# Scalable Planning and Learning for Multiagent POMDPs: Extended Version

Christopher Amato
CSAIL, MIT
Cambridge, MA 02139
camato@csail.mit.edu

Frans A. Oliehoek
Informatics Institute, University of Amsterdam
Dept. of CS, University of Liverpool
frans.oliehoek@liverpool.ac.uk

**Abstract**

Online, sample-based planning algorithms for POMDPs have shown great promise in scaling to problems with large state spaces, but they become intractable for large action and observation spaces. This is particularly problematic in multiagent POMDPs where the action and observation space grows exponentially with the number of agents. To combat this intractability, we propose a novel scalable approach based on sample-based planning and factored value functions that exploits structure present in many multiagent settings. This approach applies not only in the planning case, but also in the Bayesian reinforcement learning setting. Experimental results show that we are able to provide high quality solutions to large multiagent planning and learning problems.

## 1 Introduction

Online planning methods for POMDPs have demonstrated impressive performance (Ross et al., 2008) on large problems by interleaving planning with action selection. The leading such method, partially observable Monte Carlo planning (POMCP) (Silver and Veness, 2010), achieves performance gains by extending sample-based methods based on Monte Carlo tree search (MCTS) to solve POMDPs.

While online, sample-based methods show promise in solving POMDPs with large state spaces, they become intractable as the number of actions or observations grow. This is particularly problematic in the case of multiagent systems. Specifically, we consider multiagent partially observable Markov decision processes (MPOMDPs), which assume all agents share the same partially observable view of the world and can coordinate their actions. Because the MPOMDP model is centralized, POMDP methods apply, but the fact that the number of *joint* actions and observations scales exponentially in the number of agents renders current POMDP methods intractable.

To combat this intractability, we provide a novel sample-based online planning algorithm that exploits multiagent structure. Our method, called *factored-value partially observable Monte Carlo planning (FV-POMCP)*, is based on POMCP and is the first MCTS method that exploits *locality of interaction*: in many MASs, agents interact directly with a subset of other agents. This structure enables a decomposition of the value function into a set of overlapping factors, which can be used to produce high quality solutions (Guestrin, Koller, and Parr, 2001; Nair et al., 2005; Kok and Vlassis, 2006). But unlike these previous approaches, we will not assume a factored model, but only that the value function can be approximately factored. We present two variants of FV-POMCP that use different amounts of factorization of the value function to scale to large action and observation spaces.

Not only is our FV-POMCP approach applicable to large MPOMDPs, but it is potentially even more important for Bayesian learning where the agents have uncertainty about the underlying model as modeled by Bayes-Adaptive POMDPs (BA-POMDPs) (Ross et al., 2011). These models translate the learning problem to a planning problem, but since the resulting planning problems have an infinite number of states, scalable sample-based planning approaches are critical to their solution.

We show experimentally that our approach allows both planning and learning to be significantly more efficient in multiagent POMDPs. This evaluation shows that our approach significantly outperforms regular (non-factored) POMCP, indicating that FV-POMCP is able to effectively exploit locality of interaction in both settings.

# 2 Background

We first discuss multiagent POMDPs and previous work on Monte Carlo tree search and Bayesian reinforcement learning (BRL) for POMDPs.

## 2.1 Multiagent POMDPs

An MPOMDP (Messias, Spaan, and Lima, 2011) is a multiagent planning model that unfolds over a number of steps. At every stage, agents take individual actions and receive individual observations. However, in an MPOMDP, all individual observations are shared via communication, allowing the team of agents to act in a 'centralized manner'. We will restrict ourselves to the setting where such communication is free of noise, costs and delays.

An MPOMDP is a tuple $\langle I, S, \{A_i\}, T, R, \{Z_i\}, O, h\rangle$ with: $I$, a set of agents; $S$, a set of states with designated initial state distribution $b_0$; $A = \times_i A_i$, the set of joint actions, using action sets for each agent, $i$; $T$, a set of state transition probabilities: $T^{s\vec{a}s'} = \Pr(s'|s,\vec{a})$, the probability of transitioning from state $s$ to $s'$ when actions $\vec{a}$ are taken by the agents; $R$, a reward function: $R(s,\vec{a})$, the immediate reward for being in state $s$ and taking actions $\vec{a}$; $Z = \times_i Z_i$, the set of joint observations, using observation sets for each agent, $i$; $O$, a set of observation probabilities: $O^{\vec{a}s'\vec{z}} = \Pr(\vec{z}|\vec{a},s')$, the probability of seeing observations $\vec{o}$ given actions $\vec{a}$ were taken and resulting state $s'$; $h$, the horizon.

An MPOMDP can be reduced to a POMDP with a single centralized controller that takes joint actions and receives joint observations (Pynadath and Tambe, 2002). Therefore, MPOMDPs can be solved with POMDP solution methods, some of which will be described in the remainder of this section. However, such approaches do not exploit the particular structure inherent to many MASs. In Sec. 4, we present a first online planning method that overcomes this deficiency.

## 2.2 Monte Carlo Tree Search for (M)POMDPs

Most research for (mutliagent) POMDPs has focused on *planning*: given a full specification of the model, determine an optimal policy, $\pi$, mapping past observation histories (which can be summarized by distributions $b(s)$ over states called beliefs) to actions. An optimal policy can be extracted from an optimal Q-value function, $Q(b,a) = \sum_s R(s,a) + \sum_z P(z|b,a) \max_{a'} Q(b',a')$, by acting greedily. Computing $Q(b,a)$, however, is complicated by the fact that the space of beliefs is continuous.

POMCP (Silver and Veness, 2010), is a scalable method which extends Monte Carlo tree search (MCTS) to solve POMDPs. At every stage, the algorithm performs online planning, given the current belief, by incrementally building a lookahead tree that contains (statistics that represent) $Q(b,a)$. The algorithm, however, avoids expensive belief updates by creating nodes not for each belief, but simply for each action-observation history $h$. In particular, it samples hidden states $s$ only at the root node (called 'root sampling') and uses that state to sample a trajectory that first traverses the lookahead tree and then performs a (random) rollout. The return of this trajectory is used to update the statistics for all visited nodes. Because this search tree can be enormous, the search is directed to the relevant parts by selecting actions to maximize the 'upper confidence bounds': $U(h,a) = Q(h,a) + c\sqrt{\log(N+1)/n}$. Here, $N$ is the number of times the history has been reached and $n$ is the number of times that action $a$ has been taken in that history. POMCP can be shown to converge to an $\epsilon$-optimal value function. Moreover, the method has demonstrated good performance in large domains with a limited number of simulations.

## 2.3 Bayesian RL for (M)POMDPs

*Reinforcement learning (RL)* considers the more realistic case where the model is not (perfectly) known in advance. Unfortunately, effective RL in POMDPs is very difficult. Ross et al. (2011) introduced a framework, called the Bayes-Adaptive POMDP (BA-POMDP), that reduces the learning problem to a planning problem, thus enabling advances in planning methods to be used in the learning problem.

In particular, the BA-POMDP utilizes Dirichlet distributions to model uncertainty over transitions and observations. Intuitively, if the agent could observe states and observations, it could maintain vectors $\phi$ and $\psi$ of counts for transitions and observations respectively. Let $\phi^a_{ss'}$ be the transition count of the number times state $s'$ resulted from taking action $a$ in state $s$ and $\psi^a_{s'z}$ be the observation count representing the number of times observation $z$ was seen after taking action $a$ and transitioning to state $s'$. These counts induce a probability distribution over the possible transition and observation models. Even though the agent cannot observe the states and has uncertainty about the actual count vectors, *this uncertainty can be represented using the POMDP formalism* — by including the count vectors as part of the hidden state of a special POMDP, called a BA-POMDP.

The BA-POMDP can be extended to the multiagent setting (Amato and Oliehoek, 2013), yielding the Bayes-Adaptive multiagent POMDP (BA-MPOMDP) framework. BA-MPOMDPs are POMDPs, but with an infinite state space since there can be infinitely many count vectors. While a quality-bounded reduction to a finite state space is possible (Ross et al., 2011), the problem is still intractable; sample-based planning is needed to provide solutions. Unfortunately, current methods, such as POMCP, do not scale well to multiple agents.

# 3 Exploiting Graphical Structure

POMCP is not directly suitable for multiagent problems (in either the planning or learning setting) due to the fact that the number of joint actions and observations are exponential in the number of agents. We first elaborate on these problems, and then sketch an approach to mitigate them by exploiting locality between agents.

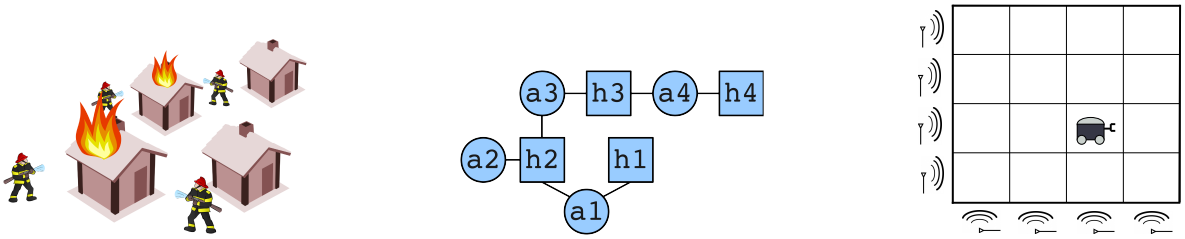
## 3.1 POMCP for MPOMDPs: Bottlenecks

The large number of joint observations is problematic since it leads to a lookahead tree with very high branching factor. Even though this is theoretically not a problem in MDPs (Kearns, Mansour, and Ng, 2002), in partially observable settings that use particle filters it leads to severe problems. In particular, in order to have a good particle representation at the next time step, the actual joint observation received must be sampled often enough during planning for the previous stage. If the actual joint observation had not been sampled frequently enough (or not at all), the particle filter will be a bad approximation (or collapse). This results in sampling starting from the initial belief again, or alternatively, to fall back to acting using a separate (history independent) policy such as a random one.

The issue of large numbers of joint actions is also problematic: the standard POMCP algorithm will, at each node, maintain separate statistics, and thus separate upper confidence bounds, for each of the exponentially many joint actions. Each of the exponentially many joint actions will have to be selected at least a few times to reduce their confidence bounds (i.e., exploration bonus). This is a principled problem: in cases where each combination of individual actions may lead to completely different effects, it may be necessary to try all of them at least a few times. In many cases, however, the effect of a joint action is factorizable as the effects of the action of individual agents or small groups of agents. For instance, consider a team of agents that is fighting fires at a number of burning houses, as illustrated in Fig. 1(a). The rewards depend only on the amount of water deposited on each house rather than the exact joint action taken (Oliehoek et al., 2008). While this problem lends itself to a natural factorization, other problems may also be factorized to permit approximation.

## 3.2 Coordination Graphs

In certain multiagent settings, *coordination (hyper) graphs (CGs)* (Guestrin, Koller, and Parr, 2001; Nair et al., 2005) have been used to compactly represents interactions between subsets of agents. In this paper we extend




(a) (b) (c)

Figure 1: (a) Illustration of 'fire fighting' (b) Coordination graph with 4 houses and 3 agents (c) Illustration of a sensor network problem on a grid that is used in the experiments.

this approach to MPOMDPs. We first introduce the framework of CGs in the single shot setting.

A CG specifies a set of payoff components $E = \{Q_e\}$, and each component $e$ is associated with a subset of agents. These subsets (which we also denote using $e$) can be interpreted as (hyper)-edges in a graph where the nodes are agents. The goal in a CG is to select a joint action that maximizes the sum of the local payoff components $Q(\vec{a}) = \sum_e Q_e(\vec{a}_e)$.[1] A CG for the fire fighting problem is shown in Fig. 1(b). We follow the cited literature in assuming that a suitable factorization is easily identifiable by the designer, but it may also be learnable. Even if a payoff function $Q(\vec{a})$ does not factor exactly, it can be approximated by a CG. For the moment assuming a *stateless problem* (we will consider the case where histories are included in the next section), an action-value function can be approximated by

$$Q(\vec{a}) \approx \sum_e Q_e(\vec{a}_e), \tag{1}$$

We refer to this as the linear approximation of $Q$, since one can show that this corresponds to an instance of linear regression (See Sec. 5).

Using a factored representation, the maximization $\max_{\vec{a}} \sum_e Q_e(\vec{a}_e)$ can be performed efficiently via variable elimination (VE) (Guestrin, Koller, and Parr, 2001), or max-sum (Kok and Vlassis, 2006). These algorithms are not exponential in the number of agents, and therefore enable significant speed-ups for larger number of agents. The VE algorithm (which we use in the experiments) is exponential in the *induced width* $w$ of the coordination graph.

### 3.3 Mixture of Experts Optimization

VE can be applied if the CG is given in advance. When we try to exploit these techniques in the context of POMCP, however, this is not the case. As such, the task we consider here is to find the maximum of an *estimated* factored function $\hat{Q}(\vec{a}) = \sum_e \hat{Q}_e(\vec{a}_e)$. Note that we do not necessarily require the best approximation to the entire $Q$, as in (1). Instead, we seek an estimation $\hat{Q}$ for which the maximizing joint action $\vec{a}^M$ is close to the maximum of the actual (but unknown) Q-value: $Q(\vec{a}^M) \approx Q(\vec{a}^*)$.

For this purpose, we introduce a technique called *mixture of experts optimization*. In contrast to methods based on linear approximation (1), we do not try to learn a best-fit factored Q function, but directly try to estimate the maximizing joint action. The main idea is that for each local action $\vec{a}_e$ we introduce an expert that predicts the *total* value $\hat{Q}(\vec{a}_e) = \mathbf{E}[Q(\vec{a}) \mid \vec{a}_e]$. For a joint action, these responses—one of each payoff component $e$—are put in a mixture with weights $\alpha_e$ and used to predict the maximization joint action: $\arg\max_{\vec{a}} \sum_e \alpha_e \hat{Q}(\vec{a}_e)$. This equation is the sum of restricted-scope functions, which is identical to the case of linear approximation (1), so VE can be used to perform this maximization effectively. In the remainder of this paper, we will integrate the weights and simply write $\hat{Q}_e(\vec{a}_e) = \alpha_e \hat{Q}(\vec{a}_e)$.

[1] Since we focus on the one-shot setting here, the Q-values in the remainder of this section should be interpreted as those for one specific joint history $h$, i.e.: $Q(\vec{a}) \equiv Q(h,\vec{a})$.

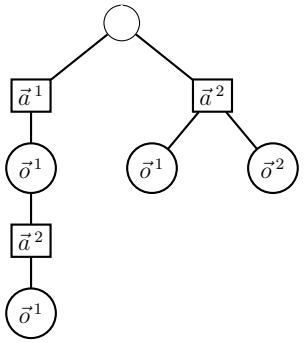



Figure 2: Factored Statistics: joint histories are maintained (for specific joint actions and observations specified by $\vec{a}^{\,j}$ and $\vec{o}^{\,k}$), but action statistics are factored at each node.

The experts themselves are implemented as maximum-likelihood estimators of the total value. That is, each expert (associated with a particular $\vec{a}_e$) keeps track of the mean payoff received when $\vec{a}_e$ was performed, which can be done very efficiently. An additional benefit of this approach is that it allows for efficient estimation of upper confidence bounds by also keeping track of how often this local action was performed, which in turns facilitates easy integration in POMCP, as we describe next.

# 4 Factored-Value POMCP

This section presents our main algorithmic contribution: Factored-Value POMCP, which is an online planning method for POMDPs that can exploit approximate structure in the value function by applying mixture of experts optimization in the POMCP lookahead search tree. We introduce two variants of FV-POMCP. The first technique, *factored statistics*, only addresses the complexity introduced by joint actions. The second technique, *factored trees*, additionally addresses the problem of many joint observations. FV-POMCP is the first MCTS method to exploit structure in MASs, achieving better sample complexity by using factorization to generalize the value function over joint actions and histories. While this method was developed to scale POMCP to larger MPOMDPs in terms of number of agents, the techniques may be beneficial in other multiagent models and other factored POMDPs.

## 4.1 Factored Statistics

We first introduce *Factored Statistics* which directly applies mixture of experts optimization to each node in the POMCP search tree. As shown in Fig. 2, the tree of joint histories remains the same, but the statistics retained at for each history is now different. That is, rather than maintaining one set of statistics in each node (i.e, joint history $\vec{h}$) for the expected value of each joint action $Q(\vec{h}, \vec{a})$, we maintain a set of statistic for each component $e$ that estimates the values $Q_e(\vec{h}, \vec{a}_e)$ and corresponding upper confidence bounds.

Joint actions are selected according to the maximum (factored) upper confidence bound:

$$\max_{\vec{a}} \sum_e U_e(\vec{h}, \vec{a}_e),$$

Where $U_e(\vec{h}, \vec{a}_e) \triangleq Q_e(\vec{h}, \vec{a}_e) + c\sqrt{\log(N_{\vec{h}} + 1)/n_{\vec{a}_e}}$ using the Q-value and the exploration bonus added for that factor. For implementation, at each node for a joint history $\vec{h}$, we store the count for the full history $N_{\vec{h}}$ as well as the Q-values, $Q_e$, and the counts for actions, $n_{\vec{a}_e}$, separately for each component $e$.

Since this method retains fewer statistics and performs joint action selection more efficiently via VE, we expect that it will be more efficient than plain application of POMCP to the BA-MPOMDP. However, the complexity due to joint observations is not directly addressed: because joint histories are used, reuse of nodes

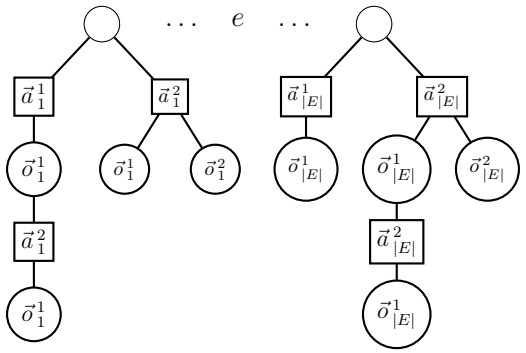



Figure 3: Factored Trees: local histories for are maintained for each factor (resulting in factored history and action statistics). Actions and observations for component $i$ are represented as $\vec{a}\,^j_i$ and $\vec{o}\,^k_i$)

and creation of new nodes for all possible histories (including the one that will be realized) may be limited if the number of joint observations is large.

### 4.2 Factored Trees

The second technique, called *Factored Trees*, additionally tries to overcome the large number of joint observations. It further decomposes the local $Q_e$'s by splitting joint histories into local histories and distributing them over the factors. That is, in this case, we introduce an expert for each local $\vec{h}_e, \vec{a}_e$ pair. During simulations, the agents know $\vec{h}$ and action selection is conducted by maximizing over the sum of the upper confidence bounds:

$$\max_{\vec{a}} \sum_e U_e(\vec{h}_e, \vec{a}_e),$$

where $U_e(\vec{h}_e, \vec{a}_e) = Q_e(\vec{h}_e, \vec{a}_e) + c\sqrt{\log(N_{\vec{h}_e} + 1)/n_{\vec{a}_e}}$. We assume that the set of agents with relevant actions and histories for component $Q_e$ are the same, but this can be generalized. This approach further reduces the number of statistics maintained and increases the reuse of nodes in MCTS and the chance that nodes in the trees will exist for observations that are seen during execution. As such, it can increase performance by increasing generalization as well as producing more robust particle filters.

This type of factorization has a major effect on the implementation: rather than constructing a single tree, we now construct a number of trees in parallel, one for each factor $e$ as shown in Fig. 3. A node of the tree for component $e$ now stores the required statistics: $N_{\vec{h}_e}$, the count for the local history, $n_{\vec{a}_e}$, the counts for actions taken in the local tree and $Q_e$ for the tree. Finally, we point out that this decentralization of statistics has the potential to reduce communication since the components statistics in a decentralized fashion could be updated without knowledge of all observation histories.

## 5 Theoretical Analysis

Here, we investigate the approximation quality induced by our factorization techniques.[2] The most desirable quality bounds would express the performance relative to 'optimal', i.e., relative to flat POMCP, which converges in probability an $\epsilon$-optimal value function. Even for the one-shot case, this is extremely difficult for any method employing factorization based on linear approximation of Q, because Equation (1) corresponds to a special case of linear regression. In this case, we can write (1) in terms of basis functions and weights as: $\sum_e Q_e(\vec{a}_e) = \sum_{e,\vec{a}_e} w_{e,\vec{a}_e} h_{e,\vec{a}_e}(\vec{a})$, where the $h_{e,\vec{a}_e}$ are the basis functions: $h_{e,\vec{a}_e}(\vec{a}) = 1$ iff $\vec{a}$ specifies $\vec{a}_e$ for component $e$ (and 0 otherwise). As such, providing guarantees with respect to the optimal $Q(\vec{a})$-value would

[2]Proofs can be found in Appendix B.

require developing a priori bounds for the approximation quality of (a particular type of) basis functions. This is a very difficult problem for which there is no good solution, even though these methods are widely studied in machine learning.

However, we do not expect our methods to perform well on arbitrary $Q$. Instead, we expect them to perform well when $Q$ is nearly factored, such that (1) approximately holds, since then the local actions contain enough information to make good predictions. As such, we analyze the behavior of our methods when the samples of $Q$ come from a factored function (i.e., as in (1) ) contaminated with zero-mean noise. In such cases, we can show the following.

**Theorem 1.** *The estimate $\hat{Q}$ of $Q$ made by a mixture of experts converges in probability to the true value plus a sample policy dependent bias term: $\hat{Q}(\vec{a}) \xrightarrow{p} Q(\vec{a}) + B_{\vec{\pi}}(\vec{a})$. The bias is given by a sum of biases induced by pairs $e$,$e'$:*

$$B_{\vec{\pi}}(\vec{a}) \triangleq \sum_{e} \sum_{e' \neq e} \sum_{\overline{\vec{a}_{e' \setminus e}}} \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}).$$

*Here, $\vec{a}_{e' \cap e}$ is the action of the agents that participate both in $e$ and $e'$ (specified by $\vec{a}$) and $\overline{\vec{a}_{e' \setminus e}}$ are the actions of agents in $e'$ that are not in $e$.*

Because we observe the global reward for a given set of actions, the bias is caused by correlations in the sampling policy and the fact that we are overcounting value from other components. When there is no overlap, and the sampling policy we use is 'component-wise': $\vec{\pi}(\vec{a}_{e' \setminus e} | \vec{a}_e) = \vec{\pi}(\vec{a}_{e' \setminus e} | \vec{a}'_e) = \vec{\pi}(\vec{a}_{e' \setminus e})$, this over counting is the same for all local actions $\vec{a}_e$:

**Theorem 2.** *When value components do not overlap and a component-wise sampling policy is used, mixture of experts optimization recovers the maximizing joint action.*

Similar reasoning can be used to establish bounds on the performance in the case of overlapping components, subject to assumptions about properties of the true value function. Let $\mathcal{N}(e)$ denote the neighborhood of component $e$: the set of other components $e'$ which have an overlap with $e$ (those that have at least one agent participating in them that also participates in $e$).

**Theorem 3.** *If for all overlapping components $e$,$e'$, and any two 'intersection action profiles' $\vec{a}_{e' \cap e}$,$\vec{a}'_{e' \cap e}$ for their intersection, the true value function satisfies*

$$\forall \vec{a}_{e' \setminus e} \qquad Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}_{e' \cap e}) - Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}'_{e' \cap e}) \leq \frac{\epsilon}{|E| \cdot |\mathcal{N}(e)| \cdot |\vec{\mathcal{A}}_{e' \setminus e}| \cdot \vec{\pi}(\vec{a}_{e' \setminus e})}, \tag{2}$$

*with $|\vec{\mathcal{A}}_{e' \setminus e}|$ the number of intersection action profiles, then mixture of experts optimization, in the limit, will return a joint action whose value lies within $\epsilon$ of the optimal solution.*

The analysis shows that a sufficiently local Q-function can be effectively optimized when using a sufficiently local sampling policy. Under the same assumptions, we can also derive guarantees for the sequential case. It is not directly possible to derive bounds for FV-POMCP itself (since it is not possible to demonstrate that the UCB exploration policy is component-wise), but it seems likely that UCB exploration leads to an effective policy that nearly satisfies this property. Moreover, since bias is introduced by the interaction between action correlations and differences in 'non-local' components, even when using a policy with correlations, the bias may be limited if the Q-function is sufficiently structured.

In the factored tree case, we can introduce a strong result. Because histories for other agents outside the factor are not included and we do not assume independence between factors, the approximation quality may suffer: where $\vec{h}$ is Markov, this is not the case for the local history $\vec{h}_e$. As such, the expected return for such a local history depends on the future policy as well as the past one (via the distribution over histories of agents not included in $e$). This implies that convergence is no longer guaranteed:

**Proposition 1.** *Factored-Trees FV-POMCP may diverge.*

*Proof.* FT-FV-POMCP (with $c = 0$) corresponds to a general case of Monte Carlo control (i.e., SARSA(1)) with linear function approximation that is greedy w.r.t. the current value function. Such settings may result in divergence (Fairbank and Alonso, 2012). □

Even though this is a negative result, and there is no guarantee of convergence for FT-FV-POMCP, in practice this need not be a problem; many reinforcement learning techniques that can diverge (e.g., neural networks) can produce high-quality results in practice, e.g., (Tesauro, 1995; Stone and Sutton, 2001). Therefore, we expect that if the problem exhibits enough locality, the factored trees approximation may allow good quality policies to be found very quickly.

Finally, we analyze the computational complexity. FV-POMCP is implemented by modifying POMCP's SIMULATE function (as described in Appendix A). The maximization is performed by variable elimination, which has complexity $O(n|\mathcal{A}_{max}|^w)$ with $w$ the induced width and $|\mathcal{A}_{max}|$ the size of the largest action set. In addition, the algorithm updates each of the $|E|$ components, bringing the total complexity of one call of simulate to $O(|E| + n|\mathcal{A}_{max}|^w)$.

# 6 Experimental Results

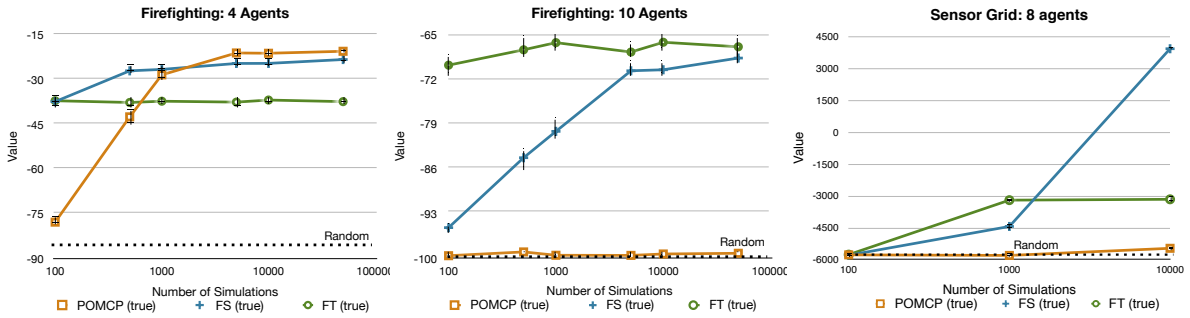
Here, we empirically investigate the effectiveness of our factorization methods by comparing them to non-factored methods in the planning and learning settings.

**Experimental Setup.** We test our methods on versions of the firefighting problem from Section 4 and on sensor network problems. In the firefighting problems, fires are suppressed more quickly if a larger number of agents choose that particular house. Fires also spread to neighbor's houses and can start at any house with a small probability. In the sensor network problems (as shown by Fig. 1(c)), sensors were aligned along discrete intervals on two axes with rewards for tracking a target that moves in a grid. Two types of sensing could be employed by each agent (one more powerful than the other, but using more energy) or the agent could do nothing. A higher reward was given for two agents correctly sensing a target at the same time. The firefighting problems were broken up into $n - 1$ overlapping factors with 2 agents in each (representing the agents on the two sides of a house) and the sensor grid problems were broken into $n/2$ factors with $n/2 + 1$ agents in each (representing all agents along the y axis and one agent along the x axis). For the firefighting problem with 4 agents, $|S|$ = 243, $|A|$ = 81 and $|Z|$ = 16 and with 10 agents, $|S|$ = 177147, $|A|$ = 59049 and $|Z|$ = 1024. For the sensor network problems with 4 agents, $|S|$ = 4, $|A|$ = 81 and $|Z|$ = 16 and with 8 agents, $|S|$ = 16, $|A|$ = 6561 and $|Z|$ = 256.

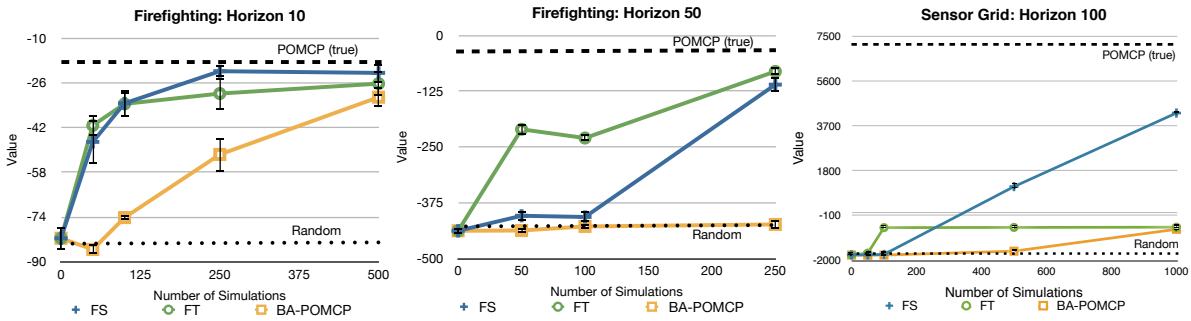
Each experiment was run for a given number of *simulations*, the number of samples used at each step to choose an action, and averaged over a number of *episodes*. We report undiscounted return with the standard error. Experiments were run on a single core of a 2.5 GHz machine with 8GB of memory. In both cases, we compare our factored representations to the flat version using POMCP. This comparison uses the same code base so it directly shows the difference when using factorization. POMCP and similar sample-based planning methods have already been shown to be state-of-the-art methods in both POMDP planning (Silver and Veness, 2010) and learning (Ross et al., 2011).

**MPOMDPs.** We start by comparing the factored statistics (FS) and factored tree (FT) versions of FV-POMCP in multiagent planning problems. Here, the agents are given the true MPOMDP model (in the form of a simulator) and use it to plan. For this setting, we compare to two baseline methods: *POMCP*: regular POMCP applied to the MPOMDP, and *random*: uniform random action selection. Note that while POMCP will converge to an $\epsilon$-optional solution, the solution quality may be poor when using small number of simulations.

The results for 4-agent and 10-agent firefighting problems with horizon 10 are shown in Figure 4(a). For the 4-agent problem, POMCP performs poorly with a few simulations, but as the number of simulations increases it outperforms the other methods (presumably converging to an optimal solution). FT provides a high-quality solution with a very small number of simulations, but the resulting value plateaus due to approximation error. FS also provides a high-quality solution with a very small number of simulations, but is then able to converge to a solution that is near POMCP. In the 10-agent problem, POMCP is only able to generate a solution that is slightly better than random while the FV-POMCP methods are able to perform much better. In fact, FT performs




(a) MPOMDP results




(b) BA-MPOMDP results

Figure 4: Results for (a) the planning (MPOMDP) case (log scale x-axis) and (b) the learning (BA-MPOMDP) case for the firefighting and sensor grid problems.

very well with a small number of samples and FS continues to improve until it reaches solution that is similar to FT.

Similar results are seen in the sensor grid problem. POMCP outperforms a random policy as the number of simulations grows, but FS and FT produce much higher values with the available simulations. FT seems to converge to a low quality solution (in both planning and learning) due to the loss of information about the target's previous position that is no longer known to local factors. In this problem, POMCP requires over 10 minutes for an episode of 10000 simulations, making reducing the number of simulations crucial in problems of this size. These results clearly illustrate the benefit of FV-POMCP by exploiting structure for planning in MASs.

**BA-MPOMDPs.** We also investigate the learning setting (i.e., when the agents are only given the BA-POMDP model). Here, at the end of each episode, both the state and count vectors are reset to their initial values. Learning in partially observable environments is extremely hard, and there may be many equivalence classes of transition and observation models that are indistinguishable when learning. Therefore, we assume a reasonably good model of the transitions (e.g., because the designer may have a good idea of the dynamics), but only a poor estimate of the observation model (because the sensors may be harder to model).

For the BRL setting, we compare to the following baseline methods: *POMCP*: regular POMCP applied to the true model using 100,000 simulations (this is the best proxy for, and we expect this to be very close to, the optimal value), and *BA-POMCP*: regular POMCP applied to the BA-POMDP.

Results for a four agent instance of the fire fighting problem are shown in Fig. 4(b), for $h = 10, 50$. In both cases, the FS and FT variants approach the POMCP value. For a small number of simulations FT learns very quickly, providing significantly better values than the flat methods and better than FS for the increased horizon. FS learns more slowly, but the value is better as the number of simulations increases (as seen in the horizon 10 case) due to the use of the full history. After more simulations in the horizon 10 problem, the performance of the flat model (BA-MPOMDP) improves, but the factored methods still outperform it and this increase is less

visible for the longer horizon problem.

Similar results are again seen in the four agent sensor grid problem. FT performs the best with a small number of simulations, but as the number increases, FS outperforms other methods. Again, for these problems, BA-POMCP requires over 10 minutes for each episode for the largest number of simulations, showing the need for more efficient methods. These experiments show that even in challenging multiagent settings with state uncertainty, BRL methods can learn by effectively exploiting structure.

# 7 Related Work

MCTS methods have become very popular in games, a type of multiagent setting, but no action factorization has been exploited so far (Browne et al., 2012). Progressive widening (Coulom, 2007) and double progressive widening (Couëtoux et al., 2011) have had some success in games with large (or continuous) action spaces. The progressive widening methods do not use the structure of the coordination graph in order to generalize value over actions, but instead must find the correct joint action out of the exponentially many that are available (which may require many trajectories). They are also designed for fully observable scenarios, so they do not address the large observation space in MPOMDPs.

The factorization of the history in FTs is not unlike the use of linear function approximation for the state components in TD-Search (Silver, Sutton, and Müller, 2012). However, in contrast to that method, due to our particular factorization, we can still apply UCT to aggressively search down the most promising branches of the tree. While other methods based on Q-learning (Guestrin, Lagoudakis, and Parr, 2002; Kok and Vlassis, 2006) exploit action factorization, they assume agents observe individual rewards (rather than the global reward that we consider) and it is not clear how these could be incorporated in a UCT-style algorithm.

Locality of interaction has also been considered previously in decentralized POMDP methods (Oliehoek, 2012; Amato et al., 2013) in the form of factored Dec-POMDPs (Oliehoek, Whiteson, and Spaan, 2013; Pajarinen and Peltonen, 2011) and networked distributed POMDPs (ND-POMDPs) (Nair et al., 2005; Kumar and Zilberstein, 2009; Dibangoye et al., 2014). These models make strict assumptions about the information that the agents can use to choose actions (only the past history of individual actions and observations), thereby significantly lowering the resulting value (Oliehoek, Spaan, and Vlassis, 2008). ND-POMDPs also impose additional assumptions on the model (transition and observation independence and a factored reward function). The MPOMDP model, in contrast, does not impose these restrictions. Instead, in MPOMDPs, each agent knows the *joint* action-observation history, so there are not different perspectives by different agents. Therefore, 1) factored Dec-POMDP and ND-POMDP methods do not apply to MPOMDPs; they specify mappings from individual histories to actions (rather than joint histories to joint actions), 2) ND-POMDP methods assume that the value function is *exactly* factored as the sum of local values ('perfect locality of interaction') while in an MPOMDP, the value is only *approximately* factored (since different components can correlate due to conditioning the actions on central information). While perfect locality of interaction allows a natural factorization of the MPOMDP value function, but our method can be applied to any MPOMDP (i.e., given any factorization of the value function). Furthermore, current factored Dec-POMDP and ND-POMDP models generate solutions given the model in an offline fashion, while we consider online methods using a simulator in this paper.

Our approach builds upon coordination-graphs (Guestrin, Koller, and Parr, 2001), to perform the joint action optimization efficiently, but factorization in one-shot problems has been considered in other settings too. Amin et al. (Amin, Kearns, and Syed, 2011) present a method to optimize graphical bandits, which relates to our optimization approach. Since their approach replaced the UCB functionality, it is not obvious how their approach could be integrated in POMCP. Moreover, their work, focuses on minimizing regret (which is not an issue in our case), and does not apply when the factorization does not hold. Oliehoek et al. (Oliehoek, Whiteson, and Spaan, 2012) present an factored-payoff approach that extends coordination graphs to imperfect information settings where each agent has its own knowledge. This is not relevant for our current algorithm, which assumes that joint observations will be received by a centralized decision maker, but could potentially be useful to relax this assumption.

# 8 Conclusions

We presented the first method to exploit multiagent structure to produce a scalable method for Monte Carlo tree search for POMDPs. This approach formalizes a team of agents as a multiagent POMDP, allowing planning and BRL techniques from the POMDP literature to be applied. However, since the number of joint actions and observations grows exponentially with the number of agents, naïve extensions of single agent methods will not scale well. To combat this problem, we introduced FV-POMCP, an online planner based on POMCP (Silver and Veness, 2010) that exploits multiagent structure using two novel techniques—factored statistics and factored trees— to reduce 1) the number of joint actions and 2) the number of joint histories considered. Our empirical results demonstrate that FV-POMCP greatly increases scalability of online planning for MPOMDPs, solving problems with 10 agents. Further investigation also shows scalability to the much more complex learning problem with four agents. Our methods could also be used to solve POMDPs and BA-POMDPs with large action and observation spaces as well the recent Bayes-Adaptive extension (Ng et al., 2012) of the self interested I-POMDP model (Gmytrasiewicz and Doshi, 2005).

# Acknowledgments

F.O. is funded by NWO Innovational Research Incentives Scheme Veni #639.021.336. C.A was supported by AFOSR MURI project #FA9550-09-1-0538 and ONR MURI project #N000141110688.

# References

Amato, C., and Oliehoek, F. A. 2013. Bayesian reinforcement learning for multiagent systems with state uncertainty. In *Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 76–83.

Amato, C.; Chowdhary, G.; Geramifard, A.; Ure, N. K.; and Kochenderfer, M. J. 2013. Decentralized control of partially observable Markov decision processes. In *CDC*, 2398–2405.

Amin, K.; Kearns, M.; and Syed, U. 2011. Graphical models for bandit problems. In *UAI*, 1–10.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games* 4(1):1–43.

Couëtoux, A.; Hoock, J.-B.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011. Continuous upper confidence trees. In *Learning and Intelligent Optimization*. 433–445.

Coulom, R. 2007. Computing elo ratings of move patterns in the game of go. *International Computer Games Association (ICGA) Journal* 30(4):198–208.

Dibangoye, J. S.; Amato, C.; Buffet, O.; and Charpillet, F. 2014. Exploiting separability in multi-agent planning with continuous-state MDPs. In *AAMAS*.

Fairbank, M., and Alonso, E. 2012. The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *International Joint Conference on Neural Networks*, 1–8. IEEE.

Gmytrasiewicz, P. J., and Doshi, P. 2005. A framework for sequential planning in multi-agent settings. *JAIR* 24.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Multiagent planning with factored MDPs. In *NIPS*, 15.

Guestrin, C.; Lagoudakis, M.; and Parr, R. 2002. Coordinated reinforcement learning. In *ICML*, 227–234.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *MLJ* 49(2-3).

Kok, J. R., and Vlassis, N. 2006. Collaborative multiagent reinforcement learning by payoff propagation. *JMLR* 7.

Kumar, A., and Zilberstein, S. 2009. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *AAMAS*.

Messias, J. V.; Spaan, M.; and Lima, P. U. 2011. Efficient offline communication policies for factored multiagent POMDPs. In *NIPS 24*.

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs. In *AAAI*.

Ng, B.; Boakye, K.; Meyers, C.; and Wang, A. 2012. Bayes-adaptive interactive POMDPs. In *AAAI*.

Oliehoek, F. A.; Spaan, M. T. J.; Whiteson, S.; and Vlassis, N. 2008. Exploiting locality of interaction in factored Dec-POMDPs. In *AAMAS*.

Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. 2008. Optimal and approximate Q-value functions for decentralized POMDPs. *JAIR* 32:289–353.

Oliehoek, F. A.; Whiteson, S.; and Spaan, M. T. J. 2012. Exploiting structure in cooperative Bayesian games. In *UAI*, 654–664.

Oliehoek, F. A.; Whiteson, S.; and Spaan, M. T. J. 2013. Approximate solutions for factored Dec-POMDPs with many agents. In *AAMAS*, 563–570.

Oliehoek, F. A. 2012. Decentralized POMDPs. In Wiering, M., and van Otterlo, M., eds., *Reinforcement Learning: State of the Art*. Springer.

Pajarinen, J., and Peltonen, J. 2011. Efficient planning for factored infinite-horizon DEC-POMDPs. In *IJCAI*, 325–331.

Pynadath, D. V., and Tambe, M. 2002. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR* 16.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for POMDPs. *JAIR* 32(1).

Ross, S.; Pineau, J.; Chaib-draa, B.; and Kreitmann, P. 2011. A Bayesian approach for learning and planning in partially observable Markov decision processes. *JAIR* 12.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large POMDPs. In *NIPS 23*.

Silver, D.; Sutton, R. S.; and Müller, M. 2012. Temporal-difference search in computer Go. *MLJ* 87(2):183–219.

Stone, P., and Sutton, R. S. 2001. Scaling reinforcement learning toward RoboCup soccer. In *ICML*, 537–544.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38(3):58–68.

# A FV-POMCP Pseudo code

Here we describe in more detail the algorithms for the proposed FV-POMCP variants. Both methods can be described as a modification of POMCP's SIMULATE procedure, which is shown in Algorithm 1. Each simulation is started by calling this procedure on the root node (corresponding to the empty history, or 'now') with a state sampled from the current belief. The comments in Algorithm 1 should make the code self-explanatory, but for a further explanation we refer to (Silver and Veness, 2010).

**Algorithm 1** POMCP

**procedure** SIMULATE($s$,$h$,$depth$)
  **if** $\gamma^{depth} < \epsilon$ **then**
    **return** $0$ ▷ Stop when desired precision reached
  **end if**
  **if** $h \notin T$ **then** ▷ if we did not visit this $h$ yet
    **for all** $a \in A$ **do**
      $T(h,a) \leftarrow (N_{init}(h,a),V_{init}(h,a),\emptyset)$ ▷ initialize counts for all $a$, and particle filter
    **end for**
    **return** Rollout($s$,$h$,$depth$) ▷ do a random rollout from this node
  **end if**
  $a \leftarrow \arg\max'_a \, Q(h,a') + c\sqrt{\frac{logN(h)}{N(h,a')}}$ ▷ max via enumeration
  $(s',o,r) \sim \mathcal{G}(s,a)$ ▷ sample a transition, observation and reward
  $R \leftarrow r + \gamma$Simulate($s'$,$(hao)$,$depth+1$) ▷ Recurse and receive the return $R$
  $B(h) \leftarrow B(h) \cup \{s\}$ ▷ Add $s$ to the particle filter maintained for $h$
  $N(h) \leftarrow N(h) + 1$ ▷ Update the number of times $h$ is visited...
  $N(h,a) \leftarrow N(h,a) + 1$ ▷ ... and how often we selected action $a$ here
  $Q(h,a) \leftarrow Q(h,a) + \frac{R - Q(h,a)}{N(h,a)}$ ▷ Incremental update of mean return
  **return** $R$
**end procedure**

**Algorithm 2** Factored Statistics

**procedure** SIMULATE($s$,$\vec{h}$,$depth$)
  **if** $\gamma^{depth} < \epsilon$ **then**
    **return** $0$
  **end if**
  **if** $\vec{h} \notin T$ **then**
    **for all** $e \in E$ **do** ▷ initialize statistics for component $e$
      **for all** $\vec{a}_e \in \vec{\mathcal{A}}_e$ **do** ▷ only loop over *local* joint actions
        $T(\vec{h},\vec{a}_e) \leftarrow (N_{init}(\vec{h},\vec{a}_e),V_{init}(\vec{h},\vec{a}_e),\emptyset)$
      **end for**
    **end for**
    **return** Rollout($s$,$\vec{h}$,$depth$)
  **end if**
  $\vec{a} \leftarrow \arg\max'_{\vec{a}} \sum_e \left[ Q_e(\vec{h},\vec{a}'_e) + c\sqrt{\frac{logN(\vec{h}+1)}{n_{\vec{a}'_e}}} \right]$ ▷ via variable elimination
  $(s',\vec{o},r) \sim \mathcal{G}(s,\vec{a})$
  $R \leftarrow r + \gamma$Simulate($s'$,$(\vec{h}\vec{a}\vec{o})$,$depth+1$)
  $B(\vec{h}) \leftarrow B(\vec{h}) \cup \{s\}$
  $N(\vec{h}) \leftarrow N(\vec{h}) + 1$
  **for all** $\vec{e} \in E$ **do** ▷ update the statistics for each component
    $n_{\vec{a}_e} \leftarrow n_{\vec{a}_e} + 1$
    $Q_e(\vec{h},\vec{a}_e) \leftarrow Q_e(\vec{h},\vec{a}_e) + \frac{R - Q_e(\vec{h},\vec{a}_e)}{n_{\vec{a}_e}}$ ▷ update the estimation of the expert for $\vec{a}_e$
  **end for**
  **return** $R$
**end procedure**

The pseudo code for the factored statistics case (FS-FV-POMCP) is shown in Algorithm 2. The comments highlight the important changes: there is no need to loop over the joint actions for initialization, or for selecting the maximizing action. Also, the same return $R$ is used to update the active expert in each component $e$.

Finally, Algorithm 3 shows the pseudo code for the factored trees variant of FV-POMCP. Like FSs, FT-FV-POMCP performs the simulations in lockstep. However, as we now maintain statistics and particle filters for each component $e$ in separate trees, the initialization and updating of these statistics is slightly different. As such, the algorithm makes clear that there is no computational advantage to factored trees (when compared to FSs), but that the big difference is in the additional generalization it performs.

The actual planning could take place in a number of ways: one agent could be designated the planner, which would require this agent to broadcast the computed joint action. Alternatively, each agent can in parallel perform an identical planning process (by, in the case of randomized planning, syncing the random number generators). Then each agent will compute the same joint action and execute its component. An interesting direction of future work is whether the planning itself can be done more effectively by distributing the task over the agents.

**Algorithm 3** Factored Trees

**procedure** SIMULATE($s$,$\vec{h}$,$depth$)
  **if** $\gamma^{depth} < \epsilon$ **then**
    **return** $0$
  **end if**
  **for all** $e \in E$ **do** ▷ check all components $e$
    **if** $\vec{h}_e \notin T_e$ **then** ▷ if there is no node for $\vec{h}_e$ in the tree for component $e$ yet
      **for all** $\vec{a}_e \in \vec{\mathcal{A}}_e$ **do** ▷ only loop over *local* joint actions
        $T_e(\vec{h}_e, \vec{a}_e) \leftarrow (N_{init}(\vec{h}_e, \vec{a}_e), V_{init}(\vec{h}_e, \vec{a}_e), \emptyset)$
      **end for**
    **end if**
  **end for**
  **return** Rollout($s$,$\vec{h}$,$depth$)
  $\vec{a} \leftarrow \arg\max'_{\vec{a}} \sum_e \left[ Q_e(\vec{h}_e, \vec{a}'_e) + c\sqrt{\frac{logN(\vec{h}_e+1)}{n_{\vec{a}'_e}}} \right]$ ▷ via variable elimination
  $(s', \vec{o}, r) \sim \mathcal{G}(s, \vec{a})$
  $R \leftarrow r + \gamma$Simulate($s'$,$(\vec{h}\vec{a}\vec{o})$,$depth+1$)
  **for all** $\vec{e} \in E$ **do** ▷ update the statisitics for each component
    $B(\vec{h}_e) \leftarrow B(\vec{h}_e) \cup \{s\}$ ▷ update the particle filter of each tree $e$ with the same $s$
    $N(\vec{h}_e) \leftarrow N(\vec{h}_e) + 1$
    $n_{\vec{a}_e} \leftarrow n_{\vec{a}_e} + 1$
    $Q_e(\vec{h}_e, \vec{a}_e) \leftarrow Q_e(\vec{h}_e, \vec{a}_e) + \frac{R - Q_e(\vec{h}_e, \vec{a}_e)}{n_{\vec{a}_e}}$ ▷ update expert for $\vec{h}_e, \vec{a}_e$
  **end for**
  **return** $R$
**end procedure**

# B Analysis of Mixture of Experts Optimization

Here we analyze the behavior of our mixtures of experts under sample policy $\vec{\pi}$. In performing this analysis we compare to the case where the true value function $Q(\vec{a})$ is factored in $E$ components

$$Q(\vec{a}) = \sum_{e=1}^{E} Q_e(\vec{a}_e)$$

and corrupted by zero-mean noise $\nu$. As such we establish the performance in cases where the actual value function is 'close' to factored. In the below, we will write $\mathcal{N}(e)$ for the neighborhood of component $e$. That

is the set of other components $e'$ which have an overlap with $e$: those that have at least one agent participating in them that also participates in $e$). In this analysis, we will assume that all experts are weighted uniformly ($\alpha_e = \frac{1}{E}$), and ignore this constant which is not relevant for determining the maximizing action.

**Theorem 4.** *The estimate $\hat{Q}$ of $Q$ made by a mixture of experts converges in probability to the true value plus a sample policy dependent bias term:*

$$\hat{Q}(\vec{a}) \xrightarrow{p} Q(\vec{a}) + B_{\vec{\pi}}(\vec{a}).$$

*The bias is given by a sum of biases induced by pairs $e$,$e'$ of overlapping value components:*

$$B_{\vec{\pi}}(\vec{a}) \triangleq \sum_e \sum_{e' \neq e} \sum_{\overline{\vec{a}_{e' \setminus e}}} \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}).$$

*Here $\vec{a}_{e' \cap e}$ is the action of the agents that participate both in $e$ and $e'$ (specified by $\vec{a}$) and $\overline{\vec{a}_{e' \setminus e}}$ are the actions of agents in $e'$ that are not in $e$ (these are summed over as emphasised by the overlining).*

*Proof.* Suppose that we have drawn a set of samples $r_1, \ldots, r_K$ according to $\vec{\pi}$. For each component $e$ and $\vec{a}_e$, we have an expert that estimates $\hat{Q}(\vec{a}_e)$. Let $\mathcal{R}(\vec{a}_e)$ be the subset of samples received where we took local joint action $\vec{a}_e$. The corresponding expert will estimate

$$\hat{Q}(\vec{a}_e) := \frac{1}{\mathcal{R}(\vec{a}_e)} \sum_{r_{\vec{a}_e} \in \mathcal{R}(\vec{a}_e)} r_{\vec{a}_e}$$

Now, the expected sample that this expert receives is

$$\begin{aligned}
\mathbf{E}\left[r_{\vec{a}_e} | \vec{\pi}\right] &= \mathbf{E}_\nu \left\{ \sum_{\vec{a}_{-e}} \vec{\pi}(\vec{a}_{-e} | \vec{a}_e) Q(\vec{a}) + \nu \right\} \\
&= \sum_{\vec{a}_{-e}} \vec{\pi}(\vec{a}_{-e} | \vec{a}_e) \left[ \sum_{e'} Q_{e'}(\vec{a}_{e'}) \right] + \mathbf{E}_\nu \nu \\
&= Q_e(\vec{a}_e) + \sum_{\vec{a}_{-e}} \vec{\pi}(\vec{a}_{-e} | \vec{a}_e) \sum_{e' \neq e} Q_{e'}(\vec{a}_{e'}) + 0 \\
&= Q_e(\vec{a}_e) + \sum_{e' \neq e} \sum_{\vec{a}_{e' \setminus e}} \vec{\pi}(\vec{a}_{e' \setminus e} | \vec{a}_e) Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}_{e' \cap e})
\end{aligned}$$

which means that the estimate of an expert converges in probability to a biased estimate

$$\hat{Q}(\vec{a}_e) \xrightarrow{p} Q_e(\vec{a}_e) + \sum_{e' \neq e} \sum_{\vec{a}_{e' \setminus e}} \vec{\pi}(\vec{a}_{e' \setminus e} | \vec{a}_e) Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}_{e' \cap e}).$$

This, in turn means that the mixture of experts

$$\begin{aligned}
\hat{Q}(\vec{a}) = \sum_e \hat{Q}_e(\vec{a}_e) &\xrightarrow{p} \sum_e \left[ Q_e(\vec{a}_e) + \sum_{e' \neq e} \sum_{\overline{\vec{a}_{e' \setminus e}}} \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) \right] \\
&= \sum_e Q_e(\vec{a}_e) + \sum_e \sum_{e' \neq e} \sum_{\overline{\vec{a}_{e' \setminus e}}} \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) \\
&= \sum_e Q_e(\vec{a}_e) + B_{\vec{\pi}}(\vec{a}),
\end{aligned}$$

as claimed. □

As is clear from the definition of the bias term, it is caused by correlations in the sample policy and the fact that we are over counting value from other components. When there is no overlap in the payoff components, and we use a sample policy we use is '*component-wise*', i.e., $\vec{\pi}(\overline{\vec{a}_{e'\setminus e}}|\vec{a}_e) = \vec{\pi}(\overline{\vec{a}_{e'\setminus e}}|\vec{a}'_e) = \vec{\pi}(\overline{\vec{a}_{e'\setminus e}})$, the effect of this bias can be disregarded: in such a case, even though all components $e' \neq e$ contribute to the bias of expert $e$ this bias is constant for those components $e' \notin \mathcal{N}(e)$. Since we care only about the relative values of joint actions $\vec{a}$, only overlapping components actually contribute to introducing error. This is clearly illustrated for the case where there are no overlapping components.

**Theorem 5.** *In the case that the value components do not overlap, mixture of experts optimization recovers the maximizing joint action.*

*Proof.* In this case,

$$\hat{Q}_e(\vec{a}_e) \xrightarrow{p} Q_e(\vec{a}_e) + \sum_{e' \neq e} \sum_{\vec{a}_{e'}} \vec{\pi}_{e'}(\vec{a}_{e'}) Q_{e'}(\vec{a}_{e'})$$

and the bias does not depend on $\vec{a}_e$, such that

$$\arg\max_{\vec{a}_e} \hat{Q}_e(\vec{a}_e) \xrightarrow{p} \arg\max_{\vec{a}_e} Q_e(\vec{a}_e) + C = \arg\max_{\vec{a}_e} Q_e(\vec{a}_e).$$

And for the joint optimization:

$$\begin{aligned}
\hat{Q}(\vec{a}) = \sum_e \hat{Q}_e(\vec{a}_e) &\xrightarrow{p} \sum_e \left[ Q_e(\vec{a}_e) + \sum_{e' \neq e} \sum_{\vec{a}_{e'}} \vec{\pi}_{e'}(\vec{a}_{e'}) Q_{e'}(\vec{a}_{e'}) \right] \\
&= \sum_e Q_e(\vec{a}_e) + \sum_e \sum_{e' \neq e} \sum_{\vec{a}_{e'}} \vec{\pi}_{e'}(\vec{a}_{e'}) Q_{e'}(\vec{a}_{e'}) \\
&= \sum_e Q_e(\vec{a}_e) + (E-1) \sum_e \sum_{\vec{a}'_e} \vec{\pi}_e(\vec{a}'_e) Q_e(\vec{a}'_e) \\
&= \sum_e \left( Q_e(\vec{a}_e) + (E-1) SA(e,\vec{\pi}) \right)
\end{aligned}$$

where $SA(e,\vec{\pi}) = \sum_{\vec{a}'_e} \vec{\pi}_e(\vec{a}'_e) Q_e(\vec{a}'_e)$ is the sampled average payoff of component $e$ which affects our value estimates, but which does not affect the maximizing joint action:

$$\arg\max_{\vec{a}} \left[ \sum_e \left( Q_e(\vec{a}_e) + (E-1) SA(e,\vec{\pi}) \right) \right] = \arg\max_{\vec{a}} \sum_e Q_e(\vec{a}_e)$$

□

Similar reasoning can be used to establish bounds on the performance of mixture of expert optimization in cases with overlap, as is shown by the next theorem.

**Theorem 6.** *If for all overlapping components $e$,$e'$, and any two 'intersection action profiles' $\vec{a}_{e' \cap e}$,$\vec{a}'_{e' \cap e}$ for their intersection, the true value function satisfies that*

$$\forall \vec{a}_{e' \setminus e} \qquad Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}_{e' \cap e}) - Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}'_{e' \cap e}) \leq \frac{\epsilon}{E \cdot |\mathcal{N}(e)| \cdot \left| \vec{\mathcal{A}}_{e' \setminus e} \right| \cdot \vec{\pi}(\vec{a}_{e' \setminus e})},$$

*with $\left| \vec{\mathcal{A}}_{e' \setminus e} \right|$ the number of intersection action profiles, then mixture of experts optimization, in the limit will return a joint action whose value solution that lies within $\epsilon$ of the optimal solution.*

*Proof.* Bias by itself is no problem, but different bias for different joint actions is, because that may cause us to select the wrong action. As such, we set out to bound

$$\forall_{\vec{a},\vec{a}'} \qquad |B_{\vec{\pi}}(\vec{a}) - B_{\vec{\pi}}(\vec{a}')| \leq \epsilon.$$

As explained, only terms where the bias is different for two actions $\vec{a}$,$\vec{a}'$ matter. As such we will omit terms that cancel out. In particular, we have that if two components $e$,$e'$ do not overlap, the expression

$$\sum_{\overline{\vec{a}_{e'\setminus e}}} \vec{\pi}(\overline{\vec{a}_{e'\setminus e}}|\vec{a}_e)Q_{e'}(\overline{\vec{a}_{e'\setminus e}},\vec{a}_{e'\cap e}) - \sum_{\overline{\vec{a}_{e'\setminus e}}} \vec{\pi}(\overline{\vec{a}_{e'\setminus e}}|\vec{a}'_e)Q_{e'}(\overline{\vec{a}_{e'\setminus e}},\vec{a}'_{e'\cap e})$$

reduces to $\sum_{\overline{\vec{a}_{e'}}} \vec{\pi}(\overline{\vec{a}_{e'}}|\vec{a}_e)Q_{e'}(\overline{\vec{a}_{e'}}) - \sum_{\overline{\vec{a}_{e'}}} \vec{\pi}(\overline{\vec{a}_{e'}}|\vec{a}'_e)Q_{e'}(\overline{\vec{a}_{e'}})$ and hence vanishes under a componentwise policy. We use this insight to define the bias in terms of neighborhood bias. Let us write $\mathcal{N}(e)$ for the set of edges $\{e'\}$ that have overlap with $e$, and let's write $\vec{a}_{\mathcal{N}(e)}$ for the joint action that specifies actions for that entire overlap, then we can write this as:

$$B_{\vec{\pi}}(\vec{a}) \triangleq \sum_{e=1}^{E} B^e_{\vec{\pi}}(\vec{a}_{\mathcal{N}(e)})$$

with

$$B^e_{\vec{\pi}}(\vec{a}_{\mathcal{N}(e)}) = \sum_{e'\in\mathcal{N}(e)} B^{e'\cap e}_{\vec{\pi}}(\vec{a}_{e'\cap e})$$

component $e$'s 'neighborhood bias', with

$$B^{e'\to e}_{\vec{\pi}}(\vec{a}_e) \triangleq \sum_{\overline{\vec{a}_{e'\setminus e}}} \vec{\pi}(\overline{\vec{a}_{e'\setminus e}}|\vec{a}_e)Q_{e'}(\overline{\vec{a}_{e'\setminus e}},\vec{a}_{e'\cap e})$$

the 'intersection bias': the bias introduced on $e$ via the overlap between $e'$ and $e$.

Now, we show how we can guarantee that the bias is small, by guaranteeing that the intersection biases are small. This gives a conservative bound, since different biases may very well cancel out. Artbitrarily select two actions, W.l.o.g. we select $\vec{a}$ to be the larger-biased joint action. We need to guarantee that

$$\begin{aligned}
& B_{\vec{\pi}}(\vec{a}) - B_{\vec{\pi}}(\vec{a}') \leq \epsilon \\
\leftrightarrow \quad & \sum_{e=1}^{E}\sum_{e'\in\mathcal{N}(e)} B^{e'\to e}_{\vec{\pi}}(\vec{a}_e) - \sum_{e=1}^{E}\sum_{e'\in\mathcal{N}(e)} B^{e'\to e}_{\vec{\pi}}(\vec{a}'_e) \leq \epsilon \\
\leftrightarrow \quad & \sum_{e=1}^{E}\left[\sum_{e'\in\mathcal{N}(e)} B^{e'\to e}_{\vec{\pi}}(\vec{a}_e) - \sum_{e'\in\mathcal{N}(e)} B^{e'\to e}_{\vec{\pi}}(\vec{a}'_e)\right] \leq \epsilon \\
\leftarrow \quad & \sum_{e'\in\mathcal{N}(e)} B^{e'\to e}_{\vec{\pi}}(\vec{a}_e) - \sum_{e'\in\mathcal{N}(e)} B^{e'\to e}_{\vec{\pi}}(\vec{a}'_e) \leq \frac{\epsilon}{E} \quad \forall e \\
\leftrightarrow \quad & \sum_{e'\in\mathcal{N}(e)} \left[B^{e'\to e}_{\vec{\pi}}(\vec{a}_e) - B^{e'\to e}_{\vec{\pi}}(\vec{a}'_e)\right] \leq \frac{\epsilon}{E} \quad \forall e \\
\leftarrow \quad & \left[B^{e'\to e}_{\vec{\pi}}(\vec{a}_e) - B^{e'\to e}_{\vec{\pi}}(\vec{a}'_e)\right] \leq \frac{\epsilon}{E\times|\mathcal{N}(e)|} \quad \forall e,e'
\end{aligned}$$

let $\epsilon' \triangleq \frac{\epsilon}{E \times |\mathcal{N}(e)|}$, we want

$$\begin{aligned}
& \left[B_{\vec{\pi}}^{e' \to e}(\vec{a}_e) - B_{\vec{\pi}}^{e' \to e}(\vec{a}'_e)\right] \leq \epsilon' \\
& \sum_{\overline{\vec{a}_{e' \setminus e}}} \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) - \sum_{\overline{\vec{a}_{e' \setminus e}}} \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}'_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}'_{e' \cap e}) \leq \epsilon' \\
\leftrightarrow \quad & \sum_{\overline{\vec{a}_{e' \setminus e}}} \left[\vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) - \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}'_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}'_{e' \cap e})\right] \leq \epsilon' \\
\leftarrow \quad & \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) - \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}'_e) Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}'_{e' \cap e}) \leq \frac{\epsilon'}{\left|\vec{\mathcal{A}}_{e' \setminus e}\right|} \qquad \forall_{\overline{\vec{a}_{e' \setminus e}}}
\end{aligned}$$

Under a component-wise policy $\vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}_e) = \vec{\pi}(\overline{\vec{a}_{e' \setminus e}} | \vec{a}'_e) = \vec{\pi}(\overline{\vec{a}_{e' \setminus e}})$ and thus

$$\begin{aligned}
& \vec{\pi}(\overline{\vec{a}_{e' \setminus e}}) \left[Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) - Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}'_{e' \cap e})\right] \leq \frac{\epsilon'}{\left|\vec{\mathcal{A}}_{e' \setminus e}\right|} \qquad \forall_{\overline{\vec{a}_{e' \setminus e}}} \\
\leftrightarrow \quad & Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) - Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}'_{e' \cap e}) \leq \frac{\epsilon'}{\left|\vec{\mathcal{A}}_{e' \setminus e}\right| \vec{\pi}(\overline{\vec{a}_{e' \setminus e}})} \qquad \forall_{\overline{\vec{a}_{e' \setminus e}}}
\end{aligned}$$

Putting it all together, if the factorized Q function satisfies that, for all the component intersections, the following holds:

$$\forall \overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}, \vec{a}'_{e' \cap e} \qquad Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}_{e' \cap e}) - Q_{e'}(\overline{\vec{a}_{e' \setminus e}}, \vec{a}'_{e' \cap e}) \leq \frac{\epsilon}{E \left|\mathcal{N}(e)\right| \left|\vec{\mathcal{A}}_{e' \setminus e}\right| \vec{\pi}(\overline{\vec{a}_{e' \setminus e}})}$$

we are guaranteed to find an $\epsilon$-(absolute-error)-approximate solution. □