Constraint-Based Dynamic Programming for Decentralized POMDPs with Structured Interactions

Akshat Kumar and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA, 01003
{akshat, shlomo}@cs.umass.edu

ABSTRACT

Decentralized partially observable MDPs (DEC-POMDPs) provide a rich framework for modeling decision making by a team of agents. Despite rapid progress in this area, the limited scalability of solution techniques has restricted the applicability of the model. To overcome this computational barrier, research has focused on restricted classes of DEC-POMDPs, which are easier to solve yet rich enough to capture many practical problems. We present CBDP, an efficient and scalable point-based dynamic programming algorithm for one such model called ND-POMDP (Network Distributed POMDP). Specifically, CBDP provides magnitudes of speedup in the policy computation and generates better quality solution for all test instances. It has linear complexity in the number of agents and horizon length. Furthermore, the complexity per horizon for the examined class of problems is exponential only in a small parameter that depends upon the interaction among the agents, achieving significant scalability for large, loosely coupled multi-agent systems. The efficiency of CBDP lies in exploiting the structure of interactions using constraint networks. These results extend significantly the effectiveness of decision-theoretic planning in multi-agent settings.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: [Dynamic programming, Multiagent systems, Intelligent agents]

Keywords

Multiagent planning, DEC-POMDPs

1. INTRODUCTION

DEC-POMDPs have emerged in recent years as an important framework for modeling team decision problems under uncertainty. They can effectively capture situations when the environment state dynamics and the reward system depend upon the joint action of all agents, but agents must act based on different partial knowledge of the overall situation. Many problems such as multi-robot coordination [14],

broadcast channel protocols [2], and target tracking in sensor networks [8] can be formulated as DEC-POMDPs.

Solving DEC-POMDPs optimally is NEXP-hard [2], with optimal algorithms able to provide solution for very small problems involving two agents and small horizons [6, 17]. Approximate algorithms can scale up the horizon, but they remain limited to a few agents [14, 3]. An emerging approach to improve scalability has been to consider restricted forms of interaction that arise frequently in practice [1, 11]. In particular, ND-POMDP [11] is one such general model which is inspired by a realistic sensor network coordination problem [8]. The key assumption in ND-POMDP is that of transition and observation independence and the *locality of interaction*, which help construct efficient algorithms that exploit the independence among the agents.

A rich portfolio of algorithms has been developed for solving ND-POMDPs, featuring locally optimal policy search [11], approximation schemes [18, 10] and a globally optimal algorithm [11]. Most of these algorithms are based on policy search. To the best of our knowledge, no bottom up dynamic programming (DP) algorithm exists. The advantage of the DP approach lies in its ability to focus planning on the reachable part of the belief space using a forward heuristic search. Such approximate DP algorithms–often referred to as *point-based dynamic programming* techniques–have shown great success in solving POMDPs. The algorithm we introduce, *Constraint-Based Dynamic Programing* (CBDP), shares its motivation with such point-based approaches.

The efficiency of CBDP lies in effectively capturing the independence and structure among agents using constraints. CBDP uses a factored MDP-based heuristic function to identify relevant belief points. Solving a factored MDP is not easy, as the space of both the joint actions and states is exponential in the number of agents. We provide a novel way to approximate this value function efficiently using constraint optimization. The planning (backup operation in DP) for these beliefs is performed using a constraint-based formulation rather than a naive search through the exponential number of policies. Such formulation makes the complexity of CBDP exponential only in the *induced width* of the depth-first search (DFS) ordering of the agent interaction graph, while linear in the number of agents. This enhances the scalability of CBDP for our sensor network domain, allowing us to solve large loosely-coupled multi-agent planning problems. Our results show that CBDP can handle very large domains. It can solve efficiently problems with hundreds of horizon steps while providing orders of magnitude time savings over the previous best algorithm.
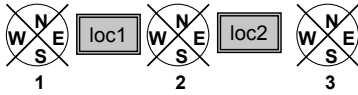
Figure 1: A 3-chain sensor network configuration. Each sensor can scan in 4 directions. Target moves stochastically between loc1 and loc2.

2. BACKGROUND

This section briefly introduces the motivating sensor network domain and the ND-POMDP model. More details can be found in [8, 11].

2.1 Coordinated target tracking

Our target application is illustrated in Figure 1, which shows a 3-chain sensor network configuration. Each sensor has scanning ability in four directions (North, South, East, West). The target moves stochastically between loc1 and loc2. To track the target and obtain the associated reward the sensors with the overlapping area must scan the area together. For example, if the target is present in loc1 then sensor 1 must scan East and sensor 2 must scan West to track it. Thus, coordination plays an important role. The movement of targets is unaffected by sensor agents. Sensors have imperfect observability of the target, so there can be false positive and negative observations. On successfully tracking a target, sensors receive a reward. Otherwise they incur a cost, when they scan an area in an uncoordinated fashion or when the target is absent. The specifications of the model are taken from [11].

2.2 ND-POMDP model

The problem of target tracking in sensor networks can be easily represented as a DEC-POMDP [2]. However, explicit utilization of independence among the agents can help us construct more efficient algorithms. For example, sensors in our domain are mostly independent–their state transitions given the target location and the observations are independent of the actions of the other agents. Agents are coupled only through the joint reward function. The ND-POMDP model [11] can express such independence relations. It is described below for a group of n sensor agents.

S $= \times_{1 \leq i \leq n} S_i \times S_u$. S_i refers to the local state of agent i. S_u refers to a set of uncontrollable states that are independent of the actions of the agents. In the sensor network example, S_i is empty, while S_u corresponds to the set of locations where targets can be present.

A $= \times_{1 \leq i \leq n} A_i$ where A_i is the set of actions for agent i. For the sensor network example $A_1 = \{N, W, E, S, Off\}$.

$\Omega = \times_{1 \leq i \leq n} \Omega_i$ is the joint observation set.

P $P(s'|s,a) = P_u(s'_u|s_u) \cdot \Pi_{1 \leq i \leq n} P_i(s'_i|s_i, s_u, a_i)$, where $a = \langle a_1, \ldots, a_n \rangle$ is the joint action performed in joint state $s = \langle s_u, s_1, \ldots, s_n \rangle$ resulting in joint state $s' = \langle s'_u, s'_1, \ldots, s'_n \rangle$. This relies on transition independence.

O $O(\omega|s,a) = \Pi_{1 \leq i \leq n} O_i(\omega_i|s_i, s_u, a_i)$, where s is the joint state resulting after taking joint action a and receiving joint observation ω. This relies on observation independence.

R $R(s,a) = \Sigma_l R_l(s_l, s_u, a_l)$. The reward function is decomposable among sub groups of agents referred by l. If k agents $i_1, \ldots, i_k$ are involved in a particular sub group l, then s_l denotes the state along the link i.e. $\langle s_{i_1}, \ldots, s_{i_k} \rangle$. Similarly, $a_l = \langle a_{i_1}, \ldots, a_{i_k} \rangle$. In the sensor domain in Figure 1, the reward is decomposed among two subgroups (sensor 1–sensor 2) and (sensor 2–sensor 3), hence $R = R_{12}(s_1, s_2, s_u, a_1, a_2) + R_{23}(s_2, s_3, s_u, a_2, a_3)$. Based on the reward function, an interaction hypergraph can be constructed. A hyperlink l connects the subset of agents which form the reward component R_l. This hypergraph visually depicts the dependency among agents and is an important tool for the complexity analysis in later sections.

b_o $b_o = (b_u, b_1, \ldots, b_n)$ is the initial belief for joint state $s = \langle s_u, s_1, \ldots, s_n \rangle$ and $b(s) = b(s_u) \cdot \Pi_{i=1}^n b(s_i)$.

Solving the above decision problem requires to compute the joint policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ that maximizes the total expected reward of all agents over a finite horizon T starting from the initial belief b_o.

The following theorem illustrates an important advantage of transition and observation independence in ND-POMDP.

THEOREM 1. *Given transition and observation independence, and a decomposable reward function, the value of a joint policy π is also decomposable. That is,*

$$V_\pi^t(s^t, \overrightarrow{\omega}^t) = \sum_l V_{\pi_l}^t(s_l^t, s_u^t, \overrightarrow{\omega}_l^t).$$

Hence, the value function of a policy in ND-POMDP can be represented as a sum of *local* value functions, one for each link. For proof please refer to [11].

3. POINT-BASED DYNAMIC PROGRAMMING

The intractability of optimal (DEC) POMDP algorithms can be attributed to planning for the complete belief space. DEC-POMDPs are further disadvantaged as the joint policy has to be represented explicitly and cannot be extracted directly from the value function. The idea of planning for a finite set of belief points has been explored by Lovejoy [9], leading to several successful point-based POMDP algorithms such as PBVI [13], Perseus [16], and FSVI [15] among others. For DEC-POMDPs, a similar point-based algorithm called MBDP has been developed by Seuken and Zilberstein [14]. However, extending MBDP to multiple agents (>2) is nontrivial because the computational cost of planning as well as the MDP heuristic will require exponential effort. Even representing the problem parameters (reward, transition function) will be exponential in the number of agents, clearly infeasible. CBDP shares its motivation with these successful point-based algorithms. However, it is significantly different from MBDP as it exploits the structure of interactions in the ND-POMDP, whereas MBDP is oblivious to such structure. Next, we presents several novel techniques we employ in CBDP to enable efficient planning for multiple agents.

Algorithm 1 shows the general point-based dynamic programming framework that CBDP follows. The input is a finite horizon problem with T decision steps. The algorithm first computes the heuristic function used for sampling belief points for each time step t. Then, the main bottom-up

Algorithm 1: *Point-based DP framework*

1 $T \leftarrow$ horizon of the problem
2 Belief set $B \leftarrow \{\}$
3 Precompute the belief selection heuristic
4 $\overrightarrow{Q}^T \leftarrow$ all one step policy trees
5 **for** $t = T - 1$ *until 1* **do**
6 $\widetilde{Q}^t \leftarrow$ DoBackup($\overrightarrow{Q}^{t+1}$)
7   Populate B using heuristic function
8 $\overrightarrow{Q}^t \leftarrow$ findBest($\widetilde{Q}^t$,B)
9 Output $\overrightarrow{Q}^1$ as the solution

dynamic programming loop starts. For each step t, the set $\overrightarrow{Q}^t = \{q_i^t | 1 \leq i \leq n\}$ contains the best policies for each agent. The set q_i^t includes policies for agent i that contributed to the best joint policy for *some* sampled belief.

The algorithm starts with the last time step, where there are only immediate actions to take. These actions, which are relatively few, are stored in the set $\overrightarrow{Q}^T$ without any belief sampling. In each further iteration, the backup of the last step's best policies is stored in $\widetilde{Q}^t$. Then, a desired number of beliefs are sampled using the precomputed heuristic function. For each of the sampled beliefs, the best joint policy is extracted from the set $\widetilde{Q}^t$ and stored in $\overrightarrow{Q}^t$, which forms the basis for the next iteration. Finally, at $t = 1$, the best policy for the initial belief b_o is found and returned as the solution. Note that the space and time complexity is defined by the number of sampled beliefs as planning is restricted only to this set. This is the key factor in the efficiency and scalability of point-based approaches.

We now examine the main steps of Algorithm 1 as implemented in CBDP: 1) Computation of the heuristic function; 2) Belief sampling using heuristic function; and 3) Finding the best joint policy for a particular belief. Contrary to a single-agent POMDP, each of these steps is non-trivial for multiple agents. First, the well known MDP heuristic for POMDPs translates to a factored MDP, which is difficult to solve due to exponential state and action space [7]. Second, belief sampling requires reasoning about joint states and actions, which again requires exponential effort if done naively. Finally, computing the best joint policy for a given belief could require search through the exponential number of candidates. We show in the following sections how we perform efficiently each one of these steps of CBDP.

3.1 Heuristic function computation

We use as a heuristic the factored MDP derived from the original ND-POMDP model by ignoring the observations. The optimal policy of this MDP is used for belief sampling in CBDP. One would intuitively believe that the factored state and the reward dynamics of this MDP will make the value function structured, making it easier to compute the optimal policy. Unfortunately, this is not the case. The value function eventually depends on all the states as has been shown by Koller and Parr [7]. This makes the policy harder to compute. To address this, we introduce a way to approximate the value function such that it becomes structured and efficiently computable.

To simplify the equations, we introduce some shorthand notation. Let r_l^t denote the immediate reward along the link

l at time step t when the involved agents perform the action a_l, i.e. $r_l^t = R_l(s_l^t, s_u^t, a_l)$. Let p_u^t denote the transition of the unaffected state at time t, i.e. $p_u^t = P(s_u^{t+1}|s_u^t)$. Let p_i^t denote the transition probability of agent i, i.e. $p_i^t = P_i(s_i^{t+1}|s_i^t, s_u^t, a_i)$. Similar shorthand notation can be defined for a link l as well: p_l^t is the product of the transition probabilities of all the agents (say k) involved in the link l, i.e. $p_l^t = p_{l_1} \ldots p_{l_k}$. The context of states and actions will become clear from the usage.

The MDP value of any joint state s^{t-1}, $V(s^{t-1})$, can be defined as follows:

$$V(s^{t-1}) = \max_a (R(s^{t-1}, a) + \sum_{s^t} P(s^t|s^{t-1}, a)V(s^t)).$$

which can be simplified as follows using the above notation and the transition independence of the factored MDP.

$$V(s^{t-1}) = \max_a (\sum_l r_l^{t-1} + \sum_{s_u^t, s_1^t \ldots s_n^t} p_u^{t-1} p_1^{t-1} \ldots p_n^{t-1} V(s^t)) \tag{1}$$

The time complexity of solving Eq. 1 for all horizons is $O(T|A_i|^n|S_u|^2|S_i|^{2n})$. The presence of two exponential terms, $|A_i|^n$ and $|S_i|^{2n}$, makes it harder to compute. We inductively define an approximate function $\hat{V}$, which is an upper bound on the optimal value function. $\hat{V}$ is decomposable along the links and is thus easier to compute.

For the last time step T, $\hat{V}$ is defined as the sum of immediate rewards for each link. Thus, it is trivially decomposable along the links. Let us assume inductively that for time step t, $\hat{V}$ is decomposable, i.e. $\hat{V}(s^t) = \sum_l \hat{V}_l^t(s_u^t s_l^t|s^t)$. The conditioning over the current state is necessary as the two different joint states s, s' may share the state component $s_u s_l$ along the link l, and the value $V(s_u s_l)$ may be different in states s and s'. For example, let $s = s_u s_1 s_2 s_3$, $s' = s_u s_1 s_2 s_4$ and the state along a link be $s_1 s_2$. The value $V(s_u s_1 s_2)$ can be different for s and s' although they share the state along the link.

The derivation of $\hat{V}$ for the previous step $t-1$ is as follows. Define a new function $\hat{V}'$ for the step $t-1$:

$$\hat{V}'(s^{t-1}) = \max_a \{\sum_l r_l^{t-1} + \sum_{s^t} p_u^{t-1} \ldots p_n^{t-1} \hat{V}(s^t)\}$$

Clearly, $\hat{V}'(s^{t-1})$ is an upper bound on the exact value $V(s^{t-1})$ because we assumed $\hat{V}(s^t)$ was an upper bound on $V(s^t)$. In the following we derive $\hat{V}$ from $\hat{V}'$

$$\begin{aligned}
\hat{V}'(s^{t-1}) &= \max_a \{\sum_l r_l^{t-1} + \sum_{s^t} p_u^{t-1} \ldots p_n^{t-1} \hat{V}(s^t)\} \\
&= \max_a \{\sum_l r_l^{t-1} + \sum_{s^t} p_u^{t-1} \ldots p_n^{t-1} \sum_l \hat{V}_l^t(s_u^t s_l^t|s^t)\} \\
&\leq \max_a \{\sum_l r_l^{t-1} + \sum_{s_u^t, s_1^t \ldots s_n^t} p_u^{t-1} \ldots p_n^{t-1} \sum_l \underbrace{\max_{s^t} \hat{V}_l^t}\}
\end{aligned}$$

The quantity under the brace is uniquely defined for each of the link l and thus can be replaced with $\widetilde{V}_l^t(s_u^t s_l^t) = \max_{s^t} \hat{V}_l^t(s_u^t s_l^t|s^t)$. Intuitively, this approximation will perform better when the sensor agents along a link in a particular state (s_l) with a given external state (s_u) provide similar contribution to the value function across all joint states (s^t).

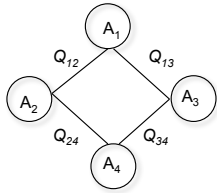Upon rearranging the terms of the previous equation we

Figure 2: A coordination graph for 4 agents

get

$$= \max_a \{ \sum_l r_l^{t-1} + \sum_l \sum_{s_u^t, s_1^t \ldots s_n^t} p_u^{t-1} \ldots p_n^{t-1} \, \widetilde{V}_l^t \}$$

$$= \max_a \{ \sum_l r_l^{t-1} + \sum_l \sum_{s_u^t, s_l^t, s_{-l}^t} p_u^{t-1} p_l^{t-1} p_{-l}^{t-1} \, \widetilde{V}_l^t \}$$

s_l^{t-1} refers to the state of agents along the link l, s_{-l}^t refers to the state of the rest of the agents. Similarly, p_{-l}^{t-1} abbreviates the transition probability product of agents except those on link l. p_{-l}^{t-1} is independent of the rest of the terms and will sum up to 1. Finally, we obtain the decomposable $\hat{V}$ which was our goal.

$$\hat{V}(s^{t-1}) = \max_a \sum_l (r_l^{t-1} + \sum_{s_u^t, s_l^t} p_u^{t-1} p_l^{t-1} \, \widetilde{V}_l^t) \qquad (2)$$

3.1.1 Joint action selection

Solving Eq. 2 still requires iterating over exponential number of actions. We now show how it can be solved efficiently for each state s^t using coordination graphs. These graphs, which were first introduced in the context of factored MDPs [5], are essentially constraint networks whose solution corresponds to the solution of the underlying constraint optimization problem.

A *coordination graph* is very similar to the interaction graph defined using reward functions. There is a node for each agent and an edge (or an hyper-edge) connects nodes for agents that are part of the same reward component R_l. Agents have Q functions defining the local rewards, which also depends upon some other agents–exactly those which share the same reward component. For example, in Figure 2 agent A_2 has two Q functions, Q_{12} shared with agent A_1 along the link $A_1 - A_2$ and Q_{24}. Eq. 2 can be solved for every state s^t by formulating a joint task for the agents to find the joint action $a^\star$ that maximizes the sum of all Q functions $\sum_l Q_l$. The Q function is defined for each link l as follows.

$$s^t : Q_l(a_l) = r_l^t + \sum_{s_u^{t+1}, s_l^{t+1}} p_u^t p_l^t \, \widetilde{V}_l^{t+1}$$

The above optimization problem can be easily solved using techniques from constraint optimization. In particular, the *bucket elimination algorithm* [4] can solve such problems efficiently using dynamic programming. The complexity of bucket elimination is exponential in the induced width of the chosen DFS ordering of the coordination graph and linear in the number of nodes of the graph which is the same as the number of agents. Note that the induced width is much smaller than the number of agents. For a tree-like network, the induced width is 1 no matter how many agents

are involved. Therefore, coordination graphs can be solved efficiently for large loosely coupled systems.

3.1.2 Complexity

Computing the policy for our factored MDP requires solving Eq. 2 for each state and time step t using coordination graphs. The time complexity is $O(nT|S_u|^2|S_i|^{2n}|A_i|^d)$ where d is the induced width of the coordination graph. The space complexity is $O(T|S_u||S_i|^n + n|A_i|^d)$. The term $|A_i|^d$ refers to the space requirement of the bucket elimination algorithm. The exponential factor corresponding to the state space still remains. However, note that for the sensor network domain, the internal states are empty. This makes the time complexity $O(nT|S_u|^2||A_i|^d)$ and space complexity $O(T|S_u| + n|A_i|^d)$ which are polynomial in the state space and linear in the number of agents. Hence, for the sensor network domain we can effectively solve large problems with low induced width.

3.2 Belief sampling

This section describes the belief sampling for any horizon t. We use a portfolio of three heuristic functions: one based purely on the underlying MDP, another taking observations into account, and finally the random heuristic.

3.2.1 MDP-based sampling

In the pure MDP-based sampling, the underlying joint state is revealed after each step. In POMDPs this is easy as we draw a state from the current belief b. However, in a DEC-POMDP the belief is over joint states, hence one has to reason about the joint state space. But fortunately, in a ND-POMDP, $b(s) = b_u(s_u)b_1(s_1)\ldots b_n(s_n)$. This allows us to sample each factored state from its corresponding distribution. Once we get the current underlying joint state $s^t = (s_u^t \ldots s_n^t)$, we select the action $a^\star$ using the precomputed MDP policy. Then, the next unaffected state is sampled from the distribution $s_u^{t+1} = Sample(P(\cdot|s_u^t))$. Each state s_i^{t+1} can be sampled as $s_i^{t+1} = Sample(P(\cdot|s_i^t, s_u^t, a_i^\star))$. Repeated simulations using such a scheme gives us a belief point for any horizon starting from the initial belief b_o.

3.2.2 Direct belief space exploration

One shortcoming of the MDP heuristic is that it ignores the effect of observations which are important to determine the belief state accurately in a partially observable environment. To overcome this limitation, Shani *et al.* proposed a new MDP-based heuristic that directly explores the belief space [15]. Our second heuristic is motivated by this approach, presented in the Algorithm 2. The algorithm returns a belief after applying a desired number of updates. The first few steps are similar to those of the pure MDP heuristic. First, the current underlying state is sampled from the current belief b^t, the MDP action for it is found and the next state is sampled using the corresponding transition functions (steps 3 to 6 of the algorithm). Next, the most probable observation is sampled for each agent i (step 7). Then, the belief for each state component is updated using the action $a^\star$ and the observation $\omega = (\omega_1 \ldots, \omega_n)$. This updated belief is passed to the next iteration of BeliefExplore until the desired number of updates are performed (step 1). The call to BeliefExplore is initialized using the initial belief b_o. Further discussion of belief updates follows.

The belief update of b' based on b, action a and obser-

Algorithm 2: BeliefExplore(b^t)

1 **if** $t == updatesRequired$ **then**
2   **return** b^t
3   Sample s^t from b^t
4 $a^\star \leftarrow$ GetMDPAction(s^t)
5 $s_u^{t+1} \leftarrow Sample(P(\cdot|s_u^t))$
6 $s_i^{t+1} \leftarrow Sample(P(\cdot|s_i^t, s_u^t, a_i^\star)) \quad \forall i = 1\ to\ n$
7 $\omega_i \leftarrow Sample(O(\cdot|s_i^{t+1}, s_u^{t+1}, a_i^\star)) \forall i = 1\ to\ n$
8 $b_u^{t+1} \leftarrow \tau_u(b_u^t, a^\star, \omega)$
9 $b_i^{t+1} \leftarrow \tau_i(b_i^t, a^\star, \omega) \forall i = 1\ to\ n$
10 **return** BeliefExplore(b^{t+1})

vation ω is represented by $b' = \tau(b, a, \omega)$. Performing such updates on the joint belief space is costly. Therefore, we use an approximation scheme which updates the factored belief space individually, i.e. the current belief $b^t = b_u^t b_1^t \ldots b_n^t$ is updated as $b^{t+1} = b_u^{t+1} b_1^{t+1} \ldots b_n^{t+1}$. The approximation is if we are updating the belief over S_i then the rest of the joint state is fixed to the state sampled using MDP policy.

$$b_u^{t+1}(s_u)^{a, \omega} = \alpha \prod_{i=1}^{n} O_i(\omega_i | \underbrace{s_i^{t+1}}, s_u, a_i) \sum_{s_u'} b_u^t(s_u') P(s_u|s_u')$$

The above equation shows the update of the unaffected state space. The part of the joint state other than the unaffected state is fixed (see the term under the brace). Each s_i^{t+1} is found using the MDP policy (in step 6 of Algorithm 2). α is the normalization constant. Similarly, other factored state spaces can be updated as

$$b_i^{t+1}(s_i)^{a, \omega} = \alpha O_i(\omega_i | s_i, s_u^{t+1}, a_i) \prod_{j=1}^{n \backslash i} O_j(\omega_j | s_j^{t+1}, s_u^{t+1}, a_j) \cdot$$
$$\sum_{s_i'} b_i^t(s_i') P(s_i | s_i', s_u^{t+1}, a_i)$$

3.3 Best joint policy computation

This section describes the last step of the DP framework– how to find the best policy for a sampled belief point. After the backup of policies from the last iteration, each agent can have a maximum of $|A_i||B|^{|\Omega_i|}$ policies where $|B|$ refers to the number of sampled belief points, which can be fixed *a priori* to a threshold *maxBelief*. Thus, the total number of joint policies can be $O((|A_j| maxBelief^{|\Omega_k|})^n)$ where A_j is the maximum action space of any agent and Ω_k is the maximum observation space. Searching for the best joint policy will require exponential time in the number of agents if done naively. We formulate below a constraint optimization problem for this task, which performs much better than naive search.

The value for a joint belief b upon executing the joint policy π is given by

$$V_\pi(b) = \sum_s b(s) V_\pi(s, \langle\rangle)$$

Using the independence relations in ND-POMDP and The-

orem 1, we get

$$V_\pi(b) = \sum_{s_u, s_1, \ldots, s_n} b_u(s_u) b_1(s_1) \ldots b_n(s_n) \sum_l V_{\pi_l}(s_l, s_u, \langle\rangle)$$
$$= \sum_l \sum_{s_u, s_l, s_{-l}} b_u(s_u) b_l(s_l) b_{-l}(s_{-l}) V_{\pi_l}(s_l, s_u, \langle\rangle)$$

where $b_l(s_l)$ refers to the product of beliefs for agents on the link l and b_{-l} refers to the rest of the agent. b_{-l} is independent of V_{π_l} and will sum up to 1. So we are left with

$$V_\pi(b) = \sum_l \sum_{s_u, s_l} b_u(s_u) b_l(s_l) V_{\pi_l}(s_l, s_u, \langle\rangle) \qquad (3)$$

The above equation shows that the value of a policy for a belief can be decomposed into contributions along the interaction links. The best joint policy for a given belief is thus the solution of the following optimization problem:

$$V_{\pi^\star}(b) = \max_\pi \{ \sum_l \sum_{s_u, s_l} b_u(s_u) b_l(s_l) V_{\pi_l}(s_l, s_u, \langle\rangle) \}$$

This problem can be solved using a constraint graph similar to the coordination graphs defined in Section 3.1.1. There is a policy variable Π_i for each agent i. The domain $D(\Pi_i) = \{\pi_i\}$ is the set of all backed up policies of agent i. Variables Π_i and Π_j are connected by an edge if their corresponding agents i and j share a reward component R_l. The structure of this graph is similar to the interaction graph, except that the agents are replaced by their corresponding policy variables. A constraint is defined for each link l with the valuation $C_l(\pi_l) = \sum_{s_u, s_l} b_u(s_u) b_l(s_l) V_{\pi_l}(s_l, s_u, \langle\rangle)$. For example, consider the interaction structure of Figure 2. A constraint is defined for each of the 4 links and $C_{12}(\pi_1, \pi_2) = \sum_{s_u, s_1, s_2} b_u(s_u) b_1(s_1) b_2(s_2) V_{\pi_1 \pi_2}(s_1, s_2, s_u, \langle\rangle)$.

The optimization problem is to find the complete assignment $\pi^\star$ to the variables which maximizes the sum $\sum_l C_l$, which is also the best joint policy for the particular belief. Using the bucket elimination algorithm, this problem can be solved in $O(n(|A_j| maxBelief^{|\Omega_k|})^d)$ time and space where d is the induced width of the interaction graph. Thus, we have reduced the complexity from being exponential in the number of agents to being exponential in the induced width, which typically will be much smaller and linear in the number of agents.

4. COMPLEXITY

THEOREM 2. *CBDP has a linear time and space complexity with respect to the problem horizon.*

PROOF. CBDP plans only for the sampled belief set whose size can be fixed to *maxBelief*. Hence, the number of policy trees for each agent is between $Q_{LB} = maxBelief$ (assuming each sampled belief requires a unique policy of the agent) and $Q_{UB} = |A_j| maxBelief^{|\Omega_k|}$. For each horizon, only Q_{LB} policy trees are retained. To construct the new trees after backup, we can use a pointer sharing mechanism in which a tree from the previous iteration can be part of multiple new policy trees. Thus, the size of a T horizon policy can be $O(Q_{LB} \cdot T)$. For n agents the total space required is $O(n \cdot (Q_{LB}.T + Q_{UB}))$. Taking into account the additional space required by the bucket elimination algorithm for the best policy computation, the total space required is
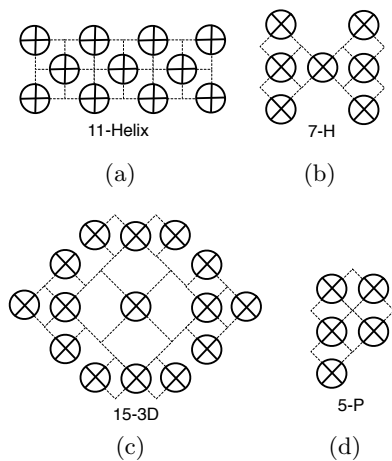
Figure 3: Sensor network configurations

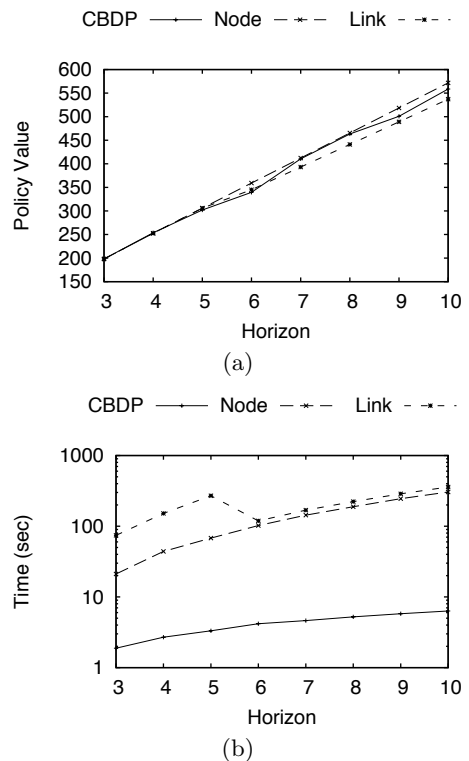**Figure 4: a) compares the solution quality of CBDP and different versions of FANS (Node, Link) on the domain 5P. b) Compares the execution time.**

$O(n \cdot (Q_{LB} \cdot T + Q_{UB}) + (Q_{UB})^d)$ which is linear in the horizon T.

As shown in Section 3.1.2, the complexity of calculating the heuristic also increases linearly with T. Once the heuristic is computed, the complexity depends upon the main dynamic programing loop (steps 5-8) of Algorithm 1. If the size of the sampled belief set is fixed then the running time of the loop is independent of T. Hence, CBDP requires linear time complexity w.r.t. the horizon length. □

The next two theorems show how well the CBDP and the heuristic computation scale when we change the problem parameters–state space, action space or number of agents.

THEOREM 3. *For each horizon, CBDP has polynomial time and space complexity w.r.t. the factored state space, exponential in the induced width d of the interaction graph and linear in the number of agents.*

PROOF. The time complexity of CBDP per horizon depends upon the best joint policy computation. Its complexity is $O(maxBelief \cdot ((Q_{UB})^k L |S_u|^2 |S_i|^{2k} |\Omega_j|^k + n(Q_{UB})^d))$. L denotes the number of links in the interaction graph, and k denotes the maximum number of agents involved in an interaction link and n being the number of agents. The first term denotes the time required to setup the constraint formulation, second term is the time needed to solve it. Similarly, the space complexity is dominated by the bucket elimination algorithm, which is $O(n(Q_{UB})^d)$, again linear in the number of agents and exponential only in the induced width. □

A direct implication of the above theorem is that CBDP can scale up well to very large problem with low induced width. For the sensor network domain, we can further show the following property. A discussion of how to generalize it is included in the last section.

THEOREM 4. *The MDP heuristic requires polynomial time and space per horizon for the sensor network domain.*

PROOF. This follows directly from Section 3.1.2. The time complexity is $O(n|S_u|^2 ||A_i|^d)$ and space complexity $O(|S_u| + n|A_i|^d)$. □

5. EXPERIMENTS

This section compares CBDP with FANS [10]–the most efficient of the existing algorithms including SPIDER [18] and LID-JESP [11]. We conducted experiments on the sensor network configurations introduced in [10], shown in Figure 3. All experiments were conducted on a Dual core machine with 1GB RAM, 2.4GHz CPU. The main purpose of the experiments is to show the scalability of CBDP, which has linear time and space complexity w.r.t. the horizon. We use a publicly available constraint solver implementing the bucket elimination algorithm [12]. The *maxbelief* was set to 5 as it provided good tradeoff between speed and solution quality across a range of tried settings.

The experiments are divided into two sets. In the first set, we use relatively smaller domains (5-P and 7-H) because they are the only domains for which FANS can scale up to significant horizons. In the second set of experiments, we use the remaining larger domains (11-helix, 15-3D, 15-mod). For these domains, FANS cannot scale well (for 15-3D it fails to solve problems with horizons greater than 5).

Figure 4(a) shows a comparison of the solution quality of CBDP and two different versions of FANS, Node ($k = 0.5$) and Link, on the 5-P domain. FANS has multiple heuristics which tradeoff solution quality for speed. It has been suggested in [10] that for smaller domains, the Link and Node heuristics provide the best tradeoff. Hence, we chose these two for comparisons. All three algorithms achieve almost the same solution quality for all horizons. However, the key difference is in the runtime (Figure 4(b)). CBDP provides significant speedup over both the Node and Link heuristics.
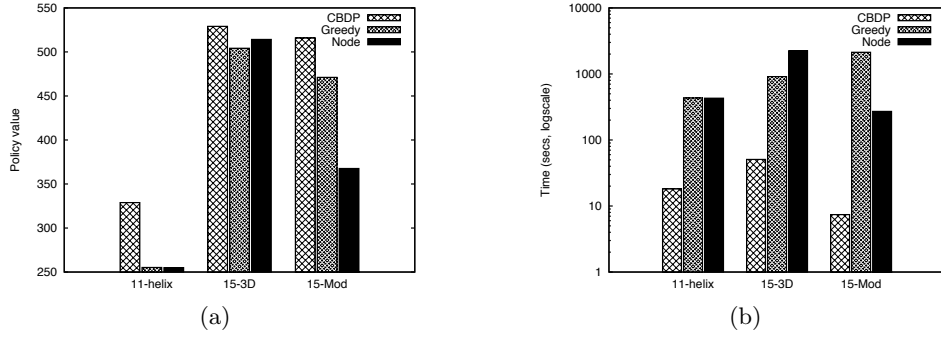
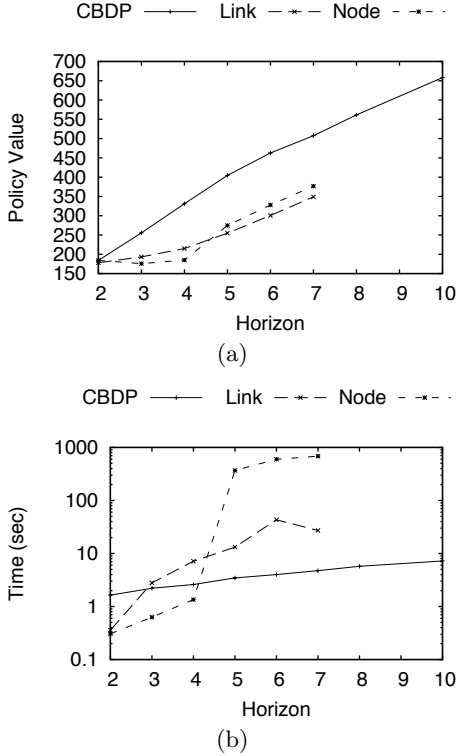Figure 6: Solution quality and execution time for 11-helix, 15-3D and 15-mod with T = 3.

Figure 5: a) compares the solution quality of CBDP and different versions of FANS (Node, Link) on the domain 7-H. b) Compares the execution time.

For horizon 10, CBDP is about 2 orders of magnitude faster than FANS with either heuristic. Another notable observation is that the increase in runtime for CBDP with the horizon is nearly linear as implied by Theorem 2.

Figure 5(a) shows a comparison of the solution quality on the 7-H domain. We again compare with FANS using the Node ($k = 0.5$) and Link heuristics. On this domain, due to increase in the number of agents, FANS can provide solutions up to horizon 7. With the increase in the domain size, FANS is no longer as competitive in the solution quality as it was in the 5-P domain. CBDP provides much better solution quality for all horizons. The runtime graph (Figure 5(b)) shows again that CBDP is much faster and that more scalable. It is 5 times faster than the Link heuristic,

and orders of magnitude faster than the Node heuristic. The average time required by CBDP per horizon is 710ms which implies CBDP can solve a 100 horizon instance within 1.5 minutes due to its linear complexity as opposed to the Node heuristic which requires 685 seconds to solve a horizon 7 instance. This further illustrates the scalability of CBDP.

Figure 6 shows the next set of experiments in which we use larger domains (11-helix, 15-3D, 15-mod). To be consistent with the results of Marecki *et al.* [10], we set the horizon to 3 because for 15-3D, FANS scales up to horizon 4 only. For these larger domains, a Greedy heuristic has been proposed [10] and it provides best tradeoff than other heuristics. Hence, we used this heuristic along with the Node ($k = 0.75$) heuristic for comparisons. Figure 6(a) shows a comparison of the solution quality. For 11-helix, CBDP provides much better solution quality than either the Greedy or Node heuristic. Similar trends are observed for the 15-3D and 15-mod domains. CBDP provides better solution quality for them as well. The runtime comparison (Figure 6(b)) again shows the stark difference between CBDP and FANS. For 15-mod, CBDP takes 0.6 secs while FANS with the Greedy heuristic takes over 2,000 secs. One can easily see the magnitude of speedup CBDP provides over FANS.
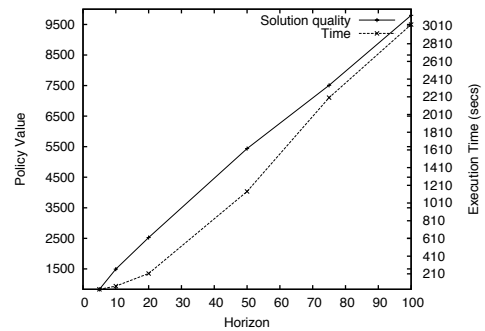

Figure 7: Solution Quality and Execution time for a range of horizons for 15-3D.

The results of the last set of experiments are shown in Figure 7. These experiments show the solution quality and execution time statistics for CBDP for different horizons on the problem 15-3D, which is the most complex problem instance for both CBDP and FANS. FANS could scale up to only horizon 5 on this problem. In contrast, CBDP can easily scale up to horizon 100. The execution time curve in this

experiment clearly supports the linear time complexity of CBDP. The solution quality too increases at a nearly constant rate with the horizon, which means that CBDP is able to maintain good solution quality for large horizons.

To summarize, our experiments consistently illustrate that CBDP provides several orders-of-magnitude of speedup over different versions of FANS, while providing better solution quality for all the instances examined. CBDP further has a linear time and space complexity w.r.t. the horizon, which makes it possible to apply it to problems with much larger horizons than was previously possible.

6. CONCLUSION AND FUTURE WORK

We have introduced a point-based dynamic programing algorithm called CBDP for ND-POMDPs–a restricted class of DEC-POMDPs characterized by a decomposable reward function. CBDP takes advantage of the structure of interaction in the domain and independence among the agents to produce significantly better scalability over the state-of-the-art algorithms. It uses constraint networks algorithms to improve the efficiency of key steps of the dynamic programming algorithm. Consequently, CBDP has linear time and space complexity with respect to the horizon length and polynomial complexity in other problem parameters. The key feature of CBDP which contributes to its scalability is that the complexity per horizon step is exponential only in the induced width of the agent interaction graph, while it is linear in the number of agents. The induced width is often much smaller than the number of agents–particularly in loosely connected multi-agent systems–implying that CBDP can scale up better and handle multi-agent systems that are significantly larger than the existing benchmark problems. Experimental results consistently validate these properties of CBDP.

Continued work on CBDP focuses on two research directions. First, we are exploring additional real-world settings that have an underlying interaction structure that CBDP can exploit. Sensor networks in particular provide a good application area where the challenge is to coordinate the operation of a large team of sensors. The second research direction aims to make the computation of the factored MDP heuristic efficient in more complex domains. For the target tracking problem, the heuristic computation is already efficient. But for problems that require agents to maintain substantial internal states, an exponential effort is required. To avoid this blowup in computation time, we plan to use the approach developed by Guestrin *et al.* [5], which has polynomial time complexity in the state space for factored MDPs such as the ones used in our work. Such an approach will further increase the scalability and applicability of DEC-POMDPs to more complex real-world problems.

Acknowledgments

7. REFERENCES

[1] R. Becker. *Exploiting Structure in Decentralized Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2006.

[2] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[3] A. Carlin and S. Zilberstein. Value-based observation compression for DEC-POMDPs. In *Proc. of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 501–508, 2008.

[4] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[5] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Proc. of the Neural Information Processing Systems*, pages 1523–1530, 2001.

[6] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

[7] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *Proc. of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1332–1339. Morgan Kaufmann, 1999.

[8] V. Lesser, M. Tambe, and C. L. Ortiz, editors. *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[9] W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1991.

[10] J. Marecki, T. Gupta, P. Varakantham, M. Tambe, and M. Yokoo. Not all agents are equal: Scaling up distributed POMDPs for agent networks. In *Proc. of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 485–492, 2008.

[11] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the Twentieth National Conference on Artificial Intelligence*, pages 133–139, 2005.

[12] A. Petcu and B. Faltings. Dpop: A scalable method for multiagent constraint optimization. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

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

[14] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligences*, pages 2009–2015, 2007.

[15] G. Shani, R. I. Brafman, and S. E. Shimony. Forward search value iteration for POMDPs. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2619–2624, 2007.

[16] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[17] D. Szer and F. Charpillet. MAA*: a heuristic search algorithm for solving decentralized POMDPs. In *Proc. of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 576–583, 2005.

[18] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo. Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In *Proc. of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1–8, 2007.