Multi-Agent Online Planning with Communication

Feng Wu
Department 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
Department of Computer Science
University of Sci. & Tech. of China
Hefei, Anhui 230027 China
xpchen@ustc.edu.cn

Abstract

We propose an online algorithm for planning under uncertainty in multi-agent settings modeled as DEC-POMDPs. The algorithm helps overcome the high computational complexity of solving such problems off-line. The key challenge is to produce coordinated behavior using little or no communication. When communication is allowed but constrained, the challenge is to produce high value with minimal communication. The algorithm addresses these challenges by communicating only when history inconsistency is detected, allowing communication to be postponed if necessary. Moreover, it bounds the memory usage at each step and can be applied to problems with arbitrary horizons. The experimental results confirm that the algorithm can solve problems that are too large for the best existing off-line planning algorithms and it outperforms the best online method, producing higher value with much less communication in most cases.

Introduction

In many applications involving cooperative robots, distributed sensor networks, or communication networks, multi-agent systems offer a natural and robust framework for planning. Besides taking actions to achieve common goals, planning in these settings includes coordination to ensure that a group of agents can work together in a coherent manner under uncertainty. In typical domains such as robot soccer, each robot operates autonomously, but is also part of a team and must cooperate with the other members of the team to play successfully. The sensors and actuators used in such systems introduce considerable uncertainty. What makes such problems particularly challenging is that each agent gets a different stream of observations at runtime and has a different partial view of the situation. And while the agents may be able to communicate with each other, sharing all the information all the time is not possible.

When agents have limited information about their teammates, they must reason about the possible policies of team members and how these policies affect their own behaviors. Communication can alleviate this problem by sharing private information such as sensory data. However, communication is often limited by bandwidth and sometimes can be costly or unreliable. For example, robots that work underground or on another planet may need to move to certain locations to initiate communication. Even when communication is readily available and cheap–for instance, in the case of indoor mobile robots–limited bandwidth and unreliability often lead to latency and robots may need to wait for a period or re-communicate several times until the critical information is fully received. In all these situations, too much communication will affect negatively the performance of the system. It is known that appropriate amounts of communication can improve the tractability and performance of multi-agent systems. The interesting and challenging question addressed by this paper is how to integrate planning with communication and use communication effectively.

The Markov Decision Process (MDP) and its partially observable counterpart (POMDP) have proved very useful for planning and learning under uncertainty. The Decentralized POMDP (DEC-POMDP) offers a natural extension of these frameworks for cooperative multi-agent settings. We use DEC-POMDPs to model multi-agent systems. Solving optimally general finite-horizon DEC-POMDPs has been shown to be NEXP-complete (Bernstein *et al.* 2000), much harder than single-agent MDPs (NP-complete) and single-agent POMDPs (PSPACE-complete). Although free communication transforms a multi-agent DEC-POMDP into a large single-agent POMDP, computing optimal communication policies when communication is not free is as hard as the general DEC-POMDP problem.

Background on Communication in DEC-POMDPs

Developing algorithms for solving DEC-POMDPs has been an active research area (Seuken & Zilberstein 2008). The literature on communication in DEC-POMDPs or equivalent models can be divided into works that compute full off-line policies and those that do not. In the former group, plans and communication strategies are determined off-line and stored for use at runtime. The DEC-POMDP-COM model (Goldman & Zilberstein 2003), which is equivalent to the COM-MTDP model (Pynadath & Tambe 2002), provides a theoretical framework for reasoning about communication off-line. Spaan *et al.* (2006) established a new model where messages are sent as part of agents' action vectors and received in the next time step as part of the recipients' observation vectors.

Generally, reasoning about communication off-line re-

quires the enumeration of all possible messages and their effect on the team. Unfortunately, the number of these messages grows exponentially and is as large as the set of all possible observation histories. The COMMUNICATIVE DP-JESP technique integrates a communication strategy into K-step pieces of the JESP algorithm and finds a Nash equilibrium of policies for multiple agents (Nair *et al.* 2004). In order to keep the algorithm tractable, it uses a fixed communication decision which enforces a rule that communication must occur at least every K steps.

In other approaches, the decision when and what to communicate occurs at execution time. Xuan *et al.* (2001) consider communication whenever the monitoring agent notices ambiguity in what an agent should plan next. Oliehoek *et al.* (2007) use a QBG (Oliehoek *et al.* 2008) heuristic to find communication policies for domains where communications have a one-step delay. More recent work extends this approach to handle stochastic delays (Spaan *et al.* 2008). Several different aspects of communication for special cases of DEC-POMDPs have been studied in recent years. Becker *et al.* (2005) developed a myopic communication strategy for transition-independent DEC-MDPs. Roth *et al.* (2007) proposed an algorithm to generate decentralized policies with minimal communication for factored DEC-MDPs. Williamson *et al.* (2008) introduced the dec_POMDP_Valued_Com model, which includes a special communication reward function in the DEC-POMDPs.

The approaches most similar to ours are BaGA-Comm (Emery-Montemerlo 2005) and Dec-Comm (Roth *et al.* 2005). In the BaGA-Comm framework, a Bayesian game is constructed and solved using BaGA-Cluster (Emery-Montemerlo *et al.* 2005) to generate policies and joint-type spaces at each time-step. The authors present three types of communication strategies: a fixed policy, an expected-value-difference (EVD) policy and an approach based on policy-difference (PD). The experimental evaluation of these method showed that EVD and PD result in similar performance and number of communication acts, and are much better than the fixed policy approach (Emery-Montemerlo 2005). That work has demonstrated that it is important to decide when and not just how often an agent should communicate to achieve good performance. In the Dec-Comm framework, each agent maintains a distribution of possible joint beliefs and chooses to communicate only when integrating its own observation history into the joint belief causes a change in the joint action selected by QPOMDP. Basically, communication in Dec-Comm is based on the PD policy. Subsequent work has also addressed the question of what to communicate (Roth *et al.* 2006).

Overview of MAOP-COMM

In this paper, we introduce a new Multi-Agent Online Planning algorithm with Communication (MAOP-COMM). Unlike BaGA-Cluster, which merges histories by their similarity in terms of the worst-case expected loss (Emery-Montemerlo *et al.* 2005), we merge histories by the similarity of the future policy structures. Moreover, our approach *bounds* the size of the histories in memory at each step while BaGA-Cluster does not. Thus, MAOP-COMM can solve problems in which agents may not communicate for a long period of time, while BaGA-Comm becomes intractable very quickly when the state and observation spaces are large. The Dec-Comm framework with particle filtering also requires a fixed amount of memory by using sampling (Roth *et al.* 2005). However, in many domains, the number of particles needed to accurately model joint beliefs may be large. With the particle representation, agents may initiate too much communication when the real joint belief is not sampled. Our approach can better address these situations as it communicates when history *inconsistency* is detected. To the best of our knowledge, this is a new way to initiate communication dynamically at runtime. Another feature of our work is that we do not require instantaneous communication. Agents can *postpone* their communication when the communication resource is unavailable or there is some delay or error in communication. This is a more realistic assumption for many multi-agent systems. Certainly, postponing communication will decrease the value of the plan, but the ability to continue to act until successful communication can be completed is important.

The rest of the paper is organized as follows. We first introduce the formal model. Then we describe how the algorithm works. Finally, we present experimental results and summarize the contributions and future work.

Decentralized POMDPs

We adopt the DEC-POMDP framework (Bernstein *et al.* 2000), however our approach and results apply to equivalent models such as MTDP (Pynadath & Tambe 2002).

Definition 1 (DEC-POMDP) *A finite-horizon decentralized partially observable Markov decision process is a tuple* $\langle I, S, \{A_i\}, \{\Omega_i\}, P, O, R, b^0 \rangle$ *where*

- I *is a finite set of agents indexed* $1, \ldots, n$.
- S *is a finite set of states.*
- A_i *is a finite set of actions available to agent* i *and* $\vec{A} = \times_{i \in I} A_i$ *is the set of joint actions, where* $\vec{a} = \langle a_1, \ldots, a_n \rangle$ *denotes a joint action.*
- Ω_i *is a finite set of observations available to agent* i *and* $\vec{\Omega} = \times_{i \in I} \Omega_i$ *is the set of joint observations, where* $\vec{o} = \langle o_1, \ldots, o_n \rangle$ *denotes a joint observation.*
- P *is a Markovian state transition probability table.* $P(s'|s, \vec{a})$ *denotes the probability that taking joint action* $\vec{a}$ *in state* s *results in a transition to state* s'.
- O *is a table of observation probabilities.* $O(\vec{o}|s', \vec{a})$ *denotes the probability of observing joint observation* $\vec{o}$ *after taking joint action* $\vec{a}$ *and reaching state* s'.
- $R : S \times \vec{A} \rightarrow \Re$ *is a reward function.* $R(s, \vec{a})$ *denotes the reward obtained from taking joint action* $\vec{a}$ *in state* s.
- $b^0 \in \Delta(S)$ *is the initial belief state distribution.*

Solving a DEC-POMDP for a given horizon T and start distribution b^0 can be seen as finding policies that maximize the expected joint reward $E[\sum_{t=0}^{T-1} R(s_t, \vec{a}_t)|b^0]$.

Formally, we define the history for agent i, h_i, as the sequence of actions taken and observations received by agent i. At any time step t, $h_i^t = (a_i^0, o_i^1, a_i^1, \ldots, o_i^{t-1}, a_i^{t-1}, o_i^t)$,

and the joint history, $h^t = \langle h_1^t, \ldots, h_n^t \rangle$. The term joint belief $b(h) \in \Delta(S)$ denotes the probability distribution over states induced by joint history h. Given a set of joint histories of the previous step, computing a set of joint belief states of current step is straightforward using Bayes' rule:

$$b^t(s') = \frac{O(\vec{o}|s', \vec{a}) \sum_s P(s'|s, \vec{a}) b^{t-1}(s)}{\sum_{s''} O(\vec{o}|s'', \vec{a}) \sum_s P(s''|s, \vec{a}) b^{t-1}(s)} \quad (1)$$

A local deterministic policy δ_i for agent i is a mapping from local histories to actions in A_i, i.e. $\delta_i(h_i) = a_i$. And a joint deterministic policy, $\delta = \langle \delta_1, \ldots, \delta_n \rangle$, is a tuple of local deterministic policies, one for each agent, i.e. $\delta(h) = \langle \delta_1(h_1), \ldots, \delta_n(h_n) \rangle = \vec{a}$. The deterministic policy can be represented as a policy tree with action nodes and observation edges and the joint deterministic policy is a group of policy trees. Similarly, a local stochastic policy for agent i, $\pi_i(a_i|h_i)$, is a mapping from a local history h_i to a distribution over A_i and a joint stochastic policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ is tuple of local stochastic policies.

The MAOP-COMM Algorithm

We propose a new algorithm, Multi-Agent Online Planning with Communication (MAOP-COMM), for finding approximate solutions of DEC-POMDPs. This algorithm is executed in parallel by all the agents in the team, interleaving planning and execution. At each step, each agent maintains the *same* joint histories for the team. This ensures that all the agents find the same joint policy and thus remain coordinated. While the algorithm is randomized, it nevertheless ensure that each agent finds the same set of joint policies by using the same pseudo-random number generator with an *identical* seed. It is important to emphasize that we only use common knowledge for planning. With the same history pool and random behavior, each agent can generate exactly the same joint policy. Each agent's local observation is used only for policy execution and inconsistency detection.

Communication is triggered when the pool of histories becomes inconsistent with the local observation that the agent receives. The agent then initiates communication as soon as the communication resource is available. When communication occurs, each agent broadcasts its own local action-observation sequence to the other agents. Consequently, each agent can construct the actual joint history and calculate the actual joint belief state. The best joint action will be selected based on the new joint belief state. And the history pool will be emptied and replaced with the actual history.

The algorithm starts by first calculating and executing the best joint action for the initial belief state using a heuristic value function. Then, the main planning loop shown in Algorithm 1 is executed. The rest of this section explains the algorithm in detail.

Searching Stochastic Policies

In settings like DEC-POMDPs, agents without knowledge of others' observations must reason about all possible belief states that could be held by others and how that affects their own action selection. In order to find agent i's policy

Algorithm 1 Multi-Agent Online Planning with Comm

Input: b^0, $seed[1..T-1]$
  for all $i \in I$ (**parallel**) **do**
    $\vec{a}^0 \leftarrow \arg\max_{\vec{a}} Q(\vec{a}, b^0)$
    Execute the action a_i^0 and initialize h_i^0
    $H^0 \leftarrow \{\vec{a}^0\}$; $B^0 \leftarrow \{b^0\}$; $\tau_{comm} \leftarrow false$
    for $t = 1$ **to** $T-1$ **do**
      Set the same random seed by $seed[t]$
      $H^t, B^t \leftarrow$ Expand histories and beliefs in H^{t-1}, B^{t-1}
      $o_i^t \leftarrow$ Get the observation from the environment
      $h_i^t \leftarrow$ Update agent i's own local history with o_i^t
      if H^t is inconsistent with o_i^t **then**
        $\tau_{comm} \leftarrow true$
      if $\tau_{comm} = true$ **and** communication available **then**
        Synch h_i^t with other agents
        $\tau_{comm} \leftarrow false$
      if agents communicated **then**
        $h^t \leftarrow$ Construct the communicated joint history
        $b^t(h^t) \leftarrow$ Calculate the joint belief state for h^t
        $\vec{a}^t \leftarrow \arg\max_{\vec{a}} Q(\vec{a}, b^t(h^t))$
        $H^t \leftarrow \{h^t\}$; $B^t \leftarrow \{b^t(h^t)\}$
      else
        $\pi^t \leftarrow$ Search the stochastic policy for H^t, B^t
        $a_i^t \leftarrow$ Select an action according to $\pi^t(a_i|h_i^t)$
        $H^t, B^t \leftarrow$ Merge histories based on π^t
      $h_i^t \leftarrow$ Update agent i's own local history with a_i^t
      Execute the action a_i^t

q_i for history h_i, agents need to reason about all the possible histories h_{-i} held by others as well as all the possible policies associated with them. It is important to point out that the joint policy created by our approach is very different from the joint policy found by Dec-Comm, which is a mapping from an agent to a policy (i.e. $\vec{q} = \langle q_1, \ldots, q_n \rangle$) (Roth *et al.* 2005). Our search process produces a truly decentralized policy, which considers the private information of each agent. Thus the resulting policy of each agent depends on its individual observations history (i.e. $\delta(h) = \langle \delta_1(h_1), \ldots, \delta_n(h_n) \rangle$).

The straightforward way of finding the best joint policy is to enumerate all possible combinations of the history-policy mappings and choose the best one. However, the size of the joint space is exponential over the size of the possible histories. In our algorithm, we use stochastic policies and a linear program to find the approximate solution. The value of a joint stochastic policy, π, is as follows:

$$V(\pi) = \sum_{h,b,\vec{q}} p(h|b) \prod_i \pi_i(q_i|h_i) Q(\vec{q}, b) \quad (2)$$

$$\text{where} \quad p(h|b) = \begin{cases} p(h) & \text{if } b = b(h) \\ 0 & \text{otherwise} \end{cases}$$

We use one-step lookahead to estimate the future value. Any off-line approaches which provide a set of value functions $V(s)$ can be used to define the heuristic. One approach we used is the solution for the underlying MDP. For the one-step lookahead case, $q_i, \vec{q}$ can be simplified to $a_i, \vec{a}$. The QMDP (Littman *et al.* 1995) heuristic is written as follows:

$$Q(\vec{a}, b) = \sum_s b(s)[R(s, \vec{a}) + \sum_{s'} P(s'|s, \vec{a}) V_{MDP}(s')].$$

Table 1: Improving the policy using linear programming

Variables: $\varepsilon, \pi_i(q_i\|h_i)$	
Objective: Maximize ε	
Improvement constraint:	
$V(\pi) + \varepsilon \leq \sum_{h,b,\vec{q}} p(h\|b)\pi_i(q_i\|h_i)\pi_{-i}(q_{-i}\|h_{-i})Q(\vec{q},b)$	
Probability constraints:	
$\forall h_i$	$\sum_{q_i} \pi_i(q_i\|h_i) = 1$
$\forall h_i, q_i$	$\pi_i(q_i\|h_i) \geq 0$

To start, each local stochastic policy π_i is initialized to be deterministic, by selecting a random action with a uniform distribution. Then, each agent is selected in turn and its policy is improved while keeping the other agents' policies fixed. This is done for agent i by finding the best parameters $\pi_i(q_i|h_i)$ satisfying the following inequality:

$$V(\pi) \leq \sum_{h,b,\vec{q}} p(h|b)\pi_i(q_i|h_i)\pi_{-i}(q_{-i}|h_{-i})Q(\vec{q},b) \quad (3)$$

The linear program shown in Table 1 is used to find the new parameters. The improvement procedure terminates and returns π when ε becomes sufficiently small for all agents. Random restarts are used to move out of local maxima. It is easy to construct situations in which two policies q_i and q_i' may have the same value for h_i. In order to guarantee coordination, each agent will choose the identical policy according to a predetermined canonical policy ordering (e.g. $q_i \prec q_i'$) if the equilibrium-selection tie happens.

Bounding Joint Histories

Note that the underlying system state as well as the observations of other agents are not available during execution time of DEC-POMDPs. Each agent must reason about all possible histories that could be observed by the other agents and how that may affect its own action selection. However, the number of possible joint histories increases exponentially with the horizon. For a two-agents tiger problem with two observations (Nair *et al.* 2003), the number of possible joint histories with 100 steps is $2^{100\times2}$. This presents a major challenge for developing online algorithms for DEC-POMDPs. Unlike existing approaches, we present a way to maintain a bounded size history pool online and use it to coordinate the strategy of the team.

The key observation here is that the history tracked by each agent is not necessarily the true individual history of that agent. As long as the true history and the stored one share the same future policies, the agent can choose the *right* best-response action for the current step based on its current local observation. A more detailed analysis shows that most of the histories kept in memory are useless. In our algorithm, we merge histories whenever they result in a similar future policy. We only keep one joint history per future policy, so the number of histories retained is bounded by the number of the future policies generated and the definition of similarity.

Theorem 1 *If the policies generated at each step are optimal, merging histories with the same policies will not affect the optimality of future steps.*

Proof: (**sketch**) See the Appendix. □

Algorithm 2 Merge Histories

Input: H^t, Q^t, π^t
  for all $i \in I$ **do**
    $\mathcal{H}_i(q_i) \leftarrow \emptyset, \forall q_i \in Q_i;\ H_i'^t \leftarrow \emptyset$
    for all $h_i \in H_i^t$ **do**
      $q_i \leftarrow$ Select a policy according to $\pi^t(q_i|h_i)$
      $\mathcal{H}_i(q_i) \leftarrow \mathcal{H}_i(q_i) \cup \{h_i \cup a_i(q_i)\}$
    for all $q_i \in Q_i^t$ **do**
      $h_i' \leftarrow$ Select a history from $\mathcal{H}_i(q_i)$ **randomly**
      $H_i'^t \leftarrow H_i'^t \cup \{h_i'\}$, **if** $h_i' \neq null$
  return H'^t

Unfortunately, the optimal future policy for each agent is not available during execution time. Finding the optimal future policy is as hard as solving the entire problem. But we can approximate future polices using limited lookahead. A k-step lookahead policy is a set of policy trees of depth-k, one for each agent. The value function therefore is decomposed into an exact evaluation of the k-steps and a heuristic estimate of the remaining part. Then, we can define similarity by comparing the depth-k policy trees. In this paper, we require that the depth-k policy trees be identical for them to be considered similar, but that can be generalized to other measures of similarity.

In our implementation, we only try one-step lookahead. So the future policy for each agent is just a one-node policy tree associated with a certain action. When storing the history, we do not need to save the whole action-observation sequence in the history pool. We only need to assign an index to a history and use $\langle\vec{\theta}, b\rangle$ to represent the joint history where θ_i is the history index for agent i and b is the joint belief induced for the joint history. Each agent's own local history used for execution is also represented by an index. At every step, we update each index as well as the joint belief state. Since we keep only one history for one policy, the history index for agent i can be represented as a tuple $\langle q_i^{t-1}, o_i^t\rangle$ where q_i^{t-1} is the policy tree index of the previous step and o_i^t is the observation of the current step. For the one-step lookahead case, this index can be further simplified as $\langle a_i^{t-1}, o_i^t\rangle$. At each step, we will update all the indexes of the histories in the pool, as well as the index of the agent's own local history. Figure 1 shows how to expand histories and update the index of the agent's local history by its observation. Figure 2 shows how to merge histories and update the index of the agent's local history after executing a policy.

Communicating When Inconsistency Arises

The technique described above can be used as a standalone algorithm for multi-agent online planning. In that case, each

Algorithm 3 Expand Histories and Update Beliefs

Input: H^{t-1}, B^{t-1}
  $H^t \leftarrow \emptyset;\ B^t \leftarrow \emptyset$
  for $\forall h \in H^{t-1}, \forall \vec{o} \in \vec{\Omega}$ **do**
    $h^t \leftarrow h \cup \vec{o};\ p(h^t) \leftarrow p(\vec{o}, \vec{a}(h)|h)p(h)$
    $b^t(h^t) \leftarrow$ Update the belief state for $b^{t-1}(h)$
    $H^t \leftarrow H^t \cup \{h^t\}, B^t \leftarrow B^t \cup \{b^t(h^t)\}$, **if** $p(h^t) > 0$
  return H^t, B^t

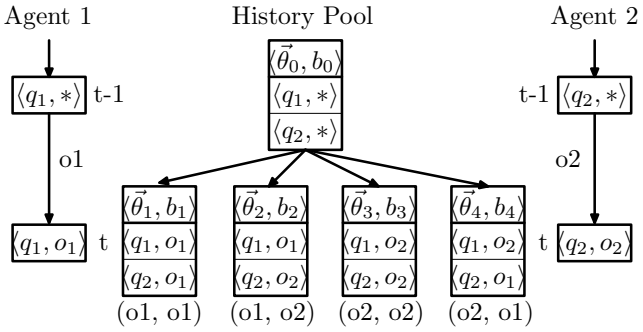

Figure 1: Example of history expansion and updating. The joint history $\langle \vec{\theta}_0, b_0 \rangle$ with two components $\langle q_1, * \rangle$ and $\langle q_2, * \rangle$ in the pool is expanded to $\langle \vec{\theta}_1, b_1 \rangle, \langle \vec{\theta}_2, b_2 \rangle, \langle \vec{\theta}_3, b_3 \rangle, \langle \vec{\theta}_4, b_4 \rangle$ by assigning all possible joint observations. Agent 1 gets the observation o_1 from the environment and updates its local history from $\langle q_1, * \rangle$ to $\langle q_1, o_1 \rangle$. Agent 2 gets the observation o_2 and updates its local history from $\langle q_2, * \rangle$ to $\langle q_2, o_2 \rangle$.

agent maintains a shared pool of histories and searches the best stochastic policy. However, this may lead to poor performance for some domains because only approximations of the exact future policies are available and errors might accumulate at every step. We analyze one major source of error in our algorithm and integrate limited communication to improve its performance.

Notice that we only keep one history per policy when merging the histories. We may merge *inconsistent* histories because we only consider partial future policies. When the partial policies look similar, the full policies they correspond to may still be different. Two histories are *inconsistent* if the corresponding optimal future policies, which are not available at execution time, are different. We can detect the problem by examining any inconsistency between the history pool and the real observation from the environment. When agent i tries some action and observes o_i, the probability of which is less than some small threshold ϵ according to the history pool, it is likely that there is something wrong with the history pool.

Let's denote agent i's local history at step t by h_i^t and the local observation agent i receives from the environment at step t by o_i^t. Note that h_i^t is the local history we maintain at step t, not the local action-observation sequence. We denote $B(h_i^t)$ a set of joint beliefs for h_i^t from the pool B^t.

Definition 2 *At time step t, the maintained history pool H^t is ϵ-inconsistent with agent i's local observation o_i^t if*

$$\forall o_{-i}^t, \max_{b \in B(h_i^t)} \sum_{s'} O(\vec{o}^t | s', \vec{a}) \sum_{s} P(s' | s, \vec{a}) b(s) < \epsilon \quad (4)$$

This definition provides a way to monitor inconsistency between agent i's local history and observation, which provides an indication of history inconsistency in the pool. The threshold ϵ is determined by the structure of the observation function. If the uncertainty of the observation is small, ϵ should be small too. The amount of communication is determined by both the observation structure and the heuristic.

Intuitively, the agent can make the right decision as long as the history pool contains the real joint history or some other similar joint history. However, agents cannot obtain the observations of other agents at execution time, so it is impossible to know the real joint history. But it can check for inconsistency between the history pool and its actual observation. If the pool is inconsistent, the agent can refresh the history pool by communicating with other agents and synchronizing the observation sequence. After synchronizing observations, the history pool contains only the real joint history and is consistent. Unlike most approaches which require instantaneous communication, our approach allows agents to postpone communication when the resource is unavailable. They can sacrifice some value and make decisions without communication. The role of communication is therefore to improve the performance when it is possible.

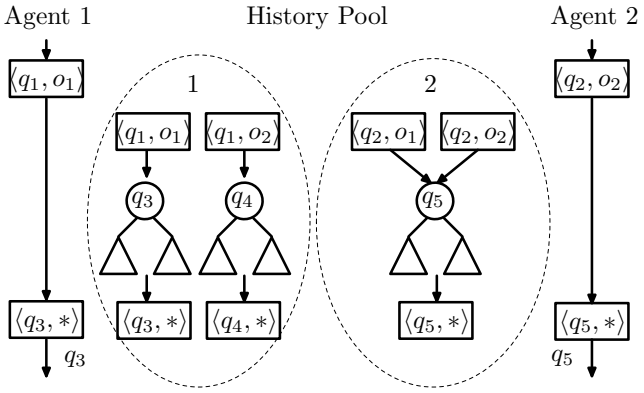

Figure 2: Example of history merging and policy execution. The histories $\langle q_2, o_1 \rangle$ and $\langle q_2, o_2 \rangle$ in the pool map to the same policy q_5, so only one history is *randomly* selected (e.g. $\langle q_2, o_1 \rangle$) for the next step and its index is updated as $\langle q_5, * \rangle$. $\langle q_1, o_1 \rangle$ maps to q_3 and is updated as $\langle q_3, * \rangle$. $\langle q_1, o_2 \rangle$ maps to q_4 and is updated as $\langle q_4, * \rangle$. The policy for the local history of agent 1 $\langle q_1, o_2 \rangle$ is q_3, so agent 1 executes q_3 and update its local history to $\langle q_3, * \rangle$. Agent 2 executes q_5 and updates its local history to $\langle q_5, * \rangle$ since $\langle q_2, o_2 \rangle$ maps to q_5.

Experimental Results

We have implemented and tested MAOP-COMM using three standard benchmark problems and a more challenging problem called grid soccer. In each of these environments, we first solved the underlying (centralized) MDP and provided the resulting value function as a heuristic to our algorithm. The reported results are averages over 20 runs of the algorithm on each of the problems. We present the average accumulated reward (R), average online runtime per step (T(s)), and average percentage of communication steps (C(%)) with different horizons (H). While communication is limited and minimizing it is an important goal, we did not add explicit cost for communication because any such cost would have been arbitrary and not particularly relevant to these applications. The main purpose of the valuation is to test whether high-valued plan can be computed quickly on-line, while using little communications. MAOP-COMM was implemented in Java and ran on a 2.4GHz Intel Core 2 Duo processor with 2GB of RAM. Linear programs were solved using lp_solve 5.5. All timing results are CPU times

with a resolution of 0.01 second.

We did try to compare MAOP-COMM with the two existing online planners that use communication (i.e., BaGA-Comm and Dec-Comm), but only Dec-Comm with particle filtering (Dec-Comm-PF) can solve the problems we focus on in this paper. As mentioned earlier, the main reason for this limitation is that BaGA-Comm and the exact version of Dec-Comm do not bound the size of histories (or beliefs). In our experiments, we observed that agents often kept silent for 10 steps or more. Consequently, the number of possible joint histories becomes very large (e.g., $5^{2\times 10}$ for 2 agent problem with 5 observations after 10 step of silence). Even the BaGA-Cluster approach could not reduce such pool of histories to a manageable size in the test domains. BaGA-Comm and the exact version of Dec-Comm ran out of memory and time very quickly. Therefore, we compared MAOP-COMM with Dec-Comm-PF, the only existing algorithm that bounds the amount of memory.

In fact, BaGA-Comm and Dec-Comm yield similar performance (average values and amount of communication) in most domains which are tractable for both of them (Emery-Montemerlo 2005). Another fact pointed out by Roth *et al.* (2005) is that Dec-Comm using an exact tree representation of joint beliefs and Dec-Comm-PF that approximates beliefs using sufficiently large particle filters provide no substantial difference in performance. Therefore, comparing to Dec-Comm-PF–the leading communicative online algorithm which is applicable to all the problems we focus on in this paper–presents the best way to assess the performance of our approach.

To put these results in perspective and better understand the role of communication, we also include the results for FULL-COMM–the case of full communication (communicating observations at each step, $\epsilon = +\infty$) and MAOP–our own online approach with no communication (no inconsistency monitoring, $\epsilon = 0$). The monitoring threshold for MAOP-COMM was set to $\epsilon = 0.01$ and the number of particles for Dec-Comm-PF was 100. According to our experiments, using more than 100 particles resulted in no substantial difference in value but an obvious increase in runtime for Dec-Comm-PF in the tested domains. We did not use a discount factor in these experiments because all the tested problems involve a finite horizon.

Benchmark Problems The first set of experiments involves three standard benchmark problems: Broadcast Channel (Bernstein *et al.* 2005) , Meeting in a Grid (Bernstein *et al.* 2005) and Cooperative Box Pushing (Seuken & Zilberstein 2007). All of them are typical cooperative multi-agent domains and well-known benchmark problems for DEC-POMDPs[1]. Because the number of possible histories is $|O|^{|I|\times|T|}$ and bounding the size of histories is one of our key contributions, we used benchmark problems with larger observation sets. Other well-known benchmark problems such as Multi-Agent Tiger (Nair *et al.* 2003), Recycling Robots (Amato *et al.* 2007) and Fire Fighting (Oliehoek *et al.* 2008) have only 2 observations.

[1]http://users.isr.ist.utl.pt/∼mtjspaan/decpomdp/index_en.html

The Broadcast Channel problem (Bernstein *et al.* 2005) is a simplified two agent networking problem. At each time step, each agent must choose whether or not to send a message. If both agents send messages, there is a collision and neither gets through. This problem has 4 states, 2 actions and 5 observations. The results in Table 2 show that in this problem all the methods achieved similar values with runtime less than 0.01. Both MAOP-COMM and Dec-Comm-PF initiated no communication. The performance without communication were almost the same as the case of full communication. These results have a simple intuitive explanation. The probability that agent's buffer will fill up on the next step is 0.9 for one agent and 0.1 for the other. Therefore, it is easy to coordinate in this case by simply giving one agent a higher priority.

In the Meeting in a Grid problem (Bernstein *et al.* 2005), two robot navigate on a grid with no obstacles. The goal is for the robots to spend as much time as possible in the same location. In order to make the problem more challenging, we used larger 3×3 grid and simulated a noisy sensor with a 0.9 chance to perceiving the right observation. This problem has 81 states, since each robot can be in any of 9 squares at any time. Each robot has 5 actions and 7 legal observations for sensing a combination of walls around. The results in Table 2 show that MAOP–the online algorithm without communication–performed surprisedly well in this case. The one-step lookahead provided a good heuristic for this problem because agents can meet anywhere and the problem resets after that. The results for FULL-COMM show that agents do not benefit much from communication. MAOP-COMM achieved a higher value than Dec-Comm-PF, but with much less communication. The runtimes of MAOP-COMM, MAOP and Dec-Comm-PF were short and quite close to each other.

In the Cooperative Box Pushing domain (Seuken & Zilberstein 2007), two agents located on a 3×4 grid are required to push boxes (two small and one large box) into a goal area. The agents benefit from cooperation because when they cooperatively push the large box into the goal area they get a very high reward. In order to make the problem more challenging, we have the agents transition to a random state when the problem resets itself. We also included uncertain observations in this domain with a 0.9 probability for the right observation and a 0.025 probability for the others. This domain has 100 states with 4 goal states and 96 non-goal states. Each agent has 4 actions and 5 observations. The results in Table 2 show that in this domain communication did improve performance significantly. MAOP without communication performed poorly. The one-step lookahead was no longer a good heuristic because agents in this domain have multiple goals (large box or small box). For this domain with longer horizons such as 50 and 100, MAOP-COMM outperformed Dec-Comm-PF, again with much less communication. MAOP and MAOP-COMM ran a little faster than Dec-Comm-PF.

To summarize, MAOP-COMM performed very well in all three benchmark problems using much less communication than Dec-Comm-PF. In some domains such as Broadcast Channel and Meeting in a Grid, MAOP could also achieve

Table 2: Experimental Results (20 trials)

H	ALG	Broadcast Channel			Meeting in a Grid			Box Pushing			Soccer 2×3			Soccer 3×3		
		R	T(s)	C(%)	R	T(s)	C(%)	R	T(s)	C(%)	R	T(s)	C(%)	R	T(s)	C(%)
10	MAOP-COMM	9.28	0.0	0.0%	1.7	0.19	8.0%	67.5	0.12	10.5%	139.0	0.13	14.5%	173.8	2.0	24.0%
	MAOP	9.16	0.0	0.0%	1.45	0.09	0.0%	31.3	0.10	0.0%	131.5	0.09	0.0%	103.3	1.7	0.0%
	FULL-COMM	9.3	0.0	100.0%	2.3	0.0	100.0%	109.25	0.0	100.0%	154.1	0.0	100.0%	178.9	0.02	100.0%
	Dec-Comm-PF	9.05	0.0	0.0%	1.5	0.22	64.5%	85.2	0.35	67.9%	131.8	1.23	30.5%	151.3	14.95	53.9%
20	MAOP-COMM	18.35	0.0	0.0%	3.35	0.26	11.0%	99.3	0.16	11.5%	290.6	0.28	14.8%	296.0	2.3	27.0%
	MAOP	18.25	0.0	0.0%	3.1	0.15	0.0%	7.5	0.14	0.0%	180.5	0.25	0.0%	190.7	1.9	0.0%
	FULL-COMM	18.9	0.0	100.0%	4.75	0.0	100.0%	222.5	0.0	100.0%	373.9	0.0	100.0%	356.0	0.02	100.0%
	Dec-Comm-PF	18.45	0.0	0.0%	2.9	0.22	70.5%	136.75	0.35	83.5%	129.5	1.36	33.5%	271.4	15.26	59.0%
50	MAOP-COMM	45.3	0.0	0.0%	8.4	0.30	11.8%	230.5	0.20	12.8%	946.5	0.14	16.4%	791.6	2.54	25.5%
	MAOP	45.2	0.0	0.0%	7.4	0.16	0.0%	-8.0	0.14	0.0%	656.0	0.12	0.0%	456.2	1.6	0.0%
	FULL-COMM	45.36	0.0	100.0%	12.8	0.0	100.0%	441.5	0.0	100.0%	949.1	0.0	100.0%	862.4	0.02	100.0%
	Dec-Comm-PF	44.75	0.0	0.0%	6.85	0.22	77.5%	233.25	0.35	71.0%	834.2	1.25	35.3%	490.1	15.01	65.4%
100	MAOP-COMM	90.35	0.0	0.0%	17.1	0.30	12.1%	441.95	0.13	12.26%	1933.9	0.16	15.4%	1679.5	2.50	26.9%
	MAOP	89.95	0.0	0.0%	15.3	0.19	0.0%	-16.0	0.13	0.0%	1157.8	0.14	0.0%	803.6	1.96	0.0%
	FULL-COMM	90.6	0.0	100.0%	24.7	0.0	100.0%	880.50	0.0	100.0%	1933.6	0.0	100.0%	1808.2	0.02	100.0%
	Dec-Comm-PF	90.0	0.0	0.0%	14.9	0.24	78.2%	296.50	0.36	59.87%	1441.6	1.28	30.8%	1044.4	9.48	70.7%

very high value without any communication. Although the tested horizon was only up to 100, our approach can solve problems with an arbitrary horizon since we bound the size of histories at each step. The experimental results varied a little with the horizon because in problems with longer horizons there is a greater chance for miscoordination, miscommunication and error accumulation. The parameter ϵ presents a good way to tradeoff between the amount of communication and overall value. In domains such as Cooperative Box Pushing, a larger ϵ would allow more communication and consequently improve performance.

Grid Soccer To demonstrate scalability, we also tested our algorithm on a more challenging problem called grid soccer. The domain includes two agents for one team and one opponent for the other team, as shown in Figure 3. Each agent has 4 possible orientations (up, down, left or right). The opponent–with full observation and reliable actions–always executes a fix policy and tries to get close to the ball as fast as possible. If the opponent bumps into an agent with the ball, it will get the ball and the game terminates with a reward of -50. If the agent with the ball enters the goal grid, the game also terminates with a reward of 100. Each agent has 6 actions: north, south, east, west, stay and pass. Each action has a 0.9 probability of success and 0.1 probability of having no impact on the current state. When an agent executes a pass action, the ball is transferred to the other agent on the next step, if and only if the other agent executes the stay action at the same time. Otherwise, the ball goes out of the field and the game terminates with a reward of -20. For each step resulting in a non-terminal state, there is a penalty of 2. After reaching a terminal state the problem is reset. Each agent gets one out of 5 possible observations describing the situation in front of it (free, wall, teammate, opponent, goal) and 2 observations indicating who controls the ball. Thus, the total number of observations is 11. The observation is noisy with a 0.9 chance to perceive the correct observation and a 0.01 chance to perceive each of the other observations. We tested our algorithm on two grid soccer problems: one is a 2×3 grid with 3843 states, and the other is a 3×3 grid with 16,131 states. These problems are the largest tackled so far by decision-theoretic algorithms for multi-agent planning.

2×3 3×3

Figure 3: Grid Soccer

The results are shown in Table 2. MAOP-COMM achieved higher value than Dec-Comm-PF, while the performance of MAOP is competitive, indicating that MAOP-COMM with a smaller ϵ would use even less communication and produce good value. The runtime of MAOP-COMM and MAOP were almost ten times faster than Dec-Comm-PF. One reason is that the operators in MAOP-COMM and MAOP are much cheaper than the particle filtering used in Dec-Comm-PF. Another reason is that the number of histories kept by MAOP-COMM and MAOP is much smaller than the number of particles used by Dec-Comm-PF. MAOP-COMM and MAOP scaled well in the 3×3 instance. The most state-sensitive operator was the Bayesian update. For a problem with 16,131 states, the update loop takes hundreds of seconds. Fortunately, the transition functions are sparse in many real applications including grid soccer, allowing us to optimize the Bayesian update in all the algorithms *including* Dec-Comm-PF. Incidentally, in problems with larger state spaces, the runtime advantage of keeping less histories became more significant.

Conclusion

We present a new online algorithm for planning under uncertainty in multi-agent settings with constrained communication. The algorithm addresses the key challenge of keeping the team of agents coordinated by maintaining a shared pool of histories that allows agents to choose local actions and detect inconsistency when it arises. The algorithm has

several important advantages. First, it can use communication very selectively to recover from inconsistency. It can also delay communication when the resource is not available and–if needed–avoid communication for a long period of time. A second advantage is scalability. The algorithm can solve existing benchmark problems much faster than the best off-line algorithms and it can solve larger problems that are beyond the scope of off-line DEC-POMDP planners. Finally, the algorithm performs very well in practice, outperforming the best existing online method by producing better value with less communication.

One benefit of the algorithm is that the amount of communication can be easily controlled using the communication threshold parameter. In fact, we could have easily obtained better results had we adjusted ϵ for each of the tested domains. We deliberately experimented with a fixed ϵ to show that there is no need for extensive tuning to get good results with MAOP-COMM. But in future work, we plan to analyze this question and develop a disciplined approach to set up the communication threshold, particularly when the cost of communication is given.

Acknowledgments

Special thanks to Maayan Roth for sharing her source code of Dec-Comm and to the reviewers for their helpful comments. This work was supported in part by the China Scholarship Council, the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0181, and the National Science Foundation under Grant No. IIS-0812149.

References

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2007. Optimizing Memory-Bounded Controllers for Decentralized POMDPs. In *Proc. of the 23rd Conf. on Uncertainty in Artificial Intelligence*.

Becker, R.; Lesser, V. R.; and Zilberstein, S. 2005. Analyzing Myopic Approaches for Multi-Agent Communication. In *Proc. of the 2005 IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, 550–557. IEEE.

Bernstein, D. S.; Zilberstein, S.; and Immerman, N. 2000. The Complexity of Decentralized Control of Markov Decision Processes. In *Proc. of the 6th 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. Joint Conf. on Artificial Intelligence*, 1287–1292.

Emery-Montemerlo, R.; Gordon, G. J.; Schneider, J. G.; and Thrun, S. 2005. Game Theoretic Control for Robot Teams. In *Proc. of the 2005 IEEE Int. Conf. on Robotics and Automation*, 1163–1169. IEEE.

Emery-Montemerlo, R. 2005. *Game-Theoretic Control for Robot Teams.* Doctoral Dissertation, Robotics Institute, Carnegie Mellon University.

Goldman, C. V., and Zilberstein, S. 2003. Optimizing information exchange in cooperative multi-agent systems. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, 137–144. ACM.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning policies for partially observable environments: Scaling up. In *Proc. of the 12th Int. Conf. on Machine Learning*, 362–370.

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. Joint Conf. on Artificial Intelligence*, 705–711.

Nair, R.; Tambe, M.; Roth, M.; and Yokoo, M. 2004. Communication for Improving Policy Computation in Distributed POMDPs. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 1098–1105. IEEE.

Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. 2007. Dec-POMDPs with delayed communication. In *The 2nd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*.

Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. 2008. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research* 32:289–353.

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.

Roth, M.; Simmons, R. G.; and Veloso, M. M. 2005. Reasoning about joint beliefs for execution-time communication decisions. In *Proc. of the 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 786–793. ACM.

Roth, M.; Simmons, R.; and Veloso, M. 2006. What to communicate? execution-time decision in multi-agent POMDPs. In *Proc. of the 8th Int. Symposium on Distributed Autonomous Robotic Systems*.

Roth, M.; Simmons, R. G.; and Veloso, M. M. 2007. Exploiting factored representations for decentralized execution in multiagent teams. In *Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 72.

Seuken, S., and Zilberstein, S. 2007. Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. In *Proc. of the 23rd Conf. in Uncertainty in Artificial Intelligence*.

Seuken, S., and Zilberstein, S. 2008. Formal Models and Algorithms for Decentralized Decision Making under Uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems* 17(2):190–250.

Spaan, M. T. J.; Gordon, G. J.; and Vlassis, N. 2006. Decentralized planning under uncertainty for teams of communicating agents. In *Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 249–256. ACM Press.

Spaan, M. T. J.; Oliehoek, F. A.; and Vlassis, N. 2008. Multiagent Planning under Uncertainty with Stochastic Communication Delays. In *Proc. of the 18th Int. Conf. on Automated Planning and Scheduling*, 338–345.

Williamson, S. A.; Gerding, E. H.; and Jennings, N. R. 2008. A principled information valuation for communication during multi-agent coordination. In *The 3rd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*.

Xuan, P.; Lesser, V.; and Zilberstein, S. 2001. Communication decisions in multi-agent cooperation: Model and experiments. In *Proc. of the 5th Int. Conf. on Autonomous Agents*, 616–623.

Appendix

Proof of Theorem 1: At step 0, if the optimal policies for 0 up to T are given, agents can select the optimal joint policy by b^0. At step t, assume agent i merges two histories h_i^t and $h_i'^t$ because they share the same optimal policy q_i^t. At any future step $t+k$, for any k step history h_i^k, history $h_i^t \cup h_i^k$ and $h_i'^t \cup h_i^k$ must still share the same optimal policy q_i^{t+k}. If not, the optimal policies for h_i^t and $h_i'^t$ are different because they have different sub-trees, contradicting the assumption that h_i^t and $h_i'^t$ have the same optimal policy. Due to the assumption of optimality, q_i^{t+k} must be a sub-policy tree of q_i^t. At step $t+k$, for any given k, agent i can still find the optimal policy after merging h_i^t and $h_i'^t$. The theorem thus holds for all steps by induction. □