Value-Based Observation Compression for DEC-POMDPs

Alan Carlin and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{acarlin,shlomo}@cs.umass.edu

ABSTRACT

Representing agent policies compactly is essential for improving the scalability of multi-agent planning algorithms. In this paper, we focus on developing a pruning technique that allows us to merge certain observations within agent policies, while minimizing loss of value. This is particularly important for solving finite-horizon decentralized POMDPs, where agent policies are represented as trees, and where the size of policy trees grows exponentially with the number of observations. We introduce a value-based observation compression technique that prunes the least valuable observations while maintaining an error bound on the value lost as a result of pruning. We analyze the characteristics of this pruning strategy and show empirically that it is effective. Thus, we use compact policies to obtain significantly higher values compared with the best existing DEC-POMDP algorithm.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Experimentation, Performance

Keywords

artificial intelligence, decentralized POMDPs, observation compression, planning under uncertainty

1. INTRODUCTION

Policy trees are often used to represent conditional plans for agents operating in partially observable environments. Each node of a policy tree determines the next action to be taken and a transition to a sub-tree that depends on the observation made by the agent. This representation has proved useful in partially-observable MDPs and their multi-agent counterparts (DEC-POMDPs) [5, 11, 14]. DEC-POMDPs offer a rich language to represent multi-agent planning problems in stochastic domains, where each agent has different partial information about the domain (determined by its observations). They have been used to model multi-robot coordination problems, multi-agent communication, and distributed load-balancing problems. A detailed survey of existing algorithms and applications of DEC-POMDPs is available in [10]. We focus in this paper on one of the key bottlenecks in DEC-POMDP algorithms – the extremely large space of joint policies. Even with small problems, it is not feasible to explore the entire space of joint policies that agents may follow. The memory requirements of this representation have limited the scalability of solution techniques for decentralized POMDPs.

Several pruning methods have been introduced to address this difficulty. One approach involves pruning of *dominated strategies* that are provably useless [5]. It allows a large number of policies to be excluded from consideration without compromising optimality. Despite the savings, this exact algorithm runs out of memory after just a few iterations. More aggressive pruning could improve scalability, but produces non-optimal plans. Amato *et al.* adapted an epsilon-pruning technique that maintains an error bound on the loss of value resulting from the pruning process for DEC-POMDPs [1]. But even with such pruning techniques, it is difficult to solve problems that involve a large number of possible observations, because the number of joint policies grows exponentially with the number of observations.

Some effective heuristic methods have been developed to select a bounded set of policies in each step, and to use this selection process to construct policies for the next step [14, 11]. Among these approaches, the MBDP algorithm has been particularly successful, making it possible to solve problems with very large planning horizons for the first time [11]. We present a disciplined approach for reducing the number of observations used in policy trees, and demonstrate our approach on MBDP. The resulting planner extends previous work by Seuken and Zilberstein, which introduced a technique for ignoring observations that are less probable [11]. Unlike that heuristic method, our approach is guided by the value lost as a result of merging certain sets of observations.

The rest of the paper is organized as follows. Section 2 defines the decentralized POMDP model and describes the key existing solution techniques. In Section 3 we describe our algorithm, which is based on MBDP with the added observation compression method. We show how to minimize the loss of value that results from merging observations. Section 4 analyzes the characteristics of the new algorithm and establishes an error bound on the solution it returns. In Section 5 we report the results of an empirical evaluation, showing significant improvement in solution quality compared with the best existing method. The value-driven approach and the associated error bound help improve the performance and robustness of DEC-POMDP solution techniques. We conclude with a discussion of the contributions of the work and future research directions that can exploit value-based observation compression.

Cite as: Title, Author(s), *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

2. DECENTRALIZED POMDPS

We study the observation compression problem within the framework of decentralized POMDPs [2]. While the results we present are valid for any number of agents, for the sake of clarity, we formalize the problem for two agents. A two agent DEC-POMDP is defined by the following tuple:

- S, the set of domain states
- $b_0 \in \Delta S$, the initial belief state (state distribution)
- A_1 and A_2, the sets of actions for each agent ($a_i \in A_i$ denotes an action of agent i)
- P, the transition model: $P(s'|s, a_1, a_2)$ is the probability of transitioning to state s' given the previous state was s and actions a_1 and a_2 were taken by agents 1 and 2 respectively
- R, the reward function: $R(s, a_1, a_2)$ is the immediate reward for taking actions a_1 and a_2 in state s
- Ω_1 and Ω_2, the sets of observations for each agent
- O, the observation probabilities: $O(o_1, o_2|s', a_1, a_2)$, the probability of agents 1 and 2 seeing observations o_1 and o_2 respectively when agent 1 takes action a_1 and agent 2 takes action a_2, causing a state transition to s'
- T, the horizon, or number of steps, in the problem

At each step, each agent chooses an action based on its own action and observation history. The goal is to create joint plans that maximize the cumulative reward over some finite horizon.

We refer to the mapping from an agent's history to an action as a policy, and use trees to represent such policies. For example, Figure 1(a) shows two policies, P_1 and P_2. With policy P_1, the agent first takes action A_1 and then, depending on the observation, moves to a subtree that determines the next action.

In the case of single-agent POMDPs, it is possible to adopt an alternate view of a policy and represent it as a mapping from a belief state to an action. Each agent can maintain a belief state that can be revised after each action using standard Bayesian updating, based on the action taken and observation that the agent receives. The agent can then select the best action, given this belief state. Maintaining belief states in the multi-agent case is much more complicated, because each belief state depends on the history of actions and observations as well as the agent's beliefs regarding the policies of the other agents. Gmytrasiewicz and Doshi have developed a multi-agent planning approach that is based on maintaining nested belief states [4]. Belief states are nested because each agent's beliefs include the beliefs of other agents, which in turn include the belief of this agent.

Action policies that are based on histories of observations allow us to avoid the complexity associated with nested beliefs. In our case, an agent's policy is expressed in the form of a tree. Each node of the tree corresponds to an action, and each branch to an observation. The root of the tree corresponds to the first action, and the tree is traversed downwards. Note that the number of possible policy trees is of the order $O(|A|^{|\Omega|^T})$ per agent. This explains why effective pruning is crucial in DEC-POMDP algorithms.

Certain types of pruning that have been developed for single-agent POMDPs can be generalized to the multi-agent case. In particular, there are a number of efficient point-based algorithms for POMDPs that evaluate different policies at select points in the belief space [8, 13, 15, 12]. In the DEC-POMDP case, the policies of the other agents determine both the value and the likelihood of each belief point, a fact that complicates the generalization of these POMDP algorithms. Other POMDP techniques involve

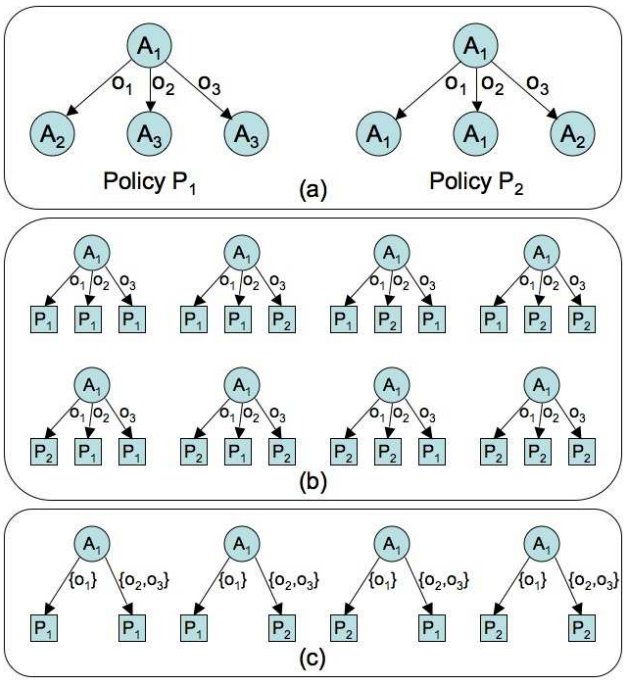

Figure 1: (a) Two policy trees. (b) The result of a full-backup of the policy trees shown in part (a) is 24 new trees. Only the trees rooted in A_1 are shown. (c) The result of running CompressObs and then a partial backup. Here $maxObs = 2$ and subpolicies of o_2 and o_3 are forced to be the same.

value-directed compression of the POMDP state space [9]. Hoey and Poupart aggregate observations in a one-dimensional observation space, and extend their technique by sampling belief points in a multi-dimensional observation space [6]. Although this technique is applicable to POMDPs, it is not clear how to extend the technique and how to sample belief points of DEC-POMDP problems.

Some useful pruning techniques for DEC-POMDPs have been developed already. Hansen *et al.* have developed an optimal dynamic programming algorithm that prunes many useless policies [5]. During the planning process, each agent must choose policies that have high value based on *any* policy the other agent may use. To formalize this, they define a belief state, $b(s, q)$, for an agent as the probability that the system state is s and the other agent will use policy q. The value of an agent's belief is then:

$$V(b) = \max_p \sum_{s,q} V(p, q, s) b(s, q)$$

which represents the maximum the agent can achieve given its set of policies weighted by the probability that the true state is s and the other agent uses policy q. Because the probability distribution of the underlying state depends on the policies of the other agent and we assume no prior knowledge about the other agent, all policies that may contribute to this maximum must be retained. Policies that do not contribute to the maximum may be removed, or *pruned*. While this method guarantees that an optimal tree will be found, the set of trees may be very large and many unnecessary trees may be generated along the way. Because of this, it requires an intractable amount of memory for any but the smallest problems.

Figure 1(b) shows the result of a full-backup where pruning did not eliminate any policies. Policy Trees P_1 and P_2 are represented within the squares. A full backup will generate every possible ac-

Algorithm 1: The MBDP-OC Algorithm (IMBDP [9] with Value-Based Observation Compression)

```
begin
    MaxTrees ← max number of trees before backup
    MaxObs ← max number of observations for backup
    T ← horizon of the DEC-POMDP
    H ← pre-compute heuristic policies for each h ∈ H
    Q_{i,1}, Q_{j,1} ← initialize all 1-step policy trees
    A_i, A_j ← the set of actions available to each agent
    for t=1 to T do
        choose h ∈ H and generate belief state b
        for each agent x do
            Q_{x,t+1} ← {}
            for each action a ∈ A_x do
                Z ←
                CompressObs(x, b, a, Q_{x,t}, Q_{-x,t}, MaxObs)
                Q_{x,t+1} ← Q_{x,t+1} ∪ PartBackup(Q_{x,t}, a, Z)
        Sel_{i,t+1}, Sel_{j,t+1} ← empty
        for k=1 to MaxTrees do
            choose h ∈ H and generate belief state b
            foreach q_i ∈ Q_{i,t+1}, q_j ∈ Q_{j,t+1} do
                evaluate each pair (q_i, q_j) wrt b
            add best policy trees to Sel_{i,t+1} and Sel_{j,t+1}
            delete these policy trees from Q_{i,t+1} and Q_{j,t+1}
        Q_{i,t+1}, Q_{j,t+1} ← Sel_{i,t+1}, Sel_{j,t+1}
    select best joint policy tree δ_T from {Q_{i,T}, Q_{j,T}}
    return δ^T
end
```

tion and observation sequence. Note that only the new policies rooted in action A_1 are shown. In this case, with 3 possible observations, 3 actions, and two previous policies, a full backup would generate $(3)2^3$ or 24 new policies.

Amato *et al.* [1] prune additional policy trees through epsilon pruning. At each step, a full backup is performed just as in [5]. Then a set of policies called the undominated set is iteratively constructed. A policy is added to the undominated set if there exists a belief state as well as a policy for the other agents, such that the new policy exceeds the value of the current undominated set by more than epsilon. A linear program is used to implement this step. The epsilon pruning technique thus produces solutions with an error bound of $n\epsilon T$, where ϵ is a user specified parameter and n is the number of times that epsilon pruning is performed per horizon step. This works well in domains with a small number of observations. But in domains with a large number of observations, the number of new policies generated before pruning will be exponential in the number of observations. This motivates the development of additional pruning that specifically targets the number of observations.

3. THE MBDP-OC ALGORITHM

In this section we present our algorithm in detail. The overall solution technique is based on modifying the IMBDP algorithm (Improved Memory-Bounded Dynamic Programming) with the value-based observation compression technique. Hence, the resulting algorithm is called MBDP-OC (MBDP with Observation Compression). We first describe the MBDP-based algorithm and then the value-based observation compression procedure.

3.1 The Modified MBDP Algorithm

MBDP and IMBDP have proved to be very effective in solving finite-horizon DEC-POMDPs [11]. They accomplish this by limiting the number of trees retained at each step to a predefined constant $MaxTrees$. At each step, a belief state is selected using a heuristic. Then the best joint policy is selected and removed from consideration. Then a new belief state (or the same one) is selected, and then the process is repeated until each agent has saved $MaxTrees$ policies. Thus, at each step, a full backup produces only $|A|MaxTrees^{|\Omega|}$ new policies. Since MBDP is a bottom up planner (that is, it starts with the last step and works its way backwards), the selected belief state can be the result of running some reasonable joint policy (such as the underlying MDP) from the first step to the current step. Noting that the result is still exponential in $|\Omega|$, the authors then limit the number of possible observations to a constant that they term $MaxObs$. IMBDP selects the most likely observations from heuristically determined belief states, and it only backs up policy trees with those observations. They term this process a *partial backup*. Complete policies are then "filled up" with the missing observations, by selecting the best available policies for these observations. Thus, for instance, if a partial backup were run for the policies shown in part A of Figure 1, and a $MaxObs$ of two were used, only 12 new policies would be generated, 4 for each root action. Although this process is quick compared to a full backup, the error is only bounded for the heuristically selected belief state. The error is given by:

$$T^2(1-\epsilon)(R_{max}-R_{min})$$

where T is the horizon of the problem, ϵ is the probability of receiving one of the remaining observations, and R is the reward function. In IMBDP, the probabilities of these observations are taken into account, but not their values.

3.2 Value-Based Observation Compression

We now present the value-based observation compression method (the CompressObs Algorithm), which reduces the complexity of the backup process by merging pairs of observations in each iteration. We present this observation compression method as a tool that can be used by bottom-up planners. We use the MBDP planner as an example of a planner that can be augmented by using this algorithm. The augmented DEC-POMDP solution method is MBDP-OC described in the previous section.

During the backup process, MBDP-OC operates in a similar manner to IMBDP in seeking to limit the number of observations and policies considered for backup [11]. Recall that each branch of a new policy tree corresponds to an observation. In contrast to IMBDP, the partial backup in MBDP-OC seeks to reduce the exponential generation of new policies by forcing different branches of the new root policies to contain the same subtrees. The distinction can be seen in Figure 1(c). IMBDP (not shown in the figure) selects the $MaxObs$ most likely observations, finds all combinations of subpolicies for those, and fills in all remaining observations with a single policy. MBDP-OC, by contrast, finds $MaxObs$ groups of policies, and in the backup process forces all observations in the same group to have the same subpolicies. Thus, the key modification to IMBDP (as shown in Algorithm 1) is just before the partial backup. In MBDP-OC, the procedure CompressObs is called to determine how to group the observations. The groupings are stored in a set Z, where Z is a set of tuples (z_i, ϵ_i). Each z_i is a set of observations, and each ϵ is an error term associated with observation group z_i. We call the backup procedure that perform these calculations $PartBackup$ to distinguish it from the partial backup in IMBDP.

Let $MaxTrees$ be the number of distinct subtrees that the user desires and $MaxObs$ be the limit on the number of observations.

Algorithm 2: CompressObs($x, b, a_x, Q_i, Q_j, MaxObs$)

input : Agent number x, belief state b, root action a_x, subpolicies Q_x and Q_{-x}, and observation limit $MaxObs$

output: Z, a set of tuples (z_k, ϵ_k), where each z_k is a set of observations, and ϵ_k is the error introduced by merging the observations in z_k

begin
- $Z \leftarrow \{(\{o_1\}, 0), (\{o_2\}, 0), ...\}$
- **for** *each observation* o_k*, each policy* q_x*, and each* $q_{-x} \in Q_{-x}$ *and* $a_{-x} \in A_{-x}$ **do**
  - Precompute $V(q_x|q_{-x}, b, a_x, a_{-x}, o_k, o_{-x})$
  - Keep track of best policies q_x^*
- **while** $|Z| > MaxObs$ **do**
  - $(z_1', z_2') \leftarrow$ $argmin_{(z_1,\epsilon_1)(z_2,\epsilon_2)\in Z} WL^*(Q_x|b, a_x, z_1 \cup z_2)$
  - $\epsilon \leftarrow WL^*(Q_x|b, a, z_1' \cup z_2')$
  - $Z \leftarrow Z \backslash \{z_1', z_2'\} \cup (z_1' \cup z_2', \epsilon)$
- **return** Z

end

We then have $MaxTrees$ policies before backup and $|A|$ actions, so we generate $|A|MaxTrees^{MaxObs}$ new policy trees for each agent at backup. The implementation then follows [11] by selecting the $MaxTrees$ best policies to retain for the next backup.

The question then becomes, which observations should be merged together, such that forcing them to contain the same subtrees does not produce a large error? In order to answer that, we need to introduce the following notation and definitions. We need to express how much value is lost by merging two observations (or two groups of observations). Intuitively, the merged observation pair will pursue the subpolicy that works best for its component observations. The value lost will be a function of the value of the chosen subpolicy, the value of the optimal policy for the component observations, and the likelihood of receiving the observations. For ease of explanation, we begin our definitions with the single agent case, and then generalize them to the multi-agent case:

- q_i^1, q_i^2..... The policy trees for agent i. During the backup process, these policy trees will become subpolicy subtrees. Throughout this paper we will often refer to a generic single subpolicy for agent i as q_i.
- $q_i^1, q_i^2 ... = Q_i$. The set Q_i has $MaxTrees$ members.
- $V(q_i|b, a, o)$. The value of following subpolicy q_i after taking action a and receiving observation o in belief state b.
- $q_i^*(b, a, o) = argmax_{q_i \in Q_i} V(q_i|b, a_i, o)$. The best subpolicy available, after taking action a_i and receiving observation o.
- $L(q_i|b, a, o) = V(q_i^*|b, a, o) - V(q_i|b, a, o)$. The value lost if subpolicy q_i is taken instead of the best available subpolicy.
- $WL(q_i|b, a, \bar{o}) = \sum_{o_k \in \bar{o}} O(o_k|b, a) L(q_i|b, a, o_k)$. The loss of following a single policy q_i for the group of policies in $\bar{o}$. Each loss is weighted by the probability of the observation.
- $WL^*(Q_i|b, a, \bar{o}) = \min_{q_i \in Q_i} WL(q_i|b, a, \bar{o})$. The Weighted Loss of following the best available single policy for the group of policies in $\bar{o}$.

Our notation is somewhat unique in that we always evaluate given (b, a, o). For a standard POMDP, one could have defined a new belief state b' given that action a was taken and o was observed from belief state b, but this notation would not scale to the multiagent case. None of these terms value the immediate reward achieved through action in belief state b, however, they all compute the probabilities of new belief states once action a is taken and o is observed, and they find subpolicy values in the new belief state. This is necessary because our pruning process requires only the value of subpolicies. When we backup, the root action is already given, and the root reward can not be changed.

We use classic POMDP methods to evaluate $V(q_i|b, a, o)$. Since we started at belief state b, took an action a and received an observation o, this defines a new belief state, which we will call $b(s')$. We can then evaluate q_i :

$$\sum_{s'} b(s') V(q_i, s')$$

the probability of being in each state times the value of q_i for that state. The value of q_i for state s' is recursively computed.

$$R(s', a') + \sum_{s'' \in S} P(s''|s', a') \sum_{o' \in \Omega} O(o'|s'', a') V(q_i', s'')$$

where a' is the root action in the policy, and $V(q_i', s'')$ is the value of continuing the policy in state s'.

We use $q_i^*(b, a, o)$ to denote the subpolicy for agent i with the highest value, given that action a was taken from belief state b and then o was observed.

In the definition of L , the difference

$$L(q_i|b, a, o) = V(q_i^*|b, a, o) - V(q_i|b, a, o)$$

is the value lost by choosing a single policy q_i (instead of the ideal q_i^*) after receiving the observation o. We then weight by the probability of observing o and call the result our weighted loss, which we abbreviate WL. This weighted loss can be used to denote the sum of the values lost by choosing one single policy for a whole group of observations, rather than the best policy for each observation.

The value of the best single policy for the observations in $\bar{o}$ is WL^*. This is the cumulative value lost if we are to group all the observations in $\bar{o}$ together.

For the multiagent case, we change the actions to joint actions, and policies to joint policies. The following notation uses the 2-agent case for ease of explanation, but is easily extended to multiple agents.

- $V(q_i|q_j, b, a_i, a_j, o_i, o_j)$ is the value of following joint policy (q_i, q_j) after taking joint action (a_i, a_j) and after the agents receive observations o_i and o_j in belief state b.
- $q_i^*(q_j, b, a_i, a_j, o_i, o_j) =$ $argmax_{q_i \in Q_i} V(q_i|q_j, b, a_i, a_j, o_i, o_j)$. When the arguments to q_i^* are clear from context, as below, we may just refer to it as q_i^*.
- $L_i(q_i|q_j, b, a_i, a_j, o_i, o_j) =$ $V(q_i^*|q_j, b, a_i, a_j, o_i, o_j) - V(q_i|q_j, b, a_i, a_j, o_i, o_j)$
- $WL_i(q_i|b, a_i, \bar{o}) =$ $\sum_{o_k \in \bar{o}} \max_{a_j \in A_j} \sum_{o_j \in \Omega_j} \max_{q_j \in Q_j}$ $O(o_k, o_j|b, a_i, a_j) L_i(q_i|q_j, b, a_i, a_j, o_k, o_j)$
- $WL_i^*(Q_i|b, a_i, \bar{o}) = \min_{q_i \in Q_i} WL_i(q_i|b, a_i, \bar{o})$

As seen above, we must consider all possible observations, actions, and policies of the other agents. The loss term is modified to be a loss for a fixed action, observation, and subpolicy of the other agent. Similarly, weighted loss terms are modified to sum over the observation probabilities of the other agent, and to find the worst-case actions and subpolicies that the other agent may take based on these observations.

The last definition means that our algorithm will consider all other policies of the other agent when deciding which observations to merge. This issue is discussed further in the conclusions.

The observation compression process itself is displayed in Algorithm 2. As the figure shows, the process can be packaged into a function call, and thus can be seamlessly integrated with IMBDP, or indeed any policy tree based algorithm. Unlike Seuken and Zilberstein, we don't force trees to follow a single policy after an observation; rather, we seek to consider which observations we can merge with a minimum loss of value. To do this, we assume we are in the backup process, and we are given the root action for some new set of policies. We precompute the value of each of the existing $MaxTrees$ subpolicies, for each observation and possible policy of the other. Possible policies of the other include the set of $|A|$ root actions as well as the set of $MaxTrees$ existing subpolicies from the previous step. Now that we have the value for each subpolicy, we identify the best subpolicy, as well as its value. We construct a table of losses, which identifies the value lost by each subpolicy if the agent should follow that one instead of the best, given the observation and the policy of the other.

We weigh this loss of value by the probability of receiving the observation. For each of the $MaxTrees$ subpolicies of the current agent, we find the weighted loss (WL) introduced by using that one subpolicy for the merging of two groups of observations, which is the sum of the loss for each observation in the group, for all possible policies of the other agent. Thus the $WL_i^*(Q_i|b, a_j, \bar{o})$ of the set of observations $\bar{o}$ is the summation of the weighted losses for all members of the combined set while using the best possible single policy. The algorithm finds WL^* for each pair of groups in Z, and merges the two groups that introduce the smallest amount of loss.

Our compression algorithm iteratively compresses groups of observations. It starts with $|\Omega_i|$ observations, and selects two groups of observations to compress together while minimizing the weighted loss. In the first iteration, each observation in Ω_i is its own group. In successive iterations, we continue to merge groups of observations, until there are only $MaxObs$ observations left. Note that since we can compute a loss bound for each iteration, we can select to iterate until a certain loss threshold is achieved.

4. ANALYSIS OF MBDP-OC

In this section we discuss the running time and space used by MBDP-OC, as well as its error bounds. We show that in certain special cases, the error introduced will be low. In the general case, we will show a loose error bound that depends on the worst possible choice of the other agent. This error bound is computable so any algorithm that uses the value functions presented in this paper can be modified to terminate after its error reaches a predetermined threshold. We first discuss running time and space for CompressObs.

THEOREM 1. *For the two agent case, the running time of* CompressObs *is* $O(MaxTrees^2|A||\Omega|^4)$*, and the space used is* $O(MaxTrees^2|A||\Omega^3|)$.

PROOF. We examine each component of the algorithm. First, all subpolicy values are computed and stored. This occurs for each subpolicy of the current agent, for each subpolicy of the other, for each initial action of the other, and for each observation vector. This takes $O(MaxTrees^2|A||\Omega^3|)$ time and space. Identifying the best policies is just a matter of scanning this list. Loss terms can be precomputed in a similar operation, and again requires $O(MaxTrees^2|A||\Omega^3|)$ time and space. The algorithm then enters a while loop where it iteratively shrinks $|Z|$ from $|\Omega|$ down to $MaxObs$. There are $(|\Omega| - MaxObs)$ iterations of this. Within the while loop, weighted losses are found and stored for each pair of groups of observations in Z (that is, for each possible (z_1, z_2), requiring $O(|\Omega|^2)$ storage). Each merged group has at most $|\Omega|$ observations, and there are $|Z|^2$ possible pairs, and we know $|Z| < |\Omega|$. Thus there are an order of $O(|\Omega|^4)$ computations of WL. We only keep track of the identities of the best groups to merge, so no additional storage is required. The computation of WL itself merely looks up joint policy values previously stored in MBDP. Since WL is referenced for each possible action and subpolicy of the other agent, and since this occurs in a while loop, the total time spent finding WL function values, including the while loop, is $O(MaxTrees^2|A||\Omega|^4)$. □

Although in this paper we analyze CompressObs as written in Algorithm 2, in practice, depending on the domain requirements, it may not be desirable to design the algorithm around the worst case. In essence, CompressObs computes online error bounds by considering the other agent as a limited adversary, so it computes a strategy for such an adversary for each observation. In practice, it may not be necessary to do this, and indeed an expectation over each observation would be more accurate than merely bounding its worst-case.

The rest of this section is devoted to an analysis of the joint value lost by running MBDP-OC in comparison to the MBDP algorithm that runs a full backup. For the MBDP class of algorithms, define a belief state's "best available policy" at a certain horizon as the best possible policy tree whose subtrees consist of the policies that MBDP saved on the previous horizon step. This is not necessarily an optimal policy, because MBDP only saves $MaxTrees$ policies per step.

Since MBDP-OC is an integration of Observation Compression with the MBDP algorithm, we will see that it inherits its error bounds from the MBDP approach. In particular, we prove no bounds on the selection of the MaxTrees, we only prove the error that observation compression introduces after that. At each step, this serves as an error bound on the heuristically chosen belief state. In addition, were observation compression to be merged with other policy-tree based algorithms[1, 5], the methodology in the proofs below could be adapted to those algorithms to produce an absolute error bound for all belief states.

We start with a lemma that says that if MBDP-OC iteratively merges an observation into larger and larger sets, one can find the total error introduced for this observation by merely examining the error on the last merge. That is, the error does not accumulate.

LEMMA 1. *Suppose that the sets of observations* $\bar{o}^1$ *and* $\bar{o}^2$ *are merged by the above algorithm. The total value lost due to the merging of* $\bar{o}^1$ *and* $\bar{o}^2$ *only depends on the components of* $\bar{o}^1$ *and* $\bar{o}^2$*, not on the value lost during the formation of* $\bar{o}^1$ *and* $\bar{o}^2$.

PROOF. The base case is trivial as the observations have not been previously merged. Note that the value lost due to the observation compression process is the difference between value of the best available policy and the value of the best possible policy once the observations are merged. This notion is captured in the definition of the loss term:

$$V(q_i^*|q_j, b, a_i, a_j, o, o_j) - V(q_i|q_j, b, a_i, a_j, o, o_j)$$

For the inductive case, we find a best policy q_i that minimizes the weighted sum of the loss of all observations o where $o \in \bar{o}^1 \cup \bar{o}^2$. As the above value difference equation indicates, the value for each observation only depends on the value of the best possible policy and the value of the selected policy, and not on previous merges. Therefore, the error does not accumulate with the value lost in any previous compression. □

We now develop the properties of a special case, where $MaxObs$ is greater than or equal to the number of $MaxTrees$. Since MBDP is capable of running with $MaxTrees = 3$, this setting is certainly possible and sometimes desirable.

PROPOSITION 1. *If $WL_i(q_i|b, a_i, \bar{o}) = 0$, then q_i is a best available subpolicy for all of the observations in $\bar{o}$.*

PROOF. The definition of $WL_i(q_i|b, a_i, \bar{o}) = 0$ consists of summations and maximums of terms of form

$$O(o_k|b, a_i, a_j)L_i(q_i|q_j, b, a_i, a_j, o_k, o_j) = 0.$$

The loss L_i is always positive, since by definition of q_i^* the value of any policy given an observation can not exceed q_i^*. Since the observation probability is always positive as well, and the sum of all the terms must be zero, then each individual term must be zero for the theorem conditions to hold. Thus we do not need to consider the nonzero observation probabilities. Unrolling the summation, we are left with the sum of several terms of the form:

$$V(q_i^*|q_j, b, a_i, a_j, o_k, o_j) - V(q_i|q_j, b, a_i, a_j, o_k, o_j)$$

where q_i is the selected policy that makes the theorem preconditions hold, and o_k is the observation in that term. Thus q_i must equal q_i^* in value, given this belief state, these actions, and this observation. Since q_i has the same value as the best available subpolicy for all the observations in belief state b, for all actions of the other, then q_i is a best available subpolicy for policies rooted in a_i, for all of the observations in $\bar{o}$ while in belief state b. □

PROPOSITION 2. *When we apply the* MBDP-OC *algorithm to a POMDP problem, if $MaxObs \geq MaxTrees$, then* MBDP-OC *constructs the same best available policy tree as* MBDP.

PROOF. Note that the algorithm can be run on a POMDP by considering the problem to be a DEC-POMDP where the second agent is restricted to one policy, action, and observation. MBDP chooses a belief state b and generates the best policy tree for that belief state. We prove that with MBDP-OC, the resulting value lost in the observation compression process is zero for belief state b. Note that a policy consists of a root action, and each subtree corresponds to an observation. There are $|\Omega|$ subtrees. However, since only $MaxTrees$ policies have been saved from the previous step, only $MaxTrees$ of these subtrees are unique.

The policy for each subtree of the best available tree contains the best available subpolicy. Since there are only $MaxTrees$ unique subtrees, this means $(|\Omega| - MaxTrees)$ of the subtrees are a duplicate of some other subtree. Take a subtree under policy branch o_2, and suppose it is a duplicate of the subtree under policy branch o_1. Clearly $q_i^*(b, a, o_1)$ and $q_i^*(b, a, o_2)$ are the same policy. Thus we choose this policy for q_i in computing $WL^*(Q_i|b, a_i, (o_1 \cup o_2))$, for any choice of a_i.

Since $L(q_i^*|b, a, o_1) = L(q_i^*|b, a, o_2) = 0$ and weighted loss can not be less than zero, the MBDP-OC algorithm selects o_1 and o_2 (or some other pair of observations whose weighted loss is also zero) for compression. □

Let Z_{a_i} be the set containing the sets of compressed observations on the policy tree rooted in action a_i found by MBDP-OC. Let $V_{\text{MBDP}}(b, a_i)$ represent the expected reward of the best joint policy for belief state b after MBDP performs a full backup and produces policy trees rooted in a_i for agent i. Likewise, let $V_{\text{MBDP-OC}}(b, a_i)$ be the expected value of the best joint policy for belief state b and trees rooted in a_i after MBDP-OC performs a partial backup using the compressed observations in Z_{a_i}.

THEOREM 2. *Let $z_1, z_2 ... z_{MaxObs}$ be the groups of observations in Z_{a_i}. Let*

$$error(b, Z_{a_i}) = V_{\text{MBDP}}(b, a_i) - V_{\text{MBDP-OC}}(b, a_i).$$

Then there exists a corresponding policy tree produced by MBDP-OC, *which is rooted in a_i and contains sub-policies $q_i^1 .. q_i^{MaxObs}$ such that*

$$error(b, Z_{a_i}) \leq \sum_{k=1}^{MaxObs} WL(q_i^k|b, a_i, z_k)$$

PROOF. Each q_i^k is the subpolicy that MBDP-OC assigns to the observation branch z_k in order to minimize the error. The partial backup in MBDP-OC produces all combinations of assignments of subpolicies for $z_1, ... z_{MaxObs}$, thus we can be assured that $q_i^1 .. q_i^{MaxObs}$ must exist if any combination of the $MaxTrees$ subpolicies retained from the previous MBDP step satisfies the inequality.

For the error term, we perform induction on the number of policies and actions available to the other agent. For the base case, when the other agent has just one policy and action available, we have seen from Lemma 1 that the error introduced when CompressObs creates each z_k is a function of the error introduced when it performed the last merge that created z_k, and not on previous merges. Examining this error, for each $o \in z_k$, fixing the policy of the other agent makes the error introduced by choosing q_i^k simply the difference between the best policy it could choose versus the policy it does choose for each z_k,

$$V(q_i^*|(q_j, b, a_i, a_j, o, o_j)) - V(q_i^k|(q_j, b, a_i, a_j, o, o_j)).$$

The contribution of each o to the total error must be weighted by its probability, and thus the contribution of q_i^k to total error is

$$WL(q_i^k|b, a_i, z_k),$$

and since no observation is in two different sets, the total error, $error(b, Z_{a_i})$, is the sum of the contributions of its components. For the inductive case, assuming that the theorem holds when the other agent's policy is limited to $|A|-1$ root actions, adding another root action choice for the other agent means that there is one more possible joint action at the root (only one, since a_i is fixed), and one more belief state must be considered when analyzing the value of joint subpolicies. The possible loss of value from MBDP must be analyzed separately for this belief state. If $WL(q_i^k|b, a_i, z_k)$ is larger for this belief state than for others, then it is the new error bound and the theorem holds. If not, then by the inductive hypothesis the theorem holds.

The inductive case on the number of subpolicies of the other agent is similar. Adding another subpolicy q_j^k to the other agent means $WL(q_i^k|b, a_i, z_k)$ may or may not be larger than the weighted losses of existing subpolicies. If it is larger, it contributes to the maximum weighted loss and the theorem holds. If not, the inductive hypothesis says that the theorem holds. □

COROLLARY 1. *Given a belief state b, in the worst case,* MBDP-OC *loses* $\max_{a_i \in A}\{error(b, Z_{a_i})\}$ *per iteration in comparison to* MBDP.

PROOF. The previous theorem bounds the loss by MBDP-OC in comparison to MBDP for policy trees rooted in a single action. There is a best joint policy produced by MBDP-OC, and that joint policy is rooted in an action a_i, and the joint expected reward lost by MBDP-OC versus MBDP for that action is $error(b, Z_{a_i})$. □

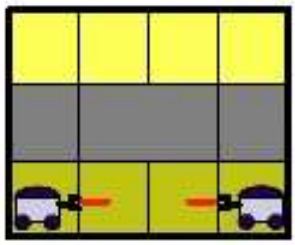

Figure 2: Box Pushing Domain.

horizon	MaxObs=2		MaxObs=3	
	IMBDP	MBDP-OC	IMBDP	MBDP-OC
1	-.2	-.2	-.2	-.2
5	51.6	69.1	79.1	72.3
10	71.3	88.6	90.9	103.9
20	55.2	127.2	96.0	149.8
50	47.8	221.7	80.8	278.7
100	38.2	350.4	72.8	503.8

Table 1: Comparison of IMBDP and MBDP-OC on the Repeated Box Pushing Domain with various horizons. The algorithms were run with *MaxTrees* = 3. Results are shown for *MaxObs* = 2 and *MaxObs* = 3.

5. EXPERIMENTS

Many of the common DEC-POMDP benchmark problems, such as the multi-agent broadcast channel (MABC) and multi-agent tiger, have 2 observations. In order to test the MBDP-OC algorithm, it was necessary to use more complex domains with more than 2 observations. Obviously, with 2 observations, the MBDP-OC algorithm does not do any compression and it devolves into the MBDP algorithm.

The first larger problem that we selected was the Cooperative Box Pushing domain [7], presented in Figure 2. In this domain, agents are required to push boxes into a goal area. The variant we used involves 2 agents, each of which can be located on 4 squares of the bottom row of a 3x4 matrix, and each agent can have 4 possible orientations (facing up, down, left or right). There are walls below and to the left of the matrix. In front of the leftmost and rightmost location are 2 small boxes. A large, 2-square box is in front of the middle two locations. The agents have 4 available actions, turn left, turn right, move forward, or stay. Each action has a .9 probability of success. They can receive 5 possible observations of what is in front of them: empty, wall, other agent, small box, or large box. If an agent moves forward, and a small box is in front of it, the box will be pushed into the top row. If both agents push the large box at the same time, it is pushed into the top row. A reward of 10 is received for pushing a small box into the top row, and a cooperative reward of 100 (50 per agent) is received when both agents push the large box into the top row at the same time. A penalty of 5 is received for bumping into a wall or trying to push the large box alone, and a penalty of 6 is received each time it bumps into the other agent.

Each time a box is pushed forward, the goal state is entered, and the problem resets. The initial state of the problem is with the agents at coordinates (3,1) and (3,4), facing upwards. In order to make the problem more challenging, and to enhance the role of observations, we would have the agents transition to a random state when the problem resets itself. Thus, for instance, agent 1 could find itself facing the back wall and need to turn around. There are 96 reachable non-goal states since the domain forces agent 1 to be left of agent 2. There are 4 possible goal states which reset the problem. We ran the experiments with parameters $MaxTrees = 3$ and tried $MaxObs = 2$ as well as $MaxObs = 3$. Results are displayed in Table 1. We ran IMBDP with the same parameters, and report results for comparison. Besides IMBDP, we know of no other solver in the literature that can produce a solution for DEC-POMDP problems this large. The results show that the value function computed by MBDP-OC produces improved policies, for both $MaxObs = 2$ as well as $MaxObs = 3$. Runtimes are shown in Table 2. There is a small time penalty for running MBDP-OC. However, the program still spends the majority of its time in the classic MBDP portion of the algorithm, where it must evaluate all possible combinations of generated joint policies in all possible states, in order to pick out the $MaxTrees$ policies to retain for the next step.

Algorithm	MaxObs	Time (s)
IMBDP	2	29.9
MBDP-OC	2	31.7
IMBDP	3	459.0
MBDP-OC	3	469.9

Table 2: Running time per horizon step for the IMBDP and MBDP-OC algorithms on the Box Pushing problem. Results are in seconds.

We next chose a domain that should be more favorable to IMBDP than BoxPushing. We ran on a 3×3 instance of the Meeting in a Grid problem introduced by Bernstein et al. [3]. In our implementation of this problem, agents start in opposite corners of a 3×3 grid. Each agent can move either up, down, left, or right. We chose for the chance of a successful action to be 60%, with a 15% chance of moving in each perpendicular direction, a 5% chance of not moving at all, and a 5% chance of moving in the opposite direction. There is an obstacle in the center square, and the agents cannot move there. Each agent receives observations as to whether the squares to the left, right, above, or below each agent are blocked. With the obstacle in the middle, and the grid being blocked at the borders, there are 6 legal combinations of observations. Thus the domain has 6 observations. Observations are 100 percent reliable. When the agents reach the same square, a reward of 1 is received, and the problem repeats itself. This domain should be more favorable to IMBDP, since its weakness, the fact that it chooses a single action and does not explore the policy space for improbable observations, is largely irrelevant in this domain. In this domain, the MBDP planners typically pick a square to meet at (they pick this implicitly, through the policies they choose to retain), and once this is done, there is a clear single choice of action for each observation in the domain. MBDP-OC, by contrast, has its weakness exposed, in that it may not be correct to consider the same policies for groups of observations. It can only be saved by using its value function, to assure that its merge operations will be as harmless as possible given the likely state. If it can do this successfully, it may be able to outperform IMBDP because it can more fully explore the policy space. Experiments were run with $MaxTrees = 3$ and $MaxObs = 2$. Results are shown in Table 3. Indeed we see that under this domain, IMBDP was able to attain more comparable results. Still we find MBDP-OC to have a small but persistent advantage. Runtime for IMBDP was 76.4 seconds per horizon step, and for MBDP-OC it was 83.36.

horizon	IMBDP	MBDP-OC
1	0.0	0.0
5	.41	.47
10	1.01	1.04
20	2.01	2.38
50	5.30	6.38
100	11.6	13.0

Table 3: Comparison of IMBDP and MBDP-OC on the 3×3 Meeting in a Grid problem.

6. CONCLUSION

We present a value-based observation compression technique that helps reduce the space and running time of finite-horizon DEC-POMDP algorithms. One key contribution is the development of a precise measure of the value lost when different observations are grouped together. This allows us to merge observations that lead to minimal loss of value and assign all the members of the group the same policy. One can think of this as an abstraction technique that creates categories of indistinguishable observations. Once the categories are generated, the agent no longer distinguishes between different observations within each category. This in turn helps to simplify the policy construction process.

The actual benefits of this observation compression technique are evaluated by integrating it into MBDP, a leading solution technique for DEC-POMDPs. We perform a rigorous theoretical analysis of the resulting algorithm and provide error bounds on the approximation. Empirically, the resulting algorithm, MBDP-OC, performs well. It is able to generate better joint policies than any existing planner for finite-horizon DEC-POMDPs on domains with several observations.

Value-based observation compression is applicable to any policy-tree based algorithm, not just MBDP. For instance, optimal planners that produce a bounded error, such as epsilon pruning [1], could benefit from compressing observations until a certain loss threshold is exceeded. In this paper, the compression process was run on a particular belief state, and the resulting analysis bounded the value lost for that belief state. This can be extended to a global error bound for *any* belief state by computing the error bound for each underlying system state and then taking the maximum value.

Another future direction for this work will include a more careful consideration of the likely policy of the other agent. In this paper, the policy generation process considered all possible policies for the other agent equally. In fact, it is more important to consider policies of the other agent that are likely to generate high value. Future work will include sampling or otherwise predicting the policy of the other agent, so that each agent can use this knowledge to guide its own policy generation process. Now that the policy generation process is not exponential in the number of observations in the domain, the largest source of computational complexity is the mere fact that each agent needs to consider each possible combination of policies for all of the other agents. If this last large source of complexity were mitigated, future planners would be able to produce good solutions for much larger DEC-POMDP domains.

7. ACKNOWLEDGMENTS

We thank Sven Seuken for providing us with the IMBDP source code. This work was supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-05-1-0254 and by the National Science Foundation under Grants No. 0328601 and 0535061. The findings and views expressed in this paper are those of the authors and do not necessarily reflect the positions of the sponsors.

8. REFERENCES

[1] C. Amato, A. Carlin, and S. Zilberstein. Bounded dynamic programming for decentralized POMDPs. In *AAMAS 2007 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, Honolulu, Hawaii, 2007.

[2] D.S. 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] D.S. Bernstein, E. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proc. of the Nineteenth International Joint Conf. on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[4] P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[5] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the Nineteenth National Conf. on Artificial Intelligence*, San Jose, CA, 2004.

[6] J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proc. of the Nineteenth International Joint Conf. on Artificial Intelligence*, 1332–1338, Edinburgh, Scotland, 2005.

[7] C. R. Kube and H. Zhang. Task modelling in collective robotics. *Autonomous Robots*, 4(1):53–72, 1997.

[8] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. of the Eighteenth International Joint Conf. on Artificial Intelligence*, 1025–1032, Acapulco, Mexico, 2003.

[9] P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems*, 15, 1547–1554, Vancouver, BC, 2002.

[10] S. Seuken and S. Zilberstein. Formal Models and Algorithms for Decentralized Control of Multiple Agents. Technical Report 05-68, Department of Computer Science, University of Massachusetts Amherst, 2005.

[11] S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proc. of the Twenty-Third Conf. on Uncertainty in Artificial Intelligence*, Vancouver, BC, 2007.

[12] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs, In *Proc. of the Twentieth Conf. on Uncertainty in Artificial Intelligence*, 520–527, Banff, Canada, 2004.

[13] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–200, 2005.

[14] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proc. of the Twenty-First Conf. on Uncertainty in Artificial Intelligence*, Edinburgh, Scotland, 2005.

[15] Y. Virin, G. Shani, S. E. Shimony, and R. I. Brafman. Scaling up: Solving POMDPs through value based clustering. In *Proc. of the Twenty-Second Conference on Artificial Intelligence*, 1910–1911, Vancouver, BC, 2007.