a}) P(\vec{a}_{-i}, s|a_i).$$

Thus, to determine all possible next states for a given action and observation, we can assume $P(\vec{a}_{-i}, s|a_i)$ is a uniform distribution and retain any state s' which has a positive probability. We will call the set of possible successor states S'.

When $P(o_i|s, \vec{a}, s')P(s'|\vec{a}, s)$ is constant for the given a_i and o_i, then it does not depend on the state or other agents' actions. Thus, $\sum_{\vec{a}_{-i}, s} P(\vec{a}_{-i}, s|a_i) = 1$ and the state is known by:

$$P(s'|a_i, o_i) = \frac{P(o_i|s, \vec{a}, s')P(s'|s, \vec{a})}{\sum_{s'} P(o_i|s, \vec{a}, s')P(s'|s, \vec{a})}$$

3.2 Algorithmic approach

Our approach is summarized in Algorithm 1. We describe it from an infinite-horizon perspective, but the same approach

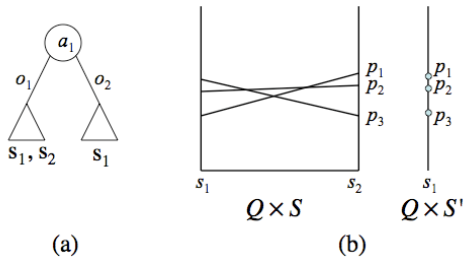

Figure 1: Example of (a) limiting states after actions are chosen and observations seen and (b) pruning with reduced states.

can be used in finite-horizon dynamic programming. For each agent, i, assume we have a controller C_i. We can create a backed up controller, C_i', by first choosing an action. Assuming we start with action a we need to decide which node in C to choose after seeing each observation. Rather than adding all possibilities from C_i as is done in the exhaustive approach, we only transition to nodes that are useful for some distribution over possible successor states s' and initial nodes of the other agents. That is, for two agents, agent i's node p is retained if the value of choosing p is higher than choosing any other node for some distribution of other agent nodes, q and next states of the system s'. This is formulated as:

$$\sum_{q,s'} x(q, s')V(p, q, s') > \sum_{q,s'} x(q, s')V(\hat{p}, q, s') \; \forall \; \hat{p}$$

where $x(q, s')$ is a distribution over nodes of the other agent and successor states, $p, \hat{p} \in C_1$, $q \in C_2$ and $s' \in S'$. Thus, we limit the possibilities after taking action a and seeing observation o to those nodes that are useful for some such distribution. The set of nodes that we retain for the given action and observation for agent i we will call $C_i^{a,o}$.

After generating all useful nodes for each observation with a fixed action, we can conduct a backup for that action. This is accomplished by creating all possible nodes which begin with the fixed action, a, followed by choosing any node from the set $C_i^{a,o}$ after o has been observed. The resulting number of nodes for agent i at this step is $\prod_o |C_i^{a,o}|$. In contrast, exhaustive generation would produce $|C_i|^{|\Omega|}$ nodes. Once all nodes for each action have been generated, we take the union of the sets for each action to get the backed up controller for the agent. We are then able to evaluate the backed up controllers and prune dominated nodes in the same way as described for policy iteration. Pruning may further reduce the number of nodes retained and increase their value.

Example An illustration of the incremental policy generation approach is given in Figure 1. The algorithm (a) first determines which states are possible after a given action has been taken and each observation has been seen. After action a_1 is taken and o_1 is seen, states s_1 and s_2 are both possible, but after o_2 is seen, only s_1 is possible. Then, (b) the values of the nodes from the previous step are determined for each resulting state and each node for the other agents.

The dimension of the value vectors of the nodes is the same as the number of states in the system times the number of

possible policies for the other agents. To clarify the illustration, we assume there is only one possible policy for the other agents and two states in the domain. As seen in the figure, when both states are considered, all nodes are useful for some possible state distribution. When only state 1 is possible, only p_1 is useful. This allows both p_2 and p_3 to be pruned for this combination of a_1 and o_2, reducing the number of possibilities to 1. Because there are still 3 useful subpolicies for the combination of a_1 and o_1, the total number of possible nodes starting with action a_1 is 3. This can be contrasted with the 9 nodes that are possible using exhaustive generation.

3.3 Analysis

We first show that using incremental policy generation, only nodes that would be pruned after exhaustive generation are not added to the set of possible next nodes, $C_i^{a,o}$.

Lemma 3.1. *Any node that is is not added by the incremental pruning algorithm is dominated by some distribution of backed up nodes.*

Proof. We will show that if node p transitions to some node in the previous controller which is not included by the incremental policy generation algorithm, then p must be dominated by some distribution of nodes $\hat{p}$. For two agents, this is determined by: $V(p,q,s) \leq \sum_{\hat{p}} x(\hat{p}) V(\hat{p},q,s) \; \forall q,s$

Intuitively, this proof is based on the fact that all reachable next states and all possible next step policies for the other agents are considered before removing a policy. We will prove this for two agents, but this proof can easily be generalized to any number of agents.

Consider the set of nodes generated for agent 1 by starting with action a. If observation, o_1^j was seen, where $p_{o_1^j}$ was not included in our set C^{a_p,o_1^j}, then we show that p must be dominated by the set of nodes that are identical to p except instead of choosing $p_{o_1^j}$, some distribution over nodes in C^{a_p,o_1^j} is transitioned to instead. We will abbreviate $p_{o_1^j}$ by p' to simplify the notation.

Because p' was not included in C^{a_p,o_1^j} we know there is some distribution of nodes in C_1that dominates it for all nodes in C_2 and at the subset of states that are possible after choosing action a_p and observing o_1.

$$V(p',q',s') \leq \sum_{\hat{p}'} x(\hat{p}') V(\hat{p}',q',s') \; \forall \; q',s' \qquad (2)$$

Given this distribution $x(\hat{p}')$ we can create a probability distribution over nodes which chooses the nodes of $\hat{p}'$ when o_1^j is seen, but otherwise transitions to the same nodes that are used by p. The value of this policy is given by

$$R(a_p,a_q,s) + \gamma \sum_{s'} P(s'|a_p,a_q,s)$$
$$\textstyle\sum_{o_1,o_2} P(o_1,o_2|a_p,a_q,s') \sum_{\hat{p}'} x(\hat{p}',o_1) V(p_{o_1},q_{o_2},s')$$

Because the nodes are otherwise the same, from the inequality in Equation 2, we know that

$$R(a_p,a_q,s) + \gamma \sum_{s'} P(s'|a_p,a_q,s) \sum_{o_1,o_2} P(o_1,o_2|a_p,a_q,s') V(p_{o_1},q_{o_2},s') \geq$$
$$R(a_p,a_q,s) + \gamma \sum_{s'} P(s'|a_p,a_q,s) \sum_{o_1,o_2} P(o_1,o_2|a_p,a_q,s') \sum_{\hat{p}'} x(\hat{p}',o_1) V(p_{o_1},q_{o_2},s')$$

This holds for for any s and q because Equation 2 holds for any initial state or node of the other agent.

This can also be viewed as a distribution of nodes, $x(\hat{p})$, one for each different node that is transitioned to with positive probability in $x(\hat{p}')$. Thus, the value of the original node p is less than or equal to the value of the distribution of nodes for all q and s. That is,

$$R(a_p,a_q,s) + \gamma \sum_{s'} P(s'|a_p,a_q,s) \sum_{o_1,o_2} P(o_1,o_2|a_p,a_q,s') V(p_{o_1},q_{o_2},s')$$
$$\leq \sum_{\hat{p}} x(\hat{p}) \left[R(a_{\hat{p}},a_q,s) + \gamma \sum_{s'} P(s'|a_{\hat{p}},a_q,s) \sum_{o_1,o_2} P(o_1,o_2|a_{\hat{p}},a_q,s') V(\hat{p}_{o_1},q_{o_2},s') \right]$$

or $V(p,q,s) \leq \sum_{\hat{p}} x(\hat{p}) V(\hat{p},q,s) \; \forall q,s$ □

Theorem 3.2. *The incremental policy generation algorithm returns an ϵ-optimal infinite-horizon solution for any initial state after a finite number of steps.*

Proof. This proof follows from Theorem 2 in [Bernstein *et al.*, 2008], which states that policy iteration converges to an ϵ-optimal controller. This is due to the use of a discount factor, which ensures that after some finite number of steps, the value of a policy will not change more than some ϵ. Thus, if enough steps of dynamic programming are performed and all policies that could contribute to the optimal policy are retained, some set of initial nodes will approach the optimal value. The pruning steps are shown to not harm this convergence and Lemma 3.1 shows that nodes that are not generated by incremental policy generation would by pruned in policy iteration. Also, it is obvious that our approach generates all undominated nodes. Thus, the incremental policy generation algorithm can also provide ϵ-optimal infinite-horizon policies in a finite number of steps. □

4 Heuristic incremental generation

In order to improve the scalability of the incremental policy generation algorithm, we have developed a heuristic version. While this algorithm can no longer guarantee ϵ-optimality is achieved, in general many more backups can be completed. This can result in higher valued solutions being produced. An overview of this approach is provided in Algorithm 2.

Our heuristic algorithm proceeds the same way as the optimal approach by using dynamic programming to improve the value of an initial solution. On each step, we generate a set

Algorithm 2: Heuristic incremental policy generation

input : A set controllers for all agents, C, and a desired number of points, m
output: A set of backed up controllers, $\hat{C}'$
begin
  $P \leftarrow$ *pointSet*(C, m)
  for *each agent,* i **do**
    $C'_i \leftarrow$ *incrPolGen*(C, i)
  $C' \leftarrow \cup_i \hat{C}'_i$
  for *each belief point,* p **do**
    $\hat{C}'_p \leftarrow$ *bestSet*(C', p)
  $\hat{C}' \leftarrow \cup_p \hat{C}'_p$
  $\hat{C}' \leftarrow$ *prune*$(\hat{C}')$
  return $\hat{C}'$
end

of belief states (state probability distributions) using action probabilities from the agents' current controllers. We then backup the controllers using incremental policy generation. Finally, we retain only those nodes that are part of the best set of controllers for each point.

4.1 Algorithmic approach

Due to uncertainty in local observations and about the other agent's polices, the agents will not know what the actual state of the problem, but during execution, only a small part of the state space may be visited. By sampling in this space, many more nodes can be eliminated that are not required for a high valued solution. This allows the policy to be focused on states that are reachable given the current policies of the agents.

To generate a set of belief points for the agents, we first estimate the probability of each agent's action given the states of the problem, $\hat{P}(a_i|s)$. This is accomplished by estimating the probability that the node of an agent's controller is q given that the current state is s, $\hat{P}(q_i|s)$. This is found by examining the probability the agent begins in the given node and state or the likelihood it transitions to the node and state later in the problem as determined by $\hat{P}(a_i|s) = \sum_{q_i} P(a_i|q_i)\hat{P}(q_i|s)$, where $\hat{P}(q_i, s) = p_0(q'_i, s') + \gamma \sum_{s,a_i} P(s'|s, a_i) \sum_{o_i} P(o_i|s', a_i) \sum_q P(a_i|q_i)P(q'_i|q_i, a_i, o_i)\hat{P}(q_i, s)$ with $p_0(q_i, s)$ defined as the initial probably of node q_i and state s. The transition and observation models are estimated for each agent by choosing random actions for the other agents in the decentralized models. This quantity can then be normalized to produce $\hat{P}(q_i|s)$.

Once we have an estimate of the probabilities of other agent's actions at each state, we can determine the resulting state distributions for each action taken and observation seen. The probability of state s', given the belief state b, and agent i's action a_i and observation o_i, $P(s'|a_i, o_i, b)$, is then given by Equation 1 with a normalizing factor of $P(o_i|a_i, b)$. Thus, given the action probabilities for the other agents, and the transition and observation models of the system, a belief state update can be calculated.

Starting with the initial problem state, a set of belief points can then be generated. Once this occurs, the current controller can be backed up using incremental policy generation. After this, instead of retaining all nodes that are produced, only those that contribute to the solution with the highest value for each point are retained. This allows many more nodes to be removed at each step while focusing the solution on the states that are reached by the agents' policies. Finally, pruning dominated nodes is conducted as described for policy iteration. The process of generating a new set of points, backing up the controller and retaining the best nodes continues until the agents' controllers no longer change.

Two Agent Tiger, $\|S\| = 2$, $\|A_i\| = 3$, $\|\Omega_i\| = 2$			
Iter.	Policy Iteration	Incremental Generation	Value
0	1 in $<$ 1s	1 in $<$1s	-150
1	(3) 3 in $<$1s	3 in $<$1s	-137
2	(27) 15 in 1s	15 $<$1s	-117.85
3	(675) 255 in 465s	255 in 70s	-98.90
4	x	(65535)* in 3057s	-90.54
Recycling Robots, $\|S\| = 4$, $\|A_i\| = 3$, $\|\Omega_i\| = 2$			
Iter.	Policy Iteration	Incremental Generation	Value
0	1 in $<$ 1s	1 in $<$ 1s	0
1	(3) 3 in $<$ 1s	3 in $<$ 1s	3.01
2	(27) 6 in 1s	(8) 6 in $<$ 1s	25.66
3	(108) 20 in 6s	20 in 1s	26.04
4	(1200) 72 in 542s	72 in 6s	26.69
5	(15552)* in 869s	272 in 201s	27.20
6	x	(1092) 1058 in 5016s	27.67
7	x	(4759)* in 5918s	28.10
Box Pushing, $\|S\| = 100$, $\|A_i\| = 4$, $\|\Omega_i\| = 5$			
Iter.	Policy Iteration	Incremental Generation	Value
0	1 in $<$ 1s	1 in $<$ 1s	-2
1	(4) 2 in 2s	2 in 1s	-2
2	(128) 9 in 209s	9 in 7s	12.84
3	x	(137,137) 131,128 in 4501s	67.87

Table 1: Results of applying policy iteration and incremental policy generation (IPG) to the test problems. The number of nodes generated is shown in parentheses beside the number retained after pruning and iterations for which pruning could not be completed with the given amount of time and memory are marked with *.

5 Experiments

We present results comparing policy iteration with incremental policy generation on a set of infinite-horizon problems. These problems are the two agent tiger [Nair *et al.*, 2003], the recycling robot [Amato *et al.*, 2007] and the box pushing [Seuken and Zilberstein, 2007] domains. The incremental policy generation results also make use of *observation compression*, which merges observations that provide the same information. For instance in the tiger problem after opening a door, each observation is possible, but they are chosen randomly. These observations provide the same information and thus, can be merged. More formally, two of agent i's observations o_i^1 and o_i^2 are the same for a given action a_i if for each set of other agent actions a_{-i}, s and s' $P(o_i^1|s, a_i, a_{-i}, s') = P(o_i^2|s, a_i, a_{-i}, s')$.

The results are shown in Table 1. Each algorithm was run until the given memory (2GB) or time (4 hours) was exhausted and the number of nodes in each agent's controller, the running time and the resulting value are provided. If the number of nodes generated was greater than the number re-

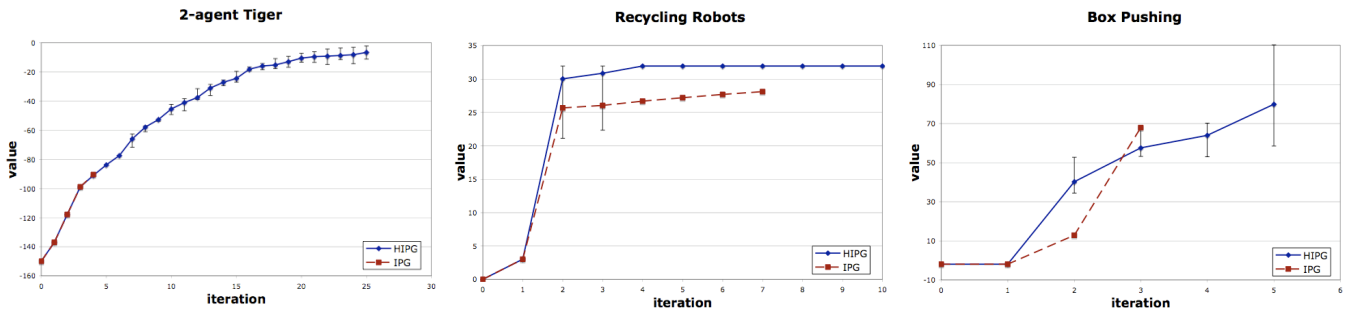

Figure 2: Comparison of the heuristic and optimal incremental policy generation algorithms on the two agent tiger, recycling robot and the box pushing problems.

tained after pruning was completed, the size after generation is given in parentheses. If the given number of iterations could not be completed, an 'x' is displayed. The initial policies for each algorithm consisted of each agent repeating the 'open left' action in the tiger problem, the 'pick up large can' action in the recycling robots problem and the 'turn left' action in the box pushing problem.

It can be seen that the incremental policy generation approach requires less time on each dynamic programming iteration and allows more iterations to be completed. This results in higher solution values for the incremental approach. The results also show that the incremental approach often generates the exact same controllers without the need for pruning. That is, while policy iteration produces a large number of nodes and then prunes many of them, the incremental approach arrives at the same controller without the added memory of generating an exhaustive set of nodes or the extra time needed for pruning these nodes. Even when nodes are removed in the incremental case, only a small number need to be pruned.

The comparison with the heuristic approach is shown in Figure 2 with incremental policy generation listed as IPG and the heuristic version listed as HIPG. Due to randomness in point generation, the heuristic algorithm was run 10 times and mean values are reported. The maximum and minimum values for each iteration are shown with bars. Also, the number of points used for each problem was 20.

Using the heuristic approach, value is improved in each problem. In the tiger problem, the same value as optimal approach is found for the first few steps, but increased value can be found due to better scalability. In the recycling robot problem, a higher valued solution is more quickly found and converged to, while the optimal approach cannot reach that value before resources are exhausted. Lastly, in the box pushing problem, the two algorithms alternate between producing the highest value, but due to increased scalability, the heuristic approach eventually outperforms the optimal approach.

6 Conclusion

In this paper, we introduced the incremental policy generation approach. This is a more efficient way to produce ϵ-optimal solutions for infinite-horizon DEC-POMDPs. We proved the optimality guarantee and showed that our method is more scalable than Bernstein et. al's policy iteration algorithm. As a result of this increased scalability, we demonstrated that incremental policy generation can provide higher quality solutions. We also presented an approximate version of our approach and showed that it is able to produce further solution quality gains on a set of test problems.

Interestingly, the incremental policy generation method is quite general and can be incorporated into other solution methods. For instance, incremental policy generation could be used to improve the efficiency of the optimal finite-horizon algorithm [Hansen *et al.*, 2004]. Also, our approach can easily be incorporated into the state-of-the-art finite-horizon approximate algorithm, IMBDP [Seuken and Zilberstein, 2007]. This would likely improve the performance of IMBDP, allowing larger problems to be solved and producing high valued solutions. We plan to explore these research avenues in the future.

References

[Amato *et al.*, 2007] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, Vancouver, Canada, 2007.

[Bernstein *et al.*, 2005] Daniel S. Bernstein, Eric Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, Edinburgh, Scotland, 2005.

[Bernstein *et al.*, 2008] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of Markov decision processes. Technical Report UM-CS-2008-044, University of Massachusetts, Department of Computer Science, Amherst, MA, 2008.

[Cassandra *et al.*, 1997] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, 1997.

[Hansen *et al.*, 2004] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, San Jose, CA, 2004.

[Hansen, 1998] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, WI, 1998.

[Nair *et al.*, 2003] Ranjit Nair, David Pynadath, Makoto Yokoo, Milind Tambe, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 705–711, Acapulco, Mexico, 2003.

[Seuken and Zilberstein, 2007] Sven Seuken and Shlomo Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, Vancouver, Canada, 2007.

[Szer and Charpillet, 2005] Daniel Szer and François Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning*, Porto, Portugal, 2005.