Exploiting bias for cooperative planning in multi-agent tree search

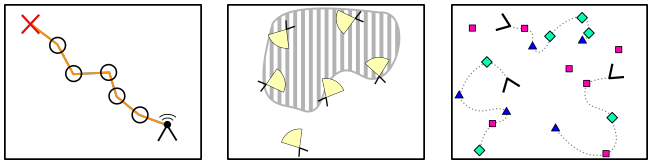
Aaron Ma Michael Ouimet Jorge Cortés

***Abstract*— Graph search over states and actions is a valuable tool for robotic planning and navigation. However, the required computation is sensitive to the size of the state and action spaces, a fact which is further exacerbated in multi-agent planning by the number of agents and the presence of sparse reward signals dependent on the cooperation of agents. To tackle these problems, we introduce an algorithm that is pre-trained in a centralized fashion but implemented on robots in a distributed way at runtime. The centralized portion uses imitation learning to iteratively construct policies that help guide an individual agent's own runtime search as well as predict other agents' future actions by exploiting previously discovered joint actions. Our algorithm includes a novel method of tree search based on a mixture of the individual and joint action space, which can be interpreted as a cascading effect where agents are biased by exploration of new actions, exploitation of previously profitable ones, and recommendation provided by deep neural nets. Simulations show the efficacy of the proposed method in cooperative scenarios with sparse rewards.**

I. Introduction

Multi-agent planning is critical across robotic applications in disaster relief scenarios, exploration, navigation, surveillance, and production. Handling these scenarios is difficult due to the large number of possible states and actions that agents can take. This complexity grows when meeting objectives requires coordination among the agents. To tackle this, our approach leverages model-based reinforcement learning to develop efficient algorithms which scale with the number of agents and incorporate the need to plan cooperatively. We rely on training deep neural networks to predict promising actions for the purpose of improving the tree search in the future. During runtime, agents use tree search in a distributed fashion, guided by the previously-trained policy to complete an objective under time constraints. Figure 1 shows illustrative examples of environments motivating our algorithm design.

Literature review: To structure the multi-agent planning algorithm, we draw motivation from coordination strategies in swarm robotics where the goal is to satisfy some global objective [1]–[6]. These algorithms utilize interaction and communication between neighboring agents to act cooperatively. Inspired by the particular case where homogeneous agents know each other's states but cannot communicate their intentions, we rely on model-based reinforcement learning. Monte-Carlo tree search (MCTS) can be used in reinforcement learning for decision making when a model is available to predict the next state of the agents and environment given the set of actions taken. This model may be learned or assumed given and can take the form of a Markov decision process. A variation on MCTS called upper confidence bound tree search (UCT) [7] chooses actions based on their upper confidence. UCT is heavily influential in model-based reinforcement learning as many variations of the tree search exist [8]. The process of training a policy to mimic the actions of another policy is called imitation learning [9]. The policy can also be trained to mimic the value of taking actions at a given state [10]. These techniques have been applied to tree searches in various ways. UCT has been used to generate data for a final policy constructed through imitation learning and deployed on Atari games [11]. Tree search actions are sometimes used to construct a *default policy* which helps simulate rollouts in UCT [12]. A variation on this is to train a policy to reflect values of certain actions at a given state in UCT, which are used instead of rollouts [13]. Recently policies have been trained to reflect the distribution of actions selected during a state in the tree search. These policies are then used to influence the selection of actions in future tree searches in both AlphaZero [14] and Expert Iteration (ExIt) [15]. This improves planning in large action spaces because agents focus on more promising choices based on previous experiences. In both AlphaZero and ExIt, agents play against a version of themselves to improve without human intervention.



(a) $\mathcal{E}_{\text{comm}}$ (b) $\mathcal{E}_{\text{cov}}$ (c) $\mathcal{E}_{\text{res}}$

Fig. 1. 2D multi-agent environments. In (a), agents must form a chain of communication from an operator to a point of interest. Agents are able to communicate if they are within a certain range of each other and agents are rewarded if a communication link between the target and operator exists. In (b), agents are able to detect in a cone-shaped region in front of them. The agents are tasked to jointly maximize their detection in an area of interest which is specified by an operator who is tasked to seek the target. In (c), static resources are scattered in the environment. Agents receive a reward for collecting a resource based on resource type, and some resources require two or more agents for collection.

In multi-agent scenarios, tree search can be used to choose actions in both a centralized and decentralized fashion. Deep learning can be utilized to create trained policies which can predict the actions of other agents and to aid an individual agent's own search [16]. Multi-agent Markov decision processes (MMDPs) [17] are used to describe Markov decision processes using joint actions from multiple agents. Another approach to multi-agent tree search is DEC-MCTS [18],

This work was supported by ONR Award N00014-16-1-2836.

A. Ma and J. Cortés are with the Department of Mechanical and Aerospace Engineering, University of California, San Diego. {aam021,cortes}@ucsd.edu. M. Ouimet is with the Naval Information Warfare Center Pacific. ouimet@spawar.navy.mil

where agents communicate compressed forms of their tree searches. In contrast, we are interested in the scenario where agents do not communicate their intentions during runtime. The work [19] combines components of linear temporal logic and hierarchical planning using MCTS with options learned from reinforcement learning, and demonstrates them on simulated autonomous driving. The work [20] uses tree search to create artificial cyclic policies which improve convergence in the multi-agent patrolling problem.

Statement of contributions: We provide two novel contributions that work together as a refinement of existing state-of-the-art model-based reinforcement learning techniques. The first contribution, *Multi-agent informed policy construction* (MIPC), is a process where we use deep imitation learning to build a heuristic offline, called *informed policy*, that guides the agents' tree search. Our strategy is to develop an informed policy for a small number of agents and use that informed policy to accelerate the tree search for environments with more agents. The second contribution, *Cascading agent tree search* (CATS), is a variation on tree search with an action selection that is biased by the informed policy and is catered for multiple agents. We prove convergence to optimal values of the Markov decision process under the Bellman operation. To evaluate the performance of this algorithm, we train a deep neural network as an informed policy, deploy the algorithm distributively across agents, and evaluate the performance across several metrics in the environments of Figure 1. In a comparison with similar tree search and model-free reinforcement learning approaches, CATS excels when the number of agents increase and when search time is limited to realistic time constraints for online deployment.

Notation: Throughout the paper, we use the following notation. Let $\mathbb{R}$ represent the set of real numbers. Let $\langle a, b, c\rangle$ and $[a, b, c]$ represent a tuple and an ordered list, respectively, of elements a, b, and c. In what follows, we use the notation $:$ to represent the concatenation of two lists. For example, if $A = [a, b]$ and $B = [c, d]$, then $A{:}\, B = [a, b, c, d]$. Finally, we use the notation $|S|$ to represent the cardinality of a set S.

II. Preliminaries

This section introduces basic preliminaries on Markov decision processes and Monte-Carlo tree searches. We also discuss imitation learning and its use to bias tree searching in multi-agent environments to improve performance.

Markov decision processes and tree search: A Markov decision process (MDP) is a tuple $\langle \mathcal{A}, \mathcal{S}, \mathcal{R}, \Pr, \gamma\rangle$, where $s \in \mathcal{S}$ and $a \in \mathcal{A}$ are state and action spaces respectively; $\Pr_{s'|s,a}$ is the transition function which encodes the probability of the next state being s' given current state s and action a. After every transition, a reward is obtained according to the reward function $\mathcal{R} : s', a, s \to r \in \mathbb{R}$. A policy π specifies the actions given a state according to $\pi : s, a \to (0, 1)$ such that $\sum_{a\in\mathcal{A}} \pi_{s,a} = 1$. Given a policy π, the *state value* is

$$V_s^\pi = \sum_{a\in\mathcal{A}} \pi_{s,a} \sum_{s'\in\mathcal{S}} \Pr_{s'|s,a}\Big(\mathcal{R}_{s',a,s} + \gamma V_{s'}^\pi\Big),$$

where $\gamma \in [0, 1]$ is a discount factor. The *state-action value* is the value of taking an action at state is

$$Q_{s,a}^\pi = \sum_{s'\in\mathcal{S}} \Pr_{s'|s,a}\Big(\mathcal{R}_{s',a,s} + \gamma V_{s'}^\pi\Big).$$

When the transition function of the MDP is known, a popular method for approximating the optimal policy is Monte-Carlo tree search [8]. There are four major steps in MCTS: selection, expansion, simulation, and backpropagation. During the selection process, actions are chosen from $\mathcal{A}$ to transition the MDP until a state has been reached for the first time, where it then expands the tree by one node. The next step is to use a predefined rollout policy to simulate future moves until a specified depth or state. If no information is known regarding the environment, it is common for the rollout policy to return a random move. Finally, rewards obtained during the tree traversal are backpropagated to update the estimated values for taking actions at the visited states.

Upper confidence bound tree search (UCT) [7] executes the first step with the following action selection policy

$$\operatorname*{argmax}_{a\in\mathcal{A}} \left(\hat{Q}_{s,a} + c_1\sqrt{\frac{\ln N_s}{N_{s,a}}}\right). \tag{1}$$

Here the first term, $\hat{Q}_{s,a}$ is the empirical value estimation for choosing action a at state s, which is exploitive and influences the action selection towards actions that yielded higher rewards in previous iterations of the tree search. In the second term, N_s and $N_{s,a}$ are the number of times that the state has been visited and the number of times that a particular action has been chosen at that state, respectively. The second term is explorative and biases the search towards actions that have been selected least often at a state. The probability that the optimal action is selected by UCT goes to 1 as shown in rigorous finite-time analysis [21], [22].

Tree search bias via imitation learning: Deep learning has also been used [11]–[15] to predict which actions to take in a never before visited state. We call the resulting network an *informed policy*, $\hat{\pi}$, which specifies that training used data created from previous iterations of tree search on the environment. In both ExIt [15] and AlphaZero [14], an informed policy is trained to resemble the *action selection distribution* $N_{s,a}/N_s$. The informed policy is then used to improve future tree searches by biasing the action selection of the tree search as follows

$$\operatorname*{argmax}_{a\in\mathcal{A}} \left(\hat{Q}_{s,a} + c_1\sqrt{\frac{\ln N_s}{N_{s,a}}} + c_2\frac{\hat{\pi}_{s,a}}{N_{s,a}}\right), \tag{2}$$

where c_2 weights the neural networks influence on action selection. Our algorithm builds on this idea to modify the action selection by tempering the explorative term using a sequential process in order to balance exploitation versus exploration. In the multi-agent domain, the informed policy can be used to enable distributed tree searches. To do this, one agent will search using its own action space (as opposed to the joint action space amongst all agents) while other

agents essentially become part of the environment and are modeled by taking the max-likelihood action according to $\hat{\pi}$.

III. Problem statement

Consider a scenario where cooperative homogeneous agents seek to maximize future discounted rewards in an environment modeled as a Markov decision process. Let $\mathcal{I}$ denote the set of agents and $s_i \in \mathbb{R}^d$ be the state of agent i. The state of the environment is then given by $s = [s_1, \ldots, s_{|\mathcal{I}|}, s^\alpha] \in \mathcal{S}$, where $s^\alpha = \mathbb{R}^p$ corresponds to the non-agent states of the environment. At every time step, each agent i chooses from a set of discrete actions given by $a_i \in \mathcal{A}_i$. Agents act simultaneously according to a *joint action* defined as $a \in \mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_{|\mathcal{I}|}$, which is determined by the distributed selection of actions amongst agents. Given a joint state s and a joint action a, the probability that a state transitions to s' is $\Pr_{s'|s,a}$. Agents receive the same reward at every step in the environment with $\mathcal{R}$ which encodes the success of cooperative tasks. The goal is to determine a policy that maximizes the state values for the Markov decision process $\langle \mathcal{A}, \mathcal{S}, \mathcal{R}, \Pr, \gamma \rangle$. Achieving this goal is challenging for multi-agent environments because the joint state and action space grow exponentially with the number of agents. We are interested in reasoning over environments where the alignment of the joint actions of the agents may have a significant impact on performance.

IV. Multi-agent tree search and bias exploitation

In this section we introduce `MIPC` and `CATS`, cf. Figure 2. First we discuss `MIPC`, an algorithm for constructing an informed policy offline. This construction requires the simulation of environments and data collection, which is performed by `CATS`.

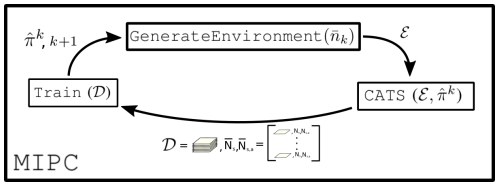


Fig. 2. Flowchart of the iterative process `MIPC`. First, many environments are generated parameterized by a specified number of agents. In parallel processes, the tree search `CATS` generates expected state-action values and action selection distributions for each of the generated environments. A convolutional neural network is trained to map an image that represents a local perspective of a single agent to the output of `CATS`, yielding an informed policy.

A. Multi-agent informed policy construction

`MIPC` is an algorithm that runs offline to iteratively build an informed policy. There are three main components to it, cf. Figure 2: `GenerateEnvironment`, `CATS`, and `Train`. Let $\bar{n} = [\bar{n}_1, \bar{n}_2, \ldots]$ be a list with entries, $\bar{n}_k \in \mathbb{R}$, that correspond to the number of agents that will be spawned in an environment during the k-th iteration of `MIPC`. Hand tuning $\bar{n}$ allows for the customization of difficulty for each iteration. In practice, we start with a small amount of agents which is slowly incremented with iterations of `MIPC`. The function `generate_environment`$(\bar{n}_k)$ creates a randomized environment with $\bar{n}_k$ agents. Iterations of `MIPC` begin by generating environments with n_1 agents. The environments are simulated in `CATS`, which gathers data using tree search. The data collected is used to create an informed policy $\hat{\pi}^1$ with `Train`. On the next iteration, environments are generated with n_2 agents and $\hat{\pi}^1$ is used to catalyze the data collection in `CATS`. The new data is used to create a new informed policy, $\hat{\pi}^2$. This process iterates up to $|\bar{n}|$ times.

The informed policy is a convolutional neural network that maps an image and an action to a number, $(x_i, a_i) \mapsto \hat{\pi}_{x_i,a_i}(x_i, a_i) \in (0, 1)$, where x_i is an image that represents the local perspective of an agent i during the initial state of the simulation that is labeled by the action selection distribution determined in `CATS`. This mapping from state to an image is translated and rotated for agent i and is low resolution in practice to allow for fast inference. We map the state to an image for two reasons. First, the number of dimensions of the full state is dependent on the number of agents. Mapping the full state to an image results in a state space dimension that is not a function of the number of agents. Second, many multi-agent deployment problems are spatial by nature. Relevant spatial information can be captured by convolutional neural networks.

B. Cascading agent tree search

`CATS` is a variation of MCTS which utilizes a neural network to guide the action selection for multiple agents. Algorithm 1 presents the pseudocode for the main components: `GenerateBias`, `Traverse`, `Expansion`, `Simulation`, and `Backpropagation`. In `GenerateBias` we generate an approximation of the neural network evaluated at the initial state of the simulation that we use to increase tree search speed. In `Traverse` we modify the tree traversal and action selection process of the tree search to accomodate for multiple agents. `Expansion`, `Simulation` and `Backpropagation` are all unchanged from the original UCT [7]. The tree search is executed from the perspective of a single agent. Actions of this agent and others are sampled with respect to a metric, *cascaded agent action selection*, where agents choose actions in a sequential manner. The order that agents choose actions is determined by $\bar{\mathcal{I}} = [i_1, \ldots, i_{|\mathcal{I}|}]$, where the first agent i_1 indicates the agent who is executing the tree search. `CATS` outputs the following, $\hat{Q}, N_s, N_{s,a}$, which are used to train informed policies or to choose actions during runtime.

1) Bias generation: Before performing tree search, we generate biases that influence actions selected. Ideally, the bias comes from the evaluation of the informed policy created by `MIPC` each time a new state is visited as a forward pass through the neural network. Unfortunately, this term has to be evaluated $\mathcal{O}(|\mathcal{A}|^D)$ times, restricting its use for large scale multi-agent systems. To avoid this problem, we devise a fast local approximation of informed policy for each agent before tree search in `GenerateBias`. To do this, we evaluate the informed policy at nearby locations for agent i while keeping all other agents at their original state.

This is done for every action in the agent's action space. Evaluations are fit using regression and the output is the *local bias* $(s_i, a_i) \mapsto \psi_{s_i,a_i}(s_i, a_i) \in (0, 1)$ evaluated at the initial state of the environment.

Algorithm 1: `MIPC` and `CATS`

```
MIPC (N, π̂_0)
    for k = 0 to |n̄|:
        D = ∅
        do in parallel:
            E ← GenerateEnvironment(n̄_k)
            Φ_{s,0}, N_s, N_{s,a} ← CATS (E, π̂^k)
            D.append(Φ_{s,0}, N_s, N_{s,a})
        π̂^{k+1}.Train (D)
    return π̂^{|n̄_k|}

CATS (E, Ī):
    for agent i in Ī
        ψ_i ← GenerateBias(E, π̂^k)
    for allotted time
        Traverse(s^c, Ī, k)
        Expansion(...)
        Simulation(...)
        Backpropagation(...)

GenerateBias (E, π̂^k, i):
    for a_i in A_i:
        for allotted time
            s'_i = sample uniformly nearby s_i
            s' = s such that with s'_i replaces s_i
            X.append(s'_i)
            Y.append(π̂_{x_i,a_i})
        ψ_{i,s_i,a_i} ← regression of X on Y
    return ψ_i = (ψ_{i,a_1}, ..., ψ_{i,a_|A|})

Traverse(s^c, Ī, k):
    if k < |Ī|:
        s^c.ā.append(argmax_{a_i ∈ A_i} f_caas(s^c, a_i))
        k = k + 1
    else:
        s' ← evolve environment
        s^c = ⟨s', [ ]⟩
        k = 0
    if s^c is unvisited:
        return s^c
    else:
        Traverse (s^c, Ī, k)
```

2) Cascading agent action selection: Our proposed action selection is biased by the informed policy via the local bias ψ and allows for exploration of joint action spaces. To choose a joint action for the selection process, an action is chosen for each agent sequentially. Let $\bar{a}$ be a *cascading joint action*, which is an ordered list of some agents' actions. The cascading joint action starts empty and is built sequentially as agents choose actions. We introduce *cascading states* to be $s^c = \langle s, \bar{a} \rangle$, which contains the current state of the environment s and the set of actions already selected by agents $\bar{a}$. Agents choose their actions with respect to the number of times a cascaded state has been visited, N_{s^c}, implying that the action selection process is conditioned on the selections of previous agents in the sequential process. Algorithm 1 outlines the sequential process under the `Traverse` pseudocode. The action selection metric is modified accordingly to reflect the change of $N_s \to N_{s^c}$ and $N_{s,a} \to N_{s^c,a_i}$ as

$$f_{caas}(s^c, a_i) = \hat{Q}_{s^c,a_i} + c_1 \sqrt{\frac{\ln N_{s^c}}{N_{s^c,a_i}}} + \frac{\psi_{s^c.s_i,a_I}}{\sqrt{N_{s^c,a_i}}}.$$

The maximization of this function leads the agent to choose the action most recommended by $\psi_{s^c.s_i,a_i}$ when visiting a state-action pair for the first time because $\hat{Q}_{s^c,a_i}$ is initialized at 0. The cascading state now transitions both when an agent selects an action and when all of the agents take a step in the environment, which is a modification of the `Traverse` process in the tree search.

Remark 4.1: (*Advantages of cascaded agent action selection*): This method of action selection has several implications. The cascade effect on agents' exploration grants the ability to discover reward signals behind joint actions. If only one agent is allowed to explore its individual action space at a time, then it is committed to plan strictly under the predictions of other agents. When those predictions do not include actions necessary for joint cooperation, then certain reward signals will be difficult to reach. The second advantage of cascading exploration is that agents are able to take a bad prior informed policy $\hat{\pi}^k$ and create a good post informed policy $\hat{\pi}^{k+1}$ given enough time. This is balanced by the exploitive local bias, which allows for search deep into the tree for tractability. This algorithm is used to generate a dataset that will train $\hat{\pi}^{k+1}$ for the first agent only. □

Remark 4.2: (*Cascaded agent action selection vs joint action selection*): Our proposed action selection does not improve the branching factor of the tree search when compared with the joint case. There are structural advantages that lead to efficient tree traversal when allowing agents to choose one at a time. A particular case is when agents receive some rewards based on non-joint actions or partial rewards for joint actions. □

C. Online and offline deployment

`CATS` is meant to be run both offline and online. `CATS` is used to generate data for training informed policies offline for `MIPC` as shown in Figure 2, where the agent order $\bar{\mathcal{I}}$ is chosen at random. In the online case, CATS is executed on each of the agents individually during deployment and all agents take actions simultaneously. Each of the agent choose themselves as first in $\bar{\mathcal{I}}$, and the rest is chosen heuristically.

V. Convergence of CATS to optimal value

Here we analyze the convergence properties of CATS. Our technical analysis proceeds in two steps. We discuss how to properly model the sequence of multi-agent state

transition and action selection in the algorithm execution via cascaded MDPs and then, building on this construction, we establish that the state-action value estimate determined by CATS converges to the optimal value of the MDP.

A. Cascaded MDPs

The sequential action selection process employed by f_{caas}, cf. Section IV-B.2, requires care in using standard MDPs to model the process of state transition and action selection. This is because agents choose their actions conditioned on the current tentative sequence of actions decided by agents which choose earlier in the sequence: in turn, this is information not contained in states of the originally defined MDP. To address this issue, we transform the original MDP with multiple agents and joint action selection into a *cascaded MDP*. Additionally we show that this transformation has a unique and convergent state value for every state in the new state space using value iteration with discount $0 < \gamma < 1$.

Let $\mathcal{C} = \langle S^c, A^c, \Pr^c, R^c, \gamma^c \rangle$, where $s^c \in S^c$ contains states defined in original MDP as well as cascaded actions $\bar{a}$. The notation $s^c_{j,n} = \langle s_j, [a_1, a_2, \ldots, a_n] \rangle$ specifies the environment state s_j from the original MDP and the actions that are in $\bar{a}$. The set of allowable actions for each agent is $a_i \in \mathcal{A}_i \in A^c$ and $\Pr^c$, R^c, and γ^c are dependent on one of two types of state transitions that we distinguish in $\mathcal{C}$. The first type of state transition, *intra-agent transitions*, occurs when the current state's cascaded action is not full, i.e., $s^c_{j,n}$ such that $n < N-1$. In this situation, the next action that will be picked will not yet complete the sequence of actions (one per agent) so this action will not yet result in a step in the environment. Picking an action results in a determined probability of transition, $\Pr^c_{s^c_{j,n+1}|s^c_{j,n},a} = 1$. The next state will contain the previously selected sequence of actions *and* the newly selected a. The reward given for this transition is always zero, $R^c_{s^c_{j,n+1},a,s^c_{j,n}} = 0$. Furthermore, the discount factor is always $\gamma^c = 1$. The second type of state transition, *environmental transitions*, happens when the last agent in the sequence chooses an action, i.e., $s^c_{j,n}$ such that $n = N-1$. At this point, the agents all take their promised action in the environment and the next state, $s^c_{j+1,0}$, contains changes in the environment and an empty cascaded action. This transition has the corresponding probability of transition in the original MDP, $\Pr^c_{s^c_{j+1,0}|s^c_{j,N-1},a} = \Pr_{s_{j+1}|s_j,\bar{a}}$, given that $s^c.\bar{a} = \bar{a}$ and $s^c.s = s$. The reward under this transition is determined by the original MDP as well, $R^c_{s^c_{j+1,0},a,s^c_{j,N-1}} = R_{s_{j+1},\bar{a},s_j}$, such that $s^c.s = s$. Finally, the discount factor used is the same as the constant discount factor from the original MDP, $\gamma^c_{s^c} s = \gamma$. Figure 3 shows the original MDP and the cascaded MDP. The convergence of MDPs under value iteration generally depend on having the discount factor $0 < \gamma < 1$. Our next result shows that given the constraints of the probability of transition, reward function, and the discount factor under intra-agent transitions and environment transitions, the state value converges under the Bellman operator $\mathcal{B}$ defined by

$$\mathcal{B}(V_s) = \max_a \left(r_{s,a} + \gamma \sum_s \Pr_{s'|s,a} V_{s'} \right).$$

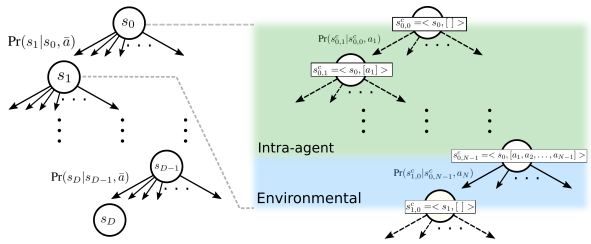


Fig. 3. Original MDP (left) and associated cascaded MDP (right). The MDPs start at the root state s_0 and $s^c_{0,0}$, respectively. The original MDP is shown to depth D. Its first step is expanded to show the additional steps of the cascaded MDP. Transition probability $\Pr(s^c_{0,1}|s^c_{0,0},a_1) = 1$ in the green region indicates an intra-agent state transition (dashed line), where the first agent in the sequence selects action a_1. Further down the tree in the blue region, $\Pr(s^c_{1,0}|s^c_{0,N-1},a_N) = \Pr(s_1|s_0,\bar{a})$ such that $\bar{a} = [a_1, a_2, \ldots, a_N]$, represents environment transitions (solid line).

For $0 < \gamma < 1$, the Bellman operator induces V^*.

Theorem 5.1 (Convergence to V^*): Consider an MDP and its associated cascaded MDP, generated as described above. Let V be the estimated state value induced by the Bellman operator in a cascaded MDP. V converges to the optimal state value of the original MDP, V^*. Furthermore, for all s^c such that $s^c.s = s$, the state values from the original MDP and cascaded MDP are equivalent, $V_{s^c} = V^*_s$.

Proof: We examine the two types of transitions in the cascaded MDP. If $s^c_{j,n}$ such that $n < N-1$ then an intra-agent transition occurs next. Let $S^c_{j,n+1}$ be the set of all states that could occur at depth $n+1$, then

$$\mathcal{B}(V_{s^c}) = \max_{s^c_{j,n+1}} V_{s^c_{j,n+1}}, \tag{3}$$

Since $r_{s^c_{j,n+1},s^c_{j,n}} = 0$, $\Pr^c_{s^c_{j,n+1}|s^c_{j,n},a} = 1$, and $\gamma^c = 1$. Therefore, a Bellman operation sets the state value equal to the next state value for intra-agent transitions. This implies that in a sequence of intra-agent transitions, all state values become equal to each other under multiple Bellman operations and

$$\begin{aligned}\mathcal{B}^{N-1}(V_{s^c_{j,0}}) &= \mathcal{B}^{N-1}(V_{s^c_{j,0}}) = \mathcal{B}^{N-2}(V_{s^c_{j,1}}) \\ &= \ldots = \max_{s^c_{j,N-1} \in S^c_{j,N-1}} V_{s^c_{j,N-1}},\end{aligned}$$

where $s^c_{j,N-1}$ is a state prior to all agents choosing their actions. After the last agent selects an action, an environmental transition occurs and the next state is of the form $s^c_{j+1,0}$. Under $N-1$ Bellman operations, $s^c_{j,0}$ becomes affected by the next environmental transition, which can be shown to be a contraction as follows,

$$\begin{aligned}&|\mathcal{B}^{N-1}(V_{s^c_{j,0}}) - \mathcal{B}^{N-1}(\bar{V}_{s^c_{j,0}})| \\ &= |\mathcal{B}^{N-2}(V_{s^c_{j,1}}) - \mathcal{B}^{N-2}(\bar{V}_{s^c_{j,1}})| \\ &= |\mathcal{B}^{N-3}(V_{s^c_{j,2}}) - \mathcal{B}^{N-3}(\bar{V}_{s^c_{j,2}})| \\ &= \ldots = |\mathcal{B}(V_{s^c_{j,N-1}}) - \mathcal{B}(\bar{V}_{s^c_{j,N-1}})|,\end{aligned}$$

where the derivation of convergence for standard MDPs can be applied. For brevity, we use $\overline{\Pr}$ to denote $\Pr_{s^c_{j+1,0}|s^c_{j,N-1},a_{n+1}}$,

$$|\mathcal{B}(V_{s^c_{j,N-1}}) - \mathcal{B}(\bar{V}_{s^c_{j,N-1}})|$$

$$
\begin{aligned}
&= \max_{a_{n+1}} \{ r_{s^c_{j,N-1}, a_{n+1}} + \gamma \sum_{s^c_{j+1,0}} \overline{\Pr} V_{s^c_{j+1,0}} \} \\
&\quad - \max_{a'_{n+1}} \{ r_{s^c_{j,N-1}, a'_{n+1}} + \gamma \sum_{s^c_{j+1,0}} \overline{\Pr} \bar{V}_{s^c_{j+1,0}} \}) \\
&\leq \max_{a_{n+1}} | \{ r_{s^c_{j,N-1}, a_{n+1}} + \gamma \sum_{s^c_{j+1,0}} \overline{\Pr} V_{s^c_{j+1,0}} \} \\
&\quad - \{ r_{s^c_{j,N-1}, a_{n+1}} + \gamma \sum_{y} \overline{\Pr} \bar{V}_{s^c_{j+1,0}} \} | \\
&= \max_{a_{n+1}} \gamma \sum_{y} \overline{\Pr} | V_{s^c_{j+1,0}} - \bar{V}_{s^c_{j+1,0}} | \\
&\leq \max_{a_{n+1}} \gamma \sum_{s'} \overline{\Pr} \| V - \bar{V} \|_\infty \\
&= \gamma \| V - \bar{V} \|_\infty \max_{a_{n+1}} \sum_{y} \overline{\Pr} = \gamma \| V - \bar{V} \|_\infty .
\end{aligned}
$$

Finally, because $\lim_{k \to \infty} \mathcal{B}^k(V_{s^c}) = V^*$, state values under Bellman operations converge. ∎

B. Convergence to optimal state value

Next, we show that the state-action value estimate determined by CATS converges to the optimal value of the MDP. This requires careful consideration of the effect of the local bias term ψ. Our treatment follows the argumentation in [21] for the convergence analysis of UCT, noting the necessary differences. First, let X_{it} be the payoff rewarded when action i is taken at time t. The average payoff is given by

$$
\bar{X}_{im} = \frac{1}{m} \sum_{t=1}^{m} X_{im}.
$$

Let $\mu_{im} = \mathbb{E}[\bar{X}_{im}]$ be the expected payoff for taking action i after m attempts and define

$$
\mu_i = \lim_{m \to \infty} \mu_{im}. \tag{4}
$$

Finally, let δ_{im} be the drift in the mean payoff,

$$
\mu_{im} = \mu_i + \delta_{im}. \tag{5}
$$

We make the following assumption.

Assumption 5.2 (Bounds on expected payoff): Fix $1 \leq i \leq K$, where $K = |A^c|$. Let $\{\mathcal{F}_{it}\}_t$ be a filtration such that $\{X_{it}\}_t$ is $\{\mathcal{F}_{it}\}$-adapted and $X_{i,t}$ is conditionally independent of $\mathcal{F}_{i,t+1}, \mathcal{F}_{i,t_2}, \dots$ given $\mathcal{F}_{i,t-1}$. Then $0 \leq X_{it} \leq 1$ and the limit of μ_{im} exists. Further, we assume that there exist a constant $C_p > 0$ and an integer N_p such that for $m \geq N_p$, for any $\delta > 0$, $\Delta_m(\delta) = C_p \sqrt{m \ln(1/\delta)}$, the following bounds hold:

$$
\begin{aligned}
\Pr(m\bar{X}_{is} \geq m\mathbb{E}[\bar{X}_{in}] + \Delta_m(\delta)) \leq \delta, \\
\Pr(m\bar{X}_{is} \leq m\mathbb{E}[\bar{X}_{in}] - \Delta_m(\delta)) \leq \delta.
\end{aligned}
$$

We let $\Delta_i = \mu^* - \mu_i$, where i and $*$ indicate suboptimal and optimal actions, respectively. Assumption 5.2 implies that δ_{it} converges to 0. Therefore, for all $\epsilon > 0$, there exists $N_0(\epsilon)$ such that if $t \geq N_0(\epsilon)$, then $|\delta_{it}| \leq \epsilon \Delta_i / 2$ and $|\delta^*_t| \leq \epsilon \Delta_i / 2$, where $|\delta^*_t|$ is the drift corresponding to the optimal action. In particular, it follows that for any optimal action, if $t > N_0(\epsilon)$, then $\delta^*_t \leq \epsilon / 2 \min_{i | \Delta_i > 0} \Delta_i$. We are ready to characterize the convergence properties of CATS.

Theorem 5.3 (Convergence of CATS): Consider CATS running to depth D where $K = |A^c|$ is the number of actions available to each agent. Assume that rewards at the leafs are in the interval $[0, 1]$. Then,

$$
|\hat{Q}_{s^c} - V^*_{s^c}| = |\delta^*_m| + \mathcal{O}\left(\frac{KND \log(m) + K^{ND}}{m} \right), \tag{6}
$$

for any initial state s^c. Further, the failure probability at the root converges to zero as the number of samples grow to infinity.

Proof: The proof follows closely the steps in [21] to establish convergence of UCT. To determine the expectation on the number of times suboptimal actions are taken, fix $\epsilon > 0$ and let $T_i(n)$ denote the number of plays of arm i. Then if i is the index of a suboptimal arm and assuming that every action is tried at least once, we bound the T_i using the indicator function, $\{I_t = i\}$, where l is an arbitrary integer,

$$
\begin{aligned}
&T_i(m) = 1 + \sum_{t=K+1}^{m} \{I_t = i\} \leq l + \sum_{t=K+1}^{m} \{I_t = i, T_i(t-1) \geq l\} \\
&\leq l + \sum_{t=K+1}^{m} \{ \bar{X}^*_{T^*(t-1)} + c_{t-1,T^*(t-1)} \leq \bar{X}_{i,T_i(t-1)} \\
&\quad + c_{t-1,T_i(t-1)}, T_i(t-1) \geq l \} \\
&\leq l + \sum_{t=K+1}^{m} \{ \min_{0<z<t} \bar{X}^*_z + c_{t-1,z} \leq \max_{l \leq z_i < t} \bar{X}_{i,z_i} + c_{t-1,z_i} \} \\
&\leq l + \sum_{t=1}^{\infty} \sum_{z=1}^{t-1} \sum_{z_i=l}^{t-1} \{ \bar{X}^*_z + c_{t,z} \leq \bar{X}_{i,z_i} + c_{t,z_i} \}.
\end{aligned}
$$

The last term, $\bar{X}^*_z + c_{t,z} \leq \bar{X}_{i,z_i} + c_{t,z_i}$, implies one of three possible cases

$$
\bar{X}^*_z \leq \mu^* - c_{t,z} \tag{7a}
$$

$$
\bar{X}_{i,z_i} \geq \mu_i + c_{t,z_i} \tag{7b}
$$

$$
\mu^* \leq \mu_i + 2c_{t,z_i}, \tag{7c}
$$

where l represents the number of times (7c) occurs, and (7a) and (7b) are characterized by Assumption 5.2. We proceed by finding upper bounds for each of these three cases. Under Assumption 5.2, we can use the Chernoff-Hoeffding bounds [23] $\mathbb{P}(\bar{X}_{iz} \geq \mathrm{E}[\bar{X}_{im}] + c) \leq e^{-2c^2 z}$ and $\mathbb{P}(\bar{X}_{iz} \leq \mathrm{E}[\bar{X}_{im}] - c) \leq e^{-2c^2 z}$ to bound cases (7a) and (7b). We apply $c = C \sqrt{\frac{\ln(t)}{z}} + \frac{1}{\sqrt{z}}$ to $e^{-c^2 z}$, where $C \geq \sqrt{2}$, $t > 0$, and $z > 0$ to get

$$
\begin{aligned}
&\mathbb{P}(\bar{X}^*_z \leq \mu^* - c_{t,z}) \leq e^{-2c^2 z} \\
&\leq e^{-2(C\sqrt{\frac{\ln(t)}{z}} + 1/z)^2 z} \leq e^{-2(\sqrt{2\frac{\ln(t)}{z}})^2 z} = t^{-4},
\end{aligned}
$$

and $\mathbb{P}(\bar{X}_{i,z_i} \geq \mu_i + c_{t,z_i})$ follows similarly. To bound l we look at (7c) where $2c_{t,z} \leq (1 - \epsilon)\Delta_i$. Solving for z yields

$$
z \leq \frac{2(\Delta_i + C^2 \log(t) - \epsilon \Delta_i)}{\epsilon^2 \Delta_i^2 - 2\epsilon \Delta_i^2 + \Delta_i^2}
$$

$$+\frac{2\sqrt{-2\epsilon C^2\Delta_i\log(t)+C^4\log^2(t)+2C^2\Delta_i\log(t))}}{\epsilon^2\Delta_i^2-2\epsilon\Delta_i^2+\Delta_i^2}$$
$$\leq\frac{WC^2\log(t)}{(1-\epsilon)^2\Delta_i^2}$$

such that some constant W , which can be upper bounded for any $t \geq 2$, $0 \geq \epsilon \geq 1$, and $d > 0$. Note that this requires Assumption 5.2, where $t > N_p$ and $t > m > N_0(\epsilon)$. Therefore l is bounded with

$$l = \max\{z, N_0(\epsilon), N_p\} \leq \frac{WC^2(\ln(t))}{(1-\epsilon)^2\Delta_i^2} + N_0(\epsilon) + N_p.$$

Our expected T_i is then

$$\mathbb{E}[T_i(m)] \leq \frac{WCp^2(\ln(t))}{(1-\epsilon)^2\Delta_i^2} + N_0(\epsilon) + N_p$$
$$+\sum_{t=1}^{\infty}\sum_{s=1}^{t-1}\sum_{s_i=l}^{t-1}\frac{1}{t^4} \leq \frac{WC^2(\ln(t))}{(1-\epsilon)^2\Delta_i^2} + N_0(\epsilon) + N_p + \frac{\pi^2}{3},$$

which is of order $\mathcal{O}(Cp^2\ln(m)+N_0)$. The rest of the proof follows from [21] where this bound is used to derive the expected regret as well as convergence of the probability of choosing suboptimal arms to 0. The statement follows via induction on the depth D^c, noting that $D^c = |A||D|$ as a result of the construction of the cascaded MDP. ■

The convergence bound is worse for CATS than for UCT (i.e., the constant W in (6) is larger than the constant in [21]) since the worst-case scenario for the local bias term has to be accounted for. In practice the local bias term helps the convergence rate, as we demonstrate in Section VI.

VI. Implementation on 2D environments

In this section we test 3 environments defined in Figure 1 and discuss performance of several algorithms in a variety of metrics. Significant parameters of the experiments are shown in Table I. The following variations of tree search and reinforcement learning algorithms are implemented.[1]

`CATS` The algorithm we propose in Section IV which utilizes the informed policy generated by `MIPC`.

`CATS-`π Our proposed algorithm with cascading action selection but withouts the bias term from the informed policy.

`CATS-APP` Our proposed algorithm without use of the local bias approximation. This algorithm evaluates the informed policy at every newly visited state to bias tree search.

`UCT` Monte-Carlo tree search using upper confidence bound for action selection on the full joint action space.

`PPO` Proximal policy optimization (PPO), a policy-based model-free reinforcement learning algorithm [24]. In our adaptation, each agent uses the same policy to choose actions given their local state image x.

Figure 4 displays results for the following experiments.

A. Performance vs. allotted simulation time (Perf. vs. Δt)

The sum of rewards per episode vs. Δt (time allotted for tree search per step) for L number of steps in the environment during an online deployment. We find that for each of the environments, `CATS` and `CATS-APP` have strong performance given limited simulation time in the tree search. It is likely that `CATS` scales better than `CATS-APP` in this experiment because evaluation of the local bias takes less time than a forward pass through $\hat{\pi}$ at each node during the tree search. Performance of `CATS-`π and `UCT` both start off low and increase as Δt increases as expected, however the cascading action selection structure enables `CATS-`π to scale better with Δt. `PPO` does not perform tree search so it is shown at constant performance. As Δt increases, `CATS` performs equal or better than `PPO` in the environments tested.

	Perf. vs. Δt			*Perf. vs.* $\lvert\mathcal{I}\rvert$			*Time to threshold*		
	$\mathcal{E}_{\text{comm}}$	$\mathcal{E}_{\text{cov}}$	$\mathcal{E}_{\text{res}}$	$\mathcal{E}_{\text{comm}}$	$\mathcal{E}_{\text{cov}}$	$\mathcal{E}_{\text{res}}$	$\mathcal{E}_{\text{comm}}$	$\mathcal{E}_{\text{cov}}$	$\mathcal{E}_{\text{res}}$
Δt	—	—	—	.1s	.1s	.1s	—	—	—
$\lvert\bar{\mathcal{I}}\rvert$	3	3	2	—	—	—	3	3	3
L	10	10	10	10	10	10	—	—	—

TABLE I
Parameters for each environment and experiment.

B. Performance vs. number of agents (Perf. vs. $|\mathcal{I}|$)

The sum of rewards per episode vs. $|\mathcal{I}|$ for L number of steps in an online deployment is tested. As the number of agents increases, the branching factor increases exponentially. This results in a shallow tree search because of limited time allotted for simulation. `CATS` sees superior performance as $|\mathcal{I}|$ increases as it is able to effectively simulate other agents during search under time constraints. Adding agents means an exponential increase in evaluations of the informed policy in `CATS-APP`. In $\mathcal{E}_{\text{comm}}$, `CATS-APP` performs poorly and likely would have benefited if it had more time to explore joint actions during tree search. `CATS-`π yields better performance than `UCT` in some cases where the cascading action selection allows an agent to find an adequate action when determining the best joint action is difficult under the time constraints. In some cases, `PPO` performs better than the tested tree search algorithms as the increased branching factor of tree search inhibits their performance.

C. Simulation time to threshold value (Time to threshold)

The amount of simulation time vs. depth of tree search (D) required during simulation to find an adequate solution in the environment, which is determined when $\hat{Q}$ is greater than a threshold value. This experiment measures how quickly an agent finds a satisfactory state in the MDP with respect to distance of that state, in terms of steps in the environment. `CATS-APP` performs worse than other algorithms because the evaluation of the informed policy for each agent at each new state becomes computationally expensive with respect to tree search depth. Performance in the other algorithms is influenced by how effectively agents are able to choose actions during tree search. `PPO` is omitted since it does not perform tree search.

[1]Hyperparameters for the algorithms and environments can be found at https://github.com/aaronma37/cats_hyperparameters.

VII. Conclusions

We provide two major contributions, first being a scalable approach for determining an informed policy using MIPC to recycle and reduce the amount of time required for data collection. The second contribution, CATS, is a variation of Monte-Carlo tree search, which provides a balance of exploitation and exploration that utilizes prior knowledge about the environment and multi-agent scenarios. Agent explore sequentially allowing for robustness to errors in the prior informed bias enabling them to distributively determine solutions for cooperative objectives. We show that this modified MDP converges to the optimal state value and CATS estimates the optimal state value when using misinformed biases. The efficacy of CATS is shown Our algorithm is compared against variations of UCT with a joint action spaces, where it excels when joint-actions are required for rewards but require low action space size for tractability.

References

[1] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[2] F. Bullo, J. Cortés, and S. Martinez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009.

[3] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*, ser. Applied Mathematics Series. Princeton University Press, 2010.

[4] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 24–39, 2012.

[5] J. Das, F. Py, J. B. J. Harvey, J. P. Ryan, A. Gellene, R. Graham, D. A. Caron, K. Rajan, and G. S. Sukhatme, "Data-driven robotic sampling for marine ecosystem monitoring," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1435–1452, 2015.

[6] J. Cortés and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 6, pp. 495–503, 2017.

[7] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *ECML*, vol. 6. Springer, 2006, pp. 282–293.

[8] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[9] A. Hussein, M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.

[10] H. Daumés, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine Learning*, vol. 75, no. 3, pp. 297–325, 2009.

[11] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline Monte-Carlo tree search planning," in *Conference on Neural Information Processing Systems*, 2014, pp. 3338–3346.

[12] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proceedings of the 24th International Conference on Machine Learning*. ACM, 2007, pp. 273–280.

[13] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[14] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[15] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Conference on Neural Information Processing Systems*, 2017, pp. 5360–5370.

[16] F. Riccio, R. Capobianco, and D. Nardi, "Q-CP: learning action values for cooperative planning," in *IEEE Int. Conf. on Robotics and Automation*, Brisbane, Australia, 2018, pp. 6469–6475.

[17] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*. Morgan Kaufmann Publishers Inc., 1996, pp. 195–210.

[18] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Dec-MCTS: Decentralized planning for multi-robot active perception," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 316–337, 2019.

[19] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov, "Combining neural networks and tree search for task and motion planning in challenging environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6059–6066.

[20] B. Kartal, J. Godoy, I. Karamouzas, and S. J. Guy, "Stochastic tree search with useful cycles for patrolling problems," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1289–1294.

[21] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo search," *Univ. Tartu, Estonia, Tech. Rep*, vol. 1, 2006.

[22] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[23] J. Schmidt, A. Siegel, and A. Srinivasan, "Chernoff–hoeffding bounds for applications with limited independence," *SIAM Journal on Discrete Mathematics*, vol. 8, no. 2, pp. 223–250, 1995.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

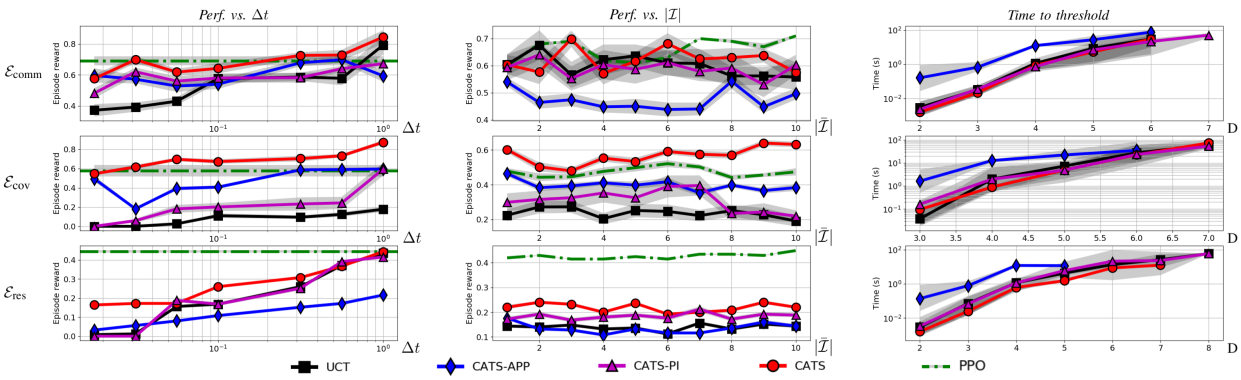


Fig. 4. Results from the experiments in each of the environments. Each datapoint is sampled 100 times and the variance is shown as the shaded region.