# Memory-Bounded Dynamic Programming for DEC-POMDPs

**Sven Seuken**[*]
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
seuken@eecs.harvard.edu

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

## Abstract

Decentralized decision making under uncertainty has been shown to be intractable when each agent has different partial information about the domain. Thus, improving the applicability and scalability of planning algorithms is an important challenge. We present the first memory-bounded dynamic programming algorithm for finite-horizon decentralized POMDPs. A set of heuristics is used to identify relevant points of the infinitely large belief space. Using these belief points, the algorithm successively selects the best joint policies for each horizon. The algorithm is extremely efficient, having linear time and space complexity with respect to the horizon length. Experimental results show that it can handle horizons that are multiple orders of magnitude larger than what was previously possible, while achieving the same or better solution quality. These results significantly increase the applicability of decentralized decision-making techniques.

## 1 Introduction

For over 50 years, researchers in artificial intelligence and operations research have been working on decision making under uncertainty. The *Markov decision process* (MDP) has been proved to be a useful framework for centralized decision making in fully observable stochastic environments. In the 1960's, partially observable MDPs (POMDPs) were introduced to account for imperfect state information. An even more general problem results when two or more agents have to cooperate to optimize a joint reward function, while having different local observations. This problem arises in many application domains such as multi-robot coordination, manufacturing, information gathering and load balancing.

The decentralized partially observable MDP (DEC-POMDP) framework is one way to model these problems. It has been shown that finite-horizon DEC-POMDPs are NEXP-complete [Bernstein *et al.*, 2000]. Thus, decentralized control of multiple agents is significantly harder than single-agent control and provably intractable. Even $\varepsilon$-approximations are hard [Rabinovich *et al.*, 2003]. Due to these complexity results, optimal algorithms have mostly theoretical significance. Consequently, researchers have started to focus on approximate algorithms to solve significantly larger problems. But scalability remains a major research challenge and the main question – how to use the available memory and time most efficiently – remains unanswered. A detailed survey of existing formal models, complexity results and planning algorithms is available in [Seuken and Zilberstein, 2005].

This paper presents a fundamentally new approach to overcome the time and space complexity of existing optimal and approximate algorithms. Most of the previous algorithms cannot solve problems with a horizon larger than 10. Our goal has been to increase this horizon barrier by several orders of magnitude. The approach is based on the first exact dynamic programming (DP) algorithm for finite-horizon DEC-POMDPs [Hansen *et al.*, 2004]. The original DP algorithm, presented in Section 3, builds the final solution *bottom-up*, starting with a 1-step policy for the last time step and successively working towards the first time step. In every iteration, it eliminates unnecessary strategies to reduce the total number of policies kept in memory. Unfortunately, the algorithm runs out of memory very quickly. The key observation is that this algorithm keeps too many policies in memory, even though only very few of them are actually necessary for optimal or near-optimal behavior. Thus, the main idea is to identify a small set of policies that are actually useful for the construction of good joint policies. This is achieved by using a set of top-down heuristics to identify tentative policies that are not necessarily optimal, but can be used to explore the belief space. Section 4 describes how a set of relevant belief states can be identified using these heuristics. The DP algorithm can then select the optimal joint policy trees for those belief states, leading to a bounded number of good policy trees. Section 5 presents the details of our memory-bounded dynamic programming algorithm and proves its linear time and space complexity with respect to the horizon length. We have applied our algorithm to a set of standard test problems from the literature. Section 6 details the results and shows that the algorithm can solve problems with horizons that are multiple orders of magnitude larger than what was previously possible.

[*]This work was done while the author was a graduate student in the Computer Science Department of the University of Massachusetts, Amherst.

## 2 Related Work

Over the last five years, researchers have proposed a wide range of optimal and approximate algorithms for decentralized multi-agent planning. In this paper we focus on finite-horizon problems; infinite-horizon problems often require different solution techniques. One important class of finite-horizon algorithms is called "Joint Equilibrium-based Search for Policies" (JESP), where the algorithms do not search for the globally optimal solution, but instead aim for local optimality [Nair *et al.*, 2003]. The authors introduce the idea of focusing on the *reachable belief states*, but the approach still leads to exponential complexity.

The approach that is closest to our work is called Point-Based Dynamic Programming for DEC-POMDPs [Szer and Charpillet, 2006]. That algorithm computes the policies based on a subset of the reachable belief states. But in contrast to our work, this approach does not employ top-down heuristics to identify the most useful belief states and it is not memory-bounded. It still leads to double-exponential worst-case complexity.

A DEC-POMDP can also be seen as a partially observable stochastic game (POSG) with common payoffs [Emery-Montemerlo *et al.*, 2004]. In this approach, the POSG is approximated as a series of smaller Bayesian games. Interleaving planning and execution, this algorithm finds good solutions for short horizons, but it still runs out of memory after horizon 10.

Another way of addressing the high complexity of DEC-POMDPs is to develop algorithms for problem classes that exhibit certain structure [Becker *et al.*, 2004; Goldman and Zilberstein, 2005]. However, such techniques still have exponential complexity unless the interaction between the agents is very limited. A model that is particularly useful if the agents do not share the same payoff function is the I-POMDP framework [Gmytrasiewicz and Doshi, 2005]. I-POMDPs have similar complexity barriers, leading the authors to introduce an approximate algorithm based on particle filtering [Doshi and Gmytrasiewicz, 2005]. This approach addresses the belief space complexity, but the policy space complexity remains too high and limits scalability.

## 3 Solution Techniques for DEC-POMDPs

We formalize the problem using the DEC-POMDP framework [Bernstein *et al.*, 2000]. Our results, however, apply to equivalent models such as the MTDP or the COM-MTDP [Pynadath and Tambe, 2002].

**Definition 1 (DEC-POMDP)** A finite-horizon decentralized partially observable Markov decision process is a tuple $\langle I, S, b_0, \{A_i\}, P, \{\Omega_i\}, O, R, T\rangle$ where

- $I$ is a finite set of agents indexed 1,...,n.
- $S$ is a finite set of states.
- $b_0 \in \Delta S$ is the initial belief state (state distribution).
- $A_i$ is a finite set of actions available to agent $i$ and $\vec{A} = \otimes_{i\in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, ..., a_n\rangle$ denotes a joint action.
- $P$ is a Markovian transition probability table. $P(s'|s,\vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state $s$ results in a transition to state $s'$.
- $\Omega_i$ is a finite set of observations available to agent $i$ and $\vec{\Omega} = \otimes_{i\in I}\Omega_i$ is the set of joint observations, where $\vec{o} = \langle o_1, ..., o_n\rangle$ denotes a joint observation.
- O is a table of observation probabilities. $O(\vec{o}|\vec{a}, s')$ denotes the probability of observing joint observation $\vec{o}$ given that joint action $\vec{a}$ was taken and led to state $s'$.
- $R : S \times \vec{A} \to \Re$ is a reward function. $R(\vec{a}, s')$ denotes the reward obtained from transitioning to state $s'$ after taking joint action $\vec{a}$.
- $T$ denotes the total number of time steps.

Because the underlying system state of a DEC-POMDP is not available to the agents during execution time, they must base their actions on beliefs about the current situation. In a single-agent setting, the *belief state*, a distribution over states, is sufficient for optimal action selection. In a distributed setting, each agent must base its actions on a *multi-agent belief state* – a distribution over states and over the other agents' policies. Unfortunately, there is no compact representation of this multi-agent belief state. It only exists implicitly for each agent if the joint policy is known and the entire history of local observations is taken into consideration.

**Definition 2 (Local and Joint Policies for a DEC-POMDP)** A **local policy** for agent $i$, $\delta_i$, is a mapping from local histories of observations $\overline{o}_i = o_{i1}\cdots o_{it}$ over $\Omega_i$, to actions in $A_i$. A **joint policy**, $\delta = \langle \delta_1, ..., \delta_n\rangle$, is a tuple of local policies, one for each agent.

Solving a DEC-POMDP means finding a joint policy that maximizes the expected total reward. A policy for a single agent $i$ can be represented as a decision tree $q_i$, where nodes are labeled with actions and arcs are labeled with observations (a so called *policy tree*). Let $Q_i^t$ denote the set of horizon-$t$ policy trees for agent $i$. A solution to a DEC-POMDP with horizon $t$ can then be seen as a vector of horizon-$t$ policy trees, a so called *joint policy tree* $\delta^t = (q_1^t, q_2^t, ..., q_n^t)$, one policy tree for each agent, where $q_i^t \in Q_i^t$. These policy trees can be constructed in two different ways: top-down or bottom-up. If the goal is the optimal policy, both techniques lead to the same final policy. But if the goal is an approximate solution, the different characteristics of the construction processes can be exploited.

**Top-down Approach** The first algorithm that used a top-down approach, MAA$^*$, makes use of heuristic search techniques [Szer *et al.*, 2005]. It is an extension of the standard A$^*$ algorithm where each search node contains a joint policy tree. For example, if $\delta^2$ is a horizon-2 joint policy tree, an expansion of the corresponding search node generates all possible joint policy trees of horizon 3 that use the joint policy tree $\delta^2$ for the first two time steps. Using a set of heuristics that are suitable for DEC-POMDPs, some parts of the search tree can be pruned. But the algorithm runs out of time very quickly, because the search space grows double exponentially.

**Bottom-up Approach: Dynamic Programming** The first non-trivial algorithm for solving DEC-POMDPs used a bottom-up approach [Hansen *et al.*, 2004]. Policy trees were constructed incrementally, but instead of successively coming closer to the frontiers of the trees, this algorithm starts at the frontiers and works its way up to the roots using dynamic programming (DP). The policy trees for each agent are kept separately throughout the construction process. In the end, the best policy trees are combined to produce the optimal joint policy. The DP algorithm has one iteration per step in the horizon. First, all policies with horizon 1 are constructed. In each consecutive iteration, the DP algorithm is given a set of horizon-$t$ policy trees $Q_i^t$ for each agent $i$. The set $Q_i^{t+1}$ is then created by an *exhaustive backup*. This operation generates every possible depth-$t+1$ policy tree that makes a transition, after an action and observation, to the root node of some depth-$t$ policy tree. After the exhaustive backup, the size of the set of policy trees is $|Q_i^{t+1}| = |A||Q_i^t|^{|O|}$. If all policy trees are generated for every step in the horizon, the total number of complete policy trees for each agent is of the order $\mathcal{O}(|A|^{(|O|^T)})$. This double exponential blow-up is the reason why the naive algorithm would quickly run out of memory.

To alleviate this problem, the algorithm uses *iterated elimination of dominated policies*. For two agents $i$ and $j$, a policy tree $q_i \in Q_i$ is *dominated*, if for every possible multi-agent belief state $mb \in \Delta(S \times Q_j)$ there is at least one other policy tree $q_k \in Q_i \setminus q_i$ that is as good as or better than $q_i$. 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. If used in every iteration, this technique significantly reduces the number of policy trees kept in memory, because for every eliminated policy tree $q$, a whole set of policy trees containing $q$ as a subtree is implicitly eliminated as well. But, even with this pruning technique, the number of policy trees still grows quickly and the algorithm runs out of memory even for very small problems.

A more detailed analysis of the construction process shows that most of the policy trees kept in memory are useless. These policy trees could be eliminated early on, but they are not. One reason is that a policy tree can only be eliminated if it is dominated for *every* belief state. But for many DEC-POMDPs, only a small subset of the belief space is actually reachable. Another reason is that a policy tree can only be eliminated if it is dominated for every possible belief over the other agents' policies. But obviously, during the construction process, the other agents also maintain a large set of policy trees that will eventually prove to be useless. Inevitably, these policy trees are also considered in the test for dominance which inhibits the elimination of a large set of policies.

Unfortunately, these drawbacks of the pruning process cannot be avoided. Before the algorithm reaches the root of the policy trees, it cannot predict which beliefs about the state and about the other agents' policies will eventually be useful. This observation leads to the idea of combining the bottom-up and top-down approaches: Using top-down heuristics to identify relevant belief states for which the dynamic programming algorithm can then evaluate the bottom-up policy trees and select the best joint policy. Figure 1 illustrates this idea.

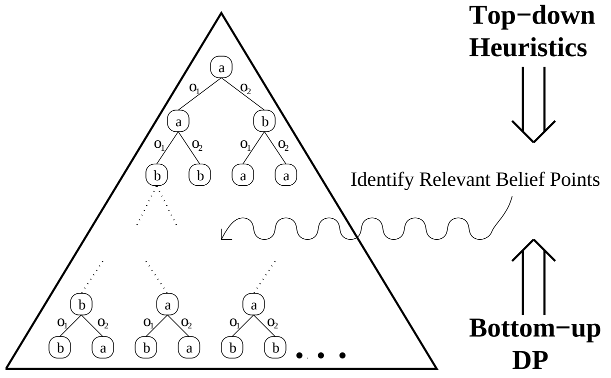

Figure 1: The construction process of a single-agent policy tree combining the top-down and the bottom-up approach.

## 4 Top-Down Heuristics for DEC-PODMPs

Even though the agents do not have access to the belief state during execution time, it can be used to evaluate the bottom-up policy trees computed by the DP algorithm. Policy trees that are good for a centralized belief state are often also good candidates for the decentralized policy. Obviously, a belief state that corresponds to the optimal joint policy is not available during the construction process. But fortunately, a set of belief states can be computed using multiple top-down heuristics – efficient algorithms that find useful top-down policies. Once a top-down heuristic policy is generated, the most likely belief state can be computed. Obviously, this does only lead to exactly *one* belief state. But depending on the specific problem, the bottom-up DP algorithm can handle up to a few dozen belief state candidates (at most 50 in our experiments). A whole set of reachable belief states can be identified using standard sampling techniques. The rest of this section describes several useful heuristics to identify these belief states.

**The MDP Heuristic** A DEC-POMDP can be turned into a fully-observable MDP by revealing the underlying state after each step. A joint policy for the resulting MDP assigns joint actions to states, thus this policy cannot be used for the real DEC-POMDP. But during the construction process, it has proven very useful to identify reachable belief states that are very likely to occur.

**The Infinite-Horizon Heuristic** Infinite-horizon DEC-POMDPs are significantly different from finite-horizon DEC-POMDPs. Instead of maximizing expected reward over $T$ time steps, the agents maximize the reward over an infinite time period. Obviously, optimal policies for finite-horizon problems with a very short horizon may be very different from those for infinite-horizon problems. But the longer the horizon, the more similar the solutions become in general. Thus, infinite-horizon policies can be used as top-down heuristics to identify useful belief states for problems with sufficiently long horizons. Infinite-horizon DEC-POMDPs are undecidable and thus it is generally impossible to determine when optimal policies for finite and infinite horizon problems match. But because the rewards are discounted over time, a bound on the difference in value can be guaranteed. Recently, approximate algorithms have been introduced that can solve those problems very efficiently [Bernstein *et al.*, 2005]. The resulting policies can be used to simulate multiple runs where the execution is simply stopped when the desired horizon and a corresponding belief state are reached.

**The Random Policy Heuristic** The MDP heuristic and the infinite-horizon heuristic can be modified by adding some $\varepsilon$-exploration – choosing random actions with probability $\varepsilon$ in every step. This helps cover a larger area of the belief space, in particular if the other heuristics are not well suited for the problem. For the same reason, a completely random heuristic can also be used – choosing actions according to a uniform distribution. This provides a set of reachable belief states and is not dependent on the specific problem. Obviously, those random policies achieve low values. But this is not a problem, because only the belief states and not the policies are used.

**Heuristic Portfolio** The usefulness of the heuristics and, more importantly, the computed belief states are highly dependent on the specific problem. For some problems, the solution computed with the MDP heuristic might be very similar to the optimal policy and accordingly the computed belief states are very useful. But for other problems, this heuristic might focus the algorithm on the *wrong* parts of the belief space and consequently it may select poor policy-tree candidates for the decentralized case. This problem can be alleviated by using what we call a *heuristic portfolio*. Instead of just using one top-down heuristic, a whole set of heuristics can be used to compute a set of belief states. Thus, each heuristic is used to select a subset of the policy trees. Learning algorithms could optimize the composition of the heuristic portfolio, but this aspect is beyond the scope of this paper.

## 5 Memory-Bounded Dynamic Programming

The MBDP algorithm combines the bottom-up and top-down approaches. The algorithm can be applied to DEC-POMDPs with an arbitrary number of agents, but to simplify notation, the description in Algorithm 1 is for two agents, $i$ and $j$. The policy trees for each agent are constructed incrementally using the bottom-up DP algorithm. But to avoid the double exponential blow-up, the parameter $maxTrees$ is chosen such that a full backup with this number of policy trees of length $T-1$ does not exceed the available memory. Every iteration of the algorithm consists of the following steps. First, a full backup of the policies from the last iteration is performed. This creates policy tree sets of size $|A||maxTrees|^{|O|}$. Next, top-down heuristics are chosen from the portfolio and used to compute a set of belief states. Then, the best policy tree pairs for these belief states are added to the new sets of policy trees. Finally, after the $T$th backup, the best joint policy tree for the start distribution is returned.

### 5.1 Theoretical Properties

Overcoming the complexity barriers of DEC-POMDPs is challenging, because there is a double exponential number of possible policy trees, each containing an exponential number of nodes in the time horizon. Accordingly, an important feature of the MBDP algorithm is that it can compute and represent the policy trees using just linear space.

**Algorithm 1**: The MBDP Algorithm

```
1  begin
2      $maxTrees \leftarrow$ max number of trees before backup
3      $T \leftarrow$ horizon of the DEC-POMDP
4      $H \leftarrow$ pre-compute heuristic policies for each $h \in H$
5      $Q_i^1, Q_j^1 \leftarrow$ initialize all 1-step policy trees
6      for $t$=1 to $T$ do
7          $Q_i^{t+1}, Q_j^{t+1} \leftarrow$ fullBackup($Q_i^t$), fullBackup($Q_j^t$)
8          $Sel_i^{t+1}, Sel_j^{t+1} \leftarrow$ empty
9          for $k$=1 to maxTrees do
10             choose $h \in H$ and generate belief state $b$
11             foreach $q_i \in Q_i^{t+1}, q_j \in Q_j^{t+1}$ do
12                 evaluate each pair $(q_i, q_j)$ with respect to $b$
13             add best policy trees to $Sel_i^{t+1}$ and $Sel_j^{t+1}$
14             delete these policy trees from $Q_i^{t+1}$ and $Q_j^{t+1}$
15         $Q_i^{t+1}, Q_j^{t+1} \leftarrow Sel_i^{t+1}, Sel_j^{t+1}$
16     select best joint policy tree $\delta^T$ from $\{Q_i^T, Q_j^T\}$
17     return $\delta^T$
18 end
```

**Theorem 1.** *The MBDP algorithm has a linear space complexity with respect to the horizon length.*

***Proof:*** Once the variable $maxTrees$ is fixed, the number of policy trees per agent is always between $k$ and $K$. The lower limit $k$ is equal to $maxTrees$, which is the number of trees after the heuristics have been used to select the $k$-best policy trees. The upper limit $K$ is equal to $|A||maxTrees|^{|O|}$, which is the number of policy trees after a full backup. By choosing $maxTrees$ appropriately, the desired upper limit can be pre-set. In every iteration of the algorithm, no more than $K$ policy trees are constructed. For the construction of the new policy trees, the algorithm uses pointers to the $k$ policy trees of the previous level and is thus able to potentially attach each subtree multiple times instead of just once as illustrated in Figure 2. Due to this efficient pointer mechanism, a final horizon-$T$ policy tree can be represented with $k \cdot T$ decision nodes. Thus, the amount of space grows linearly with the horizon length. For n agents, this is $\mathcal{O}(n(k \cdot T + K))$. $\square$

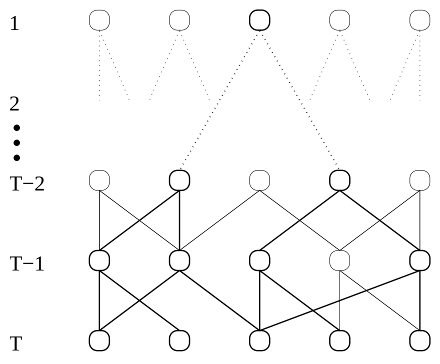

Figure 2: An exponential size policy can be represented using linear space by re-using policy trees. Here $maxTrees = 5$.

In general, the idea of the pointer mechanism could be applied to other algorithms and improve them as well. It reduces the number of nodes used for one final policy tree from $\mathcal{O}(|O|^T)$ to $\mathcal{O}(k \cdot T)$. However, if $k$ grows double exponentially – as in the optimal DP algorithm – using the pointer mechanism only has marginal effects. For an improvement in space complexity it is also necessary that $k$ – the number of possible policy trees kept in memory – be sufficiently small.

**Theorem 2.** *The MBDP algorithm has a linear time complexity with respect to the horizon length.*

***Proof:*** The main loop of the algorithm (lines 6-15) depends linearly on the horizon length $T$. Inside this loop, all operations are independent of $T$ once $k$ and $K$ are fixed. The remaining critical operation is in line 4, where the heuristic policies are pre-computed. Currently, the only heuristic that guarantees linear time is the random policy heuristic. The other heuristics have higher, but still polynomial-time complexity. In practice, the time used by the heuristics is negligible. If a top-down heuristic is used that can be computed in linear time, the time complexity of the whole algorithm is linear in the horizon length. $\square$

## 5.2 Recursive MBDP

As pointed out earlier, the effectiveness of any top-down heuristic may depend on the specific problem. To alleviate this drawback, we use a heuristic portfolio. But once the algorithm has computed a complete solution, we have a joint policy that definitely leads to relevant belief states when used as a heuristic. This is exactly the idea of *Recursive MBDP*. The algorithm can be applied recursively with an arbitrary recursion-depth. After the first recursion has finished, the final joint policy can be used as a top-down heuristic for the next run and so on.

## 5.3 Pruning Dominated Policy Trees

As described in Section 3, the exact DP algorithm uses iterated elimination of dominated strategies to reduce the size of the policy tree sets. Interestingly, this pruning technique can be combined with the MBDP algorithm in two different ways. First, after a full backup is performed, the pruning algorithm can be used to eliminate dominated policy trees, which reduces the set of remaining policy trees. Consequently, no dominated policy tree will be selected using the top-down heuristics. Second, the pruning technique can be applied after the heuristics have been used to select the policy trees for the next iteration. Pruning dominated policy trees at this step leads to a more efficient use of memory in the next iteration.

## 6 Experiments

We have implemented the MBDP algorithm and performed intensive experimental tests. The algorithm has three key parameters that affect performance. One parameter is the maximum number of trees, $maxTrees$, which defines the memory requirements; runtime is quadratically dependent on $maxTrees$. Its effect on the value of the solution is analyzed in Section 6.2. The second parameter is the depth of the recursion, which affects runtime linearly. We define a trial run of the algorithm as the entire computational process including all recursive calls. The best solution found during one trial run is returned as the final solution. Because the algorithm is partly randomized, it is obvious that on average, the higher the recursion depth, the better the solution value. The third parameter controls the selection of heuristics from the portfolio. In the following experiments we used a uniform selection method (giving the same weight to all heuristics) because it performed well without fine-tuning. We also experimented with the pruning technique described in Section 5.3, but it had little impact on the runtime or solution quality. Hence, we decided not to include it in the following experiments.

| | MABC Problem | | Tiger Problem | | |
|---|---|---|---|---|---|
| Horizon | Value | Time(s) | Value | $\sigma$ | Time(s) |
| 3 | 2.99 | 0.01 | 5.19 | 0 | 0.19 |
| 4 | 3.89 | 0.01 | 4.80 | 0 | 0.46 |
| 5 | 4.79 | 0.02 | 5.38 | 0.42 | 0.72 |
| 10 | 9.29 | 0.08 | 13.49 | 1.76 | 2.19 |
| 100 | 90.29 | 0.26 | 93.24 | 7.79 | 25.84 |
| 1,000 | 900.29 | 2.5 | 819.01 | 20.5 | 289.3 |
| 10,000 | 9,000.29 | 50 | 7930 | 52.5 | 4092 |
| 100,000 | 90,000.29 | 3000 | 78252 | 87.9 | 44028 |

Table 1: Performance of the MBDP algorithm.

## 6.1 Benchmark Problems

We have applied the MBDP algorithm to two benchmark problems from the literature: the *multi-access broadcast channel* (MABC) problem involving two agents who send messages over a shared channel and try to avoid collisions [Hansen *et al.*, 2004], and the *multi-agent tiger* problem, where two agents have to open one of two doors, one leading to a dangerous tiger and the other to a valuable treasure [Nair *et al.*, 2003]. Table 1 presents performance results of the MBDP algorithm for the two test problems. Shown are solution value and computation time, which are both averaged over 10 trial runs. For the tiger problem, we also show the standard deviation $\sigma$, because in this case the algorithm achieved different solution values over the 10 trials. For the MABC problem, we used $maxTrees = 3$ and a recursion depth of 1. For the tiger problem we used $maxTrees = 7$ and a recursion depth of 5.

To evaluate the MBDP algorithm we compared it to the optimal DP algorithm [Hansen *et al.*, 2004], the JESP algorithm [Nair *et al.*, 2003], the Point-Based Dynamic Programming algorithm (PBDP) [Szer and Charpillet, 2006] and a random-policy generating algorithm. These are all offline planning algorithms. The computation is performed in a centralized way and the final solution is a complete joint policy tree for the desired horizon, which can then be executed by multiple agents in a decentralized way. In contrast, the Bayesian Game Approximation algorithm [Emery-Montemerlo *et al.*, 2004] interleaves planning with execution. Thus, it allows for some amount of decentralized re-planning after each step. To perform a consistent comparison, we also created an online version of MBDP and in fact it outperformed the other algorithm. But a discussion of the online algorithms is beyond the scope of this paper and thus the following performance comparison does not include results for the online algorithms.

| | MABC Problem | | | | Tiger Problem | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Horizon | Optimal | PBDP | Random | **MBDP** | Optimal | JESP | PBDP | Random | **MBDP** |
| 2 | 2.00 | 2.00 | 1 | **2.00** | -4.00 | -4.00 | -4.00 | -92.44 | **-4.00** |
| 3 | 2.99 | 2.99 | 1.49 | **2.99** | 5.19 | -6.00 | 5.19 | -138.67 | **5.19** |
| 4 | 3.89 | 3.89 | 1.99 | **3.89** | 4.80 | ? | 4.80 | -184.89 | **4.80** |
| 5 | - | 4.70 | 2.47 | **4.79** | - | ? | - | -231.11 | **5.38** |
| 6 | - | 5.69 | 2.96 | **5.69** | - | ? | - | -277.33 | **9.91** |
| 7 | - | 6.59 | 3.45 | **6.59** | - | ? | - | -323.56 | **9.67** |
| 8 | - | 7.40 | 3.93 | **7.49** | - | - | - | -369.78 | **9.42** |
| 9 | - | - | 4.41 | **8.39** | - | - | - | -416.00 | **12.57** |
| 10 | - | - | 4.90 | **9.29** | - | - | - | -462.22 | **13.49** |
| 100 | - | - | 48.39 | **90.29** | - | - | - | -4,622.22 | **93.24** |
| 1,000 | - | - | 483.90 | **900.29** | - | - | - | -46,222.22 | **819.01** |
| 10,000 | - | - | 4,832.37 | **9,000.29** | - | - | - | -462,222.22 | **7930.68** |
| 100,000 | - | - | 48,323.09 | **90,000.29** | - | - | - | -4,622,222.22 | **78252.18** |

Table 2: Performance comparison of the different algorithms on the MABC problem and the Tiger problem.

Table 2 presents a performance comparison of the different algorithms for both test problems. For this comparison, the same experimental setup as described above was used. Shown are the solution values achieved for each horizon. A "?" indicates that the algorithm could compute a solution for this horizon, but the achieved value was not reported in the corresponding paper. A "-" indicates that the algorithm ran out of time or memory for this horizon. Even though the experiments have been performed on different systems with different processor speed and available memory, the two key findings of this comparison are still applicable: First, for the horizons where comparison is feasible, the MBDP algorithm achieves the same or higher value than all previous approaches. Second, the results match nicely our theoretical findings: They show that the linear complexity of the MBDP algorithm makes it possible to solve problems with much larger horizons, improving scalability by multiple orders of magnitude with respect to all existing algorithms. None of the previous approaches could compute a solution for the problems with a horizon of 10 whereas the MBDP algorithm can easily compute solutions up to horizon 100,000. For large horizons, the comparison with the random policy algorithm shows that the two test problems are non-trivial, as the difference in achieved value is significant. For example, for the MABC problem the MBDP algorithm achieves approximately 200% of the value the random policy algorithm achieves. On the tiger problem random policies lead to extremely poor performance compared to the MBDP algorithm.

## 6.2 Trade-off: Recursion Depth vs. *MaxTrees*

The parameters $maxTrees$ and the recursion depth present an important trade-off: their increase generally increases both solution value and runtime. We examine this trade-off below and identify the best parameters for the test problems.

One interesting experimental result for the MABC problem is that $maxTrees = 3$ is sufficient to produce the best solution for all horizons, even with recursion depth 1. Apparently, this problem exhibits some structure that makes only a few decision nodes necessary even for long horizons. In this regard, the tiger problem is more complex: for longer horizons, the best solution is not found in every trial, even with $maxTrees > 50$ and a recursion depth of 300. Nevertheless, just a few policy trees are necessary to find near-optimal solutions. Figure 3 shows that when $maxTrees$ is increased, the solution value levels off very quickly. The values shown are for the tiger problem with horizon 10, averaged over 10 trials with a recursion depth of 50. Experiments with other horizons yielded qualitatively similar results.

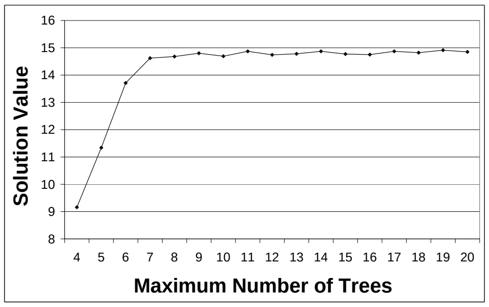

Figure 3: A small maximum number of policy trees is sufficient for finding near-optimal solutions.

The best parameter settings for $maxTrees$ and the recursion depth vary with the available runtime. Table 3 shows experimental results for the tiger problem with horizon 20, where the best parameter settings for given time limits up to 120 seconds have been identified. The solution values shown are averaged over 10 trial runs. The results illustrate the trade-off between the recursion depth and the maximum number of trees. In particular, they show that different parameter settings optimize this trade-off for different time limits.

| Time Limit (s) | MaxTrees | Recursion Depth | Value |
|---|---|---|---|
| 1 | 7 | 1 | 16.37 |
| 2 | 7 | 2 | 20.35 |
| 3 | 7 | 4 | 22.34 |
| 4 | 8 | 3 | 23.67 |
| 5 | 8 | 3 | 23.67 |
| 10 | 11 | 1 | 24.71 |
| 20 | 10 | 5 | 26.98 |
| 30 | 12 | 3 | 27.16 |
| 60 | 9 | 27 | 28.75 |
| 120 | 13 | 8 | 29.02 |

Table 3: The best parameter settings and solution values for the tiger problem with horizon 20 for given time limits.

## 7 Conclusion and Future Work

We have presented the first memory-bounded dynamic programming algorithm for finite-horizon DEC-POMDPs. It is the first to effectively combine top-down heuristics with bottom-up dynamic programming. Unlike previous approximate algorithms, the MBDP algorithm guarantees linear space and time complexity with respect to the horizon length, thereby overcoming the high worst-case complexity barrier of DEC-POMDPs. One particularly attractive feature of the algorithm is that the policy trees it generates are represented using linear space due to an efficient pointer mechanism. In contrast, a single complete policy tree normally contains an exponential number of decision nodes. Our experimental results show that the algorithm performs very well. It is truly scalable in terms of the horizon. It solves problems with horizons that are multiple orders of magnitude larger than what was previously possible, while achieving the same or better solution quality in a fraction of the runtime. We have also analyzed the important trade-off between the maximum number of policy trees kept in memory and the recursion depth of the algorithm. Interestingly, very few policy trees are sufficient to produce near-optimal results for the benchmark problems. We are currently working on a deeper analysis of the theoretical guarantees of the algorithm, i.e. deriving bounds on the approximation error and analyzing the influence of different heuristics on performance.

In future work, we plan to apply the MBDP algorithm to problems with larger state spaces as well as larger action and observation sets. A finite-horizon DEC-POMDP has an exponential complexity with regard to the number of observations. Increasing scalability in that regard is thus an important challenge. For some problem sizes, additional techniques will have to be employed to further improve scalability. One option is to explore the full potential of the pruning technique. Currently, the MBDP algorithm and the pruning algorithm can only be used as two separate entities, which has not proven to be very useful. We are investigating ways to use the top-down heuristics to identify relevant *multi-agent belief states* for which the pruning algorithm checks dominance and then selects the best bottom-up policies. The algorithm and representations used in this work open up multiple research avenues for developing effective approximation algorithms for decentralized decision making under uncertainty.

## 8 Acknowledgments

We thank Marek Petrik and Daniel Bernstein for helpful discussions on this paper. This work was supported in part by the Air Force Office of Scientific Research under Award No. FA9550-05-1-0254 and by the National Science Foundation under Grants No. IIS-0219606 and IIS-0535061.

## References

[Becker *et al.*, 2004] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Solving Transition Independent Decentralized Markov Decision Processes. *Journal of Artificial Intelligence Research (JAIR)*, 22:423–255, 2004.

[Bernstein *et al.*, 2000] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The Complexity of Decentralized Control of Markov Decision Processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 32–37, Stanford, California, June 2000.

[Bernstein *et al.*, 2005] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded Policy Iteration for Decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1287–1292, Edinburgh, Scotland, July 2005.

[Doshi and Gmytrasiewicz, 2005] Prashant Doshi and Piotr J. Gmytrasiewicz. A Particle Filtering Based Approach to Approximating Interactive POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 969–974, Pittsburg, Pennsylvania, July 2005.

[Emery-Montemerlo *et al.*, 2004] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 136–143, 2004.

[Gmytrasiewicz and Doshi, 2005] Piotr J. Gmytrasiewicz and Prashant Doshi. A Framework for Sequential Planning in Multiagent Settings. *Journal of Artificial Intelligence Research (JAIR)*, 24:49–79, 2005.

[Goldman and Zilberstein, 2005] Claudia V. Goldman and Shlomo Zilberstein. Goal-Oriented Dec-MDPs with Direct Communication. Technical Report 04-44, Department of Computer Science, University of Massachusetts Amherst, 2005.

[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 (AAAI)*, pages 709–715, San Jose, California, July 2004.

[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 Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 705–711, 2003.

[Pynadath and Tambe, 2002] David V. Pynadath and Milind Tambe. The Communicative Multiagent Team Decision Problem: Anaylzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research (JAIR)*, 16:389–423, June 2002.

[Rabinovich *et al.*, 2003] Zinovi Rabinovich, Claudia V. Goldman, and Jeffrey S. Rosenschein. The Complexity of Multiagent Systems: The Price of Silence. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1102–1103, Melbourne, Australia, 2003.

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

[Szer and Charpillet, 2006] Daniel Szer and Francois Charpillet. Point-based Dynamic Programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, Boston, USA, 2006.

[Szer *et al.*, 2005] Daniel Szer, Francois Charpillet, and Shlomo Zilberstein. MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 576–583, Edinburgh, Scotland, July 2005.