Bayes-Adaptive Monte-Carlo Planning for Type-Based Reasoning in Large Partially Observable Environments

Jonathon Schwartz
The Australian National University
Canberra, Australia
jonathon.schwartz@anu.edu.au

Hanna Kurniawati
The Australian National University
Canberra, Australia
hanna.kurniawati@anu.edu.au

ABSTRACT

Designing autonomous agents that can interact effectively with other agents without prior coordination is an important problem in multi-agent systems. Type-based reasoning methods achieve this by maintaining a belief over a set of potential behaviours for the other agents. However, current methods are limited in that they assume full observability of the state and actions of the other agent or scale poorly to larger problems. Addressing these limitations, we propose Bayes-Adaptive Partially Observable Stochastic Game Monte-Carlo Planning (BAPOSGMCP) —a general online planning method for type-based reasoning in large partially observable environments. BAPOSGMCP accounts for partial observability by maintaining a belief over the type of the other agent along with their interaction-history and the state of the environment. To handle the computational challenges this presents, BAPOSGMCP combines a number of recent advances in Monte-Carlo Tree Search with a novel meta-policy for selecting the best policy to guide planning from each belief. We provide a comprehensive evaluation and ablation studies in cooperative, competitive and mixed large partially observable multi-agent environments and demonstrate that our method is able to effectively adapt online to diverse sets of agents.

KEYWORDS

Multi-agent Decision Making, Partially Observable Stochastic Games, Type-Based Reasoning, Planning under Uncertainty

1 INTRODUCTION

The development of autonomous agents that can effectively interact with other agents without prior coordination is a core research area in multi-agent systems [2, 13, 58]. Such autonomous agents must have the ability to reason about the behaviours and intentions of other agents. Type-based reasoning methods give agents this ability by maintaining a belief over a set of *types* for the other agents [1, 4, 10, 11, 17, 57]. A type is defined by a mapping from the agent's interaction history to a probability distribution over actions, and completely specifies the agent's behaviour. If the set of types is sufficiently representative, type-based reasoning methods can lead to fast adaptation and effective interaction without prior coordination [3, 10].

A common assumption of existing type-based reasoning methods is that the agent has full observability of the state of the environment and the other agents' actions [6]. However, for many practical problems, access to the history of observations and actions of the other agents is not available. In such cases, it becomes necessary for an agent to reason not only about the type of the other agent, but also about their interaction history and the state of the environment. This introduces significant computational and technical challenges. While there has been progress on modelling frameworks to address the technical challenges [12, 29, 31], existing planning based methods have been restricted to relatively small problems due to the computational challenges [17, 25–27].

In this paper, we aim to develop an efficient planning method for type-based reasoning in partially observable environments. To be practical, such an approach should be flexible (i.e. it can be used with different sets of other agent types and in general environments) and scalable (i.e. it can be used for large problems with more than a few hundred states).

To this end, we build on the recent successes of online Monte-Carlo based planning methods. Based on Monte-Carlo Tree-Search (MCTS), Monte-Carlo planning has been effective in large competitive [15, 16, 50, 54, 55] and cooperative multi-agent problems [39], as well as partially observable single-agent environments [36, 56, 60, 66]. A number of Monte-Carlo based methods have been proposed for type-based reasoning in fully observable multi-agent environments [5, 9, 13, 58, 65, 67]. However, none can handle large partially observable environments.

In this work we propose Bayes-Adaptive Partially Observable Stochastic Game Monte-Carlo Planning (BAPOSGMCP), an online planning algorithm for type-based reasoning in large partially observable, multi-agent environments. Using the joint environment model, BAPOSGMCP maintains a belief over the type of the other agent along with their interaction history and the environment state. Such a belief allows the planning agent to model the joint uncertainty from the state of the environment, the type of the other agent, and the internal state of the other agent. BAPOSGMCP addresses a number of challenges that have typically limited MCTS based methods in partially observable environments by combining recent advances in MCTS that make use of a policy to guide search [55] with a meta-policy to select which policy to use for search [37, 63, 64]. We evaluate the proposed method on large competitive, cooperative, and mixed partially observable environments - the largest of which has four agents and on the order of 10^{14} states and 10^8 observations - and demonstrate that it is able to rapidly adapt and interact effectively without explicit prior coordination.

2 RELATED WORK

Monte-Carlo Planning: We are interested in Monte-Carlo planning methods for partially observable and multi-agent environments where the agent must adapt to a set of possible types of other agents. When coordination between agents is involved, this problem becomes an *ad-hoc teamwork* problem [13, 58]. Various planning methods have been proposed for ad-hoc teamwork and type-based reasoning, including those based on stage games [65], Bayesian beliefs [11], POMDPs [9], type-based reasoning with parameters

[5], and for the many agent setting [67]. All these methods use MCTS for online adaptation but are limited to environments where the state and actions of the other agents are fully observed. In the *agent modelling* setting [6], several online Monte-Carlo planning methods have been proposed for the Interactive POMDP (I-POMDP) [29] framework, which models the other agents using recursive-reasoning. Including methods based on finite state-automata [46], nested MCTS [52], and for open multi-agent systems [28]. Other works have focused on planning in the strictly cooperative setting. Czechowski and Oliehoek [24] propose a method for improving the coordination of a team of decentralized agents in fully-observable environments. While Choudhury et al. [18] propose a scalable MCTS based method for centralized control of a team of agents in fully-observable environments. MCTS is also widely used in game-theory [35]. Most relevant to our work is Information Set MCTS [22] for two-player imperfect-information zero-sum games and operates on known information sets, as opposed to beliefs over states.

Bayes Adaptive Planning: Our proposed method falls into the category of online Bayes-adaptive (BA) planning or Bayesian Reinforcement Learning (BRL) [59]. A number of MCTS based BRL methods have been proposed including for BA-MCP for MDPs [30], BA-POMCP for POMDPs [33], and BA-MPOMDPs for multi-agent POMDPs involving centralized control of multiple agents [7]. All these methods focus on learning parameters of the environment's transition dynamics. In our work we instead focus on learning the policy type of the other agent.

Combining Reinforcement Learning and Search: In recent years, a number of methods have been proposed that combine Reinforcement Learning (RL) with MCTS. Self-play and MCTS have been combined to improve learning and final performance in two-player fully observable zero-sum games with a known environment model [54, 55] and using a learned model [50]. Similar methods have been applied to zero-sum imperfect-information games [14, 16]. Our method builds on a number of advances made by these works in combining RL policies with search, specifically relating to using an existing policy as a prior for search. However, we apply these advances outside of self-play and zero-sum games, instead focusing on online adaption to a set of possible types of other agents with no assumptions made about the reward structure of the environment. Most closely related to the proposed method is *SPARTA* [39] which combines search with a shared blueprint policy for improved teamwork in Decentralized POMDPs. They focus on the setting where there is prior coordination for decentralized execution in a cooperative game. We generalize this approach to settings where the other agents may be using one of multiple types, where there is no prior coordination, and beyond cooperative environments.

3 PROBLEM DESCRIPTION

We consider the problem of *type-based* reasoning in partially observable environments. Suppose a single planning agent which must interact with one or more other agents each of which is assumed to be one of a fixed set of types [2, 58]. Each *type* is a complete specification of that agent's behaviour, defined as a mapping from the agent's interaction history to a probability distribution over actions. Note that this definition of type is equivalent to a *policy* in the planning under uncertainty and RL literature and so we use the terms interchangeably. The planning agent knows the set of types, but does not know the true type of the other agent *a priori*.

We model the problem as a Partially Observable Stochastic Game (POSG) [31] which consists of N agents indexed $\mathcal{I} = \{1, \dots, N\}$, a discrete set of states $\mathcal{S}$, an initial state distribution $b_0 \in \Delta(\mathcal{S})$, the joint action space $\vec{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$, the finite set of observations $\mathcal{O}_i$ for each agent $i \in \mathcal{I}$, a state transition function $\mathcal{T} : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ specifying the probability of transitioning to state s' given joint action $\vec{a}$ was performed in state s, an observation function for each agent $\mathcal{Z}_i : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{O}_i \to \mathbb{R}$ specifying the probability that performing joint action $\vec{a}$ in state s results in observation o_i for agent i, and a bounded reward function for each agent $\mathcal{R}_i : \mathcal{S} \times \vec{\mathcal{A}} \to \mathbb{R}$.

At each step, each agent $i \in \mathcal{I}$ simultaneously perform an action $a_i \in \mathcal{A}_i$ and receive an observation $o_i \in \mathcal{O}_i$ and reward $r_i \in \mathbb{R}$. The interaction can continue for a finite or infinite number of steps, where the number of steps is called the *horizon*. Each agent has no direct access to the environment state or knowledge of the other agent's actions and observations. Instead they must rely only on information in their *interaction-history* up to the current time step t: $h_{i,t} = \langle o_{i,0} a_{i,0} o_{i,1} a_{i,1} \dots a_{i,t-1} o_{i,t} \rangle$. The combined interaction histories of all agents is the *joint history*, denoted $\vec{h}_t = \langle \vec{o}_0 \vec{a}_0 \vec{o}_1 \vec{a}_1 \dots \vec{a}_{t-1} \vec{o}_t \rangle$. Agents select their next action using their *policy* π_i which is a mapping from their history $h_{i,t}$ to a probability distribution over their actions, where $\pi_i(a_i|h_{i,t})$ denotes the probability of agent i performing action a_i given history $h_{i,t}$.

In this work we make no assumption about the reward structure of the environment, however we assume the other agents are using policies from a known fixed set of policies (where each policy corresponds to an agent type). We denote the planning agent by i, and all other agents collectively using $-i$. The set of fixed policies for the other agents is $\Pi = \{\pi_{-i,m} | m = 1, \dots, M\}$, where M is the number of policies in the set. Furthermore, we assume the policies used by the other agents are selected based on a known prior distribution ρ, where $\rho(\pi_{-i,m}) = Pr(\pi_{-i,m})$. Note that when there are more than two agents in the environment, ρ defines the probability of each joint policy for the other agents (one policy per agent), thus making it possible to assign a prior over teams of agents. The goal of the planning agent is to maximize it's expected *return* with respect to ρ given by $G_{i,t} = \mathbb{E}_{\pi_{-i,m} \sim \rho} \left[\sum_{k=t}^{\infty} \gamma^{k-t} r_{i,k} | \pi_{-i,m} \right]$, where $\gamma \in [0, 1)$ is the discount factor [2, 58].

We assume the planning agent has access to a generative model of the joint dynamics of the environment. The generative model $\mathcal{G}$ provides a sample of the next state, joint observation, and joint reward, given a state and joint action: $\langle s_{t+1}, \vec{o}_{t+1}, \vec{r}_{t+1} \rangle \sim \mathcal{G}(s_t, \vec{a}_t)$.

4 METHOD

Our goal is to develop a scalable planning algorithm for type-based reasoning in partially observable environments. To this end, we introduce Bayes-Adaptive Partially Observable Stochastic Game Monte-Carlo Planning (**BAPOSGMCP**) which maintains a joint belief over the environment state, other agent type, and other agent history, allowing it to perform Bayesian learning of types online. Modelling the problem this way introduces additional computation challenges. To overcome them, BAPOSGMCP uses a tailored

multi-agent MCTS algorithm in combination with a meta-policy for selecting the best policy to guide planning from each belief.

4.1 Bayesian Learning of Types

In partially observable environments planning agents must account for three interdependent sources of uncertainty: the environment state, the type/policy of the other agent, and the other agent's interaction-history.

We adopt a similar approach to the I-POMDP framework [29] and account for these sources of uncertainty by using a joint belief over the environment state $s \in \mathcal{S}$, the other agent policy $\pi_{-i} \in \Pi$, and the joint history $\vec{h} \in \vec{\mathcal{H}}$ [1]. Each belief is thus a distribution over state-policy-history tuples, which we refer to as history-policy-states. To distinguish between environment states and history-policy-states, we denote the latter using w and it's components using dot notation: $w.s$, $w.\vec{h}$, $w.\pi_{-i}$. The space of history-policy-states is denoted $\mathcal{W}$. The planning agent's belief is a distribution over history-policy-states: $b_i(w|h_i) = Pr(w_t = w|h_{i,t} = h_i)$.

Defining the belief using history-policy-states transforms the original POSG problem into a Partially Observable Markov Decision Process (POMDP) for the planning agent where the state, other agent policy, and joint history are learned online. The agent's current belief $b_i(h_i)$ can be computed exactly using Bayes rule given the initial environment state distribution b_0 and the prior over the other agent policies ρ. Unfortunately, while we assume the policy set is discrete and finite, the size of the joint history space $\vec{\mathcal{H}}$ grows exponentially with the planning horizon. This makes exact belief updates intractable for all but the smallest problems and planning horizons. To address this, we build on previous work [7, 30, 33, 56] that solves this problem using Monte-Carlo based sampling methods.

4.2 Meta-Policy

Monte-Carlo planning has been successfully applied to very large planning problems using PUCT [49] and access to a good policy for guiding search [50, 55]. The question becomes where does this search policy come from? Prior work in the multi-agent setting [50, 55] dealt with zero-sum two-player games where it suffices to train a single Nash-equilibrium policy to get robust performance against any opponent [44]. In this work we make no assumptions about the reward structure of the environment, only that the other agents are using some policy from a known and fixed distribution ρ. In practice it is possible to design or train a policy that performs well against a known mixture of policies [43]. We aim to avoid this since any changes to ρ would require a new search policy.

In this work, we use the set of policies Π to generate a **meta-policy** for use as a search policy. We define a meta-policy as a function mapping a policy to a mixture of policies $\sigma_i : \Pi \to \Delta(\Pi)$. For our purposes its a mapping from the set of fixed policies to a distribution over this set, so that $\sigma_i(\pi_{i,j}|\pi_{-i,k}) = Pr(\pi_{i,j}|\pi_{-i,k})$ for $\pi_{i,j}, \pi_{-i,k} \in \Pi$. Ideally the meta-policy would map from the planning agent's belief to a mixture over policies. However, finding a mapping from belief to a mixture over policies reduces to the original problem of finding an optimal policy over actions for the planning agent. Furthermore computing a meta-policy in this way would require re-computing the meta-policy each time a policy is added or removed from the set Π.

To design the meta-policy we propose to use the pairwise performance of the policies in Π generated in an empirical-game. An **empirical game** is a normal-form game where the actions are policies and the expected returns for each joint policy are estimated from sample games [63, 64]. Formally, an empirical game is a tuple $\langle \Pi, U^{\Pi}, N \rangle$ where N is the number of players, $\Pi = \langle \Pi_1, \dots \Pi_N \rangle$ is the set of policies for each player and $U^{\Pi} : \Pi \to \mathbb{R}^N$ is a payoff table of expected returns for each joint policy played by all players. Empirical games have the advantage that they are relatively inexpensive to compute, depending on the representation of the policies, and adding a new policy only requires computing the payoffs for that policy.

There are a number of ways to define a meta-policy in terms of the empirical-game payoffs, the simplest being the **greedy meta-policy** σ_i^G. This meta-policy selects π_i to be the policy in Π that has the highest payoff against the given other agent policy π_{-i}, with ties broken uniform randomly.

σ_i^G has two main potential flaws. Firstly, σ_i^G is defined based on the expected performance from the start of an episode. It's possible that the best response policy from the set Π may change depending on the planning agent's current belief. Secondly, the way we have defined the meta-policy assumes full knowledge of the other agent's policy after a single step, which may lead to overly greedy behaviour when using σ_i^G.

To address these potential flaws, we also propose the **softmax meta-policy** σ_i^S where, $\sigma_i^S(\pi_i|\pi_{-i}) = \frac{1}{\eta} e^{U^{\Pi}(\pi_i,\pi_{-i})/\tau}$ with normalizing constant $\eta = \sum_{\pi'_{-i}} e^{U^{\Pi}(\pi_i,\pi'_{-i})/\tau}$ and temperature hyperparameter τ which controls how uniform or greedy the policy is.

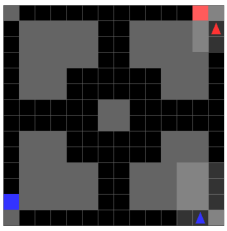
Additionally, we define the **uniform meta-policy** σ_i^U which uses a uniform distribution over Π. Together σ_i^G, σ_i^S, and σ_i^U provide a spectrum of meta-policies from most to least exploitative with respect to the empirical-game.

Of course the planning agent does not know the policy of the other agent, rather it has a belief over them. Thus, during planning at the decision node for a given history h_i the search probability for each action edge $P(h_i a_i)$ are generated as a weighted mixture, $P(h_i a_i) = \sum_{\pi_{-i} \in \Pi} b_i(\pi_{-i}|h_i) \sum_{\pi_i \in \Pi} \sigma_i(\pi_i|\pi_{-i}) \pi_i(a_i|h_i)$ where $b_i(\pi_{-i}|h_i)$ is the marginal probability computed by summing over all history-policy-states in the belief.

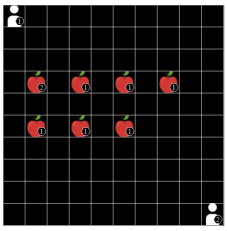
4.3 Multi-agent MCTS

Being an online planner, each step BAPOSGMCP executes a search to find the next action, followed by an update to compute the next belief given the most recent observation. To do this BAPOSGMCP builds a search tree T of agent histories using the PUCT algorithm [49, 55] and a meta-policy as the search policy. Each node of the tree corresponds to a history, where $T(h_i)$ denotes the node for history h_i. Each node maintains a belief $b_i(h_i)$ over history-policy-states, represented as a set of unweighted particles where each particle corresponds to a sample history-policy-state w. For each action $a_i \in \mathcal{A}_i$ from h_i there is an edge $h_i a_i$ that stores a set of statistics $\langle N(h_i a_i), P(h_i a_i), W(h_i a_i), Q(h_i a_i) \rangle$, where $N(h_i a_i)$ is the visit count, $P(h_i a_i)$ is the prior probability of selecting a_i

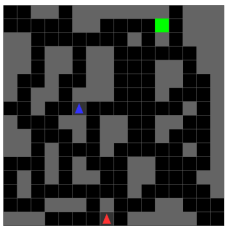
[1] Technically, only the history of the other agent is required, but we include the full joint history in practice since it makes it simpler to generalize to more than two agents.



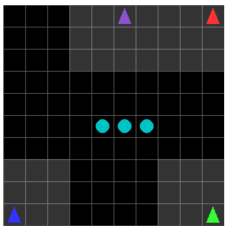
(a) Driving



(b) Level-Based Foraging (LBF)



(c) Pursuit-Evasion (PE)



(d) Predator-Prey (PP)

Figure 1: Experiment Environments

given h_i, $W(h_i a_i)$ is the total action-value, and $Q(h_i a_i)$ is the mean action-value.

4.3.1 ***Tree Search.*** For each real step at time t in the environment BAPOSGMCP runs MCTS from the planning agent's current belief $b_i(h_{i,t})$ and then selects an action $a_{i,t}$ greedily with respect to visit count at the root belief. MCTS generates a series of simulated episodes. Each simulation starts from the planning agent's current belief, the root of the tree T, and proceeds in three stages. Pseudo-code for the search procedure is shown in Algorithm 1.

Selection: Each simulation starts from the root node of the tree $T(h_{i,t})$ and finishes when the simulation reaches a leaf node $T(h_{i,L})$ after some $L-t$ simulated steps. At the start of each simulation a history-policy-state particle is sampled from the agents belief at the root node $w_t \sim b_i(h_{i,t})$. For each simulation step $l = t, t+1, \ldots, L-1, L$, actions must be selected for the planning agent i and the other agent $-i$. The planning agent's action is selected using the PUCT algorithm [49, 50]:

$$a_{i,l} = \arg\max_{a_i \in \mathcal{A}_i} \left\{ Q(h_{i,l}a_i) + C(h_{i,l}) P(h_{i,l}a_i) \frac{\sqrt{N(h_{i,l})}}{1 + N(h_{i,l}a_i)} \right\} \quad (1)$$

where $N(h_{i,l}) = \sum_{a_i \in \mathcal{A}_i} N(h_{i,l}a_i)$ and $C(h_{i,l})$ is the exploration rate, $C(h_{i,l}) = c_{init} + \log\left(\frac{N(h_{i,l}) + c_{base} + 1}{c_{base}}\right)$. The action for the other agent is sampled using their policy and history contained within the sampled history-policy-state: $a_{-i,l} \sim w_l.\pi_{-i}(w_l.h_{-i})$.

Following joint action selection, the next environment state, joint observation, and rewards are sampled using the generative model: $\langle s_{l+1}, \vec{o}_{l+1}, \vec{r}_{l+1} \rangle \sim \mathcal{G}(w_l.s, \vec{a}_l)$. These values are then used in the planning agents next-step history, $h_{i,l+1} = h_{i,l} a_{i,l} o_{i,l+1}$, and the next-step history-policy-state, $w_{l+1} = \langle s_{l+1}, w_l.\pi_{-i}, w_l.\vec{h}\vec{a}_l\vec{o}_{l+1} \rangle$. The simulation ends after $L-t$ simulation steps, when it reaches a leaf node $T(h_{i,L})$ - a node in the tree that has not been expanded.

Expansion: Once a leaf node $T(h_{i,L})$ is reached it is evaluated and added to the tree and an edge is added to it for each action. Evaluation of the leaf node involves estimating two values; the value $v_{i,L}$ and the policy $p_{i,L}$ of the node. The policy $p_{i,L}$ is computed using the meta-policy and the current simulated history-policy-state particle, where $p_{i,L}(a_i) = \sum_{\pi_i} \sigma(\pi_i | w_L.\pi_{-i}) \pi_i(a_i | h_{i,L})$.

The value $v_{i,L}$ is computed either by using the value function from the meta-policy or using a rollout. When using the value function from the meta-policy the value is computed as $v_{i,L} = \sum_{\pi_i} \sigma(\pi_i | w_L.\pi_{-i}) V^{\pi_i}(h_{i,L})$, where $V^{\pi_i}(h_{i,L})$ is the value function of policy π_i for agent i's history at the leaf node. Estimating the value using a rollout involves continuing the simulation until a terminal state is reached and then using the sum of rewards from this simulated trajectory as the value estimate. During the rollout actions for the planning agent i are selected using a policy sampled

Algorithm 1 BAPOSGMCP Search

procedure SEARCH(h_i)
  for 1, ..., N_{sims} **do**
    if $h_i = \emptyset$ **then**
      $s \sim b_0$; $\pi_{-i} \sim \rho$
      $w \leftarrow \langle s, \pi_{-i}, \emptyset \rangle$
    else
      $w \sim b_i(\cdot, h_i)$
    end if
    $\pi_i \sim \sigma_i(w.\pi_{-i})$
    SIMULATE(w, h_i, π_i, 0)
  end for
  return $\arg\max_{a_i \in \mathcal{A}_i} N(h_i a_i)$
end procedure
procedure SIMULATE(w, h_i, π_i, $depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  if $h_i \notin \mathcal{T}_i$ **then**
    return EXPAND(w, h_i, π_i, $depth$)
  end if
  $a_i \leftarrow$ PUCT(h_i)
  $a_{-i} \sim w.\pi_{-i}(\cdot | w.h_{-i})$
  $\vec{a} \leftarrow \langle a_i, a_{-i} \rangle$
  $\langle s', \vec{o}, \vec{r} \rangle \sim \mathcal{G}(w.s, \vec{a})$
  $w' \leftarrow \langle s', w.\vec{h}\vec{a}\vec{o}, w.\pi_{-i} \rangle$
  $G_i \leftarrow r_i + \gamma$ SIMULATE($w', h_i a_i o_i, \pi_i, depth+1$)
  $b_i(h_i a_i o_i) \leftarrow b_k(h_i a_i o_i) \cup \{w'\}$
  $N(h_i a_i) \leftarrow N(h_i a_i) + 1$
  $W(h_i a_i) \leftarrow W(h_i a_i) + G_i$
  $Q(h_i a_i) \leftarrow \frac{W(h_i a_i)}{N(h_i a_i)}$
  for $\hat{a}_i \in \mathcal{A}_i$ **do**
    $P(h_i \hat{a}_i) \leftarrow \frac{\pi_i(\hat{a}_i | h_i) - P(h_i \hat{a}_i)}{N(h_i)}$
  end for
  return G_i
end procedure

from the meta-strategy $\pi_i \sim \sigma(w_L.\pi_{-i})$. Actions for the other agent are selected using the policy in the history-policy-state, $w_L.\pi_{-i}$.

After the value $v_{i,L}$ and the policy $p_{i,L}$ of the leaf node are estimated, each edge $h_{i,L}a_i$ from the newly expanded node is initialized to $\langle N(h_{i,L}a_i) = 0, P(h_{i,L}a_i) = p_{i,L}, W(h_{i,L}a_i) = 0, Q(h_{i,L}a_i) = 0\rangle$.

Backup: At the end of the simulation the statistics for each edge along the simulated trajectory are updated.

For each simulation step $l = t, t{+}1, \ldots, L{-}1, L$, we form a $L{-}l$ step estimate of the discounted sum of rewards bootstrapped from the value estimate $v_{i,L}$: $\hat{G}_{i,l} = \sum_{\tau=1}^{L-l} \left[\gamma^{\tau-1} r_{i,l+\tau}\right] + \gamma^{L-l} v_{i,L}$., This is then used to update the statistics for each edge $h_{i,l}a_i$ traversed during the simulation as follows: $N(h_{i,l}a_i) = N(h_{i,l}a_i) + 1$, $W(h_{i,l}a_i) = W(h_{i,l}a_i) + \hat{G}_{i,l}$, $Q(h_{i,l}a_i) = \frac{W(h_{i,l}a_i)}{N(h_{i,l}a_i)}$. The policy prior for each edge connected to a node traversed during the simulation is also updated. For $l = t, \ldots, L$ for each $a_i \in \mathcal{A}_i$ we update the policy prior for the edge $h_{i,l}a_i$ as $P(h_{i,l}a_i) = \frac{\pi_i(a_i|h_i) - P(h_i a_i)}{N(h_{i,l})}$. This differs from previous works [50, 55] which do not update the prior once it has been set. Our setting differs in that each node in the tree is an estimate of the belief of the planning agent. When the agent first expands a node the estimate of the belief is likely very inaccurate, as it contains only a single history-policy-state particle, and thus the policy prior will also be inaccurate. As the node is visited during subsequent simulations the belief accuracy improves and so the policy prior is iteratively updated to reflect this.

4.3.2 ***Tree Update***. Once search is complete, the planning agent selects the action at the root node with the greatest visit count $a_{i,t} = \arg\max_{a_i} N(h_{i,t}a_i)$ and receives an observation $o_{i,t+1}$ from the real environment. At this point $T(h_{i,t}a_{i,t}o_{i,t+1})$ becomes the new root node of the search tree and the agent's current belief becomes $b_i(h_{i,t}a_{i,t}o_{i,t+1})$. Any branches of the tree that are not descendants of $T(h_{i,t}a_{i,t}o_{i,t+1})$ are pruned. Since BAPOSGMCP uses particle filtering for belief representation and updates, particle deprivation is possible. To alleviate this, in practice additional particles are added during the belief update using rejection sampling or a domain specific function.

5 EXPERIMENTS

5.1 Multi-Agent Environments

We evaluated the proposed method on one cooperative, one competitive, and two mixed environments (Figure 1). The largest of which has four agents and on the order of 10^{14} states and 10^8 observations. Further details for each environment are provided in Appendix A.

Driving: A general-sum grid world navigation problem requiring coordination [40, 42]. Each agent controls a vehicle and is tasked with driving the vehicle from start to destination while avoiding crashing into other vehicles.

Level-Based Foraging (LBF): is a mixed incentive environment where agents must collect food in a grid world [3, 5]. We use the partially-observable version of LBF used in [19, 48].

Pursuit-Evasion (PE): An asymmetric zero-sum grid world problem involving two agents, an evader and a pursuer [52, 53]. The evader's goal is to reach a safe location, while the the pursuer's aim is to spot the evader before it reaches it's goal.

Predator-Prey (PP): A co-operative grid world problem involving multiple predator agents working together to catch prey controlled by the environment [38, 61]. We used two-agent and four-agent versions of the environment both containing three prey. The *two-agent* version had two predators with each prey requiring two predators to capture. The *four-agent* version had four predators with each prey requiring three predators to capture.

5.2 Policies

For each environment, we created a set of policies Π which was used for the other agent policies during evaluations and also for the meta-policy σ_i and policy prior ρ. For the Driving, PE, and PP environments the policies were deep neural networks with actor-critic architectures trained using the PPO algorithm [51]. Additional details including training parameters and empirical-game payoff matrices are provided in Appendix B.

Driving: For this problem we used a set of five K-level reasoning (KLR) policies where the level k agent is trained as a best response to the level $k-1$ agent [20]. We used a uniform distribution over the policies with reasoning levels $k = 0, 1, 2, 3$ for ρ. While the meta-policy was defined using all five policies, which included the level $k = 4$ policy. This meant the planning agent had access to a best-response for all the policies in ρ and allowed a fair comparison against the Best-Response baseline (presented in Section 5.3).

Level-Based Foraging (LBF): Here we used the set of four heuristic policies used in prior work [5, 47]. Note that none of the policies in the set are best-responses to any other policy in the set, thus testing the planning agent's ability to improve on the prior-knowledge provided by the meta-policy. We used a uniform distribution over all four policies as the prior ρ and all four policies in the meta-policy.

Pursuit-Evasion (PE): For the PE environment we again trained KLR policies for $K = 4$. Since this problem is asymmetric, we trained different policies for the evader and pursuer agents. The meta-policy and prior ρ were defined same as the Driving environment.

Predator-Prey (PP): For this fully cooperative problem we trained five independent teams of agents using self-play [62] where each team consisted of identical copies of the same policy. We used a uniform distribution over five teams for the prior ρ, with each team made up of copies of the same policy from set of five trained self-play policies - one copy in the two-agent version, and three copies in the four-agent version. This setup tested the planning agent's *ad-hoc teamwork* ability. The meta-policy was defined using all five policies in both versions.

5.3 Baselines

While there have been several planning methods proposed for type-based reasoning and ad-hoc teamwork [5, 9, 11, 65, 67], to the best of the authors knowledge, the proposed method is the first to tackle the setting where all agents have partial observability. Furthermore, the size of the environments meant existing agent modelling algorithms for partially observable environments were intractable [26, 27, 45, 46]. As such, we propose a number of baselines utilizing the set of fixed-policies Π, two of which are indicative of upper and lower bounds on the performance of BAPOSGMCP, while the other two test the benefits of different components of our approach.

Meta-policy: Selects a policy from the set Π at the start of the episode based on the meta-policy with respect to the distribution ρ. This acts as a lower bound on the performance of BAPOSGMCP with access to the meta-policy but without using beliefs or search.

Best-Response: This is the best performing policy from the set of fixed policies Π against each policy, assuming full-knowledge of the policy of the other agent. This acts as an upper-bound given the policies in the set are best-response policies to other policies in the set, as is the case in the Driving, PE, and PP experiments.

PO-Meta: This baseline maintains a belief over history-policy-states and then uses the meta-policy to compute the action distribution for the next step with respect to this belief. This provides baseline performance of BAPOSGMCP using beliefs and the meta-policy but without search.

PO-MetaRollout: This baseline maintains a belief over the history-policy-states same as PO-Meta and BAPOSGMCP. Each step it evaluates available actions from the root node using one-step look-ahead search using the meta-policy for node evaluation. This provides baseline performance of BAPOSGMCP without the search tree - i.e. using only beliefs, the meta-policy, and one-step look-ahead search.

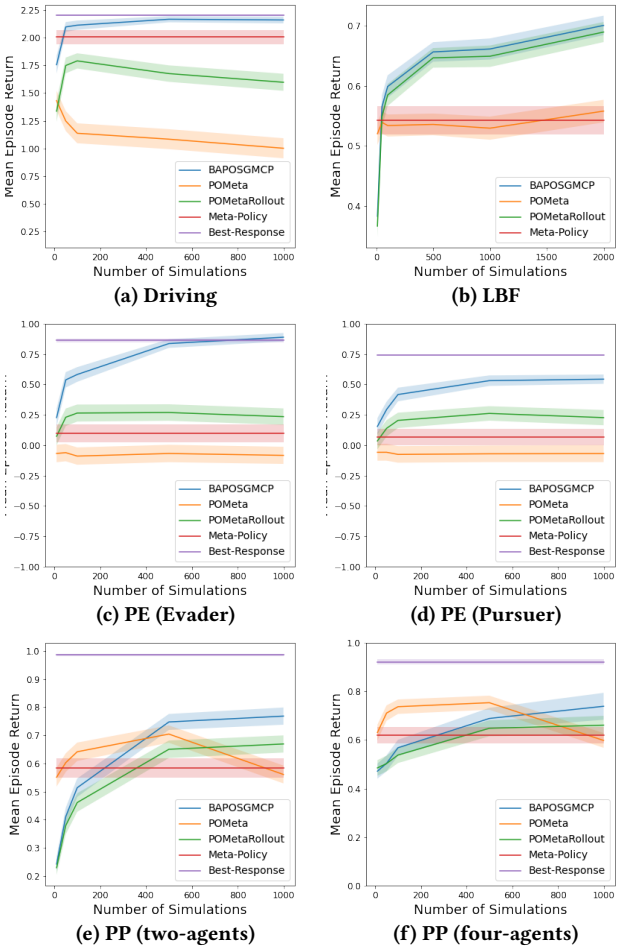
5.4 Experimental Setup

We evaluated BAPOSGMCP and baselines in each environment. For each method we ran a minimum of 400 episodes, or 48 hours of total computation time, whichever came first. For the planning based methods, we tested each using $N_{sims} \in [10, 50, 100, 1000]$ simulations in Driving, PE, and PP environments and $N_{sims} \in [10, 50, 100, 1000, 2000]$ simulations in the LBF environment. After each real step in the environment $N_{sims}/16$ additional particles were added during belief reinvigoration using rejection sampling, similar to previous work [56]. We capped the number of rejected sampled at 1000 for all experiments in order to limit update times. For all experiments we used $\epsilon = 0.01$ and a discount of $\gamma = 0.99$, giving a discount horizon beyond the maximum step limit of all environments.

We tested a number of bandit algorithms in preliminary experiments, including UCB1 [8], uniform, and PUCB [49] (results in Appendix C). PUCB clearly performed best so all reported results are based on PUCB for all methods tested. We chose search hyperparameters based on prior work combining PUCB with MCTS [49, 50, 55]. We used exploration constants $c_{init} = 1.25$ and $c_{base} = 20,000$ along with normalized Q-values to handle the returns being outside of $[0, 1]$ bounds in the tested environments. Dirichlet noise $Dir(\alpha)$ was added to the prior probabilities at each decision node with $\alpha = \frac{|\mathcal{A}_i|}{10}$ and a mix-in proportion of 0.5 for all problems.

For the Driving, PE, and PP problems each policy in Π was represented as a neural network with policy and value function outputs. As such, for these problems we used the value function of the fixed-policies (as chosen by the meta-policy) for leaf node evaluation, similar to existing approaches combining MCTS with neural network policies [39, 50, 55]. This lead to similar performance compared to using full rollouts, while significantly reducing search time (see Appendix D for details).

Our implementation is open-source available at https://github.com/Jjschwartz/ba-posgmcp.



(a) Driving (b) LBF

(c) PE (Evader) (d) PE (Pursuer)

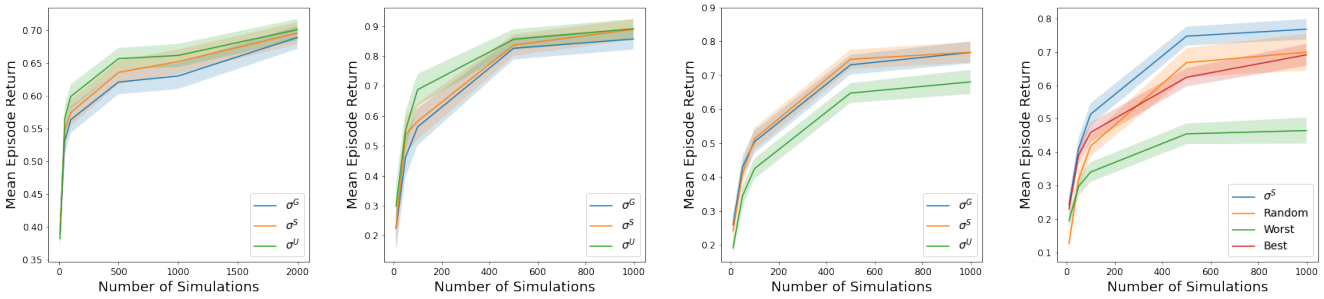
(e) PP (two-agents) (f) PP (four-agents)

Figure 2: Mean episode return of BAPOSGMCP and baseline methods. Results are for BAPOSGMCP and baselines using the best choice of meta-policy. Shaded areas show the 95% CI.

5.5 Evaluation of Returns

Figure 2 shows the mean episode returns of BAPOSGMCP and baseline methods with respect to the policy prior ρ. We found that for all environments the performance of BAPOSGMCP improved with the number of simulations and given enough simulations BAPOSGMCP equals or outperforms all non-upper bound baselines across all environments. Furthermore, in the Driving and PE problems the performance converged towards the Best-Response upper-bound, suggesting BAPOSGMCP converged towards Bayes-optimal performance as the number of simulations increased.

The importance of the different aspects of BAPOSGMCP- beliefs, search, and search tree - varied by environment, however using all three lead to overall best performance given enough planning time. In the mixed and competitive environments (Driving, LBF, PE), maintaining a belief over history-policy-states combined with the meta-policy produced similar or lower performance than using the meta-policy alone, as shown by the performance of POMeta. In the LBF and PE environments in particular we observe, from the performance POMetaRollout, that combining beliefs with a simple



(a) LBF **(b) PE (Evader)** **(c) PP (two-agents)** **(d) PP (two-agents)**

Figure 3: Mean episode returns of BAPOSGMCP using different search policies. (a-c) compares performance of greedy σ^G, softmax σ^S ($\tau = 0.25$), and uniform σ^U meta-policies. (d) shows comparison between the best meta-policy, the best and worst performing policy from the policy set Π, and the uniform random policy. Shaded areas show 95% CI.

one-step search was able to improve on the meta-policy, presumably by permitting the planning agent to improve upon the meta-policy as needed. Increasing the search depth, in the full BAPOSGMCP algorithm, only improved on these results. In the ad-hoc teamwork setting (PP environment), a large improvement was gained just thru the use of a belief. This is likely due to this setting requiring the planning agent to act in a way that aligns with it's teammates - i.e. acting similar to the aligned policy in the set Π - which occurs via following the meta-policy with an accurate belief. Although, we expect that this can be improved upon by a more powerful agent such as BAPOSGMCP with greater planning time. We observed that in some environments the performance of POMeta suffers with a higher number of simulations. This is due to the difficulty of particle filtering based belief updates combined with a design choice of our implementation, which we discuss further in Section 5.7. However, this highlights another benefit of using a search tree, as in the full BAPOSGMCP algorithm. Namely, by reusing particles generated during search, it helps alleviate the computational demand of belief updates.

5.6 Evaluation of Meta-Policies

To test the effect of meta-policy choice on performance we compared BAPOSGMCP using the **greedy** σ^G, **softmax** σ^S ($\tau = 0.25$), and **uniform** σ^U meta-policies in each environment. Meta-policies were generated as described in Section 4.2 using the empirical payoff matrix in each environment (Appendix B). Figure 3 (a-c) shows the mean episode returns of BAPOSGMCP using the different meta-policies in three of the environments (one mixed, one competitive, one cooperative). Results for all environments are available in Appendix E.

We observed that the best meta-policy varied based on the environment, with performance benefiting from a more greedy policy in some environments and a more uniform policy in others. In general, there was a benefit to some level of policy mixing, given the greedy policy has similar or lower performance to either the softmax or uniform meta-policies across all environments. We expect this is likely due to the reasons covered in Section 4.2. The softmax meta-policy was the most robust, performing best or equal best across all environments given enough simulations. It's worth noting that the level of mixing can be easily tuned using the τ parameter in the softmax meta-policy. We didn't tune this parameter at all for our experiments and expect marginal performance gains may be seen with tuned values. A question for future work is whether an optimal value for τ can be inferred from the empirical payoff-matrix.

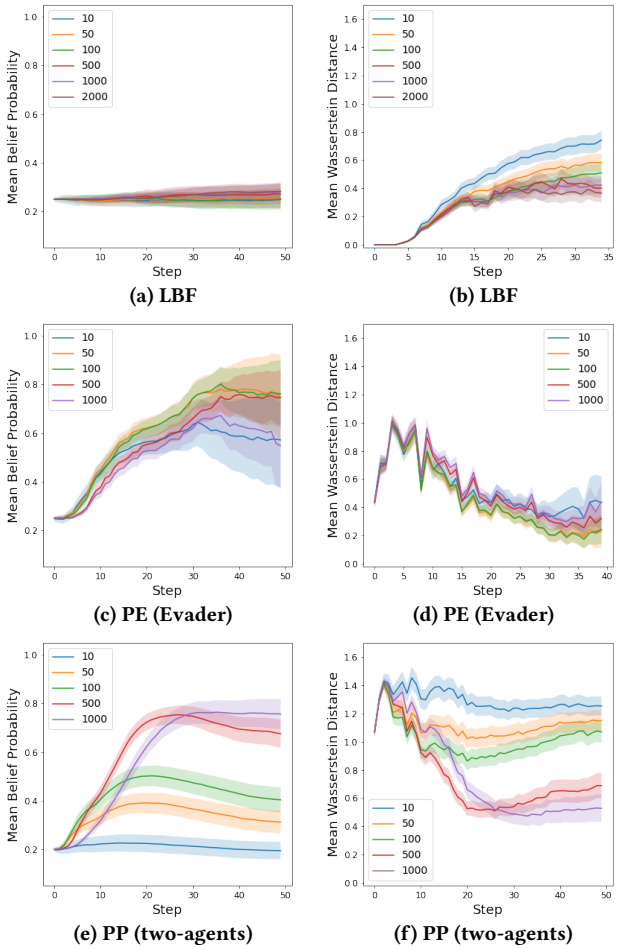
To study the benefit of using a meta-policy, as well as the robustness of BAPOSGMCP to search policy choice, we tested BAPOSGMCP using different search policies. Specifically, for each environment we replaced the meta-policy with each of the policies in the set Π, as well as the uniform random policy. Figure 3(d) shows the results for the PP (two-agents) problem, which is a representative sample (results for every environment are available in Appendix F).

We found that using a meta-policy lead to the most consistent results, with the meta-policy having similar or better performance than the best fixed-policy in each environment. While BAPOSGMCP using the best single-policy was able to perform comparable to when using the meta-policy, we typically observed a significant gap between the best and worst performing policies. Interestingly, for some environments BAPOSGMCP using the random policy had comparable performance to the meta-policy, showing that proper modelling of the uncertainty present in the problem plus MCTS is able to go a long way towards scalable type-based reasoning in many environments. The big downside of using a random policy is that it requires performing rollouts for leaf-node evaluation which increases search time significantly (see Appendix D).

5.7 Belief Evaluation

To better understand the adaptive capabilities of BAPOSGMCP we looked at the evolution of beliefs throughout an episode. In each environment we recorded the probability assigned to the true policy of the other agent in BAPOSGMCP's root belief at each step. Additionally, we recorded the accuracy of BAPOSGMCP's belief estimate of the other agent's action distribution at each step by measuring the Wasserstein distance [32] between the estimated and true action distributions. The results for the LBF, PE, and PP environments are shown in Figure 4 (results for all environments are in Appendix G).

We observed that for all environments, except LBF, BAPOSGMCP's belief in the correct type of the other agent increased as the episode progressed. A similar trend occurred for the action



(a) LBF (b) LBF

(c) PE (Evader) (d) PE (Evader)

(e) PP (two-agents) (f) PP (two-agents)

Figure 4: Accuracy of BAPOSGMCP's belief by episode step. Left column shows mean probability assigned by the belief to the true policy of the other agent. Right column shows Wasserstein distance between the belief's estimated action distribution and the true action distribution of the other agent. Each line shows BAPOSGMCP using a different number of simulations. Shaded areas show 95% CI.

distribution accuracy, with the distance between the estimated and true distributions decreasing over time. This indicates BAPOSGMCP was able to learn about the type of the other agent over time from online interactions. Interestingly, for the LBF environment BAPOSGMCP was unable to learn the type of the other agent yet it was able to predict the policy of the other agent relatively well (compared against prediction accuracy in the other environments). This suggests that the relatively rule-based other agent policies in the LBF environment had indistinguishable behaviour, yet BAPOSGMCP was able to account for this with it's belief. Another point worth highlighting is that, on average, the type of the other agent was never learned perfectly in any of the environments. It is possible this is due to the limitations of our method, however, we believe it's far more likely due to the difficulty of correctly inferring the type of the other agent in partially observable environments and highlights the benefits of maintaining a belief over types for robust performance in such settings.

As should be expected, we found roughly monotonic improvement in belief accuracy as the number of simulations increased. The exception to this was for 1000 simulations in some environments. This is likely due to our implementation using a constant cap on the number of rejected particles during belief reinvigoration. This cap is more likely to be reached when using a larger number of simulations, at which point particles are propagated without rejection leading to reduced belief accuracy. Some possible and simple fixes to this would be to remove the cap, scale the cap with the number of simulations, or use domain knowledge for belief reinvigoration, and is a suggested improvement for subsequent work.

6 CONCLUSION

We presented a scalable planning method for type-based reasoning in large partially observable, multi-agent environments. Our new algorithm, BAPOSGMCP, accounts for partial observability of the environment and the other agents by maintaining a belief over the type of the other agent, the other agent's history, and the environment state. It alleviates the computational intractability of the problem by using a novel multi-agent extension of MCTS combined with a new meta-policy approach for choosing the search policy. Detailed evaluations of BAPOSGMCP demonstrate it's ability to learn the other agent's type online and achieve robust and substantially improved performance in large competitive, cooperative, and mixed environments with up to four agents and 10^{14} states.

Multiple avenues for future research exist. In this work, we limited ourselves to discrete state-action-observation POSGs. Extending BAPOSGMCP to handle continuous spaces would make it more general. Our experiments highlighted some of the limitations of belief updates using particle filter. More efficient belief update methods would help alleviate these limitations. In this work, we proposed a number of meta-policies and validated them empirically, however based on prior work [37], a more principled approach likely exists. Lastly, we limited experiments to the case where the other agents come from the set of known types. Exploring generalization to agents outside of this set would be a useful addition.

ACKNOWLEDGMENTS

If you wish to include any acknowledgments in your paper (e.g., to people or funding agencies), please do so using the 'acks' environment. Note that the text of your acknowledgments will be omitted if you compile your document with the 'anonymous' option.

REFERENCES

[1] Stefano V. Albrecht, Jacob W. Crandall, and Subramanian Ramamoorthy. 2016. Belief and Truth in Hypothesised Behaviours. *Artificial Intelligence* 235 (2016), 63–94.

[2] Stefano V. Albrecht, Somchaya Liemhetcharat, and Peter Stone. 2017. Special Issue on Multiagent Interaction without Prior Coordination: Guest Editorial. *Autonomous Agents and Multi-Agent Systems* 31, 4 (2017), 765–766.

[3] Stefano V. Albrecht and Subramanian Ramamoorthy. 2013. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. 1155–1156.

[4] Stefano V. Albrecht and Subramanian Ramamoorthy. 2014. On Convergence and Optimality of Best-Response Learning with Policy Types in Multiagent Systems. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. 12–21.

[5] S. V. Albrecht and Peter Stone. 2017. Reasoning about Hypothetical Agent Behaviours and Their Parameters. In *16th International Conference on Autonomous Agents and Multiagent Systems 2017*. International Foundation for Autonomous Agents and Multiagent Systems, 547–555.

[6] Stefano V. Albrecht and Peter Stone. 2018. Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems. *Artificial Intelligence* 258 (May 2018), 66–95. https://doi.org/10.1016/j.artint.2018.01.002

[7] Christopher Amato and Frans A. Oliehoek. 2014. Scalable Planning and Learning for Multiagent POMDPs: Extended Version. *arXiv:1404.1140 [cs]* (Dec. 2014). arXiv:1404.1140 [cs]

[8] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine learning* 47, 2 (2002), 235–256.

[9] Samuel Barrett, Noa Agmon, Noam Hazon, Sarit Kraus, and Peter Stone. 2014. Communicating with Unknown Teammates.. In *ECAI*. 45–50.

[10] Samuel Barrett and Peter Stone. 2015. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[11] Samuel Barrett, Peter Stone, and Sarit Kraus. 2011. Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 567–574.

[12] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of OR* 27, 4 (Nov. 2002), 819–840. https://doi.org/10.1287/moor.27.4.819.297

[13] Michael Bowling and Peter McCracken. 2005. Coordination and Adaptation in Impromptu Teams. In *AAAI*, Vol. 5. 53–58.

[14] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games. *arXiv:2007.13544 [cs]* (Nov. 2020). arXiv:2007.13544 [cs]

[15] Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for Heads-up No-Limit Poker: Libratus Beats Top Professionals. *Science* 359, 6374 (2018), 418–424.

[16] Noam Brown and Tuomas Sandholm. 2019. Superhuman AI for Multiplayer Poker. *Science* 365, 6456 (2019), 885–890.

[17] Muthukumaran Chandrasekaran, Prashant Doshi, Yifeng Zeng, and Yingke Chen. 2014. Team Behavior in Interactive Dynamic Influence Diagrams with Applications to Ad Hoc Teams. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*. 1559–1560.

[18] Shushman Choudhury, Jayesh K. Gupta, Peter Morales, and Mykel J. Kochenderfer. 2022. Scalable Online Planning for Multi-Agent MDPs. *Journal of Artificial Intelligence Research* 73 (2022), 821–846.

[19] Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. 2020. Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems* 33 (2020), 10707–10717.

[20] Miguel A. Costa-Gomes and Vincent P. Crawford. 2006. Cognition and Behavior in Two-Person Guessing Games: An Experimental Study. *American economic review* 96, 5 (2006), 1737–1768.

[21] Rémi Coulom. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *International Conference on Computers and Games*. Springer, 72–83.

[22] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. 2012. Information Set Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 120–143.

[23] Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. 2021. K-Level Reasoning for Zero-Shot Coordination in Hanabi. *Advances in Neural Information Processing Systems* 34 (2021).

[24] Aleksander Czechowski and Frans A. Oliehoek. 2021. Decentralized MCTS via Learned Teammate Models. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 81–88.

[25] Prashant Doshi and Piotr J. Gmytrasiewicz. 2005. A Particle Filtering Based Approach to Approximating Interactive Pomdps. In *AAAI*. 969–974.

[26] P. Doshi and P. J. Gmytrasiewicz. 2009. Monte Carlo Sampling Methods for Approximating Interactive POMDPs. *Journal of Artificial Intelligence Research* 34 (March 2009), 297–337. https://doi.org/10.1613/jair.2630

[27] Prashant Doshi and Dennis Perez. 2008. Generalized Point Based Value Iteration for Interactive POMDPs.. In *AAAI*. 63–68.

[28] Adam Eck, Maulik Shah, Prashant Doshi, and Leen-Kiat Soh. 2020. Scalable Decision-Theoretic Planning in Open and Typed Multiagent Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 7127–7134.

[29] Piotr J. Gmytrasiewicz and Prashant Doshi. 2005. A Framework for Sequential Planning in Multi-Agent Settings. *Journal of Artificial Intelligence Research* 24 (2005), 49–79.

[30] Arthur Guez, David Silver, and Peter Dayan. 2013. Scalable and Efficient Bayes-adaptive Reinforcement Learning Based on Monte-Carlo Tree Search. *Journal of Artificial Intelligence Research* 48 (2013), 841–883.

[31] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the 19th National Conference on Artifical Intelligence (AAAI'04)*. AAAI Press, San Jose, California, 709–715.

[32] Leonid V. Kantorovich. 1960. Mathematical Methods of Organizing and Planning Production. *Management science* 6, 4 (1960), 366–422.

[33] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. 2017. Learning in POMDPs with Monte Carlo Tree Search. In *International Conference on Machine Learning*. PMLR, 1819–1827.

[34] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning*. Springer, 282–293.

[35] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. 2020. Rethinking Formal Models of Partially Observable Multiagent Decision Making. *arXiv:1906.11110 [cs]* (Oct. 2020). arXiv:1906.11110 [cs]

[36] Hanna Kurniawati and Vinay Yadav. 2016. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. In *Robotics Research*. Springer, 611–629.

[37] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. 2017. A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning. *arXiv preprint arXiv:1711.00832* (2017). arXiv:1711.00832

[38] J. Z. Leibo, V. F. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. 2017. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In *AAMAS*, Vol. 16. ACM, 464–473.

[39] Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. 2020. Improving Policies via Search in Cooperative Partially Observable Games. *AAAI* 34, 05 (April 2020), 7187–7194. https://doi.org/10.1609/aaai.v34i05.6208

[40] Adam Lerer and Alexander Peysakhovich. 2019. Learning Existing Social Conventions via Observationally Augmented Self-Play. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 107–114.

[41] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning*. PMLR, 3053–3062.

[42] Kevin R. McKee, Joel Z. Leibo, Charlie Beattie, and Richard Everett. 2022. Quantifying the Effects of Environment and Population Diversity in Multi-Agent Reinforcement Learning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 1–16.

[43] Reuth Mirsky, Ignacio Carlucho, Arrasy Rahman, Elliot Fosong, William Macke, Mohan Sridharan, Peter Stone, and Stefano V. Albrecht. 2022. A Survey of Ad Hoc Teamwork: Definitions, Methods, and Open Problems. *arXiv preprint arXiv:2202.10450* (2022). arXiv:2202.10450

[44] John F. Nash. 1950. Equilibrium Points in N-Person Games. *Proceedings of the national academy of sciences* 36, 1 (1950), 48–49.

[45] Alessandro Panella and Piotr Gmytrasiewicz. 2016. Bayesian Learning of Other Agents' Finite Controllers for Interactive POMDPs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, Phoenix, Arizona, 2530–2536.

[46] Alessandro Panella and Piotr Gmytrasiewicz. 2017. Interactive POMDPs with Finite-State Models of Other Agents. *Auton Agent Multi-Agent Syst* 31, 4 (2017), 861–904. https://doi.org/10.1007/s10458-016-9359-z

[47] Georgios Papoudakis, Filippos Christianos, and Stefano Albrecht. 2021. Agent Modelling under Partial Observability for Deep Reinforcement Learning. *Advances in Neural Information Processing Systems* 34 (2021), 19210–19222.

[48] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. 2021. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

[49] Christopher D. Rosin. 2011. Multi-Armed Bandits with Episode Context. *Annals of Mathematics and Artificial Intelligence* 61, 3 (2011), 203–230.

[50] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, and Thore Graepel. 2020. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature* 588, 7839 (2020), 604–609.

[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017). arXiv:1707.06347

[52] Jonathon Schwartz, Ruijia Zhou, and Hanna Kurniawati. 2022. Online Planning for Interactive-Pomdps Using Nested Monte Carlo Tree Search. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

[53] Iris Rubi Seaman, Jan-Willem van de Meent, and David Wingate. 2018. Nested Reasoning About Autonomous Agents Using Probabilistic Programs. *arXiv preprint arXiv:1812.01569* (2018). arXiv:1812.01569

[54] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *nature* 529, 7587 (2016), 484–489.

[55] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, and Thore Graepel. 2018. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play. *Science* 362, 6419 (2018), 1140–1144.

[56] David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. *Advances in Neural Information Processing Systems* 23 (2010), 2164–2172.

[57] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. 2005. Bayes' Bluff: Opponent Modelling in Poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. 550–558.

[58] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[59] Malcolm Strens. 2000. A Bayesian Framework for Reinforcement Learning. In *ICML*, Vol. 2000. 943–950.

[60] Zachary Sunberg and Mykel Kochenderfer. 2018. Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces. *Proceedings of the International Conference on Automated Planning and Scheduling* 28 (June 2018), 259–263.

[61] Ming Tan. 1993. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*. 330–337.

[62] Gerald Tesauro. 1994. TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural computation* 6, 2 (1994), 215–219.

[63] William E. Walsh, Rajarshi Das, Gerald Tesauro, and Jeffrey O. Kephart. 2002. Analyzing Complex Strategic Interactions in Multi-Agent Systems. In *AAAI-02 Workshop on Game-Theoretic and Decision-Theoretic Agents*. 109–118.

[64] Michael P. Wellman. 2006. Methods for Empirical Game-Theoretic Analysis. In *AAAI*. 1552–1556.

[65] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. 2011. Online Planning for Ad Hoc Autonomous Agent Teams. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[66] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. 2017. DESPOT: Online POMDP Planning with Regularization. *Journal of Artificial Intelligence Research* 58 (Jan. 2017), 231–266. https://doi.org/10.1613/jair.5328

[67] Elnaz Shafipour Yourdshahi, Thomas Pinder, Gauri Dhawan, Leandro Soriano Marcolino, and Plamen Angelov. 2018. Towards Large Scale Ad-Hoc Teamwork. In *2018 IEEE International Conference on Agents (ICA)*. IEEE, 44–49.

A ENVIRONMENTS

In this section we describe the environment used for experiments in more detail.

Driving: A general-sum 2D grid world navigation problem requiring coordination [40, 42]. Each agent controls a vehicle and is tasked with driving the vehicle from start to destination while avoiding crashing into other vehicles. Each agent observes their local area, speed, and destination, and whether they've reached their destination or crashed. Agents receive a reward of 1 if they reach their destination and -1 if they crash. To reduce exploration difficulty agents receive a reward of 0.05 each time they make progress towards their destination. The exploration bonus is analogous to GPS in that it provides guidance for navigation but not for how to coordinate with other vehicles. Episodes ended when all agents had either crashed or reached their destination, or 50 steps had passed.

Level-Based Foraging (LBF): [3, 5] is a mixed incentive environment where agents must collect food in a grid world. Each agent is assigned a level and can only pick-up a piece of food alone if the food is of equal or lower level. For food with a higher level than the agent, the agent must coordinate with the other agent. We use the partially-observable version of LBF used in [19, 48]. We slightly modified the environment so that food and agents always spawn in the same locations, while their levels are still randomly generated. This simplifies the initial belief for each agent and does not impact the cooperation-competition dynamics of the problem. Episodes ended when all food was picked-up, or 50 steps had passed.

Pursuit-Evasion (PE): An asymmetric zero-sum grid world problem involving two agents, an evader and a pursuer [52, 53]. The evader's goal is to reach a safe location, while the the pursuer's aim is to spot the evader before it reaches it's goal. The evader is considered caught if it is observed by the pursuer. Both agents have knowledge of each others starting locations, however, only the evader has knowledge of it's goal location. The pursuer only knows the set of possible safe locations. Thus, this environment requires each agent to reason about the which path the other agent will take through the dense grid environment. Each agent receives six bits of observation per step. Four bits indicate whether there is a wall or not in each of the cardinal directions, one bit indicates whether the opponent can be seen in the agent's field of vision, and the final bit indicates whether the opponent can be heard within Manhattan distance two of the agent. Due to the lack of precision of these observations, the pursuer never knows the exact position of the evader and vice versa. Similar to the Driving environment the evader agent receives a small bonus whenever it makes progress towards the safe location, while the pursuer receives the opposite reward. Episodes ended when the evader was captured or reached the safe location, or 100 steps had passed.

Predator-Prey (PP): A co-operative grid world problem involving multiple predator agents working together to catch prey [38, 61]. Prey are controlled autonomously and preference movement away from any observable predators or other prey. Predators can catch prey by being in an adjacent cell, with the number of predators required to catch a prey based on the prey strength. Both predators and prey can observe a 5-by-5 area around themselves, namely whether each cell contains a wall, predator, or prey or is empty. Each prey capture gives all predators a reward of $1/N_{prey}$. Predators start each episode from random separate locations along the edge of the grid, while prey start together in the center of the grid. We ran experiments on two different versions of the environment, where both versions had three prey. The *two-agent* version had two predators with each prey requiring two predators to capture. The *four-agent* version had four predators with prey requiring three predators to capture. Both versions required coordination between agents to capture the prey. Episodes ended when all prey were captured, or 50 steps had passed.

B FIXED POLICIES

B.1 Training

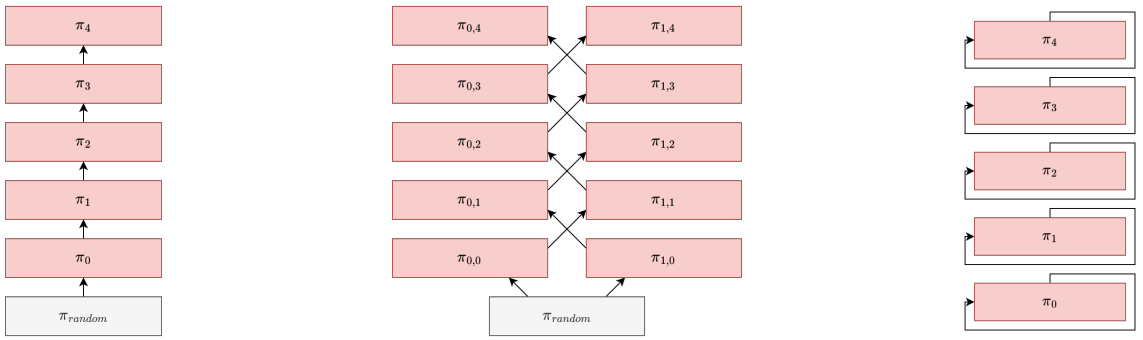
For each of the Driving, PE, and PP environments we trained a set of neural network policies using reinforcement learning. We used different multi-agent training schemes for each environment, however the same method was used the same setting for each individual policy. Specifically, for each individual policy we used the Rllib [41] implementation of the Proximal Policy Optimization (PPO) model-free, policy-gradient method [51] for training. We used the same neural network architecture for all policies, namely two fully-connected layers with 64 and 32 units, respectively, followed by a 256 unit LSTM, whose output was fed into separate fully connected output heads for the policy and value functions. The neural network architecture and training hyperparameters are shown in Table 1. Training hyperparameters were consistent across environments, except for some policies in the PP environment where we used experimented with different hyper parameter values as it lead to similar performance with faster training times. All policies were trained until convergence, as indicated by learning curve.

Hyper parameter	Driving	PE	PP
Training steps	10,240,000		
Fully Connected Network Layers	[64, 32]		
LSTM Cell Size	256		
Learning Rate	0.0003		
KL Coefficient	0.2		
KL target	0.01		
Batch size	2048		
LSTM training sequence length	20		
Entropy Bonus Cofficient	0.001		
Clip param	0.3		
γ	0.99	0.99	[0.99, 0.999]
$GAE\lambda$	0.9	0.9	[0.90, 0.95]
SGD Minibatch size	256	256	[256, 512]
Num. SGD Iterations	10	10	[10, 2]

Table 1: Driving, PE, PP policy training hyperparameters.

B.2 Driving Policies

For the Driving experiments we trained a set of five K-level reasoning (KLR) policies. Policies are trained in a hierarchy, the level $K = 0$ policy is trained against a uniform random policy, level $K = 1$ is trained against the level $K = 0$, and so on with the level K policy trained as a best response to the level $K - 1$ policy. We trained policies synchronously using the Synchronous KLR training method [23]. Figure 5 provides a visualization of the training schema used. Figure 6 shows the pairwise performance for the Driving environment policies. Each policy was evaluated against each other policy for 1000 episodes.



(a) Driving **(b) Pursuit-Evasion** **(c) Predator-Prey**

Figure 5: Multi-agent training schemas used for generating the fixed policies for the different environments (adapted from [23]). For the Pursuit-Evasion environment separate policies were trained for evader (agent 0) and pursuer (agent 1) agents.

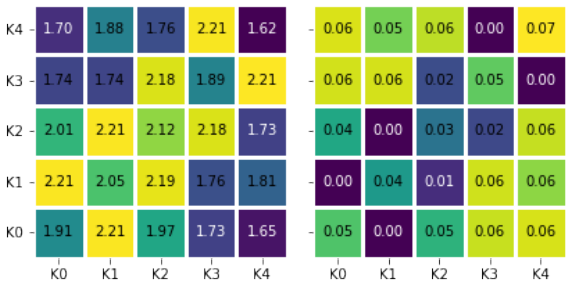


Figure 6: Payoff matrix for fixed KLR policies in the Driving environment. The left figure shows expected returns for the row policy. The right figure shows the corresponding 95% **confidence intervals.**

B.3 Level-Based Foraging Policies

For the LBF environment we used the set of four heuristic policies used in prior work [5, 47]. Figure 7 shows the pairwise performance of the four heuristic policies, generated from 1000 episodes per pairing.

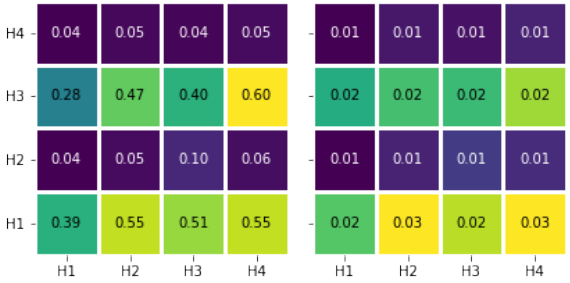


Figure 7: Payoff matrix for LBF heuristic policies. The left figure shows expected returns for the row policy. The right figure shows the corresponding 95% **confidence intervals.**

B.4 Pursuit Evasion Policies

For the PE experiments we trained a set of five KLR policies, similar to the Driving experiments. The only difference being that we trained separate policies for the Evader and Pursuer at each reasoning level. Figure 5 provides a visualization of the training schema used. Separate policies were used because the PE problem is asymmetric, with the pursuer and evader having different objectives. Figure 8 shows the pairwise performance for the PE environment policies. Each policy was evaluated against each other policy for 1000 episodes.

B.5 Predator Prey Policies

For the PP experiments we trained a set of five independent policies using self-play. Each policy was initialized with a different seed and trained only with itself. Figure 5 provides a visualization of the training schema used. We trained separate policies for the two-agent and four-agent versions of the PP environment used in the experiments. Figure 9 shows the pairwise performance for set of policies in each version of the environment, generated from 1000 episodes for each pairing. For the four-agent version we show the results from matching the row policy with a team of three versions of the same policy (e.g. T0 is three copies of policy S0).

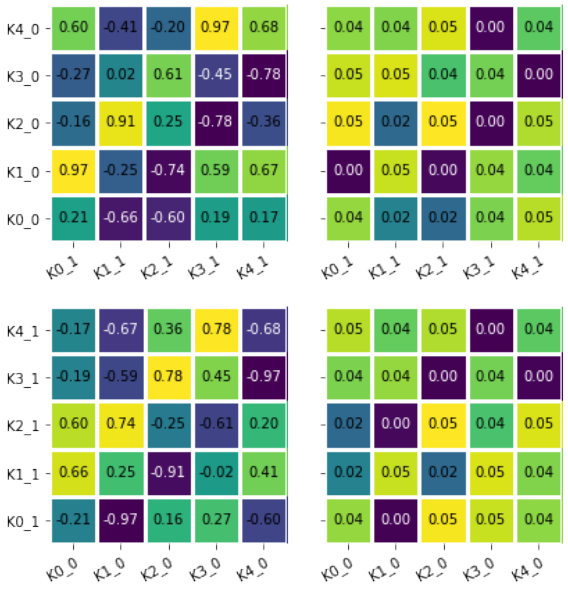


Figure 8: Payoff matrices for fixed policies in the PE environment. The top and bottom rows show the payoffs for the evader and pursuer agents, respectively. In each row the left figure shows expected returns for the row policy and the right figure shows the corresponding 95% confidence intervals.

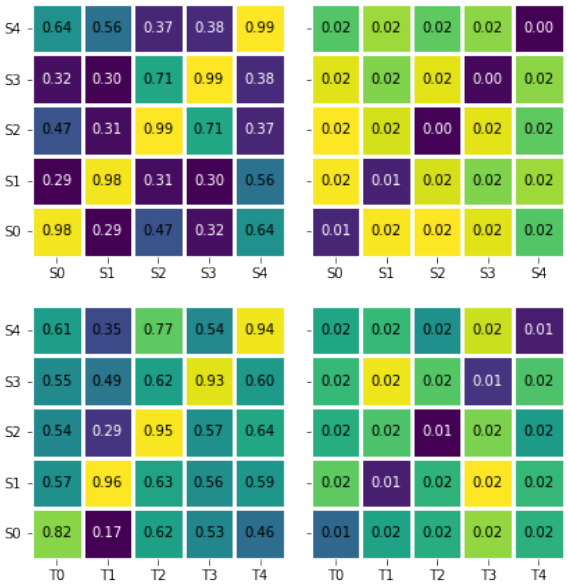
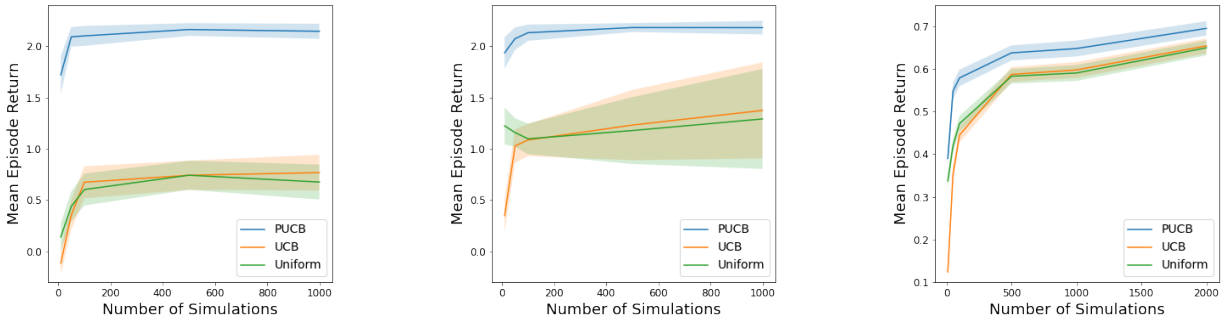


Figure 9: Payoff matrices for policies in the PP environment. The top shows payoff for the two-agent version and the bottom rows show the payoffs for four-agent version. In each row the left figure shows expected returns for the row policy and the right figure shows the corresponding 95% confidence intervals. Four the four-agent version each column represents the team of three agents consisting of three copies of the same policy (e.g. T0 is made up of three copies of policy S0).

C ACTION SELECTION STRATEGIES

Figure 10 shows results comparing different search action selection strategies for BAPOSGMCP in the Driving and LBF environments. We compared PUCB which was used in the main paper results with two baseline strategies: *uniform* and *UCB*. Each strategy controls how actions are chosen from decision nodes during search. Uniform action selection selects each action equally often by always selecting the action with the lowest visit count. When using uniform action selection the action with the highest value from the root node is chosen as the real action to use. UCB action selection uses the popular Upper Confidence Bound selection strategy for MCTS [8, 21, 34].

PUCB clearly dominates performance for both environments. This is especially true when using the value function for leaf node evaluation, as opposed to using policy rollouts.



(a) Driving using value function. **(b) Driving using rollouts.** **(c) LBF using rollouts.**

Figure 10: Performance of BAPOSGMCP using different action selections strategies. Shaded areas show 95% **confidence interval.**

D LEAF NODE EVALUATION

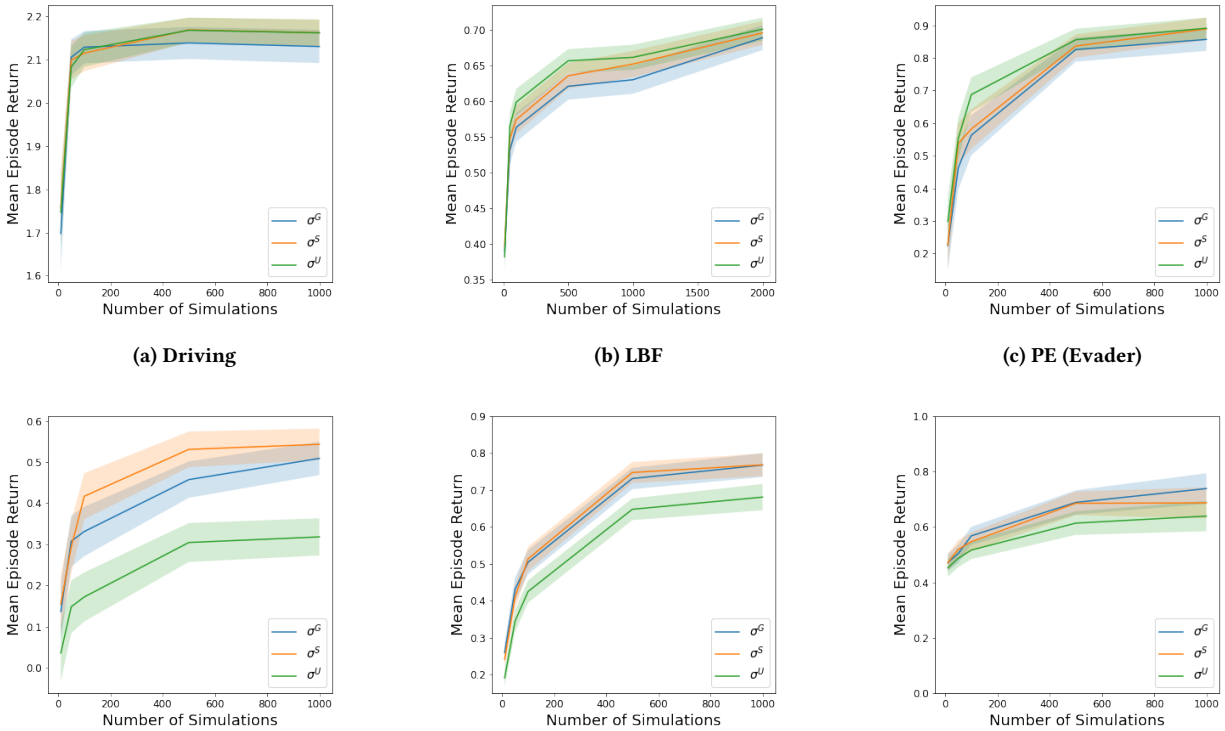
Figure 11 shows results comparing value function and rollout based leaf node evaluations. Results are for BAPOSGMCP in the Driving environment. Results are from running 100 episodes. For value function evaluations, we assume that we have access to the value function of the current policy (as selected by the meta-policy). In our experiments for the Driving, PE, and PP environments policies are Actor-Critic neural networks base policies which have both policy and value functions. The value function is used to evaluate the leaf node using the history of the planning agent at the leaf node (as per [50, 55]). For rollout evaluations the remainder of the episode is played using the current rollout policy (as selected by the meta-policy) and the state and other agent policies and histories contained in the current history-policy-state particle (similar to standard POMDP based MCTS methods [56]). The discounted sum of rewards along the rolled out trajectory is then used as a value estimate for the leaf node.

Performance in terms of episode return is comparable between the two leaf node evaluation methods. However, value function evaluation has a significant advantage in terms of mean search time per step. This of course is due to avoiding time costly rollouts. The cost of rollouts is further exacerbated in our experiments by the use of neural network based rollout policies that typically have relatively high sample cost compared with simple handcrafted rollout policies such as those used for the LBF problem.

Figure 11: Comparison of leaf node evaluation methods in BAPOSGMCP. Shaded areas show 95% confidence interval. The left figure shows mean episode return, while the right shows mean search time per real environment step.

E EVALUATION OF DIFFERENT META-POLICIES

Figure 12 shows the performance of BAPOSGMCP using the different meta-policies in each environment.



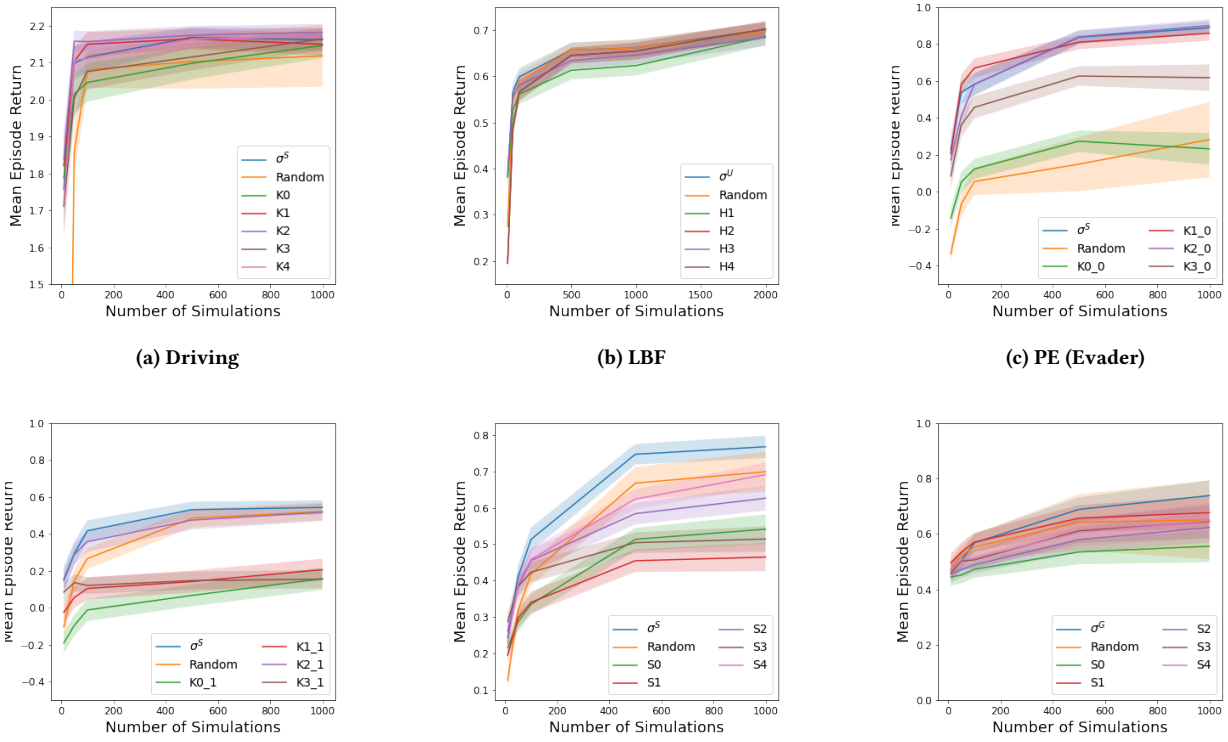
(a) Driving **(b) LBF** **(c) PE (Evader)**

(d) PE (Pursuer) **(e) PP (two-agents)** **(f) PP (four-agents)**

Figure 12: Performance of BAPOSGMCP with greedy σ^G, softmax σ^S ($\tau = 0.25$), and uniform σ^U meta-policies in each environment.

F EVALUATION OF DIFFERENT SEARCH POLICIES

Figure 13 compares the performance of BAPOSGMCP using a meta-policy against using random and fixed policies.

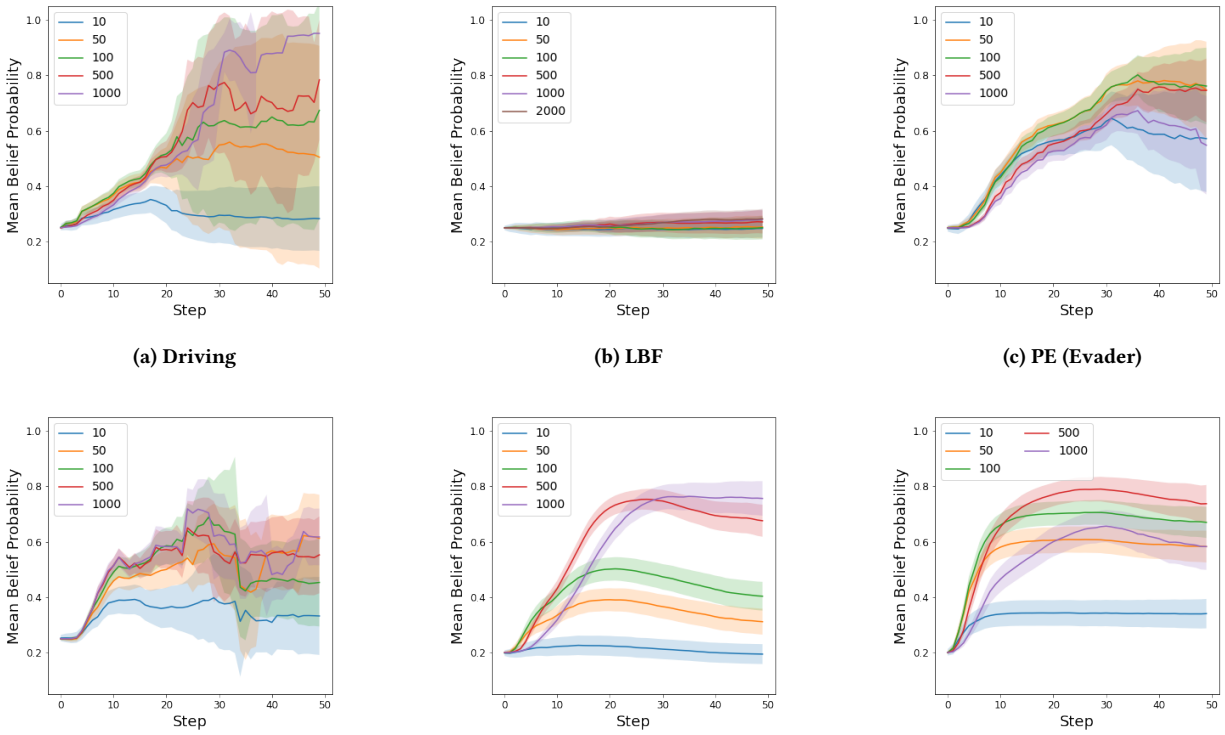


(a) Driving **(b) LBF** **(c) PE (Evader)**

(d) PE (Pursuer) **(e) PP (two-agents)** **(f) PP (four-agents) IP**

Figure 13: Comparison of BAPOSGMCP using different search policies in each environment. Each figure shows performance of BAPOSGMCP using the best performing meta-policy, the uniform random policy, and each of the available fixed policies.

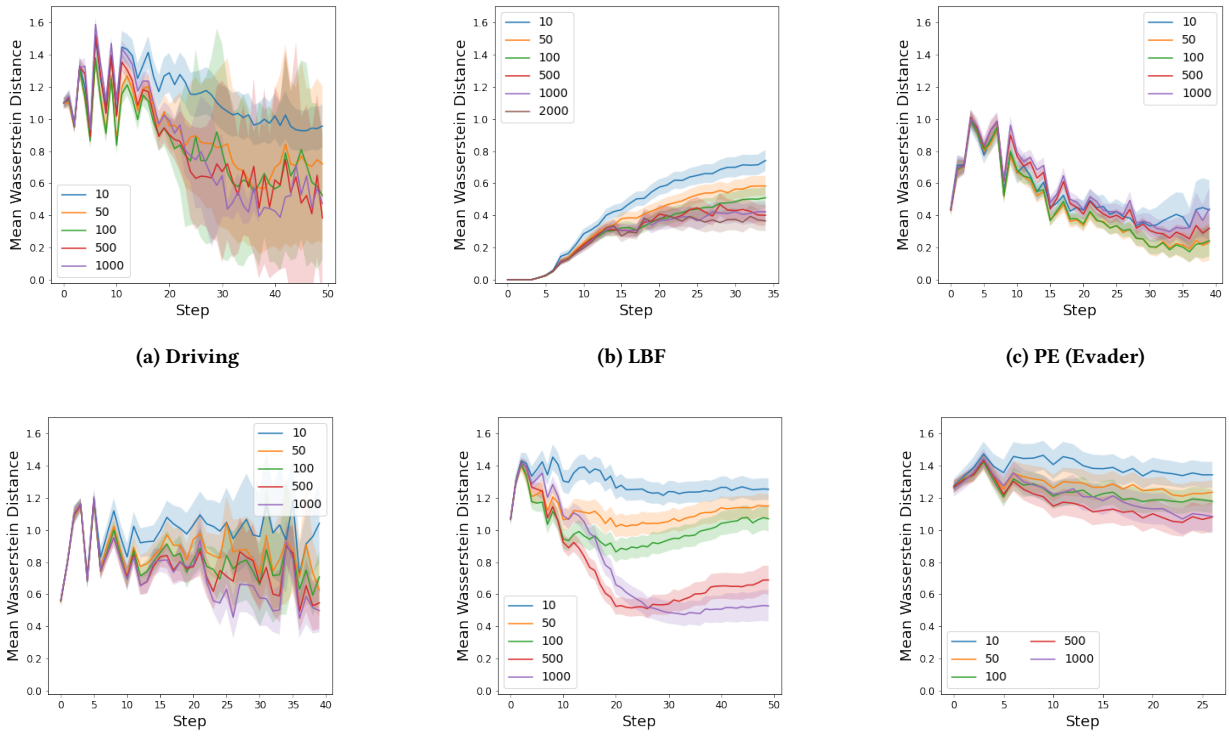
G BELIEF ACCURACY



(a) Driving (b) LBF (c) PE (Evader)

(d) PE (Pursuer) (e) PP (two-agents) (f) PP (four-agents)

Figure 14: Mean probability assigned to the true policy of the other agent in the belief of BAPOSGMCP by episode step. Each line is BAPOSGMCP using a different number of simulations. Shaded areas show 95% confidence intervals. For the Driving, LBF, and PE environments episodes lengths were often shorter than the max episode lengths, leading to larger confidence intervals for later steps.



(a) Driving (b) LBF (c) PE (Evader)

(d) PE (Pursuer) (e) PP (two-agents) (f) PP (four-agents)

Figure 15: Mean per step Wasserstein distance between the estimated policy of the other agent in the belief of BAPOSGMCP and the true policy of the other agent. Each line is BAPOSGMCP using a different number of simulations. Shaded areas show 95% confidence intervals. For the Driving, LBF, and PE environments episodes lengths were often shorter than the max episode lengths, thus why the confidence intervals tend to grow for later steps.