Point-Based Bounded Policy Iteration for Decentralized POMDPs

Youngwook Kim[1] and Kee-Eung Kim[2]

[1] Search Solutions, Seongnam-si, Korea
youngwook.kim@nhn.com,
[2] Korea Advanced Institute of Science and Technology, Daejeon, Korea
kekim@cs.kaist.ac.kr

Abstract. We present a memory-bounded approximate algorithm for solving infinite-horizon decentralized partially observable Markov decision processes (DEC-POMDPs). In particular, we improve upon the bounded policy iteration (BPI) approach, which searches for a locally optimal stochastic finite state controller, by accompanying reachability analysis on controller nodes. As a result, the algorithm has different optimization criteria for the reachable and the unreachable nodes, and it is more effective in the search for an optimal policy. Through experiments on benchmark problems, we show that our algorithm is competitive to the recent nonlinear optimization approach, both in the solution time and the policy quality.

1 Introduction

The decentralized POMDP (DEC-POMDP) is a popular framework for modeling decision making problems where two or more agents have to cooperate in order to maximize a common payoff, and to act based on imperfect state information. While the DEC-POMDP can be applied to many domains such as network routing and multi-robot coordination, it is known to be intractable for computing an optimal policy [1].

In this paper, we are interested in solving infinite-horizon DEC-POMDPs by searching in the space of fixed-size finite state controllers (FSCs). Specifically, we represent the individual policy for each agent as a stochastic FSC in which the nodes correspond to action selection strategies and the transitions correspond to observation strategies. There have been proposed a number of methods for finding FSC policies, but most relevant to our work are the bounded policy iteration for DEC-POMDPs (DEC-BPI) and the nonlinear optimization approach (NLO).

DEC-BPI [2] is a generalization of the bounded policy iteration algorithm for POMDPs [3] to the DEC-POMDP. It is a greedy local search algorithm that iteratively improves the individual policies of the agents. The policy improvement is carried out by randomly choosing a node from an individual FSC policy, and updating its parameters by solving a linear program. Although DEC-BPI is a

Variables: ϵ, $x(a_i)$, $x(a_i, z_i, q_i')$
Objective: Maximize ϵ
Improvement constraints:

$$\forall q_{-i}, s \quad V(\vec{q}, s) + \epsilon \leq \sum_{\vec{a}} P(a_{-i}|q_{-i}) \Big[x(a_i) R(s, \vec{a}) + \gamma \sum_{s', \vec{z}, \vec{q}'} x(a_i, z_i, q_i') \\ P(q_{-i}'|q_{-i}, a_{-i}, z_{-i}) T(s, \vec{a}, s') O(s', \vec{a}, \vec{z}) V(\vec{q}', s') \Big]$$

Probability constraints:

$$\sum_{a_i} x(a_i) = 1, \qquad \forall a_i, z_i \quad \sum_{q_i'} x(a_i, z_i, q_i') = x(a_i)$$

$$\forall a_i \quad x(a_i) \geq 0, \qquad \forall a_i, z_i, q_i' \quad x(a_i, z_i, q_i') \geq 0$$

Table 1. The linear program for DEC-BPI. The variable $x(a_i)$ represents $\psi_i(q_i, a_i)$, and $x(a_i, z_i, q_i')$ represents $\eta_i(q_i, a_i, z_i, q_i')$. $P(a_{-i}|q_{-i})$ denotes $\prod_{k \neq i} \psi_k(q_k, a_k)$, and $P(q_{-i}'|q_{-i}, a_{-i}, z_{-i})$ denotes $\prod_{k \neq i} \eta_k(q_k, a_k, z_k, q_k')$.

scalable algorithm, it is prone to converging to a bad local optimum when we enforce a fixed-size controller.

NLO [4] takes a more direct approach to finding an optimal policy by formulating the overall problem as a nonlinear program (NLP) and let an NLP solver take care of finding the solution. The advantage of this approach is that one can harness the efficient search techniques available in the NLP solvers, therefore obtaining high quality policies compared to those from DEC-BPI. However, most of the NLP solvers suffer from scalability, as the formulated problem is nonconvex.

We propose an improved version of DEC-BPI that addresses some of the limitations that prevent the algorithm from finding an FSC policy with a high quality. Our insight for the improvement is based on the observation that we need different optimization criteria depending on whether a controller node in FSC is reachable or not. We show the effectiveness of the proposed algorithm via experiments on standard benchmark problems.

2 Background

A decentralized partially observable Markov decision process (DEC-POMDP) is a multi-agent extension to the POMDP framework. More Formally, a DEC-POMDP is defined as tuple $\langle I, S, b_0, \{A_i\}, \{Z_i\}, T, O, R \rangle$ where

- I is a finite set of agents
- S is a finite set of states shared by all agents
- b_0 is the initial state distribution, where $b_0(s)$ denotes the probability that the system starts in state s
- A_i is a finite set of actions available to agent i; the set of *joint actions* is denoted as $\vec{A} = \prod_{i \in I} A_i$
- Z_i is a finite set of observations available to agent i; the set of *joint observations* is denoted as $\vec{Z} = \prod_{i \in I} Z_i$

– T is a transition function where $T(s, \vec{a}, s')$ denotes the probability $P(s'|s, \vec{a})$ of changing to state s' from state s by executing joint action $\vec{a}$
– O is an observation function where $O(s, \vec{a}, \vec{z})$ denotes the probability $P(\vec{z}|\vec{a}, s)$ of making joint observation $\vec{z}$ when taking joint action $\vec{a}$ and arriving in state s.
– R is a reward function where $R(s, \vec{a})$ denotes the shared reward received by all agents when taking joint action $\vec{a}$ in state s.

Each agent decides which action to take over a series of discrete time steps called a *horizon*. Since the state is not directly observable and the observations are local to each agent, the agent chooses actions based on its own local histories. This mapping from local observation histories to actions comprises a *local policy*, and the set of every agent's local history comprises a *joint policy*. The goal of DEC-POMDP planning algorithms is to find an optimal joint policy that maximizes expected cumulative reward over the horizon. In the case of infinite-horizon DEC-POMDPs, we use a discount factor $0 \leq \gamma < 1$ in order to guarantee finite cumulative rewards.

A popular representation for policies in infinite-horizon problems is to use *stochastic finite state controllers* (FSCs). The local policy for agent i is represented as a stochastic FSC $\pi_i = \langle Q_i, \psi_i, \eta_i \rangle$, where

– Q_i is the finite set of controller nodes,
– ψ_i is the action selection strategy for each node, where $\psi_i(q, a)$ denotes the probability $P(a|q)$ of choosing action a in node q,
– η_i is the observation strategy for each node, where $\eta_i(q, a, z, q')$ denotes the probability $P(q'|q, a, z)$ of changing to node q' from node q when executing action a and making observing z.

The set of π_i for each agent i comprises a joint policy $\vec{\pi}$, and the set of nodes from each agent's controller comprises a joint node. The cumulative reward (also called *value*) for state s and joint node $\vec{q}$ is defined as

$$V(\vec{q}, s) = \sum_{\vec{a}} \prod_i \psi_i(q_i, a_i) \big[R(s, \vec{a}) + \gamma \sum_{s'} T(s, \vec{a}, s') \cdot \sum_{\vec{z}} O(s', \vec{a}, \vec{z}) \sum_{\vec{q}'} \prod_i \eta_i(q_i, a_i, z_i, q_i') V(\vec{q}', s') \big] \quad (1)$$

Above system of linear equations for obtaining value function of a joint policy is called the Bellman equation. The starting joint node $\vec{q}_0$ is selected by

$$\vec{q}_0 = \text{argmax}_{\vec{q}} \sum_s b_0(s) V(\vec{q}, s)$$

which states that the controller is assumed to start in the joint node that maximizes its value from the initial state distribution.

2.1 Bounded Policy Iteration for DEC-POMDPs

Bernstein *et al.* [2]'s bounded policy iteration for DEC-POMDPs (DEC-BPI) is an extension of the bounded policy iteration algorithms for POMDPs [3] to the

Variables: $v(\vec{q}, s)$ and for each agent i, $x(q_i, a_i)$, $x(q_i, a_i, z_i, q_i')$
Objective: Maximize $\sum_s b_0(s) v(\vec{q}_0, s)$
Bellman constraints:

$$\forall \vec{q}, s \quad v(\vec{q}, s) = \sum_{\vec{a}} \left[\prod_i x(q_i, a_i)\right] \Big[R(s, \vec{a}) + \gamma \sum_{s', \vec{z}, \vec{q}'} T(s, \vec{a}, s') O(s', \vec{a}, \vec{z}) \left[\prod_i x(q_i, a_i, z_i, q_i')\right] v(\vec{q}', s')\Big]$$

Probability constraints:

$$\forall q_i \quad \sum_{a_i} x(q_i, a_i) = 1, \qquad \forall q_i, z_i, a_i \quad \sum_{q_i'} x(q_i, a_i, z_i, q_i') = 1$$

$$\forall q_i, a_i \quad x(q_i, a_i) \geq 0, \qquad \forall q_i, z_i, a_i, q_i' \quad x(q_i, a_i, z_i, q_i') \geq 0$$

Table 2. The nonlinear program for finding the optimal controller. The variable $x(q_i, a_i)$ represents $\psi_i(q_i, a_i)$, and $x(q_i, a_i, z_i, q_i')$ represents $\eta_i(q_i, a_i, z_i, q_i')$, and $v(\vec{q}, s)$ represents the value function $V(\vec{q}, s)$. Each controller have a designated initial node, forming the initial joint node $\vec{q}_0$.

multi-agent case. It is a greedy local search algorithm that iteratively improves a joint stochastic FSC with a fixed number of nodes by alternating between policy evaluation and improvement. In the policy evaluation step, DEC-BPI computes the value function of the current joint controller by solving the Bellman equation in Equation (1). In the policy improvement step, DEC-BPI randomly selects one of the nodes of an agent, and solves the linear program shown in Table 1 to obtain an improved controller.

DEC-BPI guarantees monotonic improvement in the value for all the joint nodes and the states. This however also results in only a locally optimal solution, which often degrades in quality as we have more agents or larger controllers. One way to alleviate the local convergence issue is to compute the relative occupancy distribution

$$o(\vec{q}', s') = b_0(s')\delta(\vec{q}_0, \vec{q}') + \gamma \sum_{\vec{q}, s, \vec{a}, \vec{z}} \Big[o(\vec{q}, s) T(s, a, s') \cdot \prod_i \psi_i(q_i, a_i) O(s', \vec{a}, \vec{z}) \prod_i \eta_i(q_i, a_i, z_i, q_i')\Big] \quad (2)$$

and bias the improvement by using $o(\vec{q}', s')$ as the weight for each joint node and state. The main idea is to analyze the reachability of joint node and state pairs, and concentrate the improvement on those that are more frequently visited than others.

2.2 Nonlinear Optimization Approach

Amato *et al.* [4]'s nonlinear optimization (NLO) takes a more direct approach to obtaining an optimal controller. The problem is formulated as a nonlinear program (NLP), shown in Table 2, and a state-of-the-art NLP solver is used to find solutions. Since the problem is nonconvex, most of the NLP solvers yield only a locally optimal solution, as in the case with DEC-BPI. However, since we are exploiting advanced search heuristics readily available in the NLP solver, the solution is often much better in quality than that from DEC-BPI.

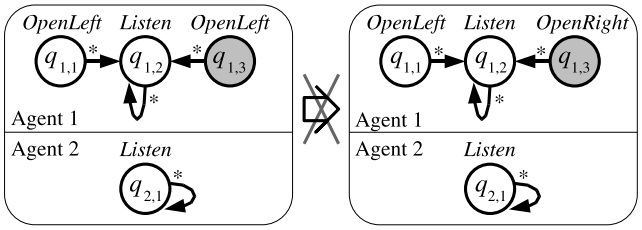

Fig. 1. DEC-BPI fails to discover a good set of policies.

There are two issues regarding this approach. First, although advanced NLP solvers are quite fast, they are not as scalable as LP solvers. Since solving a nonconvex NLP is a hard problem, we can naturally expect that NLO using a general NLP solver takes much more time than DEC-BPI. Second, we would like to pinpoint the exact cases where DEC-BPI performs a lot worse than NLO, rather than just being satisfied with a "black-box" approach that works well in practice. However, the sophisticated search heuristics implemented in NLP solvers hinder us from such an analysis.

3 Point-Based DEC-BPI

Before we present our algorithm, let us take a closer look at the policy improvement in DEC-BPI. The linear program in Table 1 tries to find better parameters for q_i, assuming that we use the controller with the new parameters for the first time step, and then the one with the old parameters from the second step on.

An important reminder is that the FSC actually represents a set of policies for each agent, of which each policy is determined by selecting one of the nodes as the starting node. We want the intermediate FSCs during the iterations of DEC-BPI to represent the set of policies that perform well with respect to various reachable state distributions starting from b_0, but DEC-BPI does not necessarily show this behavior. Figure 1 shows such a case in the decentralized tiger problem [5], where the current policy for agent 1 consists of nodes $\{q_{1,1}, q_{1,2}, q_{1,3}\}$ and that for agent 2 consists of $\{q_{2,1}\}$, and DEC-BPI chooses $q_{1,3}$ for the improvement. Note that $q_{1,1}$ and $q_{1,3}$ are identical policies that prescribe executing *OpenLeft* at the first step and then executing *Listen* from the second step on. It would be better if one of the two nodes represent a different but *useful* policy, rather than wasting an FSC node. For example, executing *OpenRight* in $q_{1,3}$ would be useful when the the tiger is behind the left door, but DEC-BPI would not yield such a policy since the monotonic improvement condition requires improving the value for *all* state distributions, including the case when the tiger is more likely to be on the right.

The main idea behind our point-based DEC-BPI is to have different optimization criteria depending on whether or not a controller node is reachable from the set of useful nodes. Formal definitions will follow shortly, but roughly stated, a joint node is *useful* if and only if it yields a maximum value at some

$B \leftarrow$ SAMPLEBELIEFS()
repeat
  $V^{\vec{\pi}} \leftarrow$ EVALUATE($\vec{\pi}$)
  $C \leftarrow$ REACHABLENODESTATES($B, \vec{\pi}, V^{\vec{\pi}}$)
  $(\epsilon, \vec{\pi}) \leftarrow$ IMPROVEPOLICY($\vec{\pi}, V^{\vec{\pi}}, C$)
until no improvement in any node of any agent

Table 3. Point-based DEC-BPI

reachable *multiagent belief* from b_0 by following the optimal joint policy, and *reachable* if and only if the probability of visiting the joint node from any useful node is non-zero under the current joint controller. The overall algorithm is shown in Table 3, and in the remainder of this section, we explain each step of the algorithm.

3.1 Sampling Beliefs

Since it is intractable to find the exhaustive set of reachable multiagent beliefs under the optimal policy, we approximate the set by sampling from a random policy, similar to [6]. Formally, given a T-step joint tree policy and a joint history $\vec{h}_T = \langle \vec{a}_1, \vec{z}_1, \ldots, \vec{a}_T, \vec{z}_T \rangle$ of actions and observations from time step 1 to T, the associated (unnormalized) state distribution $b(\vec{h}_T, \cdot)$ is recursively computed by

$$b(\vec{h}_T, s') = O(s', \vec{a}_T, \vec{z}_T) \sum_s T(s, \vec{a}_T, s') b(\vec{h}_{T-1}, s)$$

where $\vec{h}_{T-1}$ is the sub-history from time step 1 to $T-1$, and $b(\vec{h}_0, s) = b_0(s)$. We also denote by $h_{i,T}$ the local history of $\vec{h}_T$ specific to agent i, so that we can equivalently write $\vec{h}_T = \{h_{i,T} | i \in I\}$. Note that $\sum_s b(\vec{h}_T, s)$ equals the probability of the joint observation history $\langle \vec{z}_1, \ldots, \vec{z}_T \rangle$ under the given T-step tree policy.

The random policy in our case is chosen to be an arbitrary T-step tree policy, which yields $\prod_i |Z_i|^T$ joint histories, one for each leaf of the tree. Denoting this set of joint histories by $\vec{H}_T$, each random instantiation of the T-step policy (by randomly selecting actions) generates exactly one belief $b : \vec{H}_T \times S \rightarrow [0, 1]$, a probability distribution on the joint histories and the states. Denoting the set of local histories for agent i by $H_{i,T}$, we can equivalently represent the belief by $b : \prod_i H_{i,T} \times S \rightarrow [0, 1]$.

Example: If the local policies are 2-step policies with always executing *Listen*, there are 4 local histories for each agent, and the belief is represented as the following table:

$C \leftarrow \{\}$
for each belief $b \in B$ **do**
  $f^b \leftarrow \text{argmax}_{\{f_i: H_i \rightarrow Q_i\}} \sum_{\vec{h},s} b(\vec{h}, s) V^{\vec{\pi}}(f(\vec{h}), s)$
  for each joint history $\vec{h} \in \vec{H}$ and state $s \in S$ **do**
    if $b(\vec{h}, s) > 0$ **then**
      $C \leftarrow C \cup \{\langle f^b(\vec{h}), s\rangle\}$
    end if
  end for
end for
repeat
  for all $\langle \vec{q}', s'\rangle$ s.t. $\langle \vec{q}, s\rangle \in C$ and $T^{\vec{\pi}}(\vec{q}, s, \vec{q}', s') > 0$ **do**
    $C \leftarrow C \cup \{\langle \vec{q}', s'\rangle\}$
  end for
until no more node-state pair to add

Table 4. Procedure ReachableNodeStates

	HL,HL	HL,HR	HR,HL	HR,HR
HL,HL	0.2610	0.0461	0.0461	0.0081
	0.0003	0.0014	0.0014	0.0081
HL,HR	0.0461	0.0081	0.0081	0.0014
	0.0014	0.0081	0.0081	0.0461
HR,HL	0.0461	0.0081	0.0081	0.0014
	0.0014	0.0081	0.0081	0.0461
HR,HR	0.0081	0.0014	0.0014	0.0003
	0.0081	0.0461	0.0461	0.2610

where the row represents the observation history of agent 1, and the column represents the observation history of agent 2. Each cell has $b(\vec{h}$, *TigerLeft*) in the top, and $b(\vec{h}$, *TigerRight*) in the bottom.

3.2 Reachability of Nodes and States

Once we evaluate the value of current policy by Equation (1), we identify the set of useful joint nodes. Formally, a joint node $\vec{q}$ of joint controller $\vec{\pi}$ is *useful* for belief $b \in B$ if it maximizes the value at the belief. In particular, since each belief is associated with a set of joint histories, and hence associated with a set of local histories for each agent, we search for an assignment f_i of local controller node to each local history for every agent so that

$$V^{\vec{\pi}}(b) = \max_{\{f_i: H_i \rightarrow Q_i\}} \sum_{\vec{h},s} b(\vec{h}, s) V^{\vec{\pi}}(f(\vec{h}), s),$$

where $f = \{f_i\}$. The *range* of f_i's becomes the set of useful joint nodes for the belief b. Intuitively, the useful nodes are the candidate initial nodes if the system starts at the state distributions dictated by the belief b.

Variables: ϵ, $x(a_i)$, $x(a_i, z_i, q_i')$
Objective: Maximize ϵ
Improvement constraints: $\forall \langle q_i, q_{-i}, s \rangle \in C$,

$$V(\vec{q}, s) + \epsilon \leq \sum_{\vec{a}} P(a_{-i}|q_{-i}) \big[x(a_i) R(s, \vec{a}) + \gamma \sum_{s', \vec{z}, \vec{q}' : \langle \vec{q}', s' \rangle \in C} x(a_i, z_i, q_i') \\ P(q_{-i}'|q_{-i}, a_{-i}, z_{-i}) T(s, \vec{a}, s') O(s', \vec{a}, \vec{z}) V(\vec{q}', s') \big]$$

Unreachability maintenance constraints: $\forall \langle q_i, q_{-i}, s \rangle \in C$ and $\forall \langle \vec{q}', s' \rangle \notin C$

$$\sum_{\vec{a}, \vec{z}} P(a_{-i}|q_{-i}) x(a_i, z_i, q_i') P(q_{-i}'|q_{-i}, a_{-i}, z_{-i}) T(s, \vec{a}, s') O(s', \vec{a}, \vec{z}) = 0$$

Probability constraints as in Table 1

Table 5. The linear program in point-based DEC-BPI for improving reachable node q_i.

Variables: $x(a_i)$, $x(a_i, z_i, q_i')$
Objective: Maximize

$$\sum_{h_{-i}, s} b(h_i, h_{-i}, s) \sum_{\vec{a}} P(a_{-i}|f^b(h_{-i})) \Big[x(a_i) R(s, \vec{a}) + \gamma \sum_{s', \vec{z}, \vec{q}' : \langle \vec{q}', s' \rangle \in C} x(a_i, z_i, q_i') \\ P(q_{-i}'|f^b(h_{-i}), a_{-i}, z_{-i}) P(s', \vec{z}|s, \vec{a}) V(\vec{q}', s') \Big]$$

Unreachability maintenance constraints: $\forall h_{-i}, s$ with $b(h_i, h_{-i}, s) > 0$ and $\forall \langle \vec{q}', s' \rangle \notin C$

$$\sum_{\vec{a}, \vec{z}} P(a_{-i}|f^b(h_{-i})) x(a_i, z_i, q_i') P(q_{-i}'|f^b(h_{-i}), a_{-i}, z_{-i}) T(s, \vec{a}, s') O(s', \vec{a}, \vec{z}) = 0$$

Probability constraints as in Table 1

Table 6. The linear program in point-based DEC-BPI for improving unreachable node q_i with respect to local history h_i in belief b. $P(s', \vec{z}|s, \vec{a})$ is a shorthand notation for $T(s, \vec{a}, s') O(s', \vec{a}, \vec{z})$.

Once we have identified the set of useful joint nodes, we examine the reachability of all the joint nodes from the useful joint nodes. Table 4 shows the overall pseudo-code for finding the set of reachable joint nodes given the set B of beliefs. Note that we attach state information to each reachable node, so that our reachability analysis is performed on the pairs of nodes and states.

Example: Given the belief b shown in the previous example, the policy in Figure 1 will have $\langle q_{1,1}, q_{2,1} \rangle$ and $\langle q_{1,2}, q_{2,1} \rangle$ as useful joint nodes. Note that $\langle q_{1,3}, q_{2,1} \rangle$ is not selected, since it has the same value as $\langle q_{1,1}, q_{2,1} \rangle$ for all the unnormalized state distributions in the table. We can enforce f^b to assign the same joint node for the same value by imposing a lexical ordering on the nodes. The procedure will finally return

$$C = \{ \langle q_{1,1}, q_{2,1}, \mathit{TigerLeft} \rangle, \langle q_{1,1}, q_{2,1}, \mathit{TigerRight} \rangle, \\ \langle q_{1,2}, q_{2,1}, \mathit{TigerLeft} \rangle, \langle q_{1,2}, q_{2,1}, \mathit{TigerRight} \rangle \}.$$

3.3 Modified Policy Improvement

As in DEC-BPI, our algorithm randomly selects one of the node of an individual controller and uses an LP solver to find new parameter values that improve the controller. However, the LP is formulated differently depending on the reachability of the selected node. The joint node $\vec{q}$ is defined to be reachable if there exists state s such that $\langle \vec{q}, s \rangle \in C$, and the node q_i is defined to be reachable if there exists q_{-i} such that $\vec{q} = \langle q_i, q_{-i} \rangle$ is reachable.

If the node q_i selected for improvement is reachable, we solve the LP shown in Table 5. There are a number of remarks noteworthy to mention: first, the monotonic improvement is only concerned with reachable joint nodes and states. We can even have a worse value if the joint node and state is unreachable. Second, for the unreachable joint nodes and states at the next time step, they are forced to remain unreachable after the improvement. This constraint is related to the first remark, since unreachable joint nodes may degrade in value and we certainly do not want to make transition to them.

On the other hand, if the selected node q_i is unreachable, we solve the LP shown in Table 6. The LP essentially tries to make the selected node useful for some belief. In order to do so, we select a belief b from B, and change the assignment for one of the agent i's local history to q_i. Note that the assignment of nodes to local histories in b is already computed as f^b in Table 4, and here, we change the assignment for only one local history h_i to q_i while other assignments remaining the same as f^b. We also prevent the node from making transition to unreachable nodes since otherwise the node may not become useful even though it maximizes the value for the belief. The LP is solved for each local history of each belief until an improvement over the value of the old assignment is found.

Example: For the belief in our running example, f^b makes assignment for agent 1 as follows: HL,HL $\rightarrow q_{1,2}$, HL,HR $\rightarrow q_{1,2}$, HR,HL $\rightarrow q_{1,2}$, and HR,HR $\rightarrow q_{1,1}$. Assume that our algorithm selects $q_{1,3}$, which is unreachable, for improvement. Further assume that the local history HL,HL is selected as h_i. Table 6 will change $q_{1,3}$ to executing *OpenRight* and making transition to $q_{1,2}$.

3.4 Theoretical Properties

The point-based DEC-BPI guarantees monotonic improvement for all the sampled beliefs in B.

In order to prove this, we first show that

$$\forall \langle \vec{q}, s \rangle \in C, V^{\vec{\pi}'}(\vec{q}, s) \geq V^{\vec{\pi}}(\vec{q}, s)$$

where $\vec{\pi}'$ is the policy after the improvement. In the case of Table 5, this naturally follows from the observation that the transitions of $\vec{\pi}$ happens only within C, and the unreachability maintenance constraints prevent transition outside of C. Hence, it can be viewed as the original DEC-BPI policy improvement of a sub-controller for a sub-problem derived from C. In the case of Table 6, where the selected node q_i is not reachable from C, changing the parameters of q_i would not affect the value of any $\langle \vec{q}, s \rangle \in C$.

Finally, observing that for any $b \in B$,

$$V^{\vec{\pi}}(b) = \sum_{\vec{h},s} b(\vec{h}, s) V^{\vec{\pi}}(f^b(\vec{h}), s)$$

and for any joint history $\vec{h}$ and state s, either $\langle f^b(\vec{h}), s\rangle \in C$ or $b(\vec{h}, s) = 0$, we have

$$\begin{aligned} V^{\vec{\pi}'}(b) &\geq \sum_{\vec{h},s} b(\vec{h}, s) V^{\vec{\pi}'}(f^b(\vec{h}), s) \\ &\geq \sum_{\vec{h},s} b(\vec{h}, s) V^{\vec{\pi}}(f^b(\vec{h}), s) = V^{\vec{\pi}}(b) \end{aligned}$$

thus concluding the proof.

4 Experiments

We implemented all three algorithms discussed in this paper: DEC-BPI, NLO, and point-based DEC-BPI. All the algorithms were implemented in Java, and the LP and the NLP solvers used in our implementation were GPLK and IPOPT, respectively. We used three DEC-POMDP problems for the experiments: decentralized tiger [5], grid-small [2], and box-pushing [7]. We refer the readers to the above references for the exact specification of the problems.

Our implementation of DEC-BPI was actually the biased version of DEC-BPI, which takes into account the reachability of joint nodes and states by computing the occupancy distribution (Equation (2)). This is to let DEC-BPI take advantage of the modification known to be effective in improving performance.

We also accordingly modified the LPs for point-based DEC-BPI (Table 5 and Table 6) using the occupancy distribution in order to favor biased policy improvement, assuming a uniform probability of starting in one of the useful nodes. The beliefs were collected from randomly instantiated 1-step and 2-step tree policies, as well as the initial state distribution b_0. The number of beliefs for each problem is: 11 for decentralized tiger, 15 for grid-small, and 20 for box-pushing.

We ran each algorithm on each problem, starting from randomly instantiated stochastic controllers with varying number of nodes. We executed 20 runs for each controller size, measuring the value and the wall clock time of each run. We report the runs with the highest values rather than the averages, since a common approach to dealing with locally optimal solutions in local search algorithms is to re-start multiple times with different seeds and take the best local optimum as the final solution. The experiments were performed on a Linux platform with Intel 2.66GHz CPU.

Figure 2 through 4 show the value and time results on the problems. Point-based DEC-BPI was able to yield controllers that attain values much higher than those from DEC-BPI, while taking a fraction of time compared to NLO. Note that in the case of decentralized tiger and grid-small, the controllers were able to achieve almost the same level of performance as those from NLO, although they required additional number of nodes. In the case of box-pushing, we believe

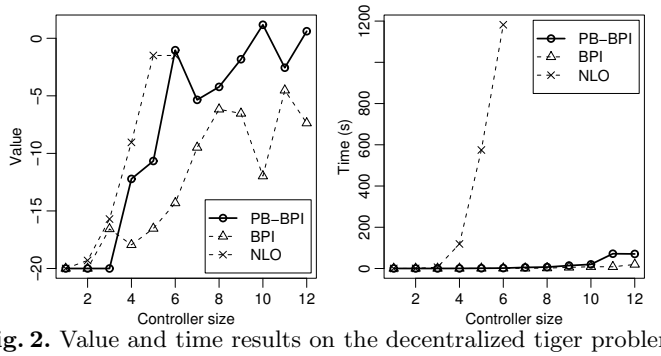

Fig. 2. Value and time results on the decentralized tiger problem.

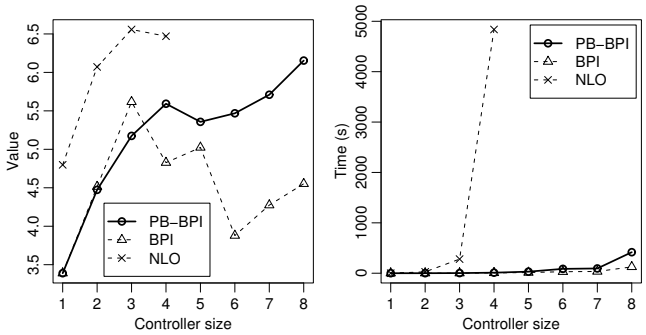

Fig. 3. Value and time results on the grid-small problem.

that we need more nodes than reported here in order to have the performance comparable to NLO, since there are many more reachable beliefs than other problems, and we may need a node for each belief in order to be effective.

5 Conclusion and Future Work

In this paper, we have presented a point-based approach to finding fixed-size stochastic controller for DEC-POMDPs. Specifically, we proposed a novel improvement technique for DEC-BPI, which addresses some of the limitations that prevent from finding a controller with a high value. First, we relaxed the monotonic improvement condition for all possible beliefs to the subset of reachable beliefs from the initial state distribution. This technique addresses bad local optima often encountered by DEC-BPI. Second, we analyze the reachability of each node in the intermediate controllers, and impose different improvement conditions depending on whether the node is reachable or unreachable. This technique makes unreachable nodes, which do not contribute to maximizing value for any belief, be useful for some reachable belief. As a result, the set of policies as dictated by the controller will spread out through the reachable beliefs, hence making the search more effective than DEC-BPI.

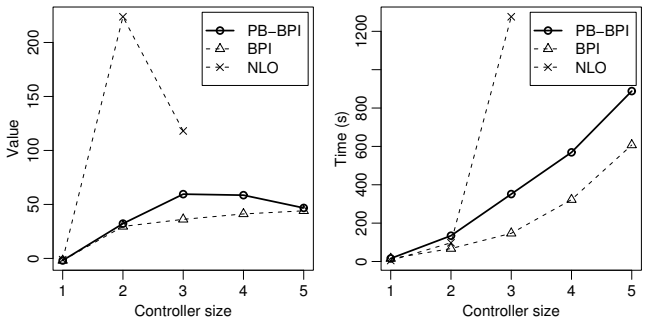

Fig. 4. Value and time results on the box-pushing problem.

Although we have presented the algorithm in the context of DEC-POMDPs, the technique could also be used without difficulty in improving BPI for POMDPs. We are also investigating making the algorithm more scalable, such as incorporating the technique for dealing with sparse controllers [8].

References

1. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of Markov decision processes. Mathematics of Operations Research **27**(4) (2002) 1192.
2. Bernstein, D.S., Hansen, E.A., Zilberstein, S.: Bounded policy iteration for decentralized POMDPs. In: Proceedings of IJCAI. (2005) 1205.
3. Poupart, P., Boutilier, C.: Bounded finite state controllers. In: Proceedings of NIPS. (2003) 1209.
4. Amato, C., Bernstein, D.S., Zilberstein, S.: Optimizing memory-bounded controllers for decentralized POMDPs. In: Proceedings of UAI. (2007) 1241.
5. Nair, R., Tambe, M., Yokoo, M., Pynadath, D., Marsella, S.: Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: Proceedings of IJCAI. (2003) 1206.
6. Szer, D., Charpillet, F.: Point-based dynamic programming for DEC-POMDPs. In: Proceedings of AAAI. (2006) 1207.
7. Seuken, S., Zilberstein, S.: Memory-bounded dynamic programming for DEC-POMDPs. In: Proceedings of IJCAI. (2007) 1208.
8. Hansen, E.A.: Sparse stochastic finite-state controllers for POMDPs. In: Proceedings of UAI. (2008) 1312.