Online Planning for Ad Hoc Autonomous Agent Teams

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

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

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

Abstract

We propose a novel online planning algorithm for ad hoc team settings—challenging situations in which an agent must collaborate with unknown teammates without prior coordination. Our approach is based on constructing and solving a series of stage games, and then using biased adaptive play to choose actions. The utility function in each stage game is estimated via Monte-Carlo tree search using the UCT algorithm. We establish analytically the convergence of the algorithm and show that it performs well in a variety of ad hoc team domains.

1 Introduction

Collaboration without prior coordination is a recognized challenge in multi-agent systems research [Stone *et al.*, 2010]. Consider, for example, a passenger arriving at a foreign airport for the first time, not knowing the native language. An autonomous robot is deployed at the airport to provide services for passengers. It can help passengers with common tasks such as baggage pickup or locating a boarding gate. The robot has no prior knowledge of each passenger's needs, but should be able to collaborate with a passenger and help perform some common tasks. This is an example of an *ad hoc team* setting, where the fundamental objective is to collaborate without pre-coordination [Stone *et al.*, 2009; 2010; Stone and Kraus, 2010; Barret *et al.*, 2011]. The robot may know in advance the airport facilities, but not the specific needs and preferences of a passenger. The challenge is to create such an autonomous agent that can efficiently and robustly work with other agents on tasks that require teamwork. In practice, many human-robot teams are ad hoc. Applications include rescue robots brought to an earthquake site from different parts of the world, or e-commerce cooperative agents created by different companies with different standards.

Planning under uncertainty for teams of agents has been widely studied using various mathematical models such as Multi-Agent MDPs (MMDPs) [Boutilier, 1999] and Decentralized POMDPs (DEC-POMDPs) [Bernstein *et al.*, 2000]. Despite recent progress, most of the existing approaches rely on substantial pre-coordination. Some assume that planning can take place completely offline [Seuken and Zilberstein, 2008]. Others allow agents to coordinate online, but assume that they employ identical planners and pre-determined coordination protocols [Wu *et al.*, 2011]. In ad hoc teams, agents must cooperate with unknown teammates such as random persons or robots programmed by different people. Such collaboration without pre-coordination is becoming increasingly important in multi-agent systems research.

We focus on a certain type of ad hoc teams in which a target agent knows the number of teammates as well as a set of their feasible actions. The system state and the joint action played at each step are fully observable by the agent. However, local features of an individual teammate such as sensing, acting, communicating and decision-making capabilities are hidden from the agent. This represents a large array of challenging ad hoc team problems, particularly scenarios that involve random people. In the above example, the robot may have information about the airport and the range of actions that a human may perform. But it cannot have a model characterizing each person. The only way for the robot to know people is by interacting with them and observing their behaviors. To succeed in such ad hoc settings, an agent must reason about the interaction history and adapt its actions to its teammates.

We propose a novel online planning algorithm for ad hoc teams. In our approach, the planning problem is approximated by solving a series of *stage games*, one for each time step. In each game, we use *biased adaptive play* (BAP) [Wang and Sandholm, 2003], which is a variant of *fictitious play* in game theory. Originally, BAP was designed to maintain coordination in fully cooperative repeated games. It is an appealing approach to ad hoc team settings because it is rational and convergent even in the presence of heterogeneous agents. We extend BAP to ad hoc team problems and analyze its performance both theoretically and empirically. When constructing each stage game, the utility function is estimated by *Monte-Carlo tree search*, using the UCT algorithm [Kocsis and Szepesvári, 2006]. UCT is a Monte-Carlo method for planning in large domains. It has outperformed previous approaches in challenging games such as Go [Gelly and Silver, 2007]. The key advantage of a Monte-Carlo method is that it requires only a generative model—a black box simulator—making it particularly suitable for our setting. We extend UCT to search the large policy space of unknown teammates. The underlying MMDP is used as a generative model. Given

*Corresponding Author at Univ. of Sci. & Tech. of China

a state and a joint action, it outputs samples for UCT, including the next state and reward. The key contribution of our work are a general framework for planning in ad hoc team, and the first algorithm that combines the advantages of BAP and UCT in these settings.

The paper is organized as follows. We first describe the essentials of MMDPs and the ad hoc team settings that we target. Next we introduce the online planning algorithm and discuss its properties. Then we present experimental results on several domains, and conclude with a summary of the contributions and future work.

2 Background

In this section, we review the Multi-agent Markov Decision Process (MMDP) framework [Boutilier, 1999] and introduce the setting of ad hoc agent teams. Although our ultimate goal is to build a single autonomous agent, teamwork considerations are essential for the agent to succeed in this setting.

2.1 The MMDP Model

Formally, a Multi-agent Markov Decision Process (MMDP) is defined as a tuple $\langle I, S, \{A_i\}, P, R, s^0, \gamma \rangle$, where:

- I is a set of agents, identified by $i \in \{1, 2, \cdots, n\}$.
- S is a set of states where $s \in S$, and s^0 is the initial state.
- A_i is a set of actions for agent i where $a_i \in A_i$, and $A = \times_{i=1}^{n} A_i$ is the set of joint actions where $\vec{a} \in A$.
- $P : S \times A \times S \to [0, 1]$ is the transition function, where $P(s'|s, \vec{a})$ denotes the probability of transiting to a new state s' when the agents take joint action $\vec{a}$ in state s.
- $R : S \times A \to \Re$ is the reward function, where $R(s, \vec{a})$ denotes the reward received by the team when the agents take joint action $\vec{a}$ in state s.
- γ is the discount factor where $0 < \gamma \leq 1$.

At every decision cycle t, each agent i independently chooses an action $a_i \in A_i$ and the joint action of the team $\vec{a} = \langle a_1, a_2, \cdots, a_n \rangle$ is executed in the environment. As the system moves from state s^t to state s^{t+1} according to the transition function, all the agents receive an identical reward. The goal of the agent team is to determine a control strategy called joint policy that maximizes the expected long-term accumulated reward of the team: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s^t, \vec{a}^t) | s^0\right]$.

A *local policy* for agent i is a mapping $\pi_i : S \times A_i \to [0, 1]$ where $\pi_i(a_i|s)$ defines the probability of agent i taking action a_i when the system state is s. A *joint policy* is a vector $\pi = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$ of local policies, one for each agent. Given a joint policy π, the value of each state $s \in S$ can be defined recursively using the so-called Bellman equation:

$$V^{\pi}(s) = \sum_{\vec{a} \in A} \pi(\vec{a}|s) \left[R(s, \vec{a}) + \gamma \sum_{s' \in S} P(s'|s, \vec{a}) V^{\pi}(s') \right].$$

Similarly, the Q-function is defined as follows, for each joint action $\vec{a}$ and state s, when the agents follow joint policy π:

$$Q(s, \vec{a}) = R(s, \vec{a}) + \gamma \sum\nolimits_{s' \in S} P(s'|s, \vec{a}) V^{\pi}(s'). \quad (1)$$

It is worth pointing out that the MMDP model represents a fully cooperative decision problem, since the reward function is common. In other words, all the agents share the same goal of maximizing the total expected reward of the team.

2.2 Ad Hoc Agent Teams

In *ad hoc team* settings, a group of agents with different decision models and capabilities must cooperate so as to complete common tasks, yet they have no prior opportunity to coordinate Stone *et al.* [2010]. Agents in ad hoc teams are likely to have heterogeneous sensing and acting capabilities that may not be common knowledge. Furthermore, they may have different communication protocols or world models. Thus neither pre-coordination nor execution-time negotiation are feasible. Rather, an agent must reason about the interaction with different teammates and adjust its behavior to cooperate with them on the fly. Although this problem pertains to teamwork, the fundamental challenge is to build a single autonomous agent with such capabilities, not an entire team.

Ad hoc teams represent a very broad concept in multi-agent systems. Agents in these settings must be prepared to collaborate with varying types of teammates. In ad hoc *human* team formation [Kildare, 2004], teamwork situations are organized along three dimensions: teammate, team and task characteristics. To simplify the challenge, it is necessary to initially limit the scope of these characteristics [Stone *et al.*, 2010]. In this paper, we restrict ourselves to settings where: (1) the system state is fully observable by the agent; (2) the agent knows the number of teammates as well as a set of their feasible actions; (3) a generative model (simulator) of the ad hoc team is available for drawing samples of team actions; and (4) at the end of each step, the agent receives an observation of the joint action taken by the team. The team can be homogeneous or heterogeneous and has no direct communication.

We make no assumptions about the teammates' acting and sensing capabilities, or their decision models. Teammates may even be unaware that they take part in an ad hoc team. The space of teammates' feasible actions includes all the actions that the agent can perceive. In practice, an agent must be able to recognize these actions in order to reason about teammates' behaviors. The generative model, represented as an MMDP, specifies the task characteristics. In some applications such as human-robot interaction, a closed-form representation of the system dynamics may not be available, but a black-box simulator (e.g., USARSim [Lewis *et al.*, 2007]) often exists. The policy space of teammates is typically very large. Without pre-coordination, the agent must reason about the past action sequences of its teammates online, learn quickly from these interactions, and act accordingly.

3 Online Planning for Ad Hoc Teams

In this section we propose the *Online Planning for Ad Hoc Agent Teams* (OPAT) algorithm. Online planning provides a convenient mechanism for reasoning about the interaction as it unfolds, and making decision during execution time. An agent operating as part of an ad hoc team must learn to cooperate with different types of teammates, taking into account future steps when optimizing its current action.

3.1 Overview of the Online Planning Algorithm

Our online planner interleaves planning and execution [Brenner and Nebel, 2009], selecting one action at a time for the current step. To estimate the value of each action, it performs forward search. In this planning process, the agent must consider the strategies of its teammates and reason about all the possible outcomes. Intuitively, a selfish policy can be arbitrarily bad if it ignores cooperation; a robot cannot help passengers if it ignores their reactions. The robot must be prepared to adjust its behavior based on the interaction.

Definition 1. A stage game $\Gamma_s = \langle I, \{A_i\}, \{Q_i\}\rangle$ is a normal-form game where I is a set of players, A_i is a set of actions and Q_i is the utility function for player $i \in I$. It is cooperative when all players have the same utility function.

From a game-theoretic standpoint, ad hoc teamwork can be modeled as a large stochastic game with unknown players [Leyton-Brown and Shoham, 2008]. However, the resulting game can be intractable, particularly with long horizons. To address that, our online planner generates actions by approximately constructing and solving smaller games called *stage games*, one game, Γ_s, for every state $s \in S$. At each step, the agent performs game-theoretic reasoning using its current knowledge of the state and teammates, much like the classical one-step lookahead approach to game playing. Thus, the overall problem is decomposed into a series of smaller games. Performance depends on how each stage game is constructed and how the agents choose actions.

To build a stage game, it is necessary to define a utility function $Q(\vec{a})$ that ideally should capture not only the immediate payoff of a joint action $\vec{a}$, but also its expected future value. State-of-the-art techniques for online planning often rely on heuristics to evaluate the resulting outcomes of joint actions. Unfortunately, it is hard to determine good heuristics for ad hoc teams due to the lack of knowledge about teammates. Such a heuristic must inherently vary when used in conjunction with different teammates. To develop a general framework for ad hoc teams, we incorporate Monte-Carlo tree search to estimate the payoff of each joint action.

After constructing a stage game, the key question is how to reason about the interaction and play adaptively. This is challenging because the characteristics of teammates are hidden and the only clue is the interaction history. In this work, we use *biased adaptive play* to create a *virtual game* $\hat{\Gamma}_s$ based on a set of "optimal" joint actions according to the estimated value of the tree search. The best response action of the virtual game is chosen by sampling the interaction history.

As shown in Algorithm 1, agent i interacts with other members of the ad hoc team online by: (1) estimating the value of each joint action with Monte-Carlo tree search; (2) constructing a stage game and a virtual game; and (3) choosing an action using biased adaptive play. The following subsections explain each part in detail.

Algorithm 1: Online Planning for Ad Hoc Teams

```
procedure OnlinePlannig(T)
    h ← ∅, r ← 0, s ← s^0 // start with state s^0.
    for t=1 to T do
        if agent i then // run online in parallel.
            foreach a⃗ ∈ A do
                Q(s, a⃗) ← TreeSearch(s, a⃗, h)
            Γ_s ← StageGame(s, Q)
            Γ̂_s ← VirtualGame(Γ_s)
            a_i ← BiasedAdaptivePlay(Γ̂_s, h_m[s])
        else
            a_{-i} ← TeammateActions()
        h[s] ← h[s] ∘ a⃗, (s', r') ~ Environment(s, a⃗)
        r ← r + γ^t · r', s ← s'
    return r // the total accumulated reward.
```

3.2 Adapting to Strategies of Teammates

To play adaptively with unknown teammates, we borrow some game-theoretic ideas. MMDPs are equivalent to *team Markov games* $\langle I, S, \{A_i\}, P, \{R_i\}, \gamma\rangle$ where all players share an identical payoff: $\forall i, j \in I, R_i = R_j$. Generally, a team Markov game can be decomposed into a set of *stage games* $\Gamma_s = \langle I, \{A_i\}, Q\rangle$, one for each state $s \in S$, where $Q(\vec{a})$ is the the utility function. Each stage game is also fully cooperative, just like team Markov games. It is known that a team Markov game converges to a Pareto-optimal Nash equilibrium if all the players coordinate in an optimal Nash equilibrium at each stage game $\Gamma_s, \forall s \in S$ [Boutilier, 1996].

Biased adaptive play (BAP) [Wang and Sandholm, 2003] is a variant of *fictitious play*. It relies on the observation that the past action choices of other agents can be used to estimate their policies. Given a stage game Γ_s, BAP constructs a *virtual game* $\hat{\Gamma}_s = \langle I, \{A_i\}, \hat{Q}\rangle$, where $\hat{Q}(\vec{a}) = 1$ if $\vec{a}$ is an optimal joint action for Γ_s and $\hat{Q}(\vec{a}) = 0$ otherwise. We denote $A^* = \{\vec{a} \in A | \hat{Q}(\vec{a}) = 1\}$ the optimal joint action set in $\hat{\Gamma}_s$ and $H^t_m = (\vec{a}^{t-m+1}, \cdots, \vec{a}^{t-1}, \vec{a}^t)$ the m most recent plays in Γ_s at time t. Given the virtual game $\hat{\Gamma}_s$ and the m most recent plays H^t_m, the main procedure of BAP is shown in Algorithm 2, which returns agent i's best response action.

If conditions $\mathcal{C}_1$ or $\mathcal{C}_2$ of Algorithm 2 are met, agent i concludes that all teammates have coordinated in an action of A^*. Therefore, it selects the best-response action a^*_i with respect to the optimal joint action set A^*. If none of these conditions hold, agent i estimates the expected payoff $EP(a_i)$ for each of its actions $a_i \in A_i$ using the K samples of H^k_m. Intuitively, $EP(a_i)$ is an estimate of the action a_i given the average policy of its teammates according to H_k. The action trace played by teammates can be viewed as an indication of their policy. Using the m most recent plays instead of the entire interaction allows BAP to cope with situations where teammates are also adjusting their strategies.

The good property of BAP is that, even in a group with heterogeneous agents such as an ad hoc team, it is still able to converge to the best-response policy. In ad hoc teams, teammates may be constrained (e.g. having reduced action sets) and thus follow some sub-optimal strategies. The auxiliary virtual games $\hat{\Gamma}_s$ in BAP are weakly acyclic with respect to the bias set A^* that contains only optimal policies. This ensures that agents with BAP will eventually converge to either a strict Nash equilibrium of $\hat{\Gamma}_s$ or a Nash equilibrium in A^*

Algorithm 2: Generative Biased Adaptive Play

procedure BIASEDADAPTIVEPLAY($\hat{\Gamma}_s, H_m^t$)
  if $|H_m^t| < m$ **then** `// generate more samples.`
    $H_m^t \leftarrow$ SAMPLEPLAY(s, H_m^t, m)
  $H_k \leftarrow$ DRAWKSAMPLES(K, H_m^t)
  if $\exists \vec{a}^* \in A^*, \forall \vec{a} \in H_k, (a_{-i} = a^*_{-i})$ **then** `//` $\mathcal{C}_1$
    $(a_i^*, a^*_{-i}) \leftarrow \arg\max_{\vec{a}^{t'}} \{t' | \vec{a}^{t'} \in H_k \wedge \vec{a}^{t'} \in A^*\}$
  else if $\exists \vec{a} \in A, (\vec{a} \in H_k \wedge \vec{a} \in A^*)$ **then** `//` $\mathcal{C}_2$
    $a_i^* \sim \{a_i \in A_i | (a_i, a^*_{-i}) \in A^*\}$
  else
    $\forall a_{-i}, N(a_{-i}) \leftarrow$ COUNTTIMES($a_{-i} \in H_k$)
    $\forall a_i, EP(a_i) \leftarrow \sum_{a_{-i} \in A_{-i}} \hat{Q}(a_i, a_{-i}) \frac{N(a_{-i})}{K}$
    $a_i^* \sim \{a_i \in A_i | EP(a_i) = \max_{a_i' \in A_i} EP(a_i')\}$
  return a_i^* `// the best-response action.`
procedure SAMPLEPLAY(s, H_m^t, m)
  $N(s) \leftarrow |H_m^t| + 1$
  foreach $\vec{a} \in A$ **do**
    $N(s, \vec{a}) \leftarrow$ COUNTTIMES($\vec{a} \in H_m^t$) $+ 1$
  for k=$|H_m^t|$ **to** m **do**
    $\vec{a} \leftarrow \arg\max_{\vec{a}' \in A} \hat{Q}(s, \vec{a}') + c\sqrt{\frac{\ln N(s)}{N(s, \vec{a}')}}$
    $N(s) \leftarrow N(s) + 1, N(s, \vec{a}) \leftarrow N(s, \vec{a}) + 1$
    $H_m^t \leftarrow H_m^t \circ \vec{a}$
  return H_m^t `// the augmented joint history.`

with probability 1 [Wang and Sandholm, 2003].

One weakness of the original BAP method is that a state s must be visited before one can sample from the action history $h[s]$. This can be satisfied in repeat games for which BAP was designed, but not in ad hoc settings, especially during the early stages of interaction or when the state space is very large. To cope with this, we sample joint actions from the *feasible* space using the UCB1 heuristic [Auer *et al.*, 2002], which balances the tradeoff between the value of a joint action and the frequency of choosing it. Other heuristics such as the action histories of "nearby" states are useful when a suitable distance measure is known [Melo and Ribeiro, 2008].

The construction of a stage game Γ_s requires that agent i has the knowledge of the optimal Q-function, that is, the utility of Γ_s. It is worth noting that this is different from the optimal Q-function of the MMDP and cannot be computed offline since the characteristics of the teammates are unknown. We therefore estimate the utility of Γ_s using Mote-Carlo tree search, as detailed in the next subsection.

3.3 Planning by Monte-Carlo Tree Search

The Monte-Carlo rollout method provides a simple solution for estimating the value of state s using random policies. It generates sequences of states using a simulator starting with s until reaching a terminal state or the planning horizon. The value of state s is estimated by the mean return of K trials starting from s. Intuitively, if we sample the state s infinitely often using an optimal policy, the averaged return will converge to the optimal expected value of s. Hence the performance of the rollout method can be improved by sampling actions selectively. To achieve this, Monte-Carlo tree search builds a lookahead tree in a sequentially best-first order. The nodes of the tree keep track of estimates of action values at the sampled states from earlier simulations. The value gathered during a simulation is added to the nodes incrementally. Therefore, if some state is visited again, the estimated action values can be used to bias the choice of actions at that state.

Instead of greedily selecting the action with the highest value, the UCT method [Kocsis and Szepesvári, 2006] chooses actions using the UCB1 heuristic [Auer *et al.*, 2002]. Each node is viewed as a separate multi-armed bandit where the arms correspond to actions and the payoffs to the estimated value. As Kocsis and Szepesvári show, the probability of selecting the optimal action converges to 1 as the number of samples grows. The UCT algorithm balances between testing the current best action and exploring alternatives to ensure that no good actions are overlooked due to early estimation errors. Domain knowledge can be used to initialize the tree nodes in the UCT algorithm. It narrowly focuses the search on promising states and thereby speeds up convergence [Gelly and Silver, 2007].

The balance between exploration and exploitation is important when searching over the teammates' policy space. Instead of searching the entire policy space of the MMDP, we want to concentrate on regions of actions that are more likely to be played by teammates. Meanwhile, we do not want to miss other actions that teammates may play in the future. Therefore, BAP is used at each step of the tree search to bias the action selection based on the interaction history. A stage game Γ_s is maintained for each tree node and updated with new values $Q(s, \cdot)$ when the node is reencountered.

In Algorithm 3, a tree node is generated for each encountered state s. Each node contains a value $Q(s, \vec{a})$ and a visitation count $N(s, \vec{a})$ for each joint action $\vec{a}$. The overall visitation count for state s is $N(s) = \sum_{\vec{a} \in A} N(s, \vec{a})$. Additionally, a stage game Γ_s is constructed for each node and updated each time the node is reencountered. In the UCB1 heuristic, the augmented term $c\sqrt{\frac{\log N(s)}{N(s, \vec{a}')}}$ is an exploration bonus that is highest for rarely tried actions; the scalar constant c is the relative ratio of exploration to exploitation. When a node is not in the search tree, the search tree is expanded and the rollout method is used to estimate the value. The search depth T is chosen based on the available online planning time.

3.4 Discussion and Analysis

In principle, it is straightforward to model an ad hoc team as a DEC-POMDP with heterogeneous actions and observations. For agent i, the observation set can be defined as: $\Omega_i = \{(s, a_{-i}) | s \in S, a_{-i} \in A_{-i}\}$ —pairs of states and teammates' actions, with the following observation function: $\forall (s', a'_{-i}) \in \Omega_i, O((s', a'_{-i}) | s, \vec{a}) = \delta(s' = s) \cdot \delta(a'_{-i} = a_{-i})$ where $\delta(p)$ is 1 if p is *true* and 0 otherwise. However, in ad hoc team settings, some knowledge about the teammates is hidden from the planner. Thus, most existing approaches for DEC-POMDPs do not apply to ad hoc team settings. In this paper, we assume that the system state is fully observable by the agent so we can focus on playing adaptively with many types of teammates. Our work can be extended to partially

Algorithm 3: Biased Monte-Carlo Tree Search

procedure TREESEARCH($s, \vec{a}, h$)
  for k=1 **to** K **do**
    $h'[s] \leftarrow h[s] \circ \vec{a}, (s', r) \sim \text{MMDP}(s, \vec{a})$
    $r_k \leftarrow r + \gamma \cdot \text{SIMULATION}(s', h', 0)$
  return $\frac{1}{K} \sum_{k=1}^{K} r_k$ `// the averaged value.`
procedure SIMULATION(s, h, t)
  if $t > T$ **return** 0 `// reach the search depth.`
  if $s \notin$ TREE **then**
    TREE $\leftarrow$ INITIALIZE(s, Q, N, Γ_s)
    return ROLLOUT(s, t)
  $\hat{\Gamma}_s \leftarrow \text{VIRTUALGAME}(\Gamma_s)$
  $a_i \leftarrow \text{BIASEDADAPTIVEPLAY}(\hat{\Gamma}_s, h_m[s])$
  $\vec{a} \leftarrow \arg\max_{\vec{a}'|a_i, a_{-i} \in A_{-i}} Q(s, \vec{a}') + c\sqrt{\frac{\log N(s)}{N(s,\vec{a}')}}$
  $h'[s] \leftarrow h[s] \circ \vec{a}, (s', r') \sim \text{MMDP}(s, \vec{a})$
  $r \leftarrow r' + \gamma \cdot \text{SIMULATION}(s', h', t+1)$
  $N(s) \leftarrow N(s) + 1, N(s, \vec{a}) \leftarrow N(s, \vec{a}) + 1$
  $Q(s, \vec{a}) \leftarrow Q(s, \vec{a}) + \frac{r - Q(s,\vec{a})}{N(s,\vec{a})}$
  $\Gamma_s \leftarrow \text{STAGEGAME}(s, Q)$
  return r `// the long-term reward.`
procedure ROLLOUT(s, t)
  if $t > T$ **return** 0
  $\vec{a} \sim \pi_{\text{ROLLOUT}}(\cdot|s), (s', r) \sim \text{MMDP}(s, \vec{a})$
  return $r + \gamma \cdot \text{ROLLOUT}(s', t+1)$

observable domains by replacing the current state with a belief state or internal memory state.

Proposition 1. *Given an infinite number of samples, OPAT will converge to the best-response actions when all the teammates play an optimal MMDP policy.*

Proof (sketch). As shown by Wang and Sandholm [2003], BAP converges in the limit to either a strict Nash equilibrium or a Nash equilibrium in A^*. The best-response action set A^* is determined by the estimated values of UCT. The policy space searched by UCT is constrained by the choice of agent i's actions, which depends on the past action sequences of the entire team. If all teammates coordinate by using an optimal policy of the underlying MMDP, BAP is convergent and will eventually select the best-response actions. Thus agent i's actions in conjunction with its teammates' actions constitute an optimal joint policy for the entire team. According to Kocsis and Szepesvári [2006], the value function returned by UCT is optimal given infinite samples. Therefore, OPAT returns the best-response actions by induction. □

A backup method is used in OPAT when no history of a particular state exists. It assumes that the other agents will act sensibly until it learns otherwise. This is important for ad hoc teams because an agent's initial actions could affect the behaviors of other actors. In some sensitive domains involving interaction with humans, it might be safer to prudently perform actions that are less reliant on the teammates while obtaining some understanding of their policies. However, this could be misinterpreted by the teammates and delay useful collaboration. While our general framework does not consider the learning capabilities of ad hoc teammates, this can be addressed by incorporating different backup methods.

The design of an ad hoc agent and its performance greatly depend on how much information is known about the domain and potential teammates. In this work, we assume the domain is known, while the behavior of teammates is unknown. They can be rational, irrational or somewhere in between. A typical domain is the pick-up soccer game where every player knows the rules of the game, but has no idea about the strategies of the other players and no chance to pre-coordinate.

Some parameters of OPAT depend on the characteristics of the teammates. The number of actions retrieved from history was originally introduced by BAP for adapting to teammates' behaviors. Presumably lower values will work better when the teammates are adapting in response to the agent's actions. The value function of the underlying MMDP can be used as domain knowledge to initialize the tree nodes in UCT and bias the choice of actions. It should be noted that the MMDP values are overestimates and may not be useful when the teammates are irrational. Long term autonomy can be achieved by considering the discounted rewards in UCT. This can be done simply by stoping the search when reaching some depth D with $\gamma^D < \epsilon$ where γ is a discount factor and ϵ is a relatively small number.

4 Experiments

Our experimental evaluation follows the paradigm introduced by Stone *et al.* [2010]. We chose a set of problem domains and designed a variety of ad hoc teams with different agents. Then, we randomly replaced an agent in each team with the agent running the OPAT algorithm and recorded the team performance, averaging results over 100 runs. The selected domains are common benchmark problems from the DEC-POMDP literature[1], offering different initial conditions that can be sampled. Performance is measured by the joint accumulated reward over some fixed time T. In the selected domains, coordination is essential for success.

We created two types of teams with different unknown teammates (UTM) in terms of their acting and sensing capabilities: UTM-1 agents play a sequence of actions according to some pre-defined patterns, while UTM-2 agents maintain a belief based on their observations and choose actions using the optimal Q-function. The pre-defined patterns are sequences of random actions with some fixed repeated lengths that are randomly chosen at the beginning of each run. For example, given a random repetition value of 2, the action pattern may be "AACCBBEE" where "ACBE" are random actions. Some pre-defined random seeds are used to guarantee that each test had the same action sequences. Note that in ad hoc teams, agents may not be aware that they take part in a team or they may not be capable of working efficiently with other agents. UTM-1 agents are referred to as "irrational" since they don't follow the optimal course of actions. The goal of UTM-1 is to test if OPAT can adapt well when its teammates' policies are not particularly effective in advancing the joint goal. For UTM-2 agents, we simulate different sensing

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

Table 1: Results for Different Ad Hoc Teams

Ad Hoc Teammate	OPAT	MMDP	RAND
Cooperative Box Pushing T=20, (-41.71, -20.16)			
UTM-1	35.25	-25.97	-29.62
UTM-2	23.30	-21.65	-26.33
Meeting in 3×3 Grid T=20, (1.75, 1.87)			
UTM-1	5.20	3.14	1.47
UTM-2	6.30	4.65	1.98
Multi-Channel Broadcast T=20, (5.45, 17.40)			
UTM-1	13.15	10.15	9.25
UTM-2	7.15	17.40	9.30

capabilities by varying the level of noise in teammates' observations. Generally, UTM-1 agents are irrational with respect to teamwork while UTM-2 agents are rational, but having only partial observations of the system state.

Table 1 shows the results of an agent running OPAT and the *optimal* MMDP policy computed offline by value iteration. Results are also provided for an agent executing random policies. Note that the MMDP agent is very competitive because it fully observes the system state and knows the optimal value of each joint action in every state. The pair of values after each domain name are the expected value of a team with two UTM-1 agents (denoted by *) and a team with two UTM-2 agents (denoted by $^+$), without replacing agents.

Figure 1 shows a comparison of OPAT when the teammate runs an identical copy of OPAT or the optimal MMDP policy. The goal is to show the performance of OPAT when all teammates are rational and have the same capabilities. Note that OPAT coordinates well in different settings without knowing the teammates' capabilities or policies. Its performance improves substantially by increasing the number of samples. The values of "MMDP-vs-MMDP" and "MMDP-vs-RAND" remain constant for different sample sizes because neither agents uses samples for decision making. They serves as the upper and lower bounds on the values in this domain.

Although all the tested domains are common DEC-POMDP benchmarks, we did not compare OPAT with existing DEC-POMDP approaches because the settings are entirely different: we assume full observability in these experiments and that offline computation of policies is not feasible.

Cooperative Box Pushing represents domains where miscoordination is very costly (e.g. collision with other agents or pushing a large box individually). To succeed, the agent must adjust its policy to avoid penalties from unexpected teammates' actions. As shown in Table 1, OPAT outperforms MMDP and gets positive rewards in both ad hoc teams. Meeting in 3×3 Grid represents problems with multiple ways of coordination (e.g. meeting in any grid location). OPAT again performs quite well. The results for the partial sensing team (UTM-2) in Multi-Channel Broadcast show a limitation of OPAT to be addressed in the future: it has no explicit model of the teammates' sensing capabilities as it directly maps teammates' actions to states. Another reason is that in this domain, only one node can broadcast at a time; the goal is to minimize

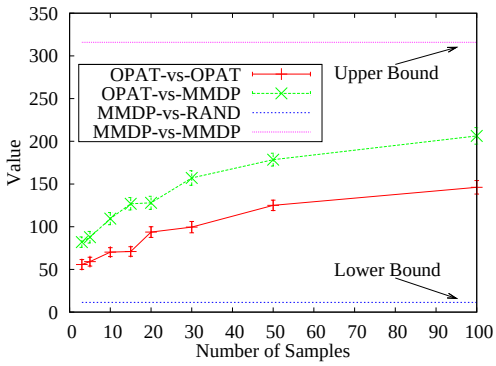

Figure 1: Comparison of different pairs of algorithms on the Cooperative Box Pushing domain (T=20).

collisions and maximize throughput. RAND broadcasts messages randomly regardless of collisions. Intuitively, OPAT may wait for its teammate to broadcast in order to avoid collisions, but sometimes the teammate fails to broadcast due to its noisy sensing. Hence the channel is more likely to be idle. On the other hand, MMDP makes very strong assumptions about its teammate's behavior and will broadcast according to the optimal Q-function. On average, MMDP performs better than RAND. In fact, in the experiments, we observed the following average channel idle time: OPAT 62.25%, RAND 52%, and MMDP 9.75%.

The computational cost of OPAT depends mainly on the search depth and the sample size of UCT. One of the key advantages of Monte-Carlo methods is that they scale very well over the state space. For problems with many agents, more sophisticated techniques should be used to search over the large joint action space. The sample size of OPAT provides a good tradeoff between value and runtime. Intuitively, the runtime will increase when larger sample size is chosen. In the experiments, we observed the following average runtime of OPAT for each domain (seconds per step): Box-Pushing 0.059, Meeting-in-Grid 0.122, Broadcast-Channel 0.008. As Figure 1 shows, the value grows with the number of samples when running with MMDP or OPAT teammates. As indicated by Proposition 1, it converges to the optimal value as the number of samples goes to infinity.

In the experiments, we observed that the agent's behavior changed when it interacted with different teammates. Consider, for example, the cooperative Box-Pushing domain. Given an irrational teammate, the agent spends more time pushing the small box alone, while given an MMDP teammate, the agent tends to push the large box together with its teammate rather than pushing a small box.

5 Conclusion

We presented OPAT—a novel online planning algorithm for ad hoc agent teams. It is designed for situations where agents must collaborate without pre-coordination and teammates may not be fully known. We empirically tested OPAT in various ad hoc teams with different types of teammates: irrational, rational but partially observable, and fully ratio-

nal teammates whose characteristics are unknown in advance. Unlike much of the previous work in the literature of multi-agent learning and game theory, we do not assume that teammates are running the same algorithms or pre-computed policies. Instead, we adopt the settings proposed by Stone *et al.* [2010], allowing arbitrary teammates that are not necessarily rational or even aware of teamwork. The adaptiveness of OPAT relies on its ability to learn quickly from the interaction history and find best-response actions that are consistent with a fixed history window—the most recent m plays.

Several approaches in the literature are related to our work. Most notably, Stone *et al.* [2009] formulate the ad hoc team problem as a matrix game, assuming that teammates play best-response policies to the observed actions of the agent. Stone and Kraus [2010] investigate a special instance of ad hoc teams where teammates have a fixed known behavior and solve the problem as a finite-horizon cooperative k-armed bandit. These works are different from ours in that they consider only one-step team strategies in restrictive settings. More recently, Barret *et al.* [2011] presented an empirical study of ad hoc teamwork in the Pursuit domain with a range of algorithms. Unlike OPAT, they do not bias the agent's action selection according to the teammates' past plays. Instead, they view the unknown behavior of teammates as part of the model uncertainty and approximate the optimal joint Q-function by a linear combination of different *pre-defined* models updated using Bayes' rule [Barret *et al.*, 2011].

In this work, planning in ad hoc teams is treated as an optimization problem in the joint policy space, which is constrained by the limited capabilities of teammates. The objective is to maximize the team's joint reward. A generative model is utilized to evaluate the agent's policies via Monte-Carlo simulation with actions that are consistent with teammates' behaviors. By doing this, the agent can choose the best-response actions regardless of whether the teammates are rational. One possible future extension is for the agent to plan its actions taking into account that they will be observed and interpreted by its teammates. The agent can then analyze the teammates' responses and play accordingly. Another interesting research direction is to consider a library of "typical" teammates' policies and adaptively use the ones that match best the interaction. This could improve the agent's ability to collaborate with initially unknown teammates.

Acknowledgments

We thank Peter Stone for valuable suggestions. Feng Wu and Xiaoping Chen were supported by the Natural Science Foundations of China, Grant No. 60745002, and the National Hi-Tech Project of China, Grant No. 2008AA01Z150. Shlomo Zilberstein was supported by the Air Force Office of Scientific Research, Grant No. FA9550-08-1-0181, and the National Science Foundation, Grant No. IIS-0812149.

References

[Auer *et al.*, 2002] P. Auer, N. Cesa-Bianchi, P. Fischer, and L. Informatik. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[Barret *et al.*, 2011] Samuel Barret, Peter Stone, and Sarit Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proc. of the 10th Int'l Conf. on Autonomous Agents and Multiagent Systems*, 2011.

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

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

[Boutilier, 1999] C. Boutilier. Sequential optimality and coordination in multi-agent systems. In *Proc. of the 16th Int'l Joint Conf. on Artificial Intelligence*, pages 478–485, 1999.

[Brenner and Nebel, 2009] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.

[Gelly and Silver, 2007] S. Gelly and D. Silver. Combining online and offline learning in UCT. In *Proc. of the 17th Int'l Conf. on Machine Learning*, pages 273–280, 2007.

[Kildare, 2004] R. A. Kildare. Ad-hoc online teams as complex systems: agents that cater for team interaction rules. In *Proc. of the 7th Asia-Pacific Conf. on Complex Systems*, 2004.

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proc. of the 17th European Conf. on Machine Learning*, pages 282–293, 2006.

[Lewis *et al.*, 2007] M. Lewis, J. Wang, and S. Hughes. USARSim: Simulation for the study of human-robot interaction. *Journal of Cognitive Engineering and Decision Making*, 1(1):98–120, 2007.

[Leyton-Brown and Shoham, 2008] K. Leyton-Brown and Y. Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan & Claypool, 2008.

[Melo and Ribeiro, 2008] F. S. Melo and M. I. Ribeiro. Emerging coordination in infinite team Markov games. In *Proc. of the 7th Int'l Conf. on Autonomous Agents and Multiagent Systems*, pages 355–362, 2008.

[Seuken and Zilberstein, 2008] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[Stone and Kraus, 2010] P. Stone and S. Kraus. To teach or not to teach? decision making under uncertainty in ad hoc teams. In *Proc. of the 9th Int'l Conf. on Autonomous Agents and Multiagent Systems*, pages 117–124, 2010.

[Stone *et al.*, 2009] P. Stone, G. A. Kaminka, and J. S. Rosenschein. Leading a best-response teammate in an ad hoc team. In *AAMAS Workshop on Agent Mediated Electronic Commerce*, 2009.

[Stone *et al.*, 2010] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proc. of the 24th AAAI Conf. on Artificial Intelligence*, pages 1504–1509, 2010.

[Wang and Sandholm, 2003] X. Wang and T. Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Proc. of the 17th Annual Conf. on Neural Information Processing Systems*, pages 1571–1578, 2003.

[Wu *et al.*, 2011] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence*, 175:2:487–511, 2011.