Trial-Based Dynamic Programming for Multi-Agent Planning

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

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

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

Abstract

Trial-based approaches offer an efficient way to solve single-agent MDPs and POMDPs. These approaches allow agents to focus their computations on regions of the environment they encounter during the trials, leading to significant computational savings. We present a novel trial-based dynamic programming (TBDP) algorithm for DEC-POMDPs that extends these benefits to multi-agent settings. The algorithm uses trial-based methods for both belief generation and policy evaluation. Policy improvement is implemented efficiently using linear programming and a sub-policy reuse technique that helps bound the amount of memory. The results show that TBDP can produce significant value improvements and is much faster than the best existing planning algorithms.

Introduction

We present a trial-based dynamic programming (TBDP) approach for planning in multi-agent settings, and show that it yields better solutions and is much faster than the state-of-the-art planning algorithms. When multiple cooperative decision-makers operate under uncertainty, they must reason about an extremely large set of possible outcomes. A further complication in these settings is the fact that each agent may have to make decisions based on different partial information about the overall situation. This problem arises in such domains as cooperative robots, sensor networks and communication networks. A useful model to study these problems is the decentralized partially observable Markov decision process (DEC-POMDP). The model has been shown to be NEXP-complete (Bernstein *et al.* 2000), leading to extensive efforts to develop efficient approximate algorithms.

The latest dynamic programming (DP) approaches for DEC-POMDPs such as memory-bounded dynamic programming (MBDP) (Seuken & Zilberstein 2007b) and its successors (Seuken & Zilberstein 2007a; Carlin & Zilberstein 2008; Dibangoye *et al.* 2009; Amato *et al.* 2009) have shown to outperform other approximation methods based on game theory or gradient descent, while having linear complexity over the horizon. However, they still suffer from limited scalability and cannot solve large problems such as multi-robot navigation. This is largely due to the "curse of dimensionality" – planning in these domains is challenging because of the high dimensional state, action and observation spaces, which also grow exponentially with the number of agents. Trial-based approaches such as real-time dynamic programming (RTDP) (Barto *et al.* 1995; Geffner & Bonet 1998) have shown to be very efficient in single-agent (PO)MDPs. They focus on computations needed to make improvements locally, but overall converge to good approximate solutions.

Extending trial-based methods to multi-agent settings is not straightforward because arbitrary trials without considering the other agents' future behavior will lead to miscoordination. The novel approach we introduce, TBDP, uses trial-based methods in each iteration both to identify the most useful states and to evaluate the candidate policies. We combine several advantages of policy improvement approaches proposed earlier to efficiently search the space of stochastic policies. Like RTDP-based algorithms, our approach can easily utilize the power of multiple processors, thus providing additional opportunities to scale up planning and learning algorithms in multi-agent domains. To the best of our knowledge, this work is the first trial-base approach to solve DEC-POMDPs and is also the first to lend itself to an efficient multi-processor implementation. Our experimental results show large improvements in solving standard benchmark problems and the ability to tackle larger domains that cannot be solved by existing algorithms.

The paper starts with background on the DEC-POMDP model and the existing dynamic programming solutions. We then describe the trial-based approach, addressing the issues of belief generation, policy improvement and policy evaluation. This is followed by the details of the multi-processor implementation of policy evaluation. Finally, we examine the performance of these algorithms on a set of test problems and conclude with a discussion of future work.

Decentralized POMDPs

The decentralized POMDP model is an extension of the partially observable Markov decision process (POMDP) to multi-agent settings. We adopt the DEC-POMDP framework and notation (Bernstein *et al.* 2000), but our approach and results apply to equivalent models such as MTDP (Pynadath & Tambe 2002) and POIPSG (Peshkin *et al.* 2000). This section describes the model and existing algorithms.

The DEC-POMDP Model

Formally, an n-agent DEC-POMDP is defined as a tuple $\langle S, \{A_i\}, P, \{\Omega_i\}, O, R, T, b^T \rangle$, where

- S is a finite set of states, $b^T \in \Delta(S)$ is a start distribution.
- A_i is a finite set of actions for agent i, and $\vec{a} = \langle a_1, a_2, \cdots, a_n \rangle$ is a joint action, where $a_i \in A_i$.
- $P(s'|s, \vec{a})$ is a transition function.
- Ω_i is a finite set of observations for agent i, and $\vec{o} = \langle o_1, o_2, \cdots, o_n \rangle$ is a joint observation, where $o_i \in \Omega_i$.
- $O(\vec{o}|s', \vec{a})$ is an observation function.
- $R(s, \vec{a})$ is a reward function, and T is the horizon.

The solution of a DEC-POMDP is to find a set of n policies, one for each agent, that maximizes the expected joint reward. Policies can usually be represented as decision trees. Let q_i denote a policy tree and Q_i a set of policy trees for agent i. Q_{-i} denotes the sets of policy trees for all agents but agent i. A joint policy $\vec{q} = \langle q_1, q_2, \cdots, q_n \rangle$ is a vector of policy trees and $\vec{Q} = \langle Q_1, Q_2, \cdots, Q_n \rangle$ denotes the sets of joint policies. The value of a joint policy $\vec{q}$ can be claculated as follows:

$$V(s, \vec{q}) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V(s', \vec{q_o}) \quad (1)$$

where $\vec{a}$ are the actions defined at the root of trees $\vec{q}$, and $\vec{q_o}$ are the subtrees of $\vec{q}$ after $\vec{o}$ have been observed.

Communication between agents is in fact modeled *implicitly* by a DEC-POMDP because the observations of one agent depend on the actions of the others, thereby providing some form of communication. While the execution of policies is inherently decentralized, the computation of policies can be *centralized* and can make use of the DEC-POMDP model, particularly when planning is performed offline.

Dynamic Programming for DEC-POMDPs

In multi-agent settings, each agent must reason about the possible future policies of the others in order to choose optimal actions. This can be done using a *multi-agent belief state* – a probability distribution over system states and the policies of all other agents: $b_i \in \Delta(S \times Q_{-i})$. Such belief states were used in an early dynamic programming approach for decentralized POMDPs – *Joint Equilibrium-based Search for Policies* (DP-JESP), which first generates a set of multi-agent belief states by keeping the policies of other agents fixed, and then computes the value and builds a policy for one agent at a time (Nair *et al.* 2003). This only guarantees local optimality and still leads to exponential complexity due to the exponential number of possible belief points.

An exact dynamic programming algorithm for DEC-POMDPs was developed by Hansen *et al.* (2004). The algorithm builds the policies from the last step towards the first step. In every iteration, it first performs an exhaustive *backup* for each policy tree of the previous iteration, and then prunes *dominated* policies. Although it can produce a globally optimal solution, this approach runs out of memory very quickly because the number of possible policies grow at a double-exponential rate over the horizon.

Point-Based DP (PBDP) exploits the fact that some regions of the belief space are not reachable in many domains (Szer & Charpillet 2006). Unlike DP-JESP, which considers the entire policies of the other agents, PBDP generates a full set of current-step policies via a backup step, and identifies the reachable beliefs by enumerating all possible top-down histories. However, it still leads to double-exponential worst case complexity due to the large number of possible policies and histories.

More recently, memory-bounded DP (MBDP) has been introduced, offering linear time and space complexity over the horizon (Seuken & Zilberstein 2007b). At each iteration, it employs top-down heuristics to identify the most useful belief states and keeps only a fixed number of best policies for these belief states. Hence, it is memory-bounded and can solve much larger problems with essentially arbitrary horizons. One of the major challenges is the *backup* operation, which still has exponential complexity over the observations. Several successive works have been proposed to improve the performance of the backup operation, but they still have exponential complexity in the worst case and scalability remains limited.

We present a novel dynamic programming approach for DEC-POMDPs, namely multi-agent trial-based dynamic programming (TBDP). We combine the main advantages of DP-JESP with MBDP to avoid the expensive *backup* operation. Furthermore, we address effectively the complexity of policy evaluation, which presents another computational bottleneck, particularly in problems with large state spaces.

Trial-Based Dynamic Programming

One important class of (PO)MDP algorithms is based on real-time dynamic programming (RTDP), where the algorithms employ trial-based approaches to improve the value of reachable (belief) states. In trial-based algorithms, agents can interact with the environment and focus their computation only on the regions that they encounter. Hence, each improvement of policies (or value) can be done locally, instead of updating the entire (belief) state space. This has been proved to be an efficient way to address the curse of dimensionality. The advantages of trial-based solutions have been amply demonstrated by the original work on RTDP and its successors (McMahan *et al.* 2005; Smith & Simmons 2006; Bonet & Geffner 2009; Sanner *et al.* 2009). Moreover, our algorithm allows agents to interact with the environment by simulating the model, or assuming that there is a central camera to observe the global state during the planning phase.

At each iteration, TBDP first samples a state distribution by multiple trials, then it makes an improvement to each joint policy. Policy evaluations are also done by sufficient trials. Algorithm 1 shows the main TBDP procedure. The rest of this section describes each part in detail.

Belief Generation

In single-agent POMDPs, the belief state is a probability distribution over domain states, $b \in \Delta(S)$. In DEC-POMDPs, a belief about the underlying system state is not sufficient because agents have to reason about the possible future behavior of all teammates to choose actions. Some researchers

Algorithm 1: Trial-Based DP for DEC-POMDPs

```
Generate a random joint policy
for t=1 to T do // bottom-up iteration
    foreach unimproved joint policy q⃗^t do
        Sample a state distribution b^t by trials
        repeat
            foreach agent i do
                Fix the policies of all the agents except i
                begin formulate a linear program
                    if V(s', q⃗^{t-1}) is required then
                        Evaluate s', q⃗^{t-1} by trials
                end
                Improve the policy q_i^t by solving the LP
        until no improvement in the policies of all agents
return the current joint policy
```

tried to address this by defining beliefs over other agents' beliefs, but this could lead to infinite regress. The *multi-agent belief state* is another way to address this using a distribution over both system states and other agents' policies. In bottom-up dynamic programming such as PBDP and MBDP, these policies are created by backing up the policies of the previous iteration. We focus here on computing a top-down state distribution or so-called joint belief.

Generally, the joint belief state (state distribution) of the current step, b', is computed recursively using Bayes' rule:

$$b'(s') = \alpha O(\vec{o}|s', \vec{a}) \sum_{s \in S} P(s'|s, \vec{a}) b(s) \qquad (2)$$

where α is the normalization factor and b is the belief state of the previous step. However, this is very time-consuming due to the summation over all states. In TBDP, we use trial-based sampling as shown in Algorithm 2. It is straightforward to perform trials either by simulating the model or interacting with the real environment. In order to identify the reachable states, sampling is performed using several types of heuristic policies, such as random policies and solutions of the corresponding MMDP (Boutilier 1996). Notice that the joint beliefs generated by Algorithm 2 are very sparse and can thus be stored in *sparse vectors* to speedup the computation.

Policy Improvement

A policy tree is a conditional plan defined recursively with an action at the root and a subtree for each observation. During execution, each agent follows a path based on its history of local observations and selects an action at each node. Unlike deterministic policy trees used in previous approaches, we allow for stochastic transitions and stochastic action selection, as this helps make policy improvement very efficient. Such stochastic policies are similar to *stochastic finite state controllers* used in infinite-horizon problems or so-called *mixed strategies* in game theory.

Formally, we define the depth-t stochastic policy for agent i, $q_i \in Q_i^t$, recursively to be a tuple $\langle \psi_i, \eta_i \rangle$, where

- ψ_i is an action selection function which defines a probability distribution over the actions $p(a_i|q_i)$.
- η_i is a transition function which specifies the probability $p(q_i'|q_i, o_i)$ of sub-policy $q_i' \in Q_i^{t-1}$ when o_i is observed.

Algorithm 2: Trial-Based Sampling

```
Input: t: the sampled step, δ: the heuristic policy
b(s) ← 0 for every state s ∈ S
for several number of trials do
    s ← draw a state from the start distribution b^T
    for k=T downto t do // top-down trial
        a⃗ ← execute a joint action according to δ
        s', o⃗ ← get the system responses
        s ← s'
    b(s) ← b(s) + 1
return the normalized b
```

The functions ψ_i and η_i define a conditional distribution $p(q_i', a_i|q_i, o_i) = p(a_i|q_i)p(q_i'|q_i, o_i)$. To start, the size of Q_i^t is set to be fixed and identical for every t and i, and each policy $q_i \in Q_i^t$ is initialized with random parameters.

The value function of a joint stochastic policy $\vec{q}$ at state s can be computed by the following equation:

$$V(s, \vec{q}) = \sum_{\vec{a}} \prod_i p(a_i|q_i) R(s, \vec{a}) + \sum_{\vec{a}, \vec{o}, s'} P(s', \vec{o}|s, \vec{a}) \sum_{\vec{q}'} \prod_i p(q_i', a_i|q_i, o_i) V(s', \vec{q}') \qquad (3)$$

where $P(s', \vec{o}|s, \vec{a}) = P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a})$. For a given joint belief state b, the value of the joint stochastic policy $\vec{q}$ is $V(b, \vec{q}) = \sum_s b(s) V(s, \vec{q})$. Thus, the policy improvement step is to find the parameters of $\vec{q}^*$ which maximize the value at a joint belief state b: $\vec{q}^* = \arg\max_{\vec{q}} V(b, \vec{q})$.

We adopt an improvement strategy that was first introduced by Nair *et al.* (2003) and refined by Bernstein *et al.* (2005). The basic idea is to improve the policy of one agent at a time while keeping the other policies fixed. This is repeated until no improvement is possible. Given agent i's policy q_i, the process of searching for new policy parameters can be formulated by the linear programming (LP) shown in Table 1, which maximizes the contribution of agent i's policy to the overall value. The constraints guarantee that all probabilities are positive and add up to 1. To use bounded memory, we take the policy reuse approach introduced in MBDP. That is, a bounded number of sub-policies are used as building blocks for each step and reused multiple times as components of the new policy tree. In general, we start with a random set of policies and then improve them from the last step back to the first step using a series of linear programs.

Policy Evaluation

Evaluating policies using Equation 3 is expensive because it requires computing a value function for every state and joint policy pairs. There are time-consuming summations on the right-hand side of the equation. As mentioned earlier, the reachable state set is very small in many domains, so it is not necessary to evaluate all state and joint policy pairs in this process. Notice that the value functions are used only when formulating the linear program in Table 1. In TBDP, we use a *lazy evaluation* approach whereby the values are computed only when needed, in order to save time and memory.

More precisely, the evaluation of $V(s', \vec{q}')$ is *required* given agent i's sub-policy q_i' in Table 1 when

$$P(s', \vec{o}|b, \vec{a}) \prod_{k \neq i} p(q_k', a_k|q_k, o_k) \neq 0 \qquad (4)$$

Maximize	$\sum_{\vec{a}} x(a_i\|q_i) \prod_{k\neq i} p(a_k\|q_k) R(b,\vec{a}) + \sum_{\vec{a},\vec{o},s'} P(s',\vec{o}\|b,\vec{a}) \sum_{\vec{q}'} x(q_i',a_i\|q_i,o_i) \prod_{k\neq i} p(q_k',a_k\|q_k,o_k) V(s',\vec{q}')$
subject to	$\sum_{a_i} x(a_i\|q_i) = 1, \forall_{a_i,o_i} \sum_{q_i'} x(q_i',a_i\|q_i,o_i) = x(a_i\|q_i), \forall_{a_i} x(a_i\|q_i) \geq 0, \forall_{a_i,o_i,q_i'} x(q_i',a_i\|q_i,o_i) \geq 0.$

Table 1: The linear program to improve agent i's policy q_i, where the variable $x(a_i|q_i)$ represents $p(a_i|q_i)$, the variable $x(q_i', a_i|q_i, o_i)$ represents $p(q_i', a_i|q_i, o_i)$, $R(b,\vec{a}) = \sum_s b(s)R(s,\vec{a})$, and $P(s',\vec{o}|b,\vec{a}) = \sum_s b(s)P(s'|s,\vec{a})O(\vec{o}|s',\vec{a})$.

Algorithm 3: Trial-Based Evaluation

Input: $s, \vec{q}^{\,t-1}, V$: the value table, c: the count table
for *several number of trials* **do**
  $s' \leftarrow s, v \leftarrow 0, w \leftarrow \emptyset$
  for k=t **downto** *1* **do** `// forward trial`
    if $c(s', \vec{q}^{\,k}) \geq numTrials$ **then**
      $w^k \leftarrow \langle s', \vec{q}^{\,k}, V(s', \vec{q}^{\,k})\rangle$ and **break**
    $\vec{a} \leftarrow$ execute a joint action according to $\vec{q}^{\,k}$
    $s'', \vec{o} \leftarrow$ get the system responses
    $r \leftarrow$ get the current reward
    $w^k \leftarrow \langle s', \vec{q}^{\,k}, r\rangle$
    $\vec{q}^{\,k-1} \leftarrow \vec{q}^{\,k}(\vec{o})$
    $s' \leftarrow s''$
  for k=*1* **to** *length(w)* **do** `// backward update`
    $s', \vec{q}, r \leftarrow w^k$
    $n \leftarrow c(s', \vec{q}), v \leftarrow v + r$
    $V(s', \vec{q}) \leftarrow [nV(s', \vec{q}) + v]/(n+1)$
    $c(s', \vec{q}) \leftarrow n + 1$
return $V(s, \vec{q}^{\,t-1})$

If this term is equal to 0, the coefficient of $x(q_i', a_i|q_i, o_i)$ in Table 1 is always 0. In that case there is no need to compute $V(s', \vec{q}')$. Therefore, instead of evaluating all state and joint policy pairs in advance, we postpone the evaluation to the improvement steps. This saves a lot of time and memory by avoiding evaluating the states which are never reachable or used in the LP. Moreover, the trial-based approach is much more efficient than solving Equation 3 and it provides a good approximation of the exact value in our experiments.

The trial-based evaluation process is presented in Algorithm 3. Intuitively, the main disadvantage of trial-based evaluation is that each trial has to travel through the future policies for several times. To evaluate the value of $\vec{q}^{\,t}$, it needs to visit the policies $\vec{q}^{\,t-1} \cdots \vec{q}^{\,1}$. For domains with long horizons, this may take a lot of time especially for the last iteration. In TBDP, we use a hash table $c(s, \vec{q})$ to count the number of trials which have been performed per state and joint policy pair $\langle s, \vec{q}\rangle$. If $\langle s, \vec{q}\rangle$ already has sufficient trials, we can stop there and return the current $V(s, \vec{q})$ without performing more trials. Each time $\langle s, \vec{q}\rangle$ is visited, we increase the count $c(s, \vec{q})$ by 1 and update the running average of $V(s, \vec{q})$ with the value of the current trial. Therefore, we can save substantial time by using the value of earlier trials. This is analogous to the value update in RTDP-based algorithms of single-agent (PO)MDPs. Similarly, we also have the convergence property as below:

Property 1 *If the sub-policies and states are visited infinitely often by trials, the value computed via trial-based evaluation converges to the exact policy value.*

Algorithm 4: Asynchronous Policy Evaluation

repeat in parallel `// run on other processors`
  $t \leftarrow$ the current step of the main process
  foreach *joint policy* $\vec{q}$ *of step* t*-1* **do**
    $\mathcal{S} \leftarrow$ sort states in descending order by $c(s, \vec{q})$
    foreach $s \in \mathcal{S}$ **do**
      while $c(s, \vec{q}) < numTrials$ **do**
        $V(s, \vec{q}) \leftarrow$ evaluate $s, \vec{q}$ by trial
until *the main process is terminated*

Asynchronous Implementation

One major advantage of RTDP-based algorithms is that it does not backup states simultaneously or in any systematically organized fashion. It is suitable for multi-processor systems with communication delays and without a common clock. Multi-processor implementations have obvious utility in speeding up computation and thus have practical significance to solve large problems using computer clusters. Our trial-based evaluation can also be implemented in an asynchronous style as shown in Algorithm 4. Each separate processor can choose any improved joint policy and evaluate it by trials. We sort the states in descending order by the visiting frequency $c(s, \vec{q})$ so the state with high $c(s, \vec{q})$ can be evaluated first. Obviously, the policy value will be closer to the exact value if more trials are performed. Furthermore, the main algorithm can benefit from the computation of other processors by avoiding long trials.

Experiments

We evaluated our algorithm using several standard benchmark problems and a more challenging problem called *cooperative recycling*. For the benchmark problems, we compared our algorithm with the best existing methods. Since none of the existing methods can solve the cooperative recycling problem, we performed in that case an extensive evaluation of our algorithm with different parameters. We report the average value and runtime over 20 runs of the algorithm on each of the problems. Timing results measure CPU times in seconds. TBDP was implemented in Java 1.5 and ran on a Mac OSX machine with 2.8GHz Quad-Core Intel Xeon CPU and 2GB of RAM available for JVM. We utilized the Java concurrency package to manage the asynchronous policy evaluation on this multi-core machine. Linear programs were solved using lp_solve 5.5 with Java wrapper.

Common Benchmark Problems

We tested our algorithm on three standard benchmark problems: Meeting in 3×3 Grid, Cooperative Box Pushing and

Table 2: Results of Benchmark Problems (20 runs)

Horizon	Algorithm	Ave Value	Ave Time
Meeting in a 3×3 Grid $\|S\|$=81, $\|A_i\|$=5, $\|\Omega_i\|$=9			
100	PBIP-IPG	92.12	3084.0s
	TBDP	92.8	427.1s
200	PBIP-IPG	193.39	13875.0s
	TBDP	192.10	1371.9s
Cooperative Box Pushing $\|S\|$=100, $\|A_i\|$=4, $\|\Omega_i\|$=5			
100	PBIP-IPG	598.40	181.0s
	TBDP	611.0	76.3s
1000	PBIP-IPG	5707.59	2147.0s
	TBDP	5857.40	1327.9s
Stochastic Mars Rover $\|S\|$=256, $\|A_i\|$=6, $\|\Omega_i\|$=8			
10	PBIP-IPG	21.18	976.0s
	TBDP	20.1	40.5s
20	PBIP-IPG	37.81	14947.0s
	TBDP	38.30	119.7s

Stochastic Mars Rover. These are the hardest benchmark domains we could find in the DEC-POMDP literature. We compare our algorithm with the best existing method called PBIP-IPG. As reported by Amato *et al.* (2009), PBIP-IPG consistently outperforms other state-of-the-art algorithms such as MBDP, IMBDP, MBDP-OC and PBIP. In this set of experiments, the number of trials used in both sampling and evaluation is 20. We show that with such a small number of trials, TBDP already performs very well compared to PBIP-IPG. The results, presented in Table 2, show that our algorithm produces competitive value (in most cases slightly higher value) in much less runtime in all the tested domains. Although our asynchronous implementation is very preliminary, it is very useful when solving a problem with a long horizon. For example, we observed that TBDP without asynchronous policy evaluation would run out of time for Cooperative Box Pushing with horizon 1000. We have not implemented TBDP on a cluster computer yet, but it should be straightforward to utilize a parallel computing model. In contrast, other algorithms such as PBIP-IPG do not lend themselves naturally to parallel computing. In fact, implementing PBIP-IPG in an asynchronous way is non-trivial.

The Cooperative Recycling Problem

Cooperative Recycling is a much more challenging domain for decision-theoretical planning mainly due to the large number of states. It is more representative of real-world applications involving cooperative robots. The structure of the domain is adapted from a robot navigation problem in the single-agent POMDP literature, which represents a map of a building (Kaelbling *et al.* 1996).

In this domain, two robots navigate in a building and empty recycle bins placed in certain locations. Each robot has 5 actions (stay, turn left, turn right, turn backward and move forward) and 4 observations indicating four types of objects in front of the robot (wall, other robot, recycle bin and free). The actions are stochastic with 0.8 probability of success. The state features are the agents' positions and orientations (N,S,E,W). Each grid cell could be occupied by at most one robot, except for cells with recycle bins. Overall, this problem has 37,824 states. The reward function is designed such that the robots benefit from coordination. There are two types of situations that require coordination: (1) if two robots try to occupy the same cell without a bin, one of them fails and they get a penalty of 5; (2) if two robots cooperatively empty the same bin at the same time, they get the highest reward of 100, or just 10 if only one robot attempts to empty a bin. Each time a bin is emptied, the problem resets with new random robot positions. The robots get a negative reward of -0.1 per time step they spend in the building.

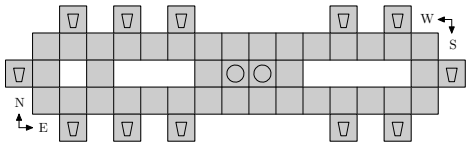

Figure 1: Cooperative Recycling: The domain includes 49 grid cells, 12 recycle bins and 2 start positions (circles).

Figure 2 shows both the value and runtime with different horizons. The number of trials is fixed to 20. As expected, the value and runtime of TBDP grow linearly with the horizon. We also report two special cases: MMDP, a case where the control of the agents is *centralized* with full observability of the global states and INDEP, a case where all agents act *independently* with local observability. Generally, these two approaches provide loose upper and lower bounds on the value of the DEC-POMDP. As shown, TBDP's value grows much like the upper bound and is much better than the baseline approach without coordination. In other words, agents can benefit from coordination using our approach. It is worth pointing out that the MMDP approach is a very loose upper bound; the actual optimal value may be much lower.

We tested TBDP with different numbers of trials for sampling and evaluation. The horizon was set to 60 and the results (averaged over 20 runs) are shown in Figure 3. These experiments show that the value grows as the number of trials increases, becoming stable after about 10 trials. As expected, runtime also grows because the set of reachable states increases with more trials. Generally, the number of trials provides a good trade-off between runtime and value.

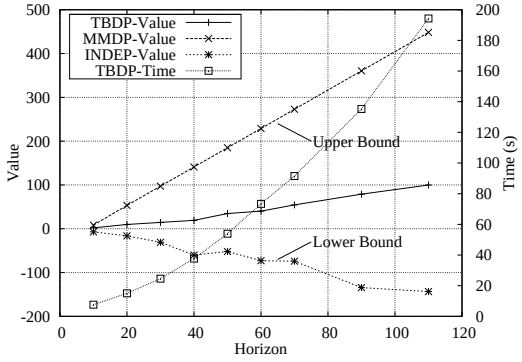

Figure 2: Cooperative Recycling with different horizons.

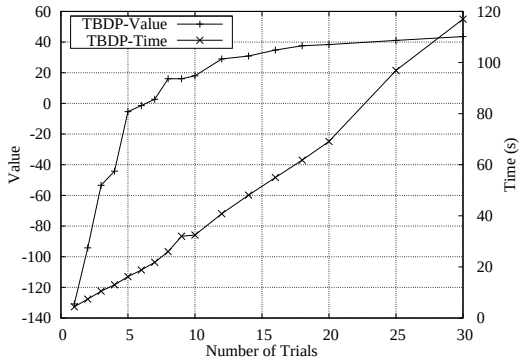

Figure 3: Cooperative Recycling with different trials.

In domains with fewer reachable states, the number of trials required to produce good results should be smaller. This demonstrates that TBDP can focus on likely states and generate good coordination policies.

Conclusion

We present a novel algorithm for multi-agent planning that uses trial-based methods to generate belief states, improve and evaluate policies. The algorithm, TBDP, offers a better way to explore the domain and exploit its reachability structure, which is particularly efficient in domains with limited state reachability. The stochastic policies introduced in TBDP use bounded memory much like MBDP, but they can be calculated efficiently without time-consuming *backup* operations. TBDP employs a lazy computation strategy, evaluating policies only when they are needed. Similar to RTDP, TBDP can be easily implemented asynchronously, and take advantage of multi-processor or multi-core machines.

The experimental results show that TBDP can compute comparative policies with order of magnitude improvement in runtime on several standard benchmark problems. More importantly, it can solve problems with much larger state spaces than previously possible, thereby improving the scalability of multi-agent planning. In the future, we plan to extend TBDP to learn the policies from direct interaction with the environment, without a model. The algorithms and representations used in this work open up several research directions for planning and learning in multi-agent systems.

Acknowledgments

This work was supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0181, the National Science Foundation under Grant No. IIS-0812149, the Natural Science Foundations of China under Grant No. 60745002, and the National Hi-Tech Project of China under Grant No. 2008AA01Z150.

References

Amato, C.; Dibangoye, J. S.; and Zilberstein, S. 2009. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proc. of the 19th Int'l Conf. on Automated Planning and Scheduling*, 2–9.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1-2):81–138.

Bernstein, D. S.; Zilberstein, S.; and Immerman, N. 2000. The complexity of decentralized control of Markov decision processes. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, 32–37.

Bernstein, D. S.; Hansen, E. A.; and Zilberstein, S. 2005. Bounded policy iteration for decentralized POMDPs. In *Proc. of the 19th Int'l Joint Conf. on Artificial Intelligence*, 1287–1292.

Bonet, B., and Geffner, H. 2009. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *Proc. of the 21st Int'l Joint Conf. on Artificial Intelligence*, 1641–1646.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *Proc. of the 6th Conf. on Theoretical Aspects of Rationality and Knowledge*, 195–210.

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

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

Geffner, H., and Bonet, B. 1998. Solving large POMDPs using real time dynamic programming. In *AAAI Fall Symposium on POMDPs*.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proc. of the 19th National Conf. on Artificial Intelligence*, 709–715.

Kaelbling, L. P.; Cassandra, A.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 963–972.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proc. of the 23rd Int'l Conf. on Machine Learning*, 569–576.

Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D. V.; and Marsella, S. 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. of the 18th Int'l Joint Conf. on Artificial Intelligence*, 705–711.

Peshkin, L.; Kim, K.-E.; Meuleau, N.; and Kaelbling, L. P. 2000. Learning to cooperate via policy search. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, 489–496.

Pynadath, D. V., and Tambe, M. 2002. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research* 16:389–423.

Sanner, S.; Goetschalckx, R.; Driessens, K.; and Shani, G. 2009. Bayesian real-time dynamic programming. In *Proc. of the 21st Int'l Joint Conf. on Artificial Intelligence*, 1784–1789.

Seuken, S., and Zilberstein, S. 2007a. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proc. of the 23rd Conf. on Uncertainty in Artificial Intelligence*, 344–351.

Seuken, S., and Zilberstein, S. 2007b. Memory-bounded dynamic programming for DEC-POMDPs. In *Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence*, 2009–2015.

Smith, T., and Simmons, R. G. 2006. Focused real-time dynamic programming for mdps: Squeezing more out of a heuristic. In *Proc. of the 21st National Conf. on Artificial Intelligence*, 1227–1232.

Szer, D., and Charpillet, F. 2006. Point-based dynamic programming for DEC-POMDPs. In *Proc. of the 21st National Conf. on Artificial Intelligence*, volume 2, 1233–1238.