Artificial Intelligence 175 (2011) 487–511

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

# Online planning for multi-agent systems with bounded communication

Feng Wu [a,b,*], Shlomo Zilberstein [b], Xiaoping Chen [a]

[a] School of Computer Science, University of Science and Technology of China, Jinzhai Road 96, Hefei, Anhui 230026, China
[b] Department of Computer Science, University of Massachusetts at Amherst, 140 Governors Drive, Amherst, MA 01003, USA

ARTICLE INFO

*Article history:*
Received 2 February 2010
Received in revised form 22 September 2010
Accepted 26 September 2010
Available online 29 September 2010

*Keywords:*
Decentralized POMDPs
Cooperation and collaboration
Planning under uncertainty
Communication in multi-agent systems

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 offline. The key challenges in decentralized operation are to maintain coordinated behavior with little or no communication and, when communication is allowed, to optimize value with minimal communication. The algorithm addresses these challenges by generating identical conditional plans based on common knowledge and communicating only when history inconsistency is detected, allowing communication to be postponed when necessary. To be suitable for online operation, the algorithm computes good local policies using a new and fast local search method implemented using linear programming. Moreover, it bounds the amount of memory used 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 offline planning algorithms and it outperforms the best online method, producing much higher value with much less communication in most cases. The algorithm also proves to be effective when the communication channel is imperfect (periodically unavailable). These results contribute to the scalability of decision-theoretic planning in multi-agent settings.

© 2010 Elsevier B.V. All rights reserved.

## 1. Introduction

A multi-agent system (MAS) consists of multiple independent agents that interact in a domain. Each agent is a decision maker that is situated in the environment and acts autonomously, based on its own observations and domain knowledge, to accomplish a certain goal. A multi-agent system design can be beneficial in many AI domains, particularly when a system is composed of multiple entities that are distributed functionally or spatially. Examples include multiple mobile robots (such as space exploration rovers) or sensor networks (such as weather tracking radars). Collaboration enables the different agents to work more efficiently and to complete activities they are not able to accomplish individually. Even in domains in which agents can be centrally controlled, a MAS can improve performance, robustness and scalability by selecting actions in parallel. In principle, the agents in a MAS can have different, even conflicting, goals. We are interested in fully-cooperative MAS, in which all the agents share a common goal.

In a cooperative setting, each agent selects actions individually, but it is the resulting joint action that produces the outcome. Coordination is therefore a key aspect in such systems. The goal of coordination is to ensure that the individual decisions of the agents result in (near-)optimal decisions for the group as a whole. This is extremely challenging especially

* Corresponding author at: Department of Computer Science, University of Massachusetts at Amherst, 140 Governors Drive, Amherst, MA 01003, USA. Tel.: +1 413 545 1985; fax: +1 413 545 1249.
*E-mail addresses:* wufeng@mail.ustc.edu.cn (F. Wu), shlomo@cs.umass.edu (S. Zilberstein), xpchen@ustc.edu.cn (X. Chen).

0004-3702/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2010.09.008

when the agents operate under high-level uncertainty. For example, in the domain of 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 their information all the time is not possible. Besides, agents in such domains may need to perform a long sequence of actions in order to reach the goal.

Different mathematical models exist to specify sequential decision-making problems. Among them, decision-theoretic models for planning under uncertainty have been studied extensively in artificial intelligence and operations research since the 1950's. Decision-theoretic planning problems can be formalized as Markov decision processes (MDPs), in which a single agent repeatedly interacts with a stochastically changing environment and tries to optimize a performance measure based on rewards or costs. Partially-observable Markov decision processes (POMDPs) extend the MDP model to handle sensor uncertainty by incorporating observations and a probabilistic model of their occurrence. In a MAS, however, each individual agent may have different partial information about the other agents and about the state of the world. Over the last decade, different formal models for this problem have been proposed. We adopt decentralized partially-observable Markov decision processes (DEC-POMDPs) to model a team of cooperative agents that interact within a stochastic, partially-observable environment.

It has been proved that decentralized control of multiple agents is significantly harder than single agent control and provably intractable. In particular, the complexity of solving a two-agent finite-horizon DEC-POMDP is NEXP-complete [12]. In the last few years, several promising approximation techniques have been developed [3,11,17,19,46,47]. The vast majority of these algorithms work offline and compute, prior to the execution, the best action to execute for all possible situations. While these offline algorithms can achieve very good performance, they often take a very long time due to the double exponential policy space that they explore. For example, PBIP-IPG – the state-of-the-art MBDP-based offline algorithm – takes 3.85 hours to solve a small problem such as Meeting in a $3\times3$ grid that involves 81 states, 5 actions and 9 observations [3]. Online algorithms, on the other hand, plan only one step at a time and they do so given all the currently available information. The potential for achieving good scalability is more promising with online algorithms. But it is extremely challenging to keep agents coordinated over a long period of time with no offline planning. Recent developments in online algorithms suggest that combining online techniques with selective communication – when communication is possible – may be the most efficient way to tackle large DEC-POMDP problems. The main goal of this paper is to present, analyze, and evaluate online methods with bounded communication, and show that they present an attractive alternative to offline techniques for solving large DEC-POMDPs.

The main contributions of this paper include: (1) a fast method for searching policies online, (2) an innovative way for agents to remain coordinated by maintaining a shared pool of histories, (3) an efficient way for bounding the number of possible histories agents need to consider, and (4) a new communication strategy that can cope with bounded or unreliable communication channels. In the presence of multiple agents, each agent must cope with limited knowledge about the environment and the other agents, and must reason about all the possible beliefs of the other agents and how that affects their decisions. Therefore, there are still many possible situations to consider even for selecting just one action given the current knowledge. We present a new linear program formulation to search the space of policies very quickly. Another challenge is that the number of possible histories (situations) grows very rapidly over time steps, and agents could run out of memory very quickly. We introduce a new approach to merging histories and thus bound the size of the pool of histories, while preserving solution quality. Finally, it is known that appropriate amounts of communication can improve the tractability and performance of multi-agent systems. When communication is bounded, which is true in many real-world applications, it is difficult to decide how to utilize the limited communication resource efficiently. In our work, agents communicate when history *inconsistency* is detected. This presents a new effective way to initiate communication dynamically at runtime.

The rest of the paper is organized as follows. In Section 2, we provide the background by introducing the formal model and discussing the offline and online algorithms as well as the communication methods in the framework of decentralized POMDPs. In Section 3, we present the multi-agent online planning with communication algorithms including the general framework, policy search, history merging, communication strategy and implementation issues. In Section 4, we report the experimental results on several common benchmark problems and a more challenging problem named grid soccer. We also report the results for the cooperative box pushing domain with imperfect communication settings. In Section 5, we survey the various existing online approaches with communications that have been applied to decentralized POMDPs, and discuss their strengths and drawbacks. Finally, we summarize the contributions and discuss the limitations and open questions in this work.

## 2. Background

In this section we provide a formal description of the problem and some essential background. We consider settings in which a group of agents coordinate with each other at discrete time steps. The agents operate over some finite number of steps, $T$, referred to as the horizon. At each time step, each agent first receives its own local observation from the environment and then takes an action. The combination of all agents' actions causes a stochastic change in the state of the environment, and produces some joint reward, after which a new decision cycle starts. In such cooperative sequential

decision-making settings, the agents have to come up with a plan that maximizes the expected long term reward of the team. Planning can take place either offline or online. In offline planning, the computed plans are distributed to each agent and executed by each agent based on its local information. Hence the planning phase can be centralized as long as the execution is decentralized. Online planning algorithms often interleave planning with execution. Thus they only need to find actions for the current step, instead of generating the whole plan at once as offline algorithms do. Agents can usually share information with each other by communication. But communication is not a free resource in many applications. We will discuss the communication model in detail later in this section.

### 2.1. Formal model of decentralized POMDPs

The Markov Decision Process (MDP) and its partially observable counterpart (POMDP) have proved very useful for planning and learning under uncertainty. The Decentralized POMDP offers a natural extension of these frameworks for cooperative multi-agent settings. We adopt the DEC-POMDP framework to model multi-agent systems, however our approach and results apply to equivalent models such as MTDP [41] and POIPSG [40].

**Definition 1** *(DEC-POMDP).* A 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 $1, \ldots, n$. Notice that when $n = 1$, a DEC-POMDP is equivalent to a single-agent POMDP.
- $S$ is a finite set of system states. A state summarizes all the relevant features of the dynamical system and satisfies the Markov property. That is, the probability of the next state depends only on the current state and the joint action, not on the previous states and joint actions: $P(s^{t+1}|s^0, \vec{a}^0, \ldots, s^{t-1}, \vec{a}^{t-1}, s^t, \vec{a}^t) = P(s^{t+1}|s^t, \vec{a}^t)$.
- $A_i$ is a finite set of actions available to agent $i$ and $\vec{A} = \bigtimes_{i\in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, \ldots, a_n\rangle$ denotes a joint action. We assume that agents do not observe which actions are taken by others at each time step.
- $\Omega_i$ is a finite set of observations available to agent $i$ and $\vec{\Omega} = \bigtimes_{i\in I} \Omega_i$ is the set of joint observations, where $\vec{o} = \langle o_1, \ldots, o_n\rangle$ denotes a joint observation. Every time step the environment emits one joint observation, but each agent only observes its own component.
- $P$ is a 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 $s'$. $P$ describes the stochastic influence of actions on the environment. We assume that the transition probabilities are stationary, which means that they are independent of the time step.
- $O$ 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 $s'$. $O$ describes how the agents perceive the state of the environment. The observation probability is also assumed to be stationary.
- $R : S \times \vec{A} \to \Re$ is a reward function. $R(s, \vec{a})$ denotes the reward obtained from taking joint action $\vec{a}$ in state $s$. $R$ specifies the agents' goal or task. It is an immediate reward for agents taking a joint action in some state. The agents do not generally observe the immediate reward at each time step, although their local observation may determine that information.
- $b^0 \in \Delta(S)$ is the initial belief state distribution. It is a vector that specifies a discrete probability distribution over $S$ that captures the agents' common knowledge about the starting state.

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 $t$,

$$h_i^t = \left(a_i^0, o_i^1, a_i^1, \ldots, o_i^{t-1}, a_i^{t-1}, o_i^t\right)$$

is the history of agent $i$, and $h^t = \langle h_1^t, \ldots, h_n^t\rangle$ is the joint history. The term joint belief $b(\cdot|h) \in \Delta(S)$ 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:

$$\forall_{s'\in S}, b^t\left(s'|h^t\right) = \frac{O(\vec{o}^t|s', \vec{a}^{t-1}) \sum_{s\in S} P(s'|s, \vec{a}^{t-1}) b^{t-1}(s|h^{t-1})}{\sum_{s''\in S} O(\vec{o}^t|s'', \vec{a}^{t-1}) \sum_{s\in S} P(s''|s, \vec{a}^{t-1}) b^{t-1}(s|h^{t-1})} \tag{1}$$

Throughout this paper, we use $b(h)$ as a shorthand of the joint belief $b(\cdot|h)$.

A local deterministic policy $\delta_i$ 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. $\delta(h) = \langle\delta_1(h_1), \ldots, \delta_n(h_n)\rangle = \vec{a}$. A deterministic policy can be represented as a policy tree with nodes representing actions and edges labeled with observations. A joint deterministic policy is a set 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 $A_i$. A joint stochastic policy, $\pi = \langle\pi_1, \ldots, \pi_n\rangle$, is tuple of local stochastic policies.

Solving a DEC-POMDP for a given horizon $T$ and start state $s^0$ can be seen as finding a policy $\delta$ that maximizes the expected cumulative reward

$$V(\delta, s^0) = E\left[\sum_{t=0}^{T-1} R(s^t, \vec{a}^t)|s^0\right] \tag{2}$$

Because of the recursive nature of DEC-POMDPs, it is more intuitive to specify the value function recursively:

$$V(\delta, h^t) = \sum_{s^t} p(s^t|h^t)\left[R(s^t, \vec{a}) + \sum_{s^{t+1},\vec{o}} P(s^{t+1}|s^t, \vec{a}) O(\vec{o}|s^{t+1}, \vec{a}) V(\delta, h^{t+1})\right] \tag{3}$$

where $\vec{a} = \delta(h^t), h^{t+1} = h^t \circ \vec{a} \circ \vec{o}$ and the state distribution $p(s^t|h^t)$ given history $h^t$ is computed recursively as follow:

$$p(s^t|h^t) = O(\vec{o}|s^t, \vec{a}) \sum_{s^{t-1}} P(s^t|s^{t-1}, \vec{a}) p(\vec{a}|h^{t-1}) p(s^{t-1}|h^{t-1}) \tag{4}$$

where $h^t = h^{t-1} \circ \vec{a} \circ \vec{o}$ and $p(s|h^0) = b^0(s), \forall s \in S$. A survey of DEC-POMDP models and algorithms is available in [48].

Previous studies have identified different categories of DEC-POMDPs characterized by different levels of observability and interaction. The computational complexity of solving these problems ranges between NEXP and P [26]. When each individual observation of an agent identifies the true state uniquely, a DEC-POMDP reduces to a multi-agent MDP (MMDP) [15]. When the joint observation identifies the true state, a DEC-POMDP is referred to as a DEC-MDP. A DEC-MDP is called transition-independent DEC-MDP (TI-DEC-MDP) when the transition models of the agents are independent of each other [9]. Other special cases that have been considered are, for instance, goal-oriented DEC-POMDPs [26], event-driven DEC-MDPs [8], network distributed POMDPs (ND-POMDPs) [33], DEC-MDPs with time and resource constraints [13,14,29], DEC-MDPs with local interactions [51] and factored DEC-POMDPs with additive rewards [36].

In this work we consider the general finite-horizon DEC-POMDPs, without any simplifying assumptions on the observations, transitions, or reward functions. The complexity of general finite-horizon DEC-POMDPs has been shown to be NEXP-complete. Due to these complexity results, optimal algorithms have mostly theoretical significance. Current research efforts in this area focus mostly on finding scalable approximation techniques [3,11,17,19,46,47].

### 2.2. Offline algorithms versus online algorithms

Developing algorithms for solving DEC-POMDPs approximately has become a thriving research area. Most existing algorithms operate offline, generating some type of a complete policy before execution begins. The policy specifies what action to take in any possible runtime situation. While good performance can be achieved using these algorithms, they often take significant time (e.g. more than a day) to solve modest problems. The reason is that they need to consider all possible policies of the other agents – or a sufficiently large set of policies – in order to preserve solution quality. In domains such as robot soccer, it is not feasible to consider all possible strategies all through to the end. Besides, small changes in the environment's dynamics require recomputing the full policy. In contrast, online algorithms only need to plan the current action and thus can be much faster. This has long been recognized in competitive game playing such as chess. A typical algorithm often performs a limited amount of lookahead and plans only for the current step, and then repeats this process online after observing the opponent's response. Furthermore, online planning can better handle emergencies and unforeseen situations, which allows online approaches to be applicable in many domains for which offline approaches are not adequate.

Implementing online algorithms for decentralized multi-agent systems is very challenging. Since the underlying system state as well as the observations of the other agents are not available during execution time, 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. In cooperative multi-agent domains, agents must ensure coordination. Consider, for example, a robot soccer problem in which two defenders ($D_1$ and $D_2$) are trying to mark two attackers ($A_1$ and $A_2$). Each defender has different observations of the environment and thus may compute a different best joint plan online based on its own local knowledge. For example, the best joint plan based on $D_1$'s knowledge is $\langle D_1 \to A_1, D_2 \to A_2\rangle$ and the best joint plan based on $D_2$'s knowledge is $\langle D_1 \to A_2, D_2 \to A_1\rangle$, where $D_i \to A_j$ denotes defender $D_i$ marking attacker $A_j$. However, the actual joint action executed is $\langle D_1 \to A_1, D_2 \to A_1\rangle$, which is an example of miscoordination. The outcome of miscoordination could be arbitrarily bad and cause severe failures in teamwork [57]. In the example mentioned above, miscoordination leads to undesired behavior, namely two defenders mark $A_1$ while $A_2$ is left unmarked. In practice, if each agent computes policies with different private information, there is a risk that the resulting policies will fail to achieve the intended effects. On the other hand, if agents ignore their private information, their policies become open-loop controllers, without considering their local observations. Thus, we define coordination as follows.

**Definition 2** *(coordination).* When agents maintain a single shared plan (joint policy) and always execute an action that is part of that plan (policy), we say that the MAS exhibits *coordination*. Otherwise, it exhibits *miscoordination*.

It is very difficult to guarantee coordination when planning is performed online and agents have different local information. Another major difficulty is that online algorithms must often meet real-time constraints, greatly reducing the available planning time, compared with offline methods. Due to these difficulties, work on online planning for DEC-POMDPs has been sparse. In this paper, we address these key challenges of multi-agent online planning and answer the following questions: (1) How to guarantee coordination when agents operate based on local observations, each having different private information? (2) How to meet planning time constraints and bound memory usage when each agent must reason about a vast number of possible outcomes and action choices of other agents?

### 2.3. Communication in decentralized POMDPs

Agents with limited source of information about their teammates must reason about the possible histories of team members and how these histories affect their own behaviors. Communication can alleviate this problem by sharing private information such as sensory data. Hence, online algorithms often incorporate communication to improve performance. 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 resend messages several times until the critical information is fully received. In all these situations, too much communication will affect negatively the performance of the system. Thus, an interesting and challenging question is how to integrate online planning with communication and use communication effectively.

Bounded communication is recognized as an important characteristic of multi-agent systems due to several factors: (1) Agents may move into regions that have no coverage of the communication signal. For example, search and rescue robots working in subterranean tunnels such as subways and mine caves may have no wireless communications because high frequency waves cannot penetrate rock, limiting radio communication to areas with line of sight between the transceivers [30]. Another example is two Mars rovers with one rover located behind some blocking obstacle [6]. In both examples, robots must move to certain spots where the wireless connection is available and then try to communicate with each other. Obviously, communication is costly if the working sites are far from the communication spots. Agents must bound the frequency of communication and maintain coordination when communication is unavailable. (2) Agents may not be able to share all the information with other agents all the time due to the limitations of the communication channel and computational device. Consider, for example, the AIBO soccer robot [42]. Communication latency of the wireless network is high. On average, messages takes 0.5 seconds to be received by all teammate robots, and in some cases, latency is observed to be as high as 5 seconds. This latency makes it difficult for the robots to communicate all the most recent information all the time. The robots also have limited computational power. They receive and process images at 20 frames per second. Because each image may contain only a few (or none) of the relevant state features, it takes time to build a world model and compute what should be shared with others. Besides, receiving messages from others at 20 Hz overloads the robot operating system's message buffers. After a few minutes of attempting to communicate their private information as they are updated, the robots' motion controllers are affected, causing the robots to slow down and occasionally crash. (3) Another important factor is communication failures, which are very common in wireless sensor networks [1]. Failures may require multiple attempts to communicate before the information is transmitted successfully. In all the scenarios mentioned above, bounded communication is preferable. Additionally, it is known that free communication reduces a DEC-POMDP to a large single agent POMDP. This is done by having each agent broadcast its local observation to the other agents at each time step. When all of the local observations are known to every agent, they can be treated as a single joint observation, giving the system the same complexity as a single-agent POMDP, PSPACE-complete. When communication is not free, finding the optimal communication policy is as hard as the general DEC-POMDP, which is NEXP-complete [41]. In this work, we adopt bounded communication which is a more suitable assumption for the domains we are interested in.

There are three possible ways in which agents can share information and coordinate their actions: indirect communication, direct communication, and using common uncontrollable observed features. In indirect communication, one agent's actions can affect the observations made by another agent. Hence these observations can serve as messages transmitted between the agents. Generally, when the observations of the agents are dependent on non-local information, each observation provides some form of indirect communication. Thus a general DEC-POMDP already includes this form of indirect communication, and the policy determines what to communicate and when. With direct communication, information can be shared by the agents by sending messages directly to each other. When there are components of the global system state that are observed by both agents but are not affected by any of these agents' actions, the agents can then act upon the common knowledge and coordinate their actions without exchanging messages directly.

In this work we consider direct communication and use the *sync* communication model [61] where each agent *broadcasts* its local information to all the others. The communication language simply allows transmission of the agents' action-observation histories. The *tell* and *query* models [61] are more complex, allowing one way communication between one pair of agents. In designing a general communication strategy, one must determine when to communicate, with whom to communicate, and what to send. However, when the *sync* model is used, the main question is when to initiate communication. Once communication is initiated by any agent, all the agents share their local information. Communication is generally

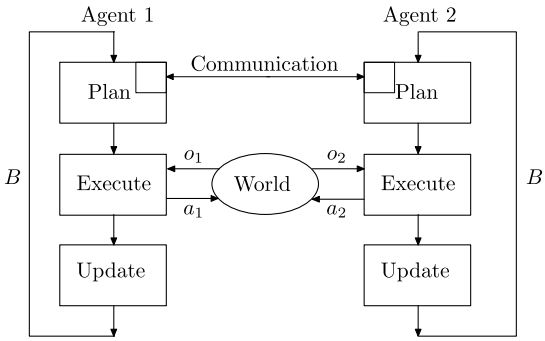


**Fig. 1.** Online planning framework for two agents.

assumed to be instantaneous – a message is received without delay as soon as it is sent. But in this work we consider the impact of possible stochastic delays.

The *sync* communication model is essential to simplify the already high complexity of planning with different sets of partial information. Nevertheless, this does not necessarily mean that agents must transmit all their local information to the other agents. Agents need only transmit the information which is relevant to establishing coordination. For example, in the multi-access broadcast channel problem [11], the actual information communicated may be only several bytes to indicate the status of the buffer, but the messages in the buffer themselves may be several mega-bytes or giga-bytes large. Similarly, it is common for a file sharing system to broadcast the file names in a directory to all the peers instead of transmitting all the files. In DEC-POMDPs, the messages exchanged among agents are sequences of high-level observations, not the sensor readings that are the basis for each observation. Additionally, it is possible to incorporate selective communication methods [43] within our model. Generally, this can be done by choosing observations with larger impact on the beliefs – an interesting direction that is left for future work. In practice, the usefulness of selective communication often depends on how the observation model is defined. Even with selective communication, the question of when to communicate remains open. Another challenge simplified by the *sync* model is the choice of whom to communicate with. This is particularly relevant when each agent interacts directly with a subset of the group, rather than the entire team. In this paper, we focus on general DEC-POMDP settings without assuming any special interaction structure. The domains we investigate require coordination among all the agents, which means that the action of one agent affects the optimal choices of all the others. The *sync* model is thus suitable for these settings, and the question of when to communicate that it presents is still very challenging.

Additionally, we use *and-communication* [20], assuming separate communication and action phases in each time step. Thus communication facilitates better domain-level action selection by conditioning domain actions on the specific information received, rather than replacing domain-level actions as in *or-communication* [20]. We did not factor an explicit cost for communication in this work because any such cost would have been arbitrary and not particularly relevant to the applications we are interested in. In the experimental results, we show that our approach is effective in the sense that it achieves better value with less communication compared with existing online techniques with communication. This guarantees that, if the cost of communication is specified, our approach will be beneficial for *any* such cost.

## 3. Multi-agent online planning with communication

In this section we introduce a new algorithm, Multi-Agent Online Planning with Communication (MAOP-COMM), for finding approximate solutions of general DEC-POMDPs. This algorithm is executed in parallel by all the agents in the team, interleaving planning and execution. More precisely, our online algorithm is divided into a planning phase, an executing phase and an updating phase, which are applied consecutively at each time step. An example involving two agents is shown in Fig. 1.

**Definition 3** *(belief pool).* A belief pool at time-step $t$ is defined by a tuple $\langle \{H_i^t | i \in I\}, B^t \rangle$, where $H_i^t$ is a set of histories for agent $i$ and $B^t$ is a set of joint belief states, $B^t = \{b(h^t) | h^t \in H^t\}$ where $H^t = \bigtimes_{i \in I} H_i^t$.

To illustrate this concept, consider the robot soccer problem mentioned in Section 2.2. Suppose that the defense strategy is that each defender should mark the nearest attacker. Then, the belief pool will contain knowledge about who is the nearest defender for each attacker. As long as all the defenders maintain the same belief pool and use the same tie-breaking rule, the outcome plans they compute (determining which defender marks each attacker) will be the same. This guarantees that the strategies of the defenders are coordinated.

**Algorithm 1**: Expand Histories and Update Beliefs

**Input**: $H^t, B^t, \delta^t$
$H^{t+1} \leftarrow \emptyset$; $B^{t+1} \leftarrow \emptyset$
**for** $\forall h^t \in H^t, \forall \vec{o} \in \vec{\Omega}$ **do**
  $\vec{a} \leftarrow \delta^t(h^t)$
  // append $\vec{a}, \vec{o}$ to the end of $h^t$.
  $h^{t+1} \leftarrow h^t \circ \vec{a} \circ \vec{o}$
  // calculate the distribution of $h^{t+1}$.
  $p(h^{t+1}) \leftarrow p(\vec{o}, \vec{a}|h^t)p(h^t)$
  // test if $h^{t+1}$ is a reachable joint history.
  **if** $p(h^{t+1}) > 0$ **then**
    $H^{t+1} \leftarrow H^{t+1} \cup \{h^{t+1}\}$
    // compute the belief state of $h^{t+1}$.
    $b^{t+1}(\cdot|h^{t+1}) \leftarrow$ Update belief with $b^t(\cdot|h^t), \vec{a}, \vec{o}$
    // add $b^{t+1}$ into a hash table indexed by $h^{t+1}$.
    $B^{t+1} \leftarrow B^{t+1} \cup \{b^{t+1}(h^{t+1})\}$
**return** $H^{t+1}, B^{t+1}$

**Definition 4** *(local and joint policies).* A local policy for agent $i$ is a mapping from a set of histories to a set of actions, $\delta_i : H_i \to A_i$, and $\delta_i(h_i)$ denotes the action assigned to history $h_i$. A joint policy is a set of local policies, $\delta = \langle \delta_1, \delta_2, \ldots, \delta_n \rangle$, one for each agent, and $\delta(h)$ denotes the joint action assigned to joint history $h$.

In the planning phase, each agent computes a joint policy $\delta^t$ for every possible history in the belief pool. During the executing phase, agent $i$ adds its new observation to its own local history, $h_i^t \leftarrow h_i^{t-1} \circ o_i^t$, executes an action $a_i = \delta_i^t(h_i^t)$ according to its component in the joint policy $\delta^t$ and its current local history $h_i^t$, and appends the action to the end of the local history, $h_i^t \leftarrow h_i^t \circ a_i$. After that, each agent updates its own belief pool based on the plan $\delta^t$, as shown in Algorithm 1, and continues to the next step.

As mentioned, coordination is an important issue in multi-agent planning since the outcome of uncoordinated policies can be arbitrarily bad. In offline planning, the coordination is guaranteed by distributing and executing the same pre-computed joint policy. It is much more difficult to achieve coordination in online planning because the policy is computed online and each agent receives different (or partial) information from the environment.

In our online algorithm, 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 ensures 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 belief pool and randomization scheme, each agent can generate exactly the same joint policy. Each agent's local observation is used only for policy execution.

**Definition 5** *(belief inconsistency).* When an agent believes that $p$ must be true and the agent's observation implies $\neg p$, we say that the belief of that agent is inconsistent.

Intuitively, belief inconsistency occurs when an agent's beliefs contradict the observations it obtains from the environment. For example, in the robot soccer problem mentioned in Section 2.2, suppose that $D_1$ observes that its nearest attacker is $A_2$, not $A_1$ as indicated by its (inconsistent) belief. Then $D_1$ should communicate with all the other defenders, informing them that the nearest attacker to $D_1$ is actually $A_2$ so that they can update their belief pools with more accurate information. Hence, the team benefits from communication by finding better plans based on consistent beliefs.

This is why communication is triggered when our algorithm detects belief inconsistency. The agent then initiates communication as soon as the communication resource is available. When communication occurs, each agent broadcasts its own local observation sequence to the other agents (the *sync* model). Consequently, each agent can construct the actual joint history and calculate the actual joint belief state. The best joint action is then selected based on the new joint belief state. And the belief pool is emptied and replaced with the actual history.

Notice that communication is not essential to ensure coordination in our framework, but it offers an optional mechanism to improve performance. This makes the use of communication more flexible, particularly in domains with a bounded communication resource. Communication can be easily integrated into our framework by considering it before the planning phase. Coordination is guaranteed in any case because communication just updates the common knowledge of the agents. More precisely, the belief pools of the agents are the same without communication, and they remain the same after communication occurs. The *sync* model used by our work guarantees that each agent has the same knowledge after communication.

**Algorithm 2**: Multi-Agent Online Planning with Communication

**Input**: $b^0$, $seed[1..T-1]$
**foreach** $i \in I$ (***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 $\pi^t$
    $h_i^t \leftarrow$ Update agent $i$'s own local history with $a_i^t$
    Execute the action $a_i^t$

**Theorem 1.** *The multi-agent online planning algorithm* (*MAOP-COMM*) *always guarantees coordination among agents with or without communication.*

**Proof (sketch).** We present several mechanisms to ensure that each agent maintains the same belief pool and finds the same joint policy for the team, so they can still coordinate with each other online. As shown in Algorithm 1, only common knowledge arising from the model is used to update the belief pool. Agents run the algorithm in lockstep and a set of predetermined random seeds are used to ensure that all the agents come up with the same randomized behavior. The only private information is the local history of each agent, which is tracked by remembering the actions executed and the observations received in previous steps. This local history is only used to execute the plan and has no effect on the coordination mechanisms. In Fig. 1, the inputs and outputs of the plan and update modules are the same for all agents. The "execute" module is the only part that considers the local information of each agent, but this does not make any change to the belief pool as well as the joint policy and cannot lead to miscoordination. We use the *sync* communication model to reset the belief pools with the same histories and joint belief state. Thus, each agent also maintains the same belief pool after communication. As long as the belief pools of each agent are identical, the joint policies computed based on the belief pools are still the same. Therefore, agents remain coordinated. □

Our online 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 2 is executed. Generally, there are three major challenges in implementing this framework: (1) It is an NP-hard problem to find the decentralized policies for every possible history of every agent [58]. In Section 3.1, we provide an approximate solution by solving a series of linear programs. (2) The belief pool itself can be extremely large, making it impossible be managed online. More precisely, the number of possible histories grows exponentially with the time step. In Section 3.2, we introduce policy-based techniques to bound the memory usage of the belief pool. (3) Detecting inconsistency in the belief pool is nontrivial given only the agent's local information. In Section 3.3, we describe a method to address that problem efficiently. Finally, in Section 3.4, we discuss some data structures used in our implementation to store belief pools.

### *3.1. Searching stochastic policies using linear programming*

In decentralized multi-agent systems, agents without knowledge of the observations of the other agents must reason about all the 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 $q_i$ for history $h_i$, agents need to reason about all the possible histories $h_{-i}$ held by the others as well

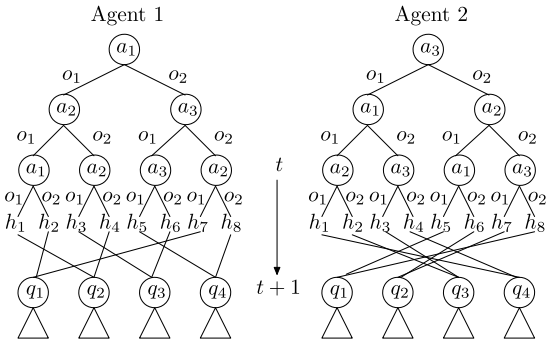


**Fig. 2.** Illustration of similarity between the one-step online planning and the one-step offline policy tree construction.

as all the possible policies associated with them. In other words, we need to find a joint policy $\delta$ that maximizes the value function below:

$$V(\delta) = \sum_{h \in H} \sum_{s \in S} p(s|h) V\big(\delta(h), s\big) \tag{5}$$

where $p(s|h)$ is the state distribution given a joint history $h$.

It is important to point out that the joint policy created by our approach is a truly decentralized policy, which depends on the private information of each agent. That is, the resulting policy of each agent depends only on its individual observation history, i.e. $\delta(h) = \langle \delta_1(h_1), \ldots, \delta_n(h_n) \rangle$. The goal of our multi-agent online planning is that each agent independently calculates the same plan $\delta(h)$ for the team and then executes its share of the plan based on its own local history. For example, agent $i$ will execute the action $a_i = \delta_i(h_i)$. The advantage of a decentralized policy is that each agent can execute its part of the plan independently based on the local information it acquired so far.

Finding decentralized policies online is analogous to the one-step policy tree construction used in offline DEC-POMDP planning algorithms. As shown in Fig. 2, the histories $(h_1, h_2, \ldots, h_8)$ are paths of the tree from the root down to the current branches. The target of both online and offline planning is to associate the histories with the "right" sub-policies that satisfy the optimality criterion. In offline planning histories are often represented as trees and the goal is to construct the best complete policy. In contrast, online algorithms store histories in a sequence form and evaluate the policy only for the current step. Obviously, the number of histories represented in the sequence form is much smaller, which is a key advantages of online planning.

The straightforward way of finding the best joint policy is to enumerate all possible mappings from histories to sub-policies and choose the best one. However, the size of the joint space is exponential over the number of the possible histories. The number of histories itself grows exponentially with the problem horizon. In fact, this problem is equivalent to the decentralized decision problem studied by [58], which has been proved to be NP-hard [58]. In our algorithm, we find an approximate solution using stochastic policies, by solving the problem as a linear program. The value of a joint stochastic policy, $\pi$, is as follows:

$$V(\pi) = \sum_{h \in H} p(h) \sum_{\vec{q}} \prod_{i \in I} \pi_i(q_i|h_i) Q\big(\vec{q}, b(h)\big) \tag{6}$$

where $p(h)$ is the probability distribution of history $h$, $b(h)$ is the belief state induced by $h$, and $Q(\vec{q}, b(h))$ is the value of policy $\vec{q}$ at $b(h)$. Note that $\vec{q}$ is a policy from the current time to the end of the problem, so $Q(\vec{q}, b(h))$ is the value that agents will achieve in future steps when starting with a state distribution $b(h)$. Unfortunately, the optimal value of $Q$ is not available online. In fact, this is a subproblem of the original DEC-POMDP with a new horizon $T - t$ where $t$ is the current time step. So trying to find the optimal value is equivalent to solving the entire problem offline by simply setting $t = 0$. But the optimal value can be estimated by certain heuristics. Usually, the heuristic close to the optimal value may take more time to compute but yield better performance and vice versa.

We use one-step lookahead to estimate the value of future steps. It means that we consider a policy with only one action node. In this case, any approach which provides a set of value functions $V(s)$ can be used to define the heuristic. Ideally, the heuristic should represent not only the immediate value of a joint action but also its expected future value. As mentioned, finding the optimal value is intractable because it requires us to do as much work as solving the entire DEC-POMDP. One approach we used is the solution of the underlying MDP. For the one-step lookahead case, $q_i, \vec{q}$ can be simplified to $a_i, \vec{a}$. The QMDP heuristic [28] can then be written as follows:

$$Q(\vec{a}, b) = \sum_{s \in S} b(s) \left[ R(s, \vec{a}) + \sum_{s' \in S} P\big(s'|s, \vec{a}\big) V_{\text{MDP}}\big(s'\big) \right] \tag{7}$$

**Table 1**
Improving the policy using linear programming.

| |
|---|
| Variables: $\varepsilon, \pi_i(q_i\|h_i)$ |
| Objective: maximize $\varepsilon$ |
| Improvement constraint: |
| $V(\pi) + \varepsilon \leqslant \sum_{h\in H} p(h) \sum_{\vec{q}} \pi_i(q_i\|h_i) \prod_{k\neq i} \pi_k(q_k\|h_k) Q(\vec{q}, b(h))$ |
| Probability constraints: |
| $\forall h_i \in H_i, \sum_{q_i} \pi_i(q_i\|h_i) = 1$ |
| $\forall h_i \in H_i, q_i, \pi_i(q_i\|h_i) \geqslant 0$ |

where $V_{\text{MDP}}$ is the value function of the underlying MDP. The QMDP heuristic is an upper bound of the optimal value since it is based on the assumption that agents fully observe the underlying system state at each step. A tighter bound would be the QPOMDP heuristic [44]:

$$Q(\vec{a}, b) = \sum_{s\in S} b(s)\left[R(s,\vec{a}) + \sum_{s'\in S} P(s'|s,\vec{a}) \sum_{\vec{o}\in\vec{\Omega}} O(\vec{o}|s',\vec{a}) V_{\text{POMDP}}\left(b_{\vec{a}}^{\vec{o}}\right)\right] \tag{8}$$

where $b_{\vec{a}}^{\vec{o}}$ is the successor belief state of $b$ with $\vec{a}, \vec{o}$ and $V_{\text{POMDP}}$ is the value function of the underlying POMDP. Intuitively, this means the agents will share their observations at each future step. When a concrete problem is considered, domain-specific knowledge can be used to better estimate the heuristic value. In our implementation, we use the QMDP heuristic because the underlying MDPs of the tested domains can be solved quickly and optimally. The one-step lookahead could be extended into a multi-step lookahead, but this is much more complex in our settings and beyond the scope of this article.

To start the search procedure, each local stochastic policy $\pi_i$ 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:

$$V(\pi) \leqslant \sum_{h\in H} p(h)\left[\sum_{\vec{q}} \pi_i(q_i|h_i)\pi_{-i}(q_{-i}|h_{-i}) Q(\vec{q}, b(h))\right] \tag{9}$$

where $\pi_{-i}(q_{-i}|h_{-i}) = \prod_{k\neq i} \pi_k(q_k|h_k)$. The linear program shown in Table 1 is used to find the new parameters of $\pi_i$. The improvement procedure terminates and returns $\pi$ when $\varepsilon$ becomes sufficiently small for all agents. Although the algorithm will terminate after a finite number of iterations, convergence to optimality is not guaranteed. It is possible to get stuck in a suboptimal Nash equilibrium in which the policy of each agent is optimal with respect to the others. In fact, the suboptimal solution may achieve a value which is arbitrarily far from the globally optimal one. Hence, algorithms which start with an arbitrary policy and make iterative improvements by alternating among the agents cannot produce polices which are within a guaranteed bound of the optimal value.

One simple technique is to use random restarts to move out of local maxima. The observation is that different starting points may converge to different locally optimal values. Hopefully, with several random restarts, one of the values may be globally optimal or close to it. This process is shown in Algorithm 3. The number of restarts is determined by the online runtime constraint. If the time per planning step is long, more restarts can be used and there is a better chance to get close to the globally optimal solution. This simple technique works very well in the test domains we experimented with.

Note that each agent shares the same random seed so that all agents have the same randomized behavior. It is easy to construct situations in which two policies $q_i$ and $q_i'$ have the same value. In order to guarantee coordination, each agent will choose the same policy according to a predetermined tie-breaking rule based on a canonical ordering of the policies (e.g. $q_i \prec q_i'$).

### 3.2. Bounding joint histories using policy-based merging

Note that the underlying system state as well as the observations of other agents are not available during the execution time of DEC-POMDPs. Each agent must reason about all the 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, which is $(|\Omega_i|^T)^{|I|}$ assuming that each agent coordinates at each step and knows the joint policy. For a small problem with 2 agents and 2 observations, the number of possible joint histories with 100 steps is $2^{100\times 2} \approx 1.6 \times 10^{60}$, which is infeasible for any planning algorithms. Even storing them in the memory is impossible. This presents a major challenge for developing online algorithms for DEC-POMDPs.

*Optimal history merging.* A more detailed analysis of the planning process shows that most of the joint histories kept in memory are useless. One reason is that the goal of reasoning about others' histories is to find out what others do and see. Clearly, each time only one sequence corresponds to this for each agent. Because they do not share private information with each other, what agents can do is to maintain a distribution over the histories based on the information they have. As

**Algorithm 3**: Search Stochastic Policies with Random Restarts

```
Input: H, B
for several restarts do
    // select the start point randomly.
    π ← Initialize the parameters to be deterministic with random actions
    repeat
        ε ← 0
        foreach i ∈ I do
            // optimize the policy alternatively.
            π_i, ε' ← Solve the linear program in Table 1 with H, B, π_{-i}
            ε ← ε + ε'
    until ε is sufficiently small
    if π is the current best policy then
        π* ← π
return π*
```

long as the distribution is sufficiently accurate, eliminating histories early on may have no effect on the decision. Another reason is that the history segments of the early stage may be useless. For example, in the multi-agent tiger problem [32], two agents are facing two doors (left and right): behind one lies a tiger and behind the other lies untold riches. The agents can independently open either door or listen for the position of the tiger. The reward function is designed to encourage coordination in that opening a door together will sustain less injury if the tiger is present or receive a greater amount of wealth if the riches are present. The listen action incurs a small cost. The position of the tiger is reset randomly after a door is opened. The resulting policy with horizon 6 may be that an agent listens twice, opens a door if it receives the same observation of the tiger's position, listens twice again and then again opens a door if it received the same two observations of the tiger's position. For all histories in which an agent opens a door on the third step, the best action for it to take by the time of the last step will not be affected by any observations received in steps 1 to 3. Those observations provide no useful information since the tiger's position is reset if a door is opened. From the perspective of the team, the only important thing for an agent is the probability that others opened a door in step 3. This quantity can be found by grouping similar histories together.

**Definition 6.** Two histories $h_i, h_i'$ for agent $i$ are probabilistically equivalent (PE) when the following holds: $\forall h_{-i}, p(h) = p(h')$ and $\forall h_{-i}, s, b(s|h) = b(s|h')$ where $h = \langle h_i, h_{-i}\rangle, h' = \langle h_i', h_{-i}\rangle$ and $h_{-i}$ is a joint history without agent $i$'s.

The PE condition means that two histories of an agent have the same distribution and also the same resulting belief state, but they may differ in action-observation sequences. The following property has been established for histories that are probabilistically equivalent.

**Lemma 1.** *(See [38].) When two histories are PE, then they are best-response equivalent and can be clustered together as one history without any loss of value.*

Although the PE condition has very nice theoretical properties, it is not very applicable in practice because it requires to know $h_{-i}$, every possible history combination of other agents. As mentioned earlier, the range of values of $h_{-i}$ will be very large and it is intractable to test every possibility. Considering every possible $h_{-i}$ is important because every single value may affect the policy of agent $i$. Hence it is worth analyzing how these values affect agent $i$'s policy. First, agent $i$ could form a belief about the current state by reasoning about the history of the other agents, $h_{-i}$. Together with its own history $h_i$, agent $i$ can calculate a state distribution $b(\langle h_i, h_{-i}\rangle)$ based on the joint history. Second, agent $i$ needs to know what policies others take because it should choose a policy which complements the others. In this context, the goal of reasoning about the histories of other agents is to compute a policy. If the optimal policy $q_i^*$ of agent $i$ with $h_i$ is given, agent $i$ can follow the policy without considering its own history as well as the others'. For example, in the multi-agent tiger problem, if an agent knows that the optimal policy from now on is to open a door whenever it hears the tiger's roar behind that door twice, remembering what happened before is not necessary.

**Definition 7** *(policy equivalent).* Two histories $h_i, h_i'$ of agent $i$ are policy equivalent (POE) when $h_i, h_i'$, have at least one identical optimal policy.

Here, a policy means a complete conditional plan from the current step to the last step. Usually, a policy is represented as a tree with nodes corresponding to actions and branches corresponding to observations. The optimal policy is a fixed point in policy space, which produces the best team performance given that other agents follow their optimal policies. Therefore, the optimal policy of an agent is part of the optimal joint policy of the team. In DEC-POMDPs, there exists at

least one optimal joint policy for the team. The optimal joint policy of small problems can be computed by optimal offline algorithms [5,27,55,56].

**Theorem 2.** *When two histories $h_i, h'_i$ of agent i are POE, then they can be merged without loss of value in policy generation by keeping either of them.*

**Proof.** At step 0, if the optimal policies for steps 0 to $T$ are given, agents can select the optimal joint policy for $b^0$. 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 $h_i^t \circ h_i^k$ and $h_i'^t \circ h_i^k$ 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. □

**Theorem 3.** *When two histories of agent i are PE, they are also POE, meaning that at least one of their optimal policies is the same.*

**Proof.** The proof is based on the theorems in [38]. Suppose that histories $h_i^t, h_i'^t$ are PE at step $t$. At any step $t+k$, the identical extensions $h_i^{t+k}, h_i'^{t+k}$, created by appending the same length-$k$ action-observation sequence to the end of $h_i^t, h_i'^t$, are also PE. Since $h_i^{t+k}, h_i'^{t+k}$ are PE, they have equal optimal Q-value function. And agent $i$ will always select the same optimal action for $h_i^{t+k}, h_i'^{t+k}$ based on the optimal Q-value function. Hence it is very easy to build a policy which is optimal both for $h_i^t, h_i'^t$ by considering every possible extension with optimal actions. When two histories of agent $i$ are PE, they have at least one equal optimal policy so they are also POE. □

However, when two histories of agent $i$ are POE, they are not necessarily PE. For example, in the multi-agent tiger domain, suppose that agent $i$ opens the left door (OL) when the history is $h_i$ but it opens the right door (OR) if the history is $h'_i$. After the door is opened, the tiger's position is reset. So agent $i$ will follow the same optimal policy. Obviously, $h_i \circ \{OL\}$ is not PE with $h'_i \circ \{OR\}$, but they are POE. Therefore, the POE condition is more general than the PE condition, $PE \subset POE$. In single-agent POMDP, the policy tree can be evaluated given a history, thus an agent can either compare the policy or value. Since multiple optimal policies may exist but have the same value, the value equivalent (VE) condition may be more general, $POE \subset VE$. However, when multiple agents are involved, only a joint policy with a joint history can be evaluated. It is not clear how to calculate the exact value of a local policy given the history of a single agent in DEC-POMDPs.

*Approximate history merging.* In DEC-POMDP offline planning, the POE condition facilitates sub-policy reuse. Reusing a policy for several histories is the same as mapping several histories to a single policy. When handling a policy, this is referred to as *reuse*. It is called *merging* when managing histories. In the optimal bottom-up dynamic programming algorithm, the best policies from the current to the last step are available. The number of policies is much lower than the number of possible histories because a small set of policies can be reused to build any complex policies in the next iteration. When considering approximate solutions, the condition of optimal policies can be relaxed. For example, consider a robot sent out to clean 10 rooms in a building. There are three type of rooms: 5 office rooms, 3 meeting rooms and 2 restrooms. The robot only needs to know how to clean these three types of rooms. In other words, the optimal policy for one type of room can be reused several times in the policy of cleaning the 10 rooms. In this scenario, the policy of cleaning one type of room is not necessarily optimal. In fact, the basic idea of MBDP is to keep a fixed number of policies and reuse them approximately when building the policies of the next iteration. The same observation can be applied to online planning as a way to merge histories.

Unfortunately, the optimal policy for each agent is not available during execution time. Finding the optimal 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 $k$-step lookahead policy therefore can be evaluated by decomposing the value function into an exact evaluation of the $k$-steps and a heuristic estimate of the remaining part. Then, we can define similarity by comparing the structure of depth-$k$ trees. The $k$-step lookahead policy is generated by pre-computed heuristics. The POE condition is approximated by the similarity of the $k$-step lookahead policies. More precisely, two histories of agent $i$ are POE when the $k$-step lookahead policies of them have similar structure. 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.

We present a way (Algorithm 4) to maintain a bounded size belief pool online and use it to coordinate the strategy of the team. Bounding the size of histories is important especially for online planning, where the planning time of each step is limited. Clustering methods often have no guarantee to reach a desired size of belief pool. If the size is too large, the algorithm may exceed the planning time and miss the action cycle. Even worse, without a policy there is no way to maintain the belief pool. In real applications, this may cause damage to the system if no special recovery process is implemented. In our algorithm, we merge histories whenever they are POE and then randomly choose just one history per policy, so the number of histories retained is bounded by the number of the policies generated and the definition of similarity. At each step, heuristics are used to perform the $k$-step lookahead and create a fixed number of policies.

**Algorithm 4**: Policy-Based History Merging

**Input**: $H^t, Q^t, \pi^t$
**foreach** $i \in I$ **do**
  $\tilde{H}_i^t \leftarrow \emptyset$
  // $\mathcal{H}_i$ is a hash table indexed by $q_i$.
  $\mathcal{H}_i(q_i) \leftarrow \emptyset, \forall q_i \in Q_i$
  // group histories based on the policy.
  **foreach** $h_i \in H_i^t$ **do**
    // get the policy of $h_i$ according to $\pi_i^t$.
    $q_i \leftarrow$ Select a policy according to $\pi_i^t(q_i|h_i)$
    // add $h_i$ to the hash table with key $q_i$.
    $\mathcal{H}_i(q_i) \leftarrow \mathcal{H}_i(q_i) \cup \{h_i\}$
  // generate a new set of histories.
  **foreach** $q_i \in Q_i^t$ **do**
    // keep one history per policy.
    **if** $\mathcal{H}_i(q_i)$ *is not empty* **then**
      $h_i \leftarrow$ Select a history from $\mathcal{H}_i(q_i)$ **randomly**
      $\tilde{H}_i^t \leftarrow \tilde{H}_i^t \cup \{h_i\}$
  // fill up the history set.
  **while** $|\tilde{H}_i^t| < |Q_i^t|$ **do**
    $q_i \leftarrow$ Select a policy from $Q_i^t$ randomly
    **if** $\mathcal{H}_i(q_i)$ *is not empty* **then**
      $h_i \leftarrow$ Select a history from $\mathcal{H}_i(q_i)$ **randomly**
      $\tilde{H}_i^t \leftarrow \tilde{H}_i^t \cup \{h_i\}$
**return** $\tilde{H}^t$

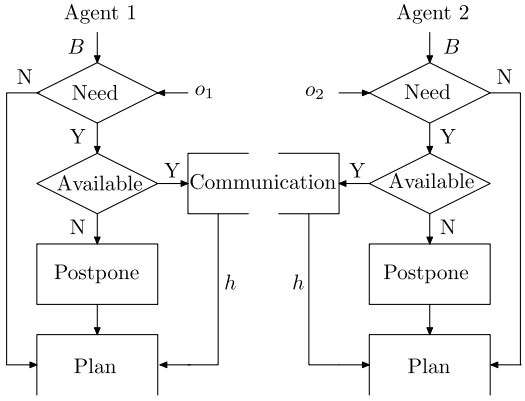


**Fig. 3.** Communication model for two agents.

### 3.3. Communicating when inconsistency arises

In our online planning framework, communication will be initiated before the planning phase, as shown in Fig. 3. First, each agent will decide if communication is necessary then check if the resource is available. If communication is not needed, agents will continue to plan without communication. If communication is needed, but the resources are unavailable, the agents will postpone communication to the next step. Otherwise, they will *sync* their local information and do planning based on the information received from teammates. The local information communicated is just each agent's local observation sequence between the last communication step and the current step. The resources include the local communication device of the agents and the communication channel. All communication failures require re-communication at the next step. The decision to communicate is thus made whenever other agents initiate communication, communication is postponed in the previous step, or *inconsistency* of the belief pool is detected.

To verify the inconsistency of a belief pool, it will be straightforward if agents know the current state of the system. However, in the DEC-POMDP model, the state is unavailable online and each agent can receive only its own local observation at the execution time. Fortunately, agents' local observation often provides partial information of the system state. Hence we can detect the problem by examining any inconsistency between the belief pool and the local observation of the agent

from the environment. Intuitively, when agent $i$ tries some action and observes $o_i$, the probability of which is less than $\epsilon$ according to the belief pool, it is likely that there is something wrong with the belief pool.

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

**Definition 8** *($\epsilon$-inconsistency).* At time step $t$, the maintained belief pool $B^t$ is said to be $\epsilon$-inconsistent with agent $i$'s local observation $o_i^t$ if

$$\max_{\forall b, \forall o_{-i}^t} \left\{ \sum_{s' \in S} O\left(\vec{o}^{\,t} | s', \vec{a}\right) \sum_{s \in S} P\left(s' | s, \vec{a}\right) b(s) \right\} < \epsilon \tag{10}$$

where $\vec{a}$ is a joint action based on $\vec{o}^{\,t}$ and the joint policy computed at the previous step, $\vec{o}^{\,t} = o_i^t \cup o_{-i}^t$, $o_{-i}^t \in \bigtimes_{k \neq i} \Omega_k$ and $b \in B(h_i^t)$.

This definition is used to monitor inconsistency between agent $i$'s local history and observation, which provides an indication of history inconsistency in the pool. The threshold $\epsilon$ is determined by the structure of the observation function. If the uncertainty regarding the observation is small, $\epsilon$ should be small too. Note however that this rule cannot detect every form of inconsistency in the belief pool and is only based on the current local observations. And when inconsistency is detected, it only means that there is a high likelihood of a problem in the belief pool.

The amount of communication is determined by both the observation structure and the heuristic. Intuitively, agents can make the right decision as long as the belief pool contains a joint history that is close to the real joint history. However, agents cannot obtain the observations of the other agents at execution time, so it is impossible to know the real joint history. But they can check for inconsistency of the history pool based on their local information. If the pool is inconsistent, the agent can refresh the belief pool by communicating with the other agents and synchronizing the observation sequence. After synchronization, the belief 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 performance when it is possible. When communication fails for a long time, state-of-the-art approaches continue to make decisions using either open-loop methods which totally ignore the local observations [44,52] or local-greedy methods which lead to miscoordination due to the different local information of each agent. Obviously, the open-loop or local-greedy policies can become arbitrarily poor given a problem with sufficiently long horizons. Our approach computes a joint policy conditioned on the local observation of each agent. The common joint policy ensures that each agent will take coordinated actions while utilizing its local information. Hence agents can coordinate their behavior even without communication.

It is worthwhile to point out that our concept of belief inconsistency is fundamentally different from the idea of belief divergence [60]. The belief divergence approach first establishes a reference point, which is the belief of agents when they last synchronized their knowledge. Then it compares the current belief state with that reference point. The distance between the belief points is measured by KL-divergence. If the divergence reaches some threshold, the agents communicate with each other to synchronize their observation sequences. This approach assumes that the other agents do not receive independently any new observations and that their beliefs remain static after the last communication step [60]. In contrast, our approach keeps updating the *joint* beliefs given the possible observations of the other agents. Belief inconsistency is detected when the agent's local observation does not match the projected joint belief.

To summarize, our approach has the unique ability to postpone communication and at the same time take into account new local observations and maintain coordination. While other approaches also allow (or could be easily modified to allow) postponement of communication, the result is a significant degradation in performance. Specifically, in Dec-Comm, this leads to open-loop operation discarding local observations. In approaches that rely on belief divergence, postponing communication leads to a greater divergence of beliefs and increases miscoordination.

### 3.4. Implementation considerations

As mentioned earlier, we only try one-step lookahead in our implementation. So the 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 belief pool. We only need to assign an index to a history and use a hash table $h^t = \langle \vec{\theta}, b^t \rangle$ to represent the joint history, where $\vec{\theta} = \langle \theta_1, \ldots, \theta_n \rangle$, $\theta_i$ is the history index for agent $i$ and $b$ is the joint belief induced for the joint history by Eq. (1). Since we keep only one history for one policy, the history index for agent $i$ can be represented as a tuple $\theta_i = \langle q_i^{t-1}, o_i^t \rangle$ 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.

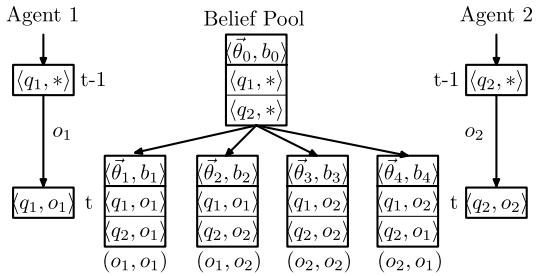


**Fig. 4.** Example of history expansion and updating. The joint history $\langle \vec{\theta}_0, b_0 \rangle$ with two components $\langle q_1, * \rangle$ and $\langle q_2, * \rangle$ in the pool is expanded to $\langle \vec{\theta}_1, b_1 \rangle, \langle \vec{\theta}_2, b_2 \rangle, \langle \vec{\theta}_3, b_3 \rangle, \langle \vec{\theta}_4, b_4 \rangle$ 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 $\langle q_2, o_2 \rangle$.

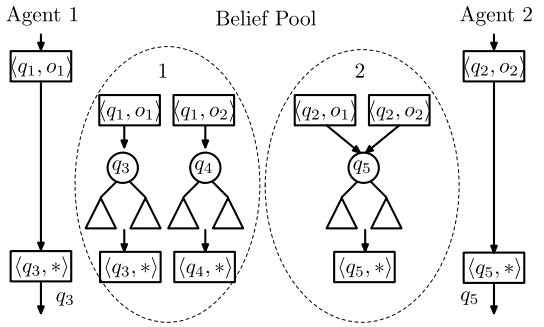


**Fig. 5.** Example of history merging and policy execution. The histories $\langle q_2, o_1 \rangle$ and $\langle q_2, o_2 \rangle$ in the pool map to the same policy $q_5$, so only one history is *randomly* selected (e.g. $\langle q_2, o_1 \rangle$) for the next step and its index is updated to $\langle q_5, * \rangle$. $\langle q_1, o_1 \rangle$ maps to $q_3$ and its updated index becomes $\langle q_3, * \rangle$. $\langle q_1, o_2 \rangle$ maps to $q_4$ and its updated index is $\langle q_4, * \rangle$. 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 updates 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 $q_5$.

For the one-step lookahead case, this index can be further simplified as $\theta_i = \langle a_i^{t-1}, o_i^t \rangle$. Therefore, the data structure we use to represent each element of the belief pool $h^t \in H^t$ is

$$h^t = \underbrace{\langle\langle \underbrace{q_1^{t-1}, o_1^t}_{\theta_1}\rangle, \underbrace{q_2^{t-1}, o_2^t}_{\theta_2}\rangle, \ldots, \underbrace{\langle q_n^{t-1}, o_n^t\rangle}_{\theta_n}\rangle}_{\vec{\theta}}, b^t\rangle$$

At every step, we update each index as well as the joint belief state. Fig. 4 shows how to expand histories and update the index of the agent's local history with its observation. Fig. 5 shows how to merge histories and update the index of the agent's local history after executing a policy. Note that the new history is created by appending first a new action and then a new observation to the end of a previous history. The history index without observation is represented by $\langle q, * \rangle$ in the figures for the intermediate histories during the update process, $h_i \circ a_i$, that are missing an observation. It is possible to design different types of indices and keep more than one history for each policy if needed.

Each agent's own local history used for execution is also represented by an index. At each step, we will update the index of the agent's own local history using $\langle q_i^t, o_i \rangle$, where $q_i^t$ is the policy agent $i$ executes and $o_i$ is the observation received from the environment at the current step $t$. If the history indexed by $\langle q_i^t, o_i \rangle$ is merged and represented by another history $\langle q_i^t, o_i' \rangle$ in the pool, we will change agent $i$'s local history index to $\langle q_i^t, o_i' \rangle$. Hence we always map the agent's local history to a history in the pool. For the purpose of communication, agent $i$ also stores an action-observation sequence $\langle a_i^0, o_i^1, a_i^1, o_i^2, \ldots, a_i^{t-1}, o_i^t \rangle$ that includes the actions executed and observation received.

To summarize, we developed new data structures to implement the belief pool as well as each agent's local history. Instead of storing all the observation-action sequences, we use indices to represent histories. Each index of a history $\langle q_i, o_i \rangle$ contains two parts: a pointer to agent $i$'s policy, $q_i$ and the current local observation $o_i$. In Fig. 4, we show how to expand the belief pool by every joint observation and how to update each agent's local history by its current observation received from the environment. In Fig. 5, we show how the expanded histories are merged and how each agent's local history is updated after executing a new policy. These two figures illustrate the key operations using the new representation of indices in our implementation.

## 4. 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 (Reward), average online runtime per step (Time(s)), and average percentage of communication steps (Comm(%)) with different horizons (Horizon). While communication is limited and minimizing it is an important goal, we did not add an explicit cost for communication because any such cost would have been arbitrary and not particularly relevant to these applications. The main purpose of the experiments is to test whether high-valued plans can be computed quickly on-line, while using little communication. Specifically, our goal was to achieve significantly better value with significantly less communication compared to the state-of-the-art. MAOP-COMM was implemented in Java and ran on a 2.4 GHz Intel Core 2 Duo processor with 2 GB of RAM. Linear programs were solved using lp_solve 5.5 with Java wrapper. 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 benchmark problems we used. 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 steps without communication). 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. Another fact pointed out by [44] 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 our algorithms to Dec-Comm-PF – the leading communicative online algorithm which is applicable to all the test problems – presents the best way to assess the benefits 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. We did not use a discount factor in these experiments because all the tested problems involve a finite horizon.

It is important to emphasize that the FULL-COMM strategy is expected to outperform approaches that use partial communication. Nonetheless, we provide the results of FULL-COMM to establish an upper bound for our online algorithm. Although perfectly reliable and instantaneous communication is generally unrealistic, it is interesting to show how well the online communication approach performs when it uses the same MDP heuristic. In our implementation, FULL-COMM runs as a POMDP with joint actions and observations. The joint action selected at each step is based on a one-step lookahead using the MDP heuristic. It takes very little time because the main computation is the Bayesian update of only one joint belief state. We do not consider the time for transmitting observations among agents. As discussed earlier, the FULL-COMM strategy simply reduces a DEC-POMDP to a POMDP, which is much easier to solve. Our goal in these experiments is to show that our proposed approach for online planning with bounded communication is effective, when compared with the FULL-COMM upper bound.

### 4.1. Perfectly reliable communication channel

Our initial set of experiments involves a communication channel that is perfectly reliable; it is assumed to be always available and without noise. We relax these assumptions in the following section.

*Standard benchmark problems.* The first set of experiments involves four standard benchmark problems: Broadcast Channel [11], Meeting in a Grid [11], Cooperative Box Pushing [46] and Stochastic Mars Rover [4]. These benchmarks have been widely used to evaluate cooperative multi-agent planning algorithms modeled as DEC-POMDPs.[1] Because the number of possible histories is extremely large 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 [32], Recycling Robots [2] and Fire Fighting [35] have only 2 observations.

The Broadcast Channel problem [11] 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

[1] Original domain descriptions are available for download from the DEC-POMDP repository: http://users.isr.ist.utl.pt/~mtjspaan/decpomdp/index_en.html.

**Table 2**
Benchmark results (20 trials).

| Horizon | Algorithm | Reward | Time (s) | Comm (%) |
|---|---|---|---|---|
| Broadcast channel: $\|S\| = 4$, $\|A_i\| = 2$, $\|\Omega_i\| = 5$ | | | | |
| 20 | MAOP | 18.25 | $< 0.01$ | 0.0 |
| | MAOP-COMM | 18.35 | $< 0.01$ | 0.0 |
| | Dec-Comm-PF | 18.45 | $< 0.01$ | 0.0 |
| | FULL-COMM | 18.90 | $< 0.01$ | 100.0 |
| 100 | MAOP | 89.95 | $< 0.01$ | 0.0 |
| | MAOP-COMM | 90.35 | $< 0.01$ | 0.0 |
| | Dec-Comm-PF | 90.0 | $< 0.01$ | 0.0 |
| | FULL-COMM | 90.60 | $< 0.01$ | 100.0 |
| Meeting in a 3×3 Grid: $\|S\| = 81$, $\|A_i\| = 5$, $\|\Omega_i\| = 7$ | | | | |
| 20 | MAOP | 3.10 | 0.15 | 0.0 |
| | MAOP-COMM | 3.35 | 0.26 | 11.0 |
| | Dec-Comm-PF | 2.90 | 0.22 | 70.50 |
| | FULL-COMM | 4.75 | $< 0.01$ | 100.0 |
| 100 | MAOP | 15.30 | 0.19 | 0.0 |
| | MAOP-COMM | 17.10 | 0.30 | 12.10 |
| | Dec-Comm-PF | 14.90 | 0.24 | 78.20 |
| | FULL-COMM | 24.70 | $< 0.01$ | 100.0 |
| Cooperative box pushing: $\|S\| = 100$, $\|A_i\| = 4$, $\|\Omega_i\| = 5$ | | | | |
| 20 | MAOP | 7.50 | 0.14 | 0.0 |
| | MAOP-COMM | 99.30 | 0.16 | 11.50 |
| | Dec-Comm-PF | 136.75 | 0.35 | 83.50 |
| | FULL-COMM | 222.50 | $< 0.01$ | 100.0 |
| 100 | MAOP | −16.0 | 0.13 | 0.0 |
| | MAOP-COMM | 441.95 | 0.13 | 12.26 |
| | Dec-Comm-PF | 296.50 | 0.36 | 59.87 |
| | FULL-COMM | 880.50 | $< 0.01$ | 100.0 |
| Stochastic Mars Rover: $\|S\| = 256$, $\|A_i\| = 6$, $\|\Omega_i\| = 8$ | | | | |
| 20 | MAOP | 18.19 | 0.97 | 0.0 |
| | MAOP-COMM | 46.19 | 0.05 | 17.0 |
| | Dec-Comm-PF | 45.02 | 2.46 | 22.0 |
| | FULL-COMM | 61.41 | $< 0.01$ | 100.0 |
| 100 | MAOP | 51.15 | 2.20 | 0.0 |
| | MAOP-COMM | 222.86 | 0.09 | 18.00 |
| | Dec-Comm-PF | 133.04 | 2.39 | 35.00 |
| | FULL-COMM | 325.04 | $< 0.01$ | 100.0 |

similar values with runtime less than 0.01 seconds. Both MAOP-COMM and Dec-Comm-PF initiated no communication. The performance without communication was almost the same as the case of full communication. These results have a simple intuitive explanation. The probability that an 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 [11], two robots 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 [46], 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 100, MAOP-COMM outperformed Dec-Comm-PF, again with much less communication. MAOP and MAOP-COMM ran a little faster than Dec-Comm-PF.

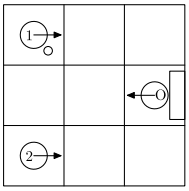


**Fig. 6.** The 3×3 grid soccer domain with 1 opponent and 2 teammates.

**Table 3**
Grid soccer results (20 trials).

| Horizon | Algorithm | Reward | Time (s) | Comm (%) |
|---|---|---|---|---|
| Grid soccer 2×3: $\|S\| = 3843$, $\|A_i\| = 6$, $\|\Omega_i\| = 11$ | | | | |
| 20 | MAOP | 180.50 | 0.25 | 0.0 |
| | MAOP-COMM | 290.6 | 0.28 | 14.80 |
| | Dec-Comm-PF | 129.50 | 1.36 | 33.5 |
| | FULL-COMM | 373.90 | $< 0.01$ | 100.0 |
| 100 | MAOP | 1157.80 | 0.14 | 0.0 |
| | MAOP-COMM | 1933.90 | 0.16 | 15.40 |
| | Dec-Comm-PF | 1441.60 | 1.28 | 30.80 |
| | FULL-COMM | 1933.60 | $< 0.01$ | 100.0 |
| Grid soccer 3×3: $\|S\| = 16{,}131$, $\|A_i\| = 6$, $\|\Omega_i\| = 11$ | | | | |
| 20 | MAOP | 190.70 | 1.90 | 0.0 |
| | MAOP-COMM | 296.00 | 2.30 | 27.0 |
| | Dec-Comm-PF | 271.40 | 15.26 | 59.0 |
| | FULL-COMM | 356.0 | $< 0.01$ | 100.0 |
| 100 | MAOP | 803.60 | 1.96 | 0.0 |
| | MAOP-COMM | 1679.50 | 2.50 | 26.90 |
| | Dec-Comm-PF | 1044.40 | 9.48 | 70.70 |
| | FULL-COMM | 1808.20 | $< 0.01$ | 100.0 |

The Stochastic Mars Rover [4] is a larger problem with 256 states and 2 agents. Each agent has 5 actions and 8 observations. The original problem has deterministic observations. We made it more challenging by introducing uncertainty into the observations. Each agent observes with 0.9 probability what is really in front of it, and with 0.025 probability any other observation. The results in Table 2 show that communication is critical in this domain as well. Without communication, MAOP gets less value than MAOP-COMM – the communicating version. Again, for the longer horizon, MAOP-COMM produces much higher value than Dec-Comm-PF with much less communications.

To summarize, MAOP-COMM performed very well in all the 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 very high value without any communication. Although the tested horizon was only up to 100, our approach can solve problems with much larger horizons 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 miscommunication and error accumulation. The parameter $\epsilon$ presents a good way to tradeoff between the amount of communication and overall value. In domains such as Cooperative Box Pushing, a larger $\epsilon$ would allow more communication and consequently improve performance.

*The grid soccer domain.* To demonstrate scalability, we also tested our algorithm on a more challenging problem called grid soccer. Shown in Fig. 6, the domain includes two agents for one team and one opponent for the other team. Each agent has 4 possible orientations (up, down, left or right). The opponent – with full observation and reliable actions – always executes a fixed policy and tries to approach 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.

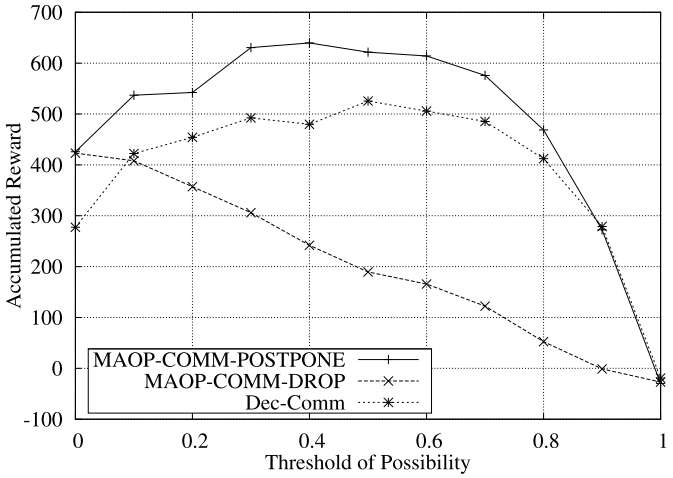


**Fig. 7.** Result of accumulated rewards with different thresholds.

The results are shown in Table 3. MAOP-COMM achieved higher value than Dec-Comm-PF, while the performance of MAOP is competitive, indicating that MAOP-COMM with a smaller $\epsilon$ would use even less communication and produce good value. The runtimes 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 $3\times3$ 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.

### *4.2. Imperfect communication channel*

In many real-world applications, communication could be unreliable due to noise or poor reception. For example, when a wireless communication network is used, connectivity could be intermittent and messages may have to be retransmitted several times. Our work allows communication to be postponed when the communication channel is not available. While communication could be postponed in other approaches, the downside is more severe. Dec-Comm [44], for example, will simply ignore the agents' local observations and run open-loop policies. The belief divergence approach [60] will make incorrect assumptions about the other agents' beliefs and run greedy policies, which are likely to be uncoordinated. In contrast, when communication is postponed, our approach still computes a conditional plan that takes into consideration all the possible observations. As discussed in Section 3, this conditional plan remains coordinated for all the agents.

In this section, we present experimental results for the cooperative box-pushing domain with an imperfect communication channel. We simulated intermittent communication by drawing a random variable from a uniform distribution each time communication is initiated. If the value is greater than some threshold, the communication channel is available; otherwise, it is unavailable. We varied the threshold from 0.0 to 1.0 and measured the accumulated rewards and percentage of communication averaged over 20 runs. Communication is always available when the threshold is 0.0 and there is no communication allowed when the threshold is 1.0. We show the results of two MAOP-COMM variants: MAOP-COMM-POSTPONE, which postpones communication until the channel is available, and MAOP-COMM-DROP, which simply drops the communication attempt when the channel is not available.

We show the results of accumulated rewards in Fig. 7 and the results of percentage of communication in Fig. 8 with different thresholds. As expected, the reward and amount of communication goes down in MAOP-COMM-DROP when the probability of unavailable communication is high. It means that these communication steps are truly critical for the cooperative box pushing domain and simply dropping them will gradually decrease the rewards. Interestingly, the rewards and amount of communication grow for a certain range of threshold values when we postpone communication until it is available.

Interestingly, as the threshold starts growing (from 0 to 0.4), the agents implemented by MAOP-COMM-POSTPONE communicate a little bit more but also get better value. The value then declines as the threshold continue to grow. We have tried to analyze and explain this phenomenon, which seems counterintuitive. There are several possible reasons contributing to this. When communication is postponed, agents still need to make decisions without communication. The inconsistent belief pool becomes more and more uncertain as time goes on. The actions computed based on these uncertain beliefs will introduce some randomness. And because the value function is based on a heuristic that overestimates future performance,

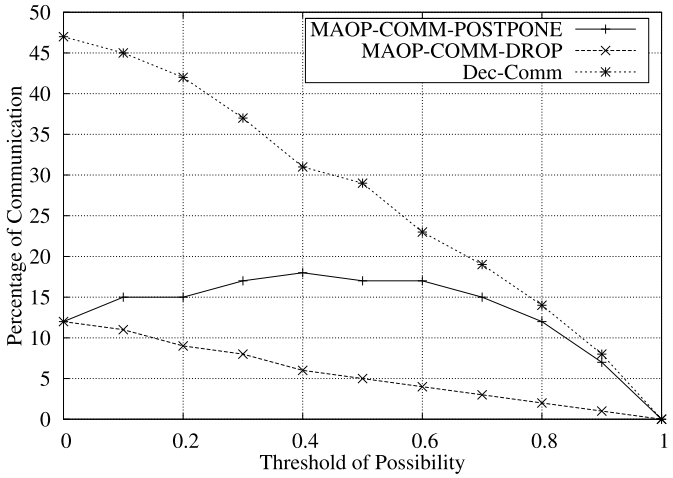


**Fig. 8.** Result of percentage of communication with different thresholds.

this randomness can have a positive effect on performance. Another possible explanation for problems with multiple goals is that agents may change their current goals after the communication. Once an agent initiates communication, all the agents in the team will be forced to refresh their belief pool. Since the heuristic is only an estimate of future value, changing goals too frequently based on the heuristic may not be desirable. Therefore agents may lose value if they communicate and change their undergoing goal too often. Consider, for example, the cooperative box pushing domain. Suppose that Agent 1 is close to the small box and decides to push it by itself based on the current belief pool. At the same time, Agent 2 is close to the large box and initiates communication, which forces Agent 1 to refresh its belief pool. Based on the new belief pool and the heuristic, the two agents establish a new goal and decide to push the large box together. However, the new goal computed by the overestimating heuristic may not be achievable while the old goal of Agent 1 could be achieved within the limited horizon. Thus, the agents could lose value by incorrectly estimating their abilities and abandoning current goals. This is also the reason that when to communicate, not the frequency of communication is more important [20].

We also compared our algorithms with Dec-Comm when the communication channel is imperfect. As the results show, MAOP-COMM-POSTPONE performs better than Dec-Comm with higher value and less communication. Interestingly, the value of Dec-Comm also grows when a certain amount of uncertainty about the reliability of the communication channel is introduced. Overall, these experiments confirm the advantage of MAOP-COMM-POSTPONE in domains with imperfect communication, which is an important factor in real-world applications.

## 5. Related work

The literature on planning and communication in DEC-POMDPs or equivalent models can be generally divided into works that compute full offline policies and those that do not. In this section, we only describe related work where explicit communication is involved. More general surveys of DEC-POMDP solution methods have been recently published by Seuken and Zilberstein [48], Oliehoek et al. [35] and Bernstein et al. [10].

### *5.1. Offline planning with communication*

One group of solution methods determines the entire plan and communication strategy *offline*, before plan execution starts. The stored plan is then used at runtime. The COM-MTDP model [41] and DEC-POMDP-COM model [25], which have equivalent complexity [48], provide theoretical frameworks for reasoning about communication offline. The COM-MTDP model offers a framework for analyzing the optimality of team performance and the computational complexity of the agents' decision problem. In terms of optimality analysis, it is able to encode existing teamwork theories and models, based on joint intentions and STEAM [57], and provide a novel algorithm which outperforms these earlier coordination strategies. The DEC-POMDP-COM model formalizes the problem for a given communication language and semantics, and examines the value of an optimal policy of action and communication with different cost models.

Pynadath and Tambe's work focuses on the optimality and complexity analysis of various communication models. The optimal solution they provide is not particularly useful in practice due to the complexity result – NEXP-complete in the general communication case. Besides, the COM-MTDP communication policies used in the joint intention instantiations require the designer to specify a joint persistent goal and allow an agent to send a message when the goal has been achieved [41]. The DEC-POMDP-COM model applies to the more general problem where the agents are allowed to communicate more than once and optimize the timing and frequency of communication. Both models require a priori semantics for the communicated messages. The semantics define the type of messages as well as the situations in which agents will communicate.

Assuming a message set $\Sigma$ with user-defined, fixed semantics simplifies the optimization problem, but makes the model design more difficult and domain-dependent.

Spaan et al. [50] 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. In contrast to the COM-MTDP and DEC-POMDP-COM models, it does not require an explicit communication language, but instead treats the semantics of communication as part of the optimization problem. They also present an iterative method for computing a joint policy for the team in a decentralized fashion, treating communication as an integral part of the reasoning process. Given a set of fixed policies for all agents but agent $i$, the policy computation process converts the DEC-POMDP into a POMDP from $i$'s perspective, which factors the expected contribution to the joint team reward of the policies of the other agents. The communication policy is based on a heuristic method in which information entropy is defined over policies and states. However, the model is restricted to *transition independent* agents where each agent's observation only depends on an agent's local state.

Generally, reasoning about communication offline requires 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 components of the JESP algorithm and finds a Nash equilibrium of policies for multiple agents [31]. The JESP (Joint Equilibrium-Based Search for Policies) approach tries to find the policy that maximizes the joint expected reward for one agent at a time, keeping the policies of the other agents fixed. the process is repeated until an equilibrium is reached [32]. DP-JESP is a dynamic programming approach to reason about an agent's policy in conjunction with its teammates. The multi-agent belief state defined in this approach is a distribution over the current state as well as the observation history of the other agents. Since the agent does not know exactly what observations the other agents have received at runtime, the algorithm allows the agents to periodically synchronize their own observation histories to reduce the likelihood of undesired behavior. 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. Thus no policy can be indexed by an observation history of length greater than $K$. It also uses the *sync* model but does not assume a separate communication phase. That is, in each decision cycle, an agent can either choose to communication or act.

Using information value theory, the value of communication can be defined as the net gain from communicating, which is the difference between the expected improvement in the agents' performance due to communication and the costs associated with communication. When the value of communication is greater than certain threshold, agents can benefit from the communication by sharing their local information. However, computing the exact value of communication is intractable especially in multi-agent systems where communication is constrained and each agent has different partial information about the overall situation. Under the myopic assumption – that communication is only possible at the present time – it is possible to estimate efficiently the value of communication. Becker et al. [7] studied the implications of the myopic assumptions and developed a myopic communication strategy for transition-independent DEC-MDPs. Carlin and Zilberstein [18] extended the work to more general DEC-POMDP models and further improved the performance with non-myopic reasoning. Both works use the *sync* model of communication and determine the communication strategy offline.

### 5.2. Online planning with communication

Online approaches provide an important alternative in which the decision when and what to communicate occurs at execution time. The approaches most similar to ours are Bayesian Game Approximation with Communication (BaGA-Comm) [20] and Avoids Coordination Errors by reasoning over Possible Joint Beliefs with Communication (ACE-PJB-Comm) [42], also known as Dec-Comm [44].

In the BaGA-Comm framework, a series of smaller Bayesian games are constructed and solved using BaGA [21] to generate policies and joint-type spaces at each time-step. An alternating-maximization algorithm is used to find locally optimal solutions for each Bayesian game. To guarantee that each agent has sufficient information to independently construct the same game, a large type space, which corresponds to possible joint histories, is maintained by each agent. In order to exert some control over the size of the type space, a followup method, BaGA-Clustering [22], introduced two types of clustering: Low Probability Clustering (LPC) and Minimum Distance Clustering (MDC). LPC removes the clusters with low probability by merging each one with its nearest remaining neighbor. MDC repeatedly finds the most similar pair of clusters and merges them. Both LPC and MDC use the worst-case expected loss as the similarity measure. Although clustering methods may alleviate the exponential growth of histories in some domains, they do not generally bound the number of histories kept in memory. In the worst case, the size of histories still grows exponentially when limited clustering is possible. Besides, clustering methods are often time-consuming. To further improve the performance, several communication strategies are used to share information among agents. 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. This work has demonstrated that it is important to decide when – not just how often – an agent should communicate to achieve good performance [20].

Unlike BaGA-Cluster, which merges histories based on their similarity in terms of the worst-case expected loss, we merge histories based on 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, our algorithm 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. For communication, Emery-Montemerlo also uses the *sync* model. But rather than transmitting local observations, agents broadcast their current type to the team. However, the type information may not be sufficiently accurate after the lossy clustering procedure is applied.

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 the QPOMDP heuristic function. This work emphasizes the challenge of avoiding coordination error by reasoning over possible joint beliefs. However, the number of possible joint beliefs often grows rapidly beyond what is feasible to store in memory. To address this, Roth et al. utilize a fixed-size method for modeling the distribution of possible joint beliefs using particle filtering. The approach requires only a fixed amount of memory due to the use of sampling. Nevertheless, in many domains, the number of particles needed to accurately model joint beliefs may be large. With the particle representation, agents may initiate too many communications when the real joint belief is not sampled. Our approach can better address these situations because it initiates communication when history *inconsistency* is detected. Basically, communication in Dec-Comm is based on the PD policy, which initiates communication when the policies before and after communication are different. Additionally, Roth et al. use the *tell* model for communication, which allows some subset of agents to broadcast their messages to others. Subsequent work has also addressed the question of what to communicate, which selects the most valuable subset of observations from an agent's observation history, instead of using the entire set as the broadcast message [43].

### 5.3. Other relevant work on communication

Several different aspects of communication within special cases of DEC-POMDPs have been studied in recent years. Roth et al. [45] proposed an algorithm to generate decentralized policies with minimal communication for factored DEC-MDPs. It employs techniques for solving factored MDPs to generate the centralized, free-communication plan for the team offline. At the runtime, each agent executes a factored policy by traversing the policy tree, choosing branches according to the values of the state variables that it encounters, until it reaches an action at a leaf. Communication is needed to facilitate the execution of those portions of the policy without context-specific independence. It utilizes the *query* communication model, where each agent asks its teammates for information when needed.

Spaan and Melo [51] introduced interaction-driven Markov games (IDMGs), assuming communication only occurs at interaction states. It explicitly distinguishes between situations in which agents should interact and situations in which they can act independently. In non-interaction states, each agent chooses its individual actions based on a simple heuristic approach that completely disregards the existence of other agents. In interaction states, each agent communicates its current individual state to the other agents. Then each agent computes a specific Nash equilibrium of the corresponding matrix-game and chooses actions accordingly. Communication is assumed to be unlimited and noise-free.

Williamson et al. [59] introduced the *dec_POMDP_Valued_Com* model, which adds a special communication reward function to DEC-POMDPs. The impact of any communication is measured using KL Divergence – the difference in information in an agent's belief state with and without communication. They extended an online approach called Real Time Belief Space Search (RTBSS) to generate the policy. Early work on RTBSS relied on extensive domain knowledge to encode explicitly how the agents should coordinate, assuming that communication is some parallel activity to other actions [39]. The online approach of Williamson et al. [59] is quite similar to the Dec-Comm algorithm, except that the former algorithm only uses the local observations to update the joint belief states. This work relies on a hand-tuned parameter to value communication as a weighted sum of the communication reward and the original reward. To overcome this, the *RS_dec_POMDP* model approximates this valuation by shaping the reward based on belief divergence [60]. Like the *dec_POMDP_Valued_Com* model, the *RS_dec_POMDP* model still requires a domain-dependent communication reward function, provided by the designer, as an add-on to the general DEC-POMDP model. Most importantly, the belief update in their work is domain-dependent. In general DEC-POMDPs, agents must consider the joint belief space, which takes into account the behaviors of the other agents. This joint belief space blows up exponentially because there are so many choices and outcomes of policies for the other agents, and each agent can only obtain its own observation. Hence we use a policy-based history merging technique to bound memory usage. But for specific domains such as RoboCup rescue, it is possible to design an ad-hoc belief monitoring mechanism so that the algorithm only maintains a single belief state at each step of agents' online decision-making [59].

Oliehoek et al. [34] use a QBG value function [37] to find communication policies for domains where communications have a one-step delay. They model the problem using Bayesian Games and adapt PERSUES, an approximate POMDP solver, to compute QBG value functions. More recent work extends this approach to handle stochastic delays [52]. They present a model which allows communication to be delayed by one or more time steps and explicitly considers future probabilities of successful communication. The Q value function is exact when the communication delays are at most one time step. In situations with delays longer than one time step, agents take coordinated decisions based on some open-loop method. Unlike our work, this approach does not consider when to communicate.

There are many other ways to use communication in multi-agent systems. Stone and Veloso [53] utilize the low-bandwidth communication to do real-time task decomposition and dynamic role assignment. Xuan [61] consider communication in DEC-MDPs whenever an agent notices ambiguity in what it should plan next. Shen et al. [49] formulate the DEC-MDP with a two-layer Bayesian network and find the near-optimal communication strategy for problems with a given structure. Goldman et al. [24] address the problem of potential misbehavior resulting from misinterpretation of messages

exchanged, and establish a formal framework to identify a collection of properties that allow agents to interpret what others are communicating. Some researchers have managed to learn communication policies using reinforcement learning in settings that do not require a complete model to be known [23,54]. Most of these approaches focus on fully-observable multi-agent domains and are based on reinforcement learning, which relies on many trials [16]. The major benefit of communication is to improve coordination since each independent learner may have different models online.

## 6. Conclusions

We present a new online algorithm for planning under uncertainty in multi-agent settings with bounded 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 offline algorithms and it can solve larger problems that are beyond the scope of offline DEC-POMDP planners. Finally, the algorithm performs very well in practice, outperforming the best existing online method by producing better value with less communication.

The performance of online planning is highly dependent on the quality of the value function which guides the behavior of agents when interacting with the environment. The optimal value function is not available at execution time because calculating it may take as much computation time as solving the entire problem. But it can be approximated by some heuristic functions such as the Q-value function of the underlying MDP. The effectiveness of such heuristics is, however, domain dependent. In multi-agent settings, it is challenging to find heuristics that take into account the interdependence among the agents. That is, the actions of one agent may affect the observations of other agents. This is an implicit form of communication, which may be critical for coordination. How to capture this type of information by the heuristic function is an important and interesting research direction for multi-agent online planning.

As discussed earlier, it is intractable to keep all the possible histories of the team. Therefore we introduce a technique for bounding the usage of memory when maintaining the belief pool. Obviously, any such bounding of memory will introduce error into the pool, thereby making the pool inconsistent with the real situation. To the best of our knowledge, we are the first to address this type of inconsistency by explicit communication among agents. The algorithm monitors the belief pool at each step and refreshes the pool by sharing the private information of agents when inconsistency is detected by any of the agents. It is worth mentioning that the effectiveness of our detection method depends on the precision of the agents' observations. We verify the belief pool using each agent's local observation at each step. If the observation is very noisy, it is hard to tell whether the detected inconsistency is due to problems with the belief pool or merely uncertainties in the local observation. In many applications, the sensor data should be accurate enough to detect inconsistencies of the belief pool. If not, the agents may keep silent without exceeding the usage of communication resources. In this paper, we use communication to share local information among agents. Another interesting type of communication is to negotiate the team's policy among the agents. This type of negotiation is nontrivial and beyond the scope of this paper.

More broadly, this paper tries to draw the attention of the AI community to online methods as a viable alternative for multi-agent decision-theoretical planning. We show that online planning with limited communication can perform quite well, while taking much less time. Surprisingly, work on this topic has been sparse [21,22,43,44]. Our work contributes to the literature by presenting a complete framework of multi-agent online planning with two new methods for bounding the usage of both memory and communication resources; it explores several promising research directions for planning and learning in multi-agent systems.

## Acknowledgements

Special thanks to Maayan Roth for sharing her source code of Dec-Comm and to the reviewers for their helpful feedback and suggestions. 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, the National Science Foundation under Grant No. IIS-0812149, the National Science Foundation of China under Grant No. 60745002, and the National High-tech Project of China under Grant No. 2008AA01Z150.

## References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.
[2] C. Amato, D.S. Bernstein, S. Zilberstein, Optimizing memory-bounded controllers for decentralized POMDPs, in: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence, 2007, pp. 1–8.
[3] C. Amato, J.S. Dibangoye, S. Zilberstein, Incremental policy generation for finite-horizon DEC-POMDPs, in: Proceedings of the 19th International Conference on Automated Planning and Scheduling, 2009, pp. 2–9.
[4] C. Amato, S. Zilberstein, Achieving goals in decentralized POMDPs, in: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2009, pp. 593–600.
[5] R. Aras, A. Dutech, F. Charpillet, Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs, in: Proceedings of the 17th International Conference on Automated Planning and Scheduling, 2007, pp. 18–25.
[6] R. Becker, A. Carlin, V. Lesser, S. Zilberstein, Analyzing myopic approaches for multi-agent communication, Computational Intelligence 25 (1) (2009) 31–50.

[7] R. Becker, V.R. Lesser, S. Zilberstein, Analyzing myopic approaches for multi-agent communication, in: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2005, pp. 550–557.
[8] R. Becker, S. Zilberstein, V.R. Lesser, Decentralized Markov decision processes with event-driven interactions, in: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2004, pp. 302–309.
[9] R. Becker, S. Zilberstein, V.R. Lesser, C.V. Goldman, Solving transition independent decentralized Markov decision processes, Journal of Artificial Intelligence Research 22 (2004) 423–455.
[10] D.S. Bernstein, C. Amato, E.A. Hansen, S. Zilberstein, Policy iteration for decentralized control of Markov decision processes, Journal of Artificial Intelligence Research 34 (2009) 89–132.
[11] D.S. Bernstein, E.A. Hansen, S. Zilberstein, Bounded policy iteration for decentralized POMDPs, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005, pp. 1287–1292.
[12] D.S. Bernstein, S. Zilberstein, N. Immerman, The complexity of decentralized control of Markov decision processes, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 2000, pp. 32–37.
[13] A. Beynier, A. Mouaddib, A polynomial algorithm for decentralized Markov decision processes with temporal constraints, in: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, 2005, pp. 963–969.
[14] A. Beynier, A. Mouaddib, An iterative algorithm for solving constrained decentralized Markov decision processes, in: Proceedings of the 21st National Conference on Artificial Intelligence, 2006, pp. 1089–1094.
[15] C. Boutilier, Planning, learning and coordination in multiagent decision processes, in: Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge, 1996, pp. 195–210.
[16] L. Busoniu, R. Babuska, B.D. Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38 (2) (2008) 156–172.
[17] A. Carlin, S. Zilberstein, Value-based observation compression for DEC-POMDPs, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2008, pp. 501–508.
[18] A. Carlin, S. Zilberstein, Myopic and non-myopic communication under partial observability, in: Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2009, pp. 331–338.
[19] J.S. Dibangoye, A. Mouaddib, B. Chaib-draa, Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs, in: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2009, pp. 569–576.
[20] R. Emery-Montemerlo, Game-theoretic control for robot teams, Doctoral Dissertation, Robotics Institute, Carnegie Mellon University, August 2005.
[21] R. Emery-Montemerlo, G.J. Gordon, J.G. Schneider, S. Thrun, Approximate solutions for partially observable stochastic games with common payoffs, in: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2004, pp. 136–143.
[22] R. Emery-Montemerlo, G.J. Gordon, J.G. Schneider, S. Thrun, Game theoretic control for robot teams, in: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005, pp. 1163–1169.
[23] M. Ghavamzadeh, S. Mahadevan, Learning to communicate and act using hierarchical reinforcement learning, in: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2004, pp. 1114–1121.
[24] C.V. Goldman, M. Allen, S. Zilberstein, Learning to communicate in a decentralized environment, Autonomous Agents and Multi-Agent Systems 15 (1) (2007) 47–90.
[25] C.V. Goldman, S. Zilberstein, Optimizing information exchange in cooperative multi-agent systems, in: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2003, pp. 137–144.
[26] C.V. Goldman, S. Zilberstein, Decentralized control of cooperative systems: Categorization and complexity analysis, Journal of Artificial Intelligence Research 22 (2004) 143–174.
[27] E.A. Hansen, D.S. Bernstein, S. Zilberstein, Dynamic programming for partially observable stochastic games, in: Proceedings of the 19th National Conference on Artificial Intelligence, 2004, pp. 709–715.
[28] M.L. Littman, A.R. Cassandra, L.P. Kaelbling, Learning policies for partially observable environments: Scaling up, in: Proceedings of the 12th International Conference on Machine Learning, 1995, pp. 362–370.
[29] J. Marecki, M. Tambe, On opportunistic techniques for solving decentralized Markov decision processes with temporal constraints, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 2007, pp. 825–832.
[30] A.C. Morris, D. Ferguson, Z. Omohundro, D. Bradley, D. Silver, C. Baker, S. Thayer, W. Whittaker, W.R.L. Whittaker, Recent developments in subterranean robotics, Journal of Field Robotics 23 (1) (2006) 35–57.
[31] R. Nair, M. Tambe, M. Roth, M. Yokoo, Communication for improving policy computation in distributed POMDPs, in: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, 2004, pp. 1098–1105.
[32] R. Nair, M. Tambe, M. Yokoo, D.V. Pynadath, S. Marsella, Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence, 2003, pp. 705–711.
[33] R. Nair, P. Varakantham, M. Tambe, M. Yokoo, Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs, in: Proceedings of the 20th National Conference on Artificial Intelligence, 2005, pp. 133–139.
[34] F.A. Oliehoek, M.T.J. Spaan, N. Vlassis, Dec-POMDPs with delayed communication, in: The 2nd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, 2007.
[35] F.A. Oliehoek, M.T.J. Spaan, N. Vlassis, Optimal and approximate q-value functions for decentralized POMDPs, Journal of Artificial Intelligence Research 32 (2008) 289–353.
[36] F.A. Oliehoek, M.T.J. Spaan, S. Whiteson, N.A. Vlassis, Exploiting locality of interaction in factored Dec-POMDPs, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, 2008, pp. 517–524.
[37] F.A. Oliehoek, N. Vlassis, Q-value functions for decentralized POMDPs, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 2007, pp. 833–840.
[38] F.A. Oliehoek, S. Whiteson, M.T.J. Spaan, Lossless clustering of histories in decentralized POMDPs, in: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2009, pp. 577–584.
[39] S. Paquet, L. Tobin, B. Chaib-draa, An online POMDP algorithm for complex multiagent environments, in: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2005, pp. 970–977.
[40] L. Peshkin, K.-E. Kim, N. Meuleau, L.P. Kaelbling, Learning to cooperate via policy search, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 2000, pp. 489–496.
[41] D.V. Pynadath, M. Tambe, The communicative multiagent team decision problem: Analyzing teamwork theories and models, Journal of Artificial Intelligence Research 16 (2002) 389–423.
[42] M. Roth, Execution-time communication decisions for coordination of multi-agent teams, Ph.D. thesis, The Robotics Institute, Carnegie Mellon University, 2007.
[43] M. Roth, R. Simmons, M. Veloso, What to communicate? Execution-time decision in multi-agent POMDPs, in: Proceedings of the 8th International Symposium on Distributed Autonomous Robotic Systems, 2006.
[44] M. Roth, R.G. Simmons, M.M. Veloso, Reasoning about joint beliefs for execution-time communication decisions, in: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, 2005, pp. 786–793.

[45] M. Roth, R.G. Simmons, M.M. Veloso, Exploiting factored representations for decentralized execution in multiagent teams, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 2007, pp. 457–463.
[46] S. Seuken, S. Zilberstein, Improved memory-bounded dynamic programming for decentralized POMDPs, in: Proceedings of the 23rd Conference in Uncertainty in Artificial Intelligence, 2007, pp. 344–351.
[47] S. Seuken, S. Zilberstein, Memory-bounded dynamic programming for DEC-POMDPs, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007, pp. 2009–2015.
[48] S. Seuken, S. Zilberstein, Formal models and algorithms for decentralized decision making under uncertainty, Journal of Autonomous Agents and Multi-Agent Systems 17 (2) (2008) 190–250.
[49] J. Shen, V.R. Lesser, N. Carver, Minimizing communication cost in a distributed bayesian network using a decentralized mdp, in: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2003, pp. 678–685.
[50] M.T.J. Spaan, G.J. Gordon, N. Vlassis, Decentralized planning under uncertainty for teams of communicating agents, in: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, 2006, pp. 249–256.
[51] M.T.J. Spaan, F.S. Melo, Interaction-driven Markov games for decentralized multiagent planning under uncertainty, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, 2008, pp. 525–532.
[52] M.T.J. Spaan, F.A. Oliehoek, N. Vlassis, Multiagent planning under uncertainty with stochastic communication delays, in: Proceedings of the 18th International Conference on Automated Planning and Scheduling, 2008, pp. 338–345.
[53] P. Stone, M.M. Veloso, Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork, Artificial Intelligence 110 (2) (1999) 241–273.
[54] D. Szer, F. Charpillet, Improving coordination with communication in multi-agent reinforcement learning, in: Proceedings of the 6th IEEE International Conference on Tools with Artificial Intelligence, 2004, pp. 436–440.
[55] D. Szer, F. Charpillet, Point-based dynamic programming for DEC-POMDPs, in: Proceedings of the 21st National Conference on Artificial Intelligence, 2006, pp. 1233–1238.
[56] D. Szer, F. Charpillet, S. Zilberstein, Maa*: A heuristic search algorithm for solving decentralized POMDPs, in: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, 2005, pp. 576–590.
[57] M. Tambe, Towards flexible teamwork, Journal of Artificial Intelligence Research 7 (1997) 83–124.
[58] J. Tsitsiklis, M. Athans, On the complexity of decentralized decision making and detection problems, IEEE Transaction on Automatic Control 30 (1985) 440–446.
[59] S.A. Williamson, E.H. Gerding, N.R. Jennings, A principled information valuation for communication during multi-agent coordination, in: The 3rd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, 2008.
[60] S.A. Williamson, E.H. Gerding, N.R. Jennings, Reward shaping for valuing communications during multi-agent coordination, in: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems, 2009, pp. 641–648.
[61] P. Xuan, V. Lesser, S. Zilberstein, Communication decisions in multi-agent cooperation: Model and experiments, in: Proceedings of the 5th International Conference on Autonomous Agents, 2001, pp. 616–623.