Distributed Policy Iteration for Scalable Approximation of Cooperative Multi-Agent Policies

Paper #162

ABSTRACT

Decision making in multi-agent systems (MAS) is a great challenge due to enormous state and joint action spaces as well as uncertainty, making centralized control generally infeasible. Decentralized control offers better scalability and robustness but requires mechanisms to coordinate on joint tasks and to avoid conflicts. Common approaches to learn decentralized policies for cooperative MAS suffer from non-stationarity and lacking credit assignment, which can lead to unstable and uncoordinated behavior in complex environments. In this paper, we propose *Strong Emergent Policy approximation (STEP)*, a scalable approach to learn strong decentralized policies for cooperative MAS with a distributed variant of policy iteration. For that, we use function approximation to learn from action recommendations of a decentralized multi-agent planning algorithm. STEP combines decentralized multi-agent planning with centralized learning, only requiring a generative model for distributed black box optimization. We experimentally evaluate STEP in two challenging and stochastic domains with large state and joint action spaces and show that STEP is able to learn stronger policies than standard multi-agent reinforcement learning algorithms, when combining multi-agent open-loop planning with centralized function approximation. The learned policies can be reintegrated into the multi-agent planning process to further improve performance.

KEYWORDS

multi-agent planning; multi-agent learning; policy iteration

1 INTRODUCTION

Cooperative multi-agent systems (MAS) are popular in artificial intelligence research and have many potential real-world applications like autonomous vehicles, sensor networks, and robot teams [9, 10, 16]. However, decision making in MAS is extremely challenging due to intractable state and joint action spaces as well as stochastic dynamics and uncertainty w.r.t. other agents' behavior.

Centralized control does not scale well in large MAS due to the *curse of dimensionality*, where state and joint action spaces grow exponentially with the number of agents [1, 6, 9, 10, 16, 19]. Therefore, decentralized control is recommended, where each agent decides its individual actions under consideration of other agents, providing better scalability and robustness [9, 10, 16, 19]. Decentralized approaches to decision making in MAS typically require a *coordination mechanism* to solve joint tasks and to avoid conflicts [6].

Learning decentralized policies with *multi-agent reinforcement learning (MARL)* in cooperative MAS faces two major challenges: One challenge is *non-stationarity*, where all agents adapt their behavior concurrently which can lead to unstable and uncoordinated policies [11, 15, 27, 31, 37]. Another challenge is *multi-agent credit assignment*, where the joint action of all agents leads to a single global reward which makes the deduction of the individual contribution of each agent difficult for adequate adaptation [8, 16, 45, 56].

Many approaches to solve these problems use reward or value decomposition to provide individual objectives [8, 19, 45] or use reward shaping to obtain objectives which are easier to optimize [12, 56]. However, these approaches are generally not sufficient or infeasible due to complex emergent dependencies within large MAS which are hard to learn and specify [16].

Recent approaches to learn strong policies are based on *policy iteration* and combine planning with deep reinforcement learning, where a neural network is used to imitate the action recommendations of a tree search algorithm. In return, the neural network provides an action selection prior for the tree search [2, 44]. This iterative procedure, called *Expert Iteration (ExIt)*, gradually improves both the performance of the tree search and the neural network [2]. ExIt has been successfully applied to zero-sum games, where a single agent improves itself by self-play. However, ExIt cannot be directly applied to large cooperative MAS, since using a centralized tree search is practically infeasible for such problems [9, 10].

In this paper, we propose *Strong Emergent Policy approximation (STEP)*, a scalable approach to learn strong decentralized policies for cooperative MAS with a distributed variant of policy iteration. For that, we use function approximation to learn from action recommendations of a decentralized multi-agent planner. STEP combines decentralized multi-agent planning with centralized learning, where each agent is able to explicitly reason about emergent dependencies to make coordinated decisions. Our approach only requires a generative model for distributed black box optimization.

We experimentally evaluate STEP in two challenging and stochastic domains with large state and joint action spaces and show that STEP is able to learn stronger policies than standard MARL algorithms, when combining multi-agent open-loop planning with centralized function approximation. The policies can be reintegrated into the planning process to further improve performance.

The rest of the paper is organized as follows. Some background about decision making is provided in Section 2. Section 3 discusses related work. STEP is described in Section 4. Experimental results are presented and discussed in Section 5. Section 6 concludes and outlines a possible direction for future work.

2 BACKGROUND

2.1 Multi-Agent Markov Decision Processes

2.1.1 MDP. A *Markov Decision Process (MDP)* is defined by a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is the (finite) set of actions, $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the transition probability function, and $\mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$ is the reward function [40]. We

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 2019, Montreal, Canada

https://doi.org/doi

always assume that $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, and $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$, where s_{t+1} is reached after executing a_t in s_t at time step t.

The goal is to find a *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ which maximizes the (discounted) return G_t at state s_t for a horizon h:

$$G_t = \sum_{k=0}^{h-1} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}, s_{t+k+1}) \quad (1)$$

where $\gamma \in [0, 1)$ is the discount factor. Alternatively, a policy may be stochastic such that $\pi(a_t|s_t) \in [0, 1]$ with $\sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) = 1$.

A policy π can be evaluated with a *state value function* $V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t] = max_{a_t \in \mathcal{A}}(Q^\pi(s_t, a_t))$, which is defined by the expected return at s_t. $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t]$ is the *action value function* of π defining the expected return when executing a_t in s_t.

π is optimal, if it is *stronger* than all other policies π' such that $V^\pi(s_t) \geq V^{\pi'}(s_t)$ for all $s_t \in S$. We denote the optimal policy by π^* and the optimal value function by $V^{\pi^*} = V^*$ or $Q^{\pi^*} = Q^*$ resp.

2.1.2 MMDP. An MDP can be extended to a *multi-agent MDP (MMDP)* with a (finite) set of agents $\mathcal{D} = \{1, ..., N\}$. $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$ is the (finite) set of *joint actions*. The goal is to find a *joint policy* $\pi = \langle \pi_1, ..., \pi_N \rangle$ which maximizes the return G_t of Eq. 1. π_i is the *local policy* of agent $i \in \mathcal{D}$. Similarly to MDPs, a value function V^π can be used to evaluate the joint policy π.

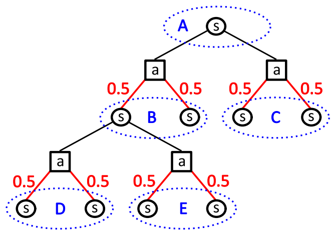
MMDPs can be used to model fully observable problems for cooperative MAS, where all agents share a common goal [6, 9, 10]. In this paper, we focus on *homogeneous* MAS with $\mathcal{A}_1 = ... = \mathcal{A}_N$ and $\pi^* = \langle \pi_1^*, ..., \pi_N^* \rangle$, where $\pi_i^* = \pi_j^*$, even if $i \neq j$ [36, 52, 59] [1].

2.2 Planning

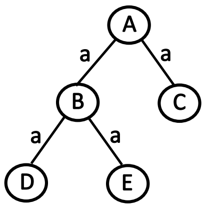
Planning searches for an (near-)optimal policy, given a model $\hat{M}$ of the environment M. $\hat{M}$ provides an approximation for $\mathcal{P}$ and $\mathcal{R}$ of the underlying MDP or MMDP [6]. *Global planning* searches the whole state space $\mathcal{S}$ to find π^*. *Policy iteration* is a global planning approach which computes π^* with alternating *policy evaluation*, where V^{π^n} is computed for the current policy π^n and *policy improvement*, where a *stronger policy* π^{n+1} is generated by selecting actions that maximize V^{π^n} for each state $s_t \in \mathcal{S}$ [6, 23]. *Local planning* only regards the current state s_t and possible future states to find a policy π_t with closed- or open-loop search [55]. *Monte Carlo planning* uses a *generative model* $\hat{M}$ as black box simulator without reasoning about explicit probability distributions [25, 28, 55].

Closed-loop planning conditions the action selection on the history of previous states and actions. *Monte Carlo Tree Search (MCTS)* is a popular closed-loop algorithm, which incrementally constructs a search tree to estimate V^* [25, 44]. It traverses the tree by selecting nodes $s_t \in \mathcal{S}$ with a policy π_{tree} until a leaf node s_{h-1} is reached. π_{tree} is commonly implemented with the UCB1 selection strategy which maximizes $UCB1 = \overline{Q(s_t, a_t)} + c\sqrt{2log(n_t)/n_{a_t}}$, where $\overline{Q(s_t, a_t)}$ is the average return, n_t is the visit count of s_t, n_{a_t} is the selection count of a_t, and c is an exploration constant [3, 25]. The node s_{h-1} is expanded by a new child node s_h, whose value $\hat{V}(s_h)$ is estimated with a rollout or a value function [44]. The observed rewards are recursively accumulated to returns G_t (Eq. 1) to update the value estimate $\overline{Q(s_t, a_t)}$ of each state-action pair in the search path. MCTS is an *anytime algorithm*, which returns an action recommendation for the root state s_0 according to the highest action value after a *computation budget* n_b has run out.

In stochastic domains, closed-loop planning needs to store each state s_{t+1} encountered when executing a_t in s_t. This may lead to large search trees with high branching factors, if $\mathcal{S}$ and $\mathcal{A}$ are very large. *Open-loop planning* conditions the action selection only on previous actions and summarized statistics about predecessor states, thus reducing the search space [38, 55]. An example is shown in Fig. 1. A closed-loop tree for a domain with $\mathcal{P}(s_{t+1}|s_t, a_t) = 0.5$ is shown in Fig. 1a. Fig. 1b shows the corresponding open-loop tree which summarizes the state nodes of Fig. 1a within the blue dotted ellipses into state distribution nodes. *Open-Loop UCB applied to Trees (OLUCT)* is an open-loop variant of MCTS with UCB1 [28].



(a) closed-loop tree



(b) open-loop tree

Figure 1: Illustration of closed- and open-loop trees for planning. (a) Closed-loop tree with states (circular nodes) and actions (rectangular nodes). Red links correspond to stochastic state transitions with a probability of 0.5. (b) Open-loop tree with links as actions and nodes as state distributions according to the blue dotted ellipses in Fig. 1a.

2.3 Reinforcement Learning

Reinforcement Learning (RL) searches for an (near-)optimal policy in an environment M without knowing the effect of executing $a_t \in \mathcal{A}$ in $s_t \in \mathcal{S}$ [6, 47]. RL agents typically obtain experience samples $E = \{e_1, ..., e_t\}$ with $e_t = \langle s_t, a_t, s_{t+1}, r_t \rangle$ by interacting with the environment. *Model-based* RL methods learn a model $\hat{M} \approx M$ by approximating $\mathcal{P}$ and $\mathcal{R}$ with E [6, 47]. $\hat{M}$ can be used for planning to find a policy. Alternatively, π^*, V^*, and/or Q^* can be approximated directly with E by using *model-free RL* [26, 46, 48, 54].

2.4 Decision Making in MMDPs

An MMDP can be formulated as a joint action MDP and solved with single-agent planning or RL by directly searching for joint policies [6]. However, this does not scale well due to the curse of dimensionality, where the state and joint action spaces grow exponentially with the number of agents [1, 6, 9, 10, 16, 19].

Alternatively, local policies can be searched with decentralized planning or RL, where each agent plans or learns independently of each other [9, 51]. Decentralized approaches typically require a *coordination mechanism* to solve joint tasks and to avoid conflicts [6]. Common mechanisms are communication to exchange private information [39, 51], synchronization to reach a consensus [13, 35], or prediction of other agents' behavior with policy models [9, 10].

[1]We assume that each agent is uniquely identified by its identifier i and individual state s_t^i to ensure that there exists an optimal local policy π_i^* for all agents [6, 43, 59].

3 RELATED WORK

Policy Iteration with Deep Learning and Tree Search. Recently, approaches to learning strong policies from MCTS recommendations with deep learning have been successfully applied to single-agent domains and zero-sum games [2, 18, 24, 44]. In zero-sum games, a single agent is trained via self-play, gradually improving itself when playing against an increasingly stronger opponent. This corresponds to the policy iteration scheme, where self-play evaluates the current policy and MCTS improves the policy by recommending stronger actions based on the evaluation.

Our approach addresses cooperative problems with *multiple* agents, where a local policy has to be learned for each agent, which maximizes the *common return.* We use *decentralized multi-agent planning* to recommend actions for local policy approximation, since centralized planning would be infeasible for large MAS [6, 9]. The local policy approximation also serves as *coordination mechanism* to predict other agents' behavior during planning [9, 10, 52].

Policy Iteration for Cooperative MAS. Previous work on policy iteration in MAS has focused on centralized offline planning, where an (near-)optimal joint policy is searched by exhaustively evaluating and updating all local policy candidates for each agent with an explicit model of the MAS [4, 5, 20, 49]. Dominated policy candidates can be discarded by using heuristics to reduce computation [42, 57]. However, these approaches do not scale well for complex domains due to the curse of dimensionality of the (joint) policy space [33].

Our approach is more scalable, since we use *decentralized multi-agent planning* for training, which can be performed online, and a single *function approximation* to learn a local policy for each agent. Our approach only requires a *generative model* for black box optimization but no explicit probability distributions of the MAS.

Multi-Agent Reinforcement Learning (MARL). MARL is a widely studied field [7, 51] and has been often combined with deep learning [14, 50]. A scalable way to MARL in cooperative MAS is to let each agent learn its local policy independently of others [29, 50, 51], but non-stationarity and the lack of credit assignment can lead to uncoordinated behavior [11, 16, 27, 45]. Non-stationarity can be addressed with stabilized or synchronized experience replay [17, 35], or with opponent modeling [15, 21, 22, 41, 58]. Approaches to credit assignment provide local rewards for each agent [19, 30], learn a filtering of local rewards from a global reward [8, 45], or use reward shaping [12, 16, 56]. In many cases, learning is centralized, where all agents share experience or parameters to accelerate learning of coordinated local policies [14, 16, 19, 51]. While training might be centralized, the execution of the policies is decentralized [14, 19].

Our approach learns a local policy from *action recommendations* of a *decentralized multi-agent planner.* A generative model is used for distributed black box optimization, where each agent *explicitly reasons* about the global effect of its *individual actions* without additional domain knowledge or reward decomposition.

4 STEP

We now describe *Strong Emergent Policy approximation (STEP)* for learning strong decentralized policies by using a distributed variant of policy iteration. STEP defines a framework to combine decentralized multi-agent planning with centralized learning.

4.1 Scalable Policy Iteration for MAS

Similarly to MDPs, policy iteration for MMDPs consists of an alternating *evaluation* and *improvement* step [6, 23]. Given a joint policy $\pi^n = \langle \pi_1^n, ..., \pi_N^n \rangle$, the global value function V^{π^n} can be computed to evaluate π^n. By selecting joint actions $a_t = \langle a_{t,1}, ..., a_{t,N} \rangle \in \mathcal{A}$ which maximize V^{π^n} for each state $s_t \in \mathcal{S}$, we obtain an improved joint policy $\pi^{n+1}(s_t) = a_t$, which is stronger than π^n. In MMDPs, the state and joint action spaces are typically too large to exactly compute V^{π^n} and π^{n+1} [6, 9]. Thus, we use function approximation to compute $\hat{V} \approx V^{\pi^n}$ and $\hat{\pi} = \langle \hat{\pi}_1, ..., \hat{\pi}_N \rangle \approx \pi^{n+1}$ [2, 44].

For *scalable policy evaluation*, we use temporal difference (TD) learning to train $\hat{V}$ with real experience and ensure generalization to avoid computing $V^{\pi^n}(s_t)$ for each state $s_t \in \mathcal{S}$ explicitly.

For *scalable policy improvement*, we use decentralized multi-agent planning, where each agent i explicitly reasons about the global effect of its individual actions $a_{t,i} \in \mathcal{A}_i$ to maximize the common return G_t instead of searching the whole joint action space $\mathcal{A}$. A function approximator $\hat{\pi}_i$ is used to learn a local policy from the action recommendations of each agent's individual planner.

The explicit reasoning can solve the credit assignment problem, since each agent i is incentivized to optimize its individual actions to maximize the common return based on the global value function $\hat{V}$ and the policy $\hat{\pi}_j$ of all other agents $j \neq i$. Since $\hat{\pi}_i$ is trained to imitate the individual planner of agent i, $\hat{\pi}_i$ can be used to predict future actions of agent i, similarly to opponent modeling to address non-stationarity, when optimizing local decisions. This can lead to coordinated actions to solve joint tasks and to avoid conflicts [9].

Combining these elements leads to a *distributed policy iteration* scheme for cooperative MAS, which only requires a generative model for distributed black box optimization.

4.2 Decentralized Open-Loop UCT

Decentralized closed-loop planning, with a worst-case branching factor of $b_{cl} = |\mathcal{S}| \cdot |\mathcal{A}_i|$, quickly becomes infeasible when the problem is too large to provide sufficient computation budget. Thus, we focus on open-loop planning, which generally explores much smaller search spaces with a branching factor of $b_{ol} = |\mathcal{A}_i| << b_{cl}$ (Fig. 1), and can be competitive to closed-loop planning, when computational resources are highly restricted [28, 38, 55]. We propose a decentralized variant of OLUCT from [28], which we call *DOLUCT.*

At every time step t, all agents perform an independent DOLUCT search in parallel. A stochastic policy function $\hat{\pi}_i(a_{t,i}|s_t) \in [0, 1]$ is used to simulate all other agents. To traverse a DOLUCT search tree, we propose a modified version of UCB1 similarly to [44]:

$$UCB1_{DOLUCT}^{\hat{\pi}_i}(Nd_t, s_t, a_{t,i}) = \overline{Q(Nd_t, a_{t,i})} + \hat{\pi}_i(a_{t,i}|s_t)c\sqrt{\frac{2log(n_{t,i})}{n_{a_{t,i}}}} \tag{2}$$

where Nd_t is a node in the open-loop tree (Fig. 1b). Note that the local action probabilities $\hat{\pi}_i(a_{t,i}|s_t)$ for the same node Nd_t can vary depending on the currently simulated state s_t, thus providing a *closed-loop prior* for the action selection.

To avoid searching the full depth of the problem, we propose to use a value function $\hat{V}$ to evaluate states at leaf nodes [39, 44].

The complete formulation of DOLUCT is given in Algorithm 1, where i identifies the current agent, s_t is the state to plan on,

$\hat{M}$ is the generative model, N is the number of agents, n_b is the computation budget, $\hat{V}$ is a value function, and $\hat{\pi}_i$ is a local policy.

Algorithm 1 Decentralized Open-Loop UCT (DOLUCT)

```
1:  procedure $DOLUCT(i, s_t, \hat{M}, N, n_b, \hat{V}, \hat{\pi}_i)$
2:      Initialize $Nd_t$: $\overline{Q(Nd_t, a_{t,i})} = n_{t,i} = n_{a_{t,i}} = 0, \forall a_{t,i} \in \mathcal{A}_i$
3:      while $n_b > 0$ do
4:          $\langle n_b, G_t \rangle \leftarrow Simulate(i, Nd_t, s_t, \hat{M}, N, n_b, \hat{V}, \hat{\pi}_i)$
5:          $p(a_{t,i}|s_t) \leftarrow \frac{n_{a_{t,i}}}{n_{t,i}}, \forall a_{t,i} \in \mathcal{A}_i$
        return $\langle argmax_{a_{t,i} \in \mathcal{A}_i}(\overline{Q(Nd_t, a_{t,i})}), p(a_{t,i}|s_t) \rangle$
1:  procedure $Simulate(i, Nd_t, s_t, \hat{M}, N, n_b, \hat{V}, \hat{\pi}_i)$
2:      if $n_b \leq 0$ then return $\langle 0, \hat{V}(s_t) \rangle$
3:      if $Nd_t$ is a leaf node then
4:          Expand $Nd_t$
5:          return $\langle n_b, \hat{V}(s_t) \rangle$
6:      $a_{t,i} \leftarrow argmax_{a_{t,i} \in \mathcal{A}_i}(UCB1^{\hat{\pi}_i}_{DOLUCT}(Nd_t, s_t, a_{t,i}))$
7:      $a_{t,j} \sim \hat{\pi}_i(a_{t,j}|s_t), \forall_{j \in \{1,...,N\}} j \neq i$ ▷ simulate other agents
8:      $a_t \leftarrow \langle a_{t,1}, ..., a_{t,N} \rangle$
9:      $\langle s_{t+1}, r_t \rangle \sim \hat{M}(s_t, a_t)$
10:     $\langle n_b, n_{t,i}, n_{a_{t,i}} \rangle \leftarrow \langle n_b - 1, n_{t,i} + 1, n_{a_{t,i}} + 1 \rangle$
11:     $\langle n_b, R_t \rangle \leftarrow Simulate(i, Nd_{t+1}, s_{t+1}, \hat{M}, N, n_b, \hat{V}, \hat{\pi}_i)$
12:     $G_t \leftarrow r_t + \gamma R_t$
13:     $\overline{Q(Nd_t, a_{t,i})} \leftarrow ((n_{t,i} - 1)\overline{Q(Nd_t, a_{t,i})} + G_t)/n_{t,i}$
14:     return $\langle n_b, G_t \rangle$
```

4.3 Strong Emergent Policy Approximation

We intend to learn π_i^* by imitating a decentralized multi-agent planning algorithm similarly to ExIt [2, 44] for the single-agent case. The planner itself is improved with $\hat{\pi}_i \approx \pi_i^*$ as an action selection prior and as prediction of other agents' behavior for coordination, and $\hat{V} \approx V^*$ as leaf state evaluator. We assume an online setting with an alternating *planning* and *learning* step for each time step t.

In the planning step, a joint action $a_t = \langle a_{t,1}, ..., a_{t,N} \rangle$ is searched with decentralized multi-agent planning for the current state s_t. The planning algorithm can exploit the policy approximation $\hat{\pi}_i$ as a prior for action selection (Eq. 2) and as prediction of other agents' behavior for coordination. All agents execute the joint action a_t and cause a state transition to s_{t+1}, while observing a global reward r_t. The transition $e_t = \langle s_t, a_t, s_{t+1}, r_t, p_t \rangle$ is stored as *experience sample* in a *central buffer* E, where $p_t = \langle p(a_{t,1}|s_t), ..., p(a_{t,N}|s_t) \rangle$ contains the relative frequencies of the action selections $p(a_{t,i}|s_t) = \frac{n_{a_{t,i}}}{n_{t,i}}, \forall a_{t,i} \in \mathcal{A}_i$ of each agent's individual planner for state s_t.

In the learning step, a parametrized function approximator $f_{i,\theta} = \langle \hat{\pi}_i, \hat{V} \rangle$ with parameter vector θ is used to approximate π_i^* and V^* by minimizing the loss $L_{STEP} = \mathbb{E}_{i \in \mathcal{D}}[L_{STEP,i}]$ for all agents of all transitions $e_t \in E$ w.r.t. θ:

$$L_{STEP,i} = (y_t - \hat{V}(s_t))^2 - p(a_{t,i}|s_t)^\top log(\hat{\pi}_i(a_{t,i}|s_t)) \qquad (3)$$

where $y_t = r_t + \gamma \hat{V}(s_{t+1})$ is the TD target for $\hat{V}$ [46]. $p(a_{t,i}|s_t)$ and $\hat{\pi}_i(a_{t,i}|s_t)$ are $|\mathcal{A}_i|$-dimensional probability vectors. The first term is the squared TD error and the second term is the cross entropy between $p(a_{t,i}|s_t)$ and $\hat{\pi}_i(a_{t,i}|s_t)$. With L_{STEP}, $f_{i,\theta}$ can be trained online because it can incrementally incorporate new experience e_t [2]. Thus, $f_{i,\theta}$ is able to adapt to changes at system runtime and does not require the problem to be episodic [39, 46].

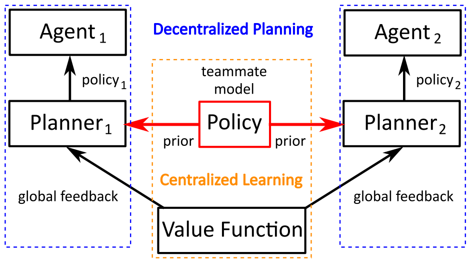


Figure 2: Architecture of STEP. The policy (red box) can be used as a prior for action selection and to predict other agents' behavior for coordination. The value function is used to evaluate leaf states for multi-agent planning [39].

The complete formulation of STEP is given in Algorithm 2, where *DPlan* is a decentralized multi-agent planning algorithm, $\hat{M}$ is the generative model, N is the number of agents, n_b is the computation budget, and $f_{i,\theta}$ is the function approximator for π_i^* and V^*.

Algorithm 2 Strong Emergent Policy Approximation (STEP)

```
1:  procedure $STEP(DPlan, \hat{M}, N, n_b, f_{i,\theta})$
2:      Initialize $\theta$ of $f_{i,\theta}$
3:      Observe $s_1$
4:      for $t = 1, T$ do ▷ $T$ is the training duration (can be infinite)
5:          $\langle \hat{\pi}_i, \hat{V} \rangle \leftarrow f_{i,\theta}$
6:          for $i \in \{1, ..., N\}$ do                ▷ decentralized planning
7:              $\langle a_{t,i}, p(a_{t,i}|s_t) \rangle \leftarrow DPlan(i, s_t, \hat{M}, N, n_b, \hat{V}, \hat{\pi}_i)$
8:          $p_t \leftarrow \langle p(a_{t,1}|s_t), ..., p(a_{t,N}|s_t) \rangle$
9:          Execute $a_t \leftarrow \langle a_{t,1}, ..., a_{t,N} \rangle$
10:         Observe global reward $r_t$ and new state $s_{t+1}$
11:         Store $e_t \leftarrow \langle s_t, a_t, s_{t+1}, r_t, p_t \rangle$ in $E$
12:         Refine $\theta$ to minimize $L_{STEP}$ for all $e_t \in E$ ▷ centralized
    learning
```

4.4 STEP Architecture

Fig. 2 shows the conceptual architecture for two agents. All agents are controlled by individual planners, which share the same local policy $\hat{\pi}_i$ and the global value function $\hat{V}$. $\hat{\pi}_i$ is used as an action selection prior and as prediction of other agents' behavior for coordination. $\hat{V}$ is used to evaluate leaf states during planning.

Although multi-agent planning is *decentralized*, learning is *centralized* to accelerate the approximation [14, 16, 19, 51]. The parameters are shared among all agents and updated in a centralized manner. Although we focus on homogeneous MAS, where all agents use the same local policy $\hat{\pi}_i$, our approach can be easily extended to heterogeneous MAS. Given a MAS with K different agent types, K different local policies need to be approximated (one for each agent type), which can still be done in a centralized fashion such that all agents have access to these K policies during training.

[2]In practice, θ is optimized on random experience batches of constant size [32].

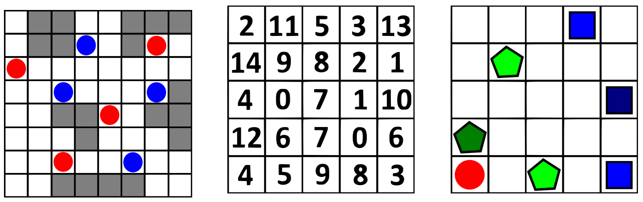
4.5 Bias Regulation

Using $f_{i,\theta}$ in decentralized tree search algorithms (e.g., DOLUCT) induces a bias in estimating π_i^* and V^* with $\hat{\pi}_i$ and $\hat{V}$ resp., thus includes approximation errors in the planning step.

The action selection bias of $\hat{\pi}_i$ can be reduced by increasing the computation budget n_b. The more node Nd_t is visited, the smaller the exploration term multiplied with c in Eq. 2 becomes, which decreases the influence of $\hat{\pi}_i$. This causes the search to focus on nodes with higher expected return $\mathbb{E}[G_t|Nd_t]$, thus the search tree is expanded into these directions. The increasing horizon h discounts the value estimate $\hat{V}(s_h)$ of newly added nodes Nd_h by a factor of γ^h, thus reducing the bias of $\hat{V}$ for frequently visited paths, if $\gamma < 1$.

5 EXPERIMENTS

5.1 Evaluation Environments



(a) Pursuit & Evasion (b) Smart Factory (SF) (c) An agent in SF

Figure 3: (a) Pursuit & Evasion ($N = 4$) with pursuers (red circles) and evaders (blue circles). (b) Machine grid of the Smart Factory (SF) with the numbers denoting the machine type. (c) An agent i (red circle) with $tasks_i = [\{9, 12\}, \{3, 10\}]$ in the SF of Fig. 3b. It should get to the green pentagonal machines first before going to the blue rectangular machines [39].

5.1.1 Pursuit & Evasion (PE). PE is a well-known benchmark problem for MARL algorithms [19, 51, 53]. We implemented this domain as 8×8 grid with N *pursuers* as learning agents, N *evaders* as randomly moving entities, and some obstacles as shown in Fig. 3a for $N = 4$, where the pursuers must collaborate to capture all evaders. All pursuers and evaders have random initial positions and are able to move north, south, west, east, or do nothing. If two pursuers occupy the same cell as an evader, a global reward of $r_t = 1$ is obtained. The reward $r_t = \sum_{i=1}^{N} r_{t,i}$ can be decomposed into local rewards $r_{t,i}$, where each of the N_{hunter} pursuers i, which occupied the cell of the captured evader, is rewarded with $r_{t,i} = \frac{1}{N_{hunter}}$.

5.1.2 Smart Factory (SF). SF was introduced in [39]. It consists of a 5×5 grid of machines as shown in Fig. 3b and N agents with each having one *item*, a list of randomly assigned tasks $tasks_i$ organized in *buckets*, and a random initial position. An example from [39] is shown in Fig. 3c for an agent i with $tasks_i = [\{9, 12\}, \{3, 10\}]$. It first needs to get processed at the machines having a machine type of 9 and 12 (green pentagons) before going to the machines with type 3 and 10 (blue rectangles). All agents are able to enqueue at their current machine, move north, south, west, east, or do nothing. At every time step, each machine processes one agent in its queue with a cost of 0.25 but does nothing with a probability of 0.1. If a task in its current bucket matches with the current machine's type, the task is removed from the agent's task list. An agent i is *complete*, if $tasks_i = \emptyset$. All agents have to coordinate to avoid conflicts to ensure fast completion of all tasks. The goal is to maximize the value of $score_t = |\mathcal{D}_{complete}| - tasks_t - cost_t - tpen_t$, where $|\mathcal{D}_{complete}|$ is the number of complete items, $tasks_t$ is the total number of unprocessed tasks, $cost_t$ is the total sum of processing cost at each machine, and $tpen_t$ is the total sum of time penalties of 0.1 per incomplete item at every time step until t. The global reward $r_t = score_{t+1} - score_t = \sum_{i=1}^{N} r_{t,i}$ can be decomposed into local rewards $r_{t,i} = score_{t+1,i} - score_{t,i}$, where $score_{t,i}$ is calculated similarly to $score_t$ by only regarding the tasks, time penalties, and machine costs concerning agent i itself.

5.2 Methods

5.2.1 Multi-Agent Planning. We implemented different planning approaches to evaluate the most suited approach for training.

DOLUCT Variants. We instantiated DOLUCT (Section 4.2) with different value functions $\hat{V}$ and policies $\hat{\pi}_i$ to simulate other agents:

- $DOLUCT_{Baseline}$ uses a random uniform policy $\hat{\pi}_{i,Random}$ and a value function $\hat{V}(s_t) = 0$ for each state $s_t \in \mathcal{S}$.
- $DOLUCT_{Random}$ uses a random uniform policy $\hat{\pi}_{i,Random}$ and approximates V^* (Section 5.2.3) to evaluate leaf states [44].
- $DOLUCT_{STEP}$ uses the current policy and value function approximation of $f_{i,\theta} = \langle \hat{\pi}_i, \hat{V} \rangle$ as described in Algorithm 2 with *DPlan* = *DOLUCT*.

Decentralized MCTS (DMCTS). We implemented $DMCTS_{STEP}$, as a closed-loop planner using STEP similarly to $DOLUCT_{STEP}$ (with $Nd_t = s_t \in \mathcal{S}$ in Eq. 2 and *DPlan* = *DMCTS* in Algorithm 2). Unlike $DOLUCT_{STEP}$, $DMCTS_{STEP}$ conditions its action selection on states and state transitions caused by simulated joint actions with a worst-case branching factor of $b_{cl} = |\mathcal{S}| \cdot |\mathcal{A}_i|$ (Fig. 1a and Section 4.2).

Direct Cross Entropy (DICE) Planning. Based on [39], we implemented an open-loop version of *DICE* [34] to perform centralized planning on the joint action MDP formulation of the MMDP. Our implementation approximates V^* (Section 5.2.3) to evaluate leaf states [39, 44]. Unlike DOLUCT and DMCTS, where all agents perform independent local planning in parallel, DICE is not parallelizable.

5.2.2 Policy and Value Function Approximation. We used deep neural networks $\hat{\pi}_\phi$, $\hat{Q}_\omega$, $\hat{V}_\rho$, and f_θ with weights ϕ, ω, ρ, and θ to implement different MARL algorithms (Section 5.2.3). All networks receive the *global state* s_t and the *individual state* information s_t^i of agent i as input, which we refer to as $s_{t,i} = \langle s_t, s_t^i \rangle$ to omit the index i of the function approximators. An experience buffer E was implemented to store the last 10,000 transitions and to sample minibatches of size 64 to perform stochastic gradient descent (SGD) using ADAM with a learning rate of 0.001. E was initialized with 5,000 experience samples generated with $DOLUCT_{Baseline}$ ($n_b = 512$). We set $\gamma = 0.95$. All value-based approaches use a *target network*, where a neural network copy with parameters ω^- or ρ^- is used to generate the TD targets [32]. The target network is updated every 5,000 SGD steps with the trained parameters ω or ρ resp.

The global and individual state can be represented as multi-channel image for both domains. The global state input s_t is a tensor of size $8 \times 8 \times 3$ (PE) or $5 \times 5 \times 35$ (SF). The individual state input s_t^i is a tensor of size $8 \times 8 \times 3$ (PE) or $5 \times 5 \times 3$ (SF). The feature planes, used for s_t and s_t^i, are described in Table 1 and 2 resp.

Both input streams are processed by separate residual towers of a convolutional layer with 128 filters of size 5×5 with stride 1, batch normalization and two subsequent residual blocks. Each residual block consists of two convolutional layers with 128 filters of size 3×3 with stride 1 and batch normalization. The input of each residual block is added to the corresponding normalized output. The concatenated output of both residual towers is processed by a fully connected layer with 256 units. All hidden layers use ReLU activation. The residual network architecture was inspired by [44]. The units and activation of the output layer depend on the approximated function and are listed in Table 3. f_θ has the same outputs as $\hat{\pi}_\phi$ and $\hat{V}_\rho$ but combined into a single neural network.

5.2.3 Multi-Agent Reinforcement Learning. We implemented different MARL algorithms with deep neural networks (Section 5.2.2). We performed *centralized learning*, where a single neural network is trained for all agents, but *decentralized execution*, where the trained neural network is deployed on each agent for testing [14, 19].

- *Strong Emergent Policy approximation (STEP)* is learned as explained in Section 4.3 and Algorithm 2, where f_θ minimizes L_{STEP} according to Eq. 3 for each $e_t \in E$.
- *Distributed V-Learning (DVL)* approximates V^* of the optimal joint policy π^* with $\hat{V}_\rho$ by minimizing $L_{DVL,i} = (r_t + \gamma \hat{V}_{\rho^-}(s_{t+1,i}) - \hat{V}_\rho(s_{t,i}))^2$ for each $e_t \in E$ [39, 46, 47].
- *Distributed Q-Learning (DQL)* approximates Q_i^* with $\hat{Q}_\omega$ by minimizing $L_{DQL,i} = (r_t + \gamma\, max_{a_{t+1,i}}(\hat{Q}_{\omega^-}(s_{t+1,i}, a_{t+1,i})) - \hat{Q}_\omega(s_{t,i}, a_{t,i}))^2$ for each $e_t \in E$ [29, 32, 50, 51, 54].
- *Distributed Actor-Critic (DAC)* approximates π_i^* with $\hat{\pi}_\phi$ by minimizing $L_{DAC,i} = -log(\hat{\pi}_\phi(a_{t,i}|s_{t,i}))(r_t + \gamma \hat{V}_\rho(s_{t+1,i}) - \hat{V}_\rho(s_{t,i}))$ [16, 26, 48], where $\hat{V}_\rho$ is trained with DVL [3].

5.3 Results

We tested all approaches on settings with $N = 2, 4, 6$ agents for PE and $N = 4, 8, 12$ agents for SF. An *episode* is reset after 50 time steps, when all evaders are captured (PE), or when all items are complete (SF). A *run* consists of 500 episodes and is repeated 30 times. As a no-learning algorithm, $DOLUCT_{Baseline}$ ($n_b = 512$) was run 100 times to determine its average performance for comparison.

The performance for PE is evaluated with the evader *capture rate* $R_{capture} = \frac{N_{captured}}{N}$, where $N_{captured}$ is the number of captured evaders and N is the total number of evaders.

The performance for SF is evaluated with the value of $score_{50}$ and the item *completion rate* $R_{completion} = \frac{|\mathcal{D}_{complete}|}{N}$.

5.3.1 STEP Training. We trained STEP with DOLUCT and DMCTS and compared them with other planning approaches (Section 5.2.1). For all decentralized algorithms, we set $n_b = 128$ and $c = 1$. DICE has a computation budget of $n_b = 128N$, proportional to the number of agents N, and a planning horizon of $h = 4$ [4].

The results are shown in Fig. 4 and Table 4. $DOLUCT_{STEP}$ converges fastest in all cases, except in PE with $N = 6$, and clearly outperforms all other approaches in SF. $DMCTS_{STEP}$ also displays progress but converges much slower than $DOLUCT_{STEP}$ in all SF settings. DICE only displays significant progress in SF but becomes more competitive with increasing N. $DOLUCT_{Random}$ fails to find meaningful policies in PE but slowly improves in SF.

5.3.2 STEP Test. We evaluated the STEP approximation f_θ of each DOLUCT run from Section 5.3.1 with 100 randomly generated test episodes to determine the average performance after every tenth training episode. The performance was compared with $\hat{Q}_\omega$ and $\hat{\pi}_\phi$ which were trained using DQL and DAC resp. (Section 5.2.3). We also implemented versions of DQL and DAC, which we call DQL_{local} and DAC_{local} resp., that learn with local rewards as described in Section 5.1 for easier multi-agent credit assignment to provide stronger baselines than DQL_{global} and DAC_{global} which were trained with the original global rewards. In addition, we provide the results of STEP policies that were trained with DOLUCT using computation budgets of $n_b = 64, 256$.

The results are shown in Fig. 5 and Table 4. STEP learns the strongest policies in all settings, except in PE with $N = 6$. DQL_{global} and DAC_{global} are unable to learn meaningful policies in SF, but DAC_{global} shows progress in PE with increasing N. DQL_{local} and DAC_{local} always outperform their global reward optimizing counterparts but are inferior to STEP, except in PE with $N = 6$.

5.4 Discussion

Our experiments show the effectiveness of STEP in two domains, which are challenging in different aspects. PE has a *sparse reward* structure and becomes *less challenging* when the number of pursuers is large because the pursuers can distribute more effectively across the map to capture evaders. This can be seen in Fig. 4a-c and Table 4, where $DOLUCT_{Baseline}$ performs better with more agents.

In contrast, SF has a *dense reward* structure and becomes *more challenging* when the number of agents is large due to more potential conflicts at simultaneously required machines as indicated by the performance of $DOLUCT_{Baseline}$ in Table 4, where it becomes increasingly difficult to find coordinated local policies due to the enormous search space. In PE, two pursuers need to occupy the same cell to capture an evader as a *joint task*, while in SF multiple agents should *avoid conflicts* by not enqueuing at the same machine.

Results from Section 5.3.1 show that DOLUCT and DMCTS are able to improve with STEP, but DOLUCT is more suited when the problem is too large to provide sufficient computation budget (Section 4.2), offering scalable performance in all domains as shown in Fig. 4 and Table 4. $DOLUCT_{STEP}$ also outperforms the centralized DICE, which directly optimizes the joint policy but requires more total computation budget to be competitive against DOLUCT.

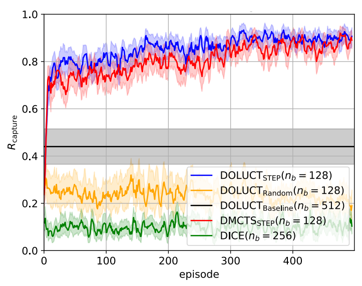
Results from Section 5.3.2 show that STEP is able to learn strong decentralized policies, which can be reintegrated into the planning process to further improve and coordinate decentralized multi-agent planning in contrast to planning with just a random policy (Fig. 4 and Table 4). In fact, $DOLUCT_{Random}$ performs very poorly in PE, where it is important to accurately predict other agents' actions to coordinate on the joint task of capturing evaders (Section 4.1).

Increasing the computation budget n_b tends to slightly improve the quality of STEP, which might be due to the decreasing bias when planning with the currently learned f_θ as explained in Section 4.5.

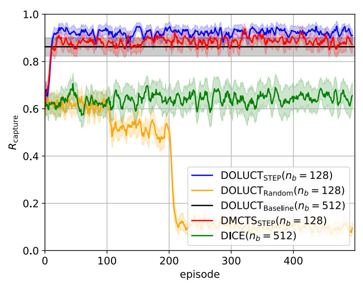
[3] $\hat{\pi}_\phi$ and $\hat{V}_\rho$ are trained separately, because it is more stable than training f_θ for DAC.
[4] We experimented with $h = 2, 4, 8$, but 4 seemed to be the best choice for DICE.

Table 1: Description of all feature planes as global state input s_t, which are adopted from [19] (PE) and [39] (SF) resp.

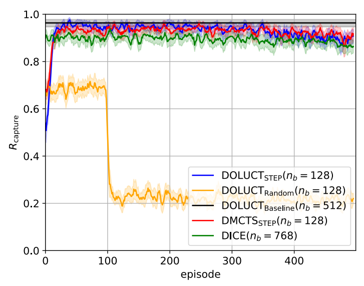
Domain	Feature	# Planes	Description
Pursuit & Evasion	Pursuer position	1	The position of each pursuer in the grid (e.g., red circles in Fig. 3a).
	Evader position	1	The position of each evader in the grid (e.g., blue circles in Fig. 3a).
	Obstacle position	1	The position of each obstacle in the grid (e.g., gray rectangles in Fig. 3a).
Smart Factory	Machine type	1	The type of each machine as a value between 0 and 14 (Fig. 3b)
	Agent state	4	The number of agents standing at machines whose types are (not) contained in their current bucket of tasks and whether they are enqueued or not.
	Tasks (1^{st} bucket)	15	Spatial distribution of agents with a matching task in their first bucket for each machine type.
	Tasks (2^{nd} bucket)	15	Same as "Tasks (1^{st} bucket)" but for the second bucket.



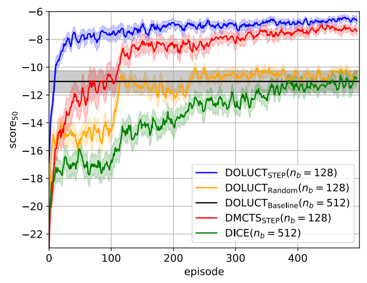
(a) Pursuit & Evasion (2 agents)



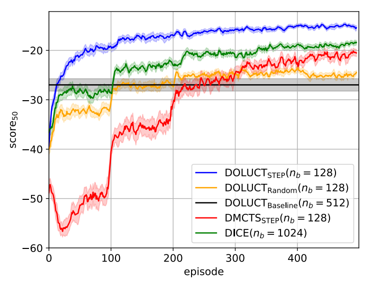
(b) Pursuit & Evasion (4 agents)



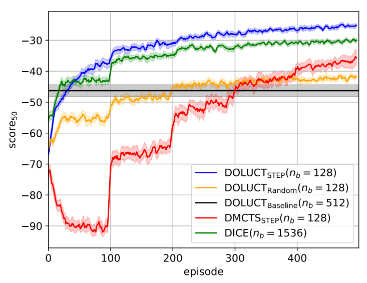
(c) Pursuit & Evasion (6 agents)



(d) Smart Factory (4 agents)



(e) Smart Factory (8 agents)



(f) Smart Factory (12 agents)

Figure 4: Average training progress of $R_{capture}$ (Pursuit & Evasion) and $score_{50}$ (Smart Factory) of 30 runs shown as running mean over 5 episodes for different multi-agent planning algorithms. Shaded areas show the 95 % confidence interval.

Table 2: Feature planes as individual state input s_t^i.

Plane	Pursuit & Evasion	Smart Factory
1	Agent i's position	Agent i's position
2	Evader positions	Machine positions (1^{st} bucket)
3	Obstacle positions	Machine positions (2^{nd} bucket)

Table 3: Output of the neural networks.

Neural Network	Layer Type	# units	activation
$\hat{\pi}_\phi$	fully connected	$\lvert\mathcal{A}_i\rvert$	softmax
$\hat{Q}_\omega$	fully connected	$\lvert\mathcal{A}_i\rvert$	linear
$\hat{V}_\rho$	fully connected	1	linear
f_θ	same as $\hat{\pi}_\phi$ and $\hat{V}_\rho$ combined into a single network		

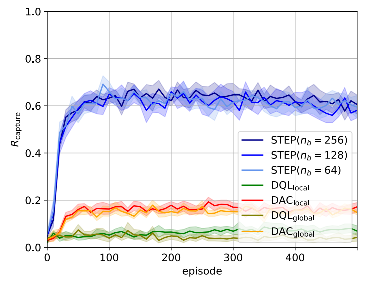
In SF (Fig. 5d-f), DQL_{global} and DAC_{global} are unable to adapt adequately due to much noise in the gradients caused by the global reward density. DAC_{global} displays exploding gradients here, which leads to premature convergence towards a poor policy. However, DAC_{global} performs better with increasing N in PE. This might be due to the sparse rewards which make the updates less noisy and the decreasing difficulty of capturing evaders, when N is large. DQL_{local} and DAC_{local} always outperform their global reward optimizing counterparts, since their individual objectives are easier to optimize. Still, they are generally inferior to STEP because they are unable to consider emergent dependencies in the future like strategic positioning (in PE) or potential conflicts (in SF). Unlike, these approaches, STEP is trained with decentralized multi-agent planning, which explicitly reasons about these dependencies.

6 CONCLUSION AND FUTURE WORK

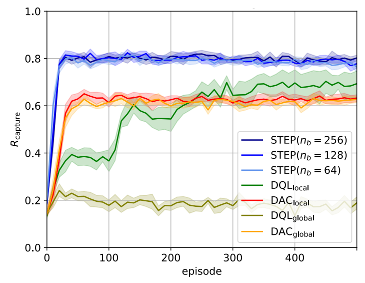
In this paper, we proposed STEP, a scalable approach to learn strong decentralized policies for cooperative MAS with a distributed variant of policy iteration. For that, we use function approximation to

Table 4: Average capture rate $R_{capture}$ (Pursuit & Evasion) and completion rate $R_{completion}$ (Smart Factory) at the end of the 500^{th} episode of all experiments within a 95% confidence interval.

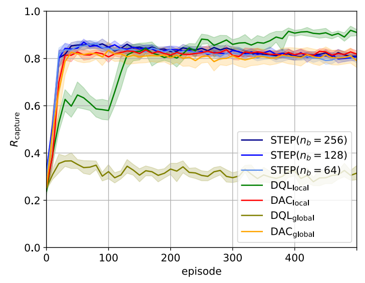
	Pursuit & Evasion ($R_{capture}$)			Smart Factory ($R_{completion}$)		
# agents N	2	4	6	4	8	12
# states $\|\mathcal{S}\|$	$\approx 4.1 \cdot 10^6$	$\approx 1.7 \cdot 10^{13}$	$\approx 6.9 \cdot 10^{19}$	$\approx 6.9 \cdot 10^{53}$	$\approx 4.8 \cdot 10^{107}$	$\approx 3.3 \cdot 10^{161}$
# joint actions $\|\mathcal{A}\| = \|\mathcal{A}_i\|^N$	$5^2 = 25$	$5^4 = 625$	$5^6 = 15,625$	$6^4 = 1,296$	$6^8 \approx 1.7 \cdot 10^6$	$6^{12} \approx 2.2 \cdot 10^9$
$DOLUCT_{STEP}$ ($n_b = 128$)	88.3 ± 8.3%	**93.3 ± 5.0%**	88.9 ± 6.7%	**100 ± 0.0%**	**99.6 ± 0.8%**	**99.2 ± 1.1%**
$DOLUCT_{Random}$ ($n_b = 128$)	23.3 ± 10.0%	9.2 ± 5.8%	22.2 ± 5.6%	96.7 ± 3.3%	92.5 ± 2.9%	88.3 ± 1.9%
$DOLUCT_{Baseline}$ ($n_b = 512$)	44.5 ± 7.5%	86.3 ± 4.0%	**96.3 ± 1.8%**	90.5 ± 3.0%	83.3 ± 2.6%	76.2 ± 3.0%
$DMCTS_{STEP}$ ($n_b = 128$)	**90.0 ± 10.0%**	90.0 ± 6.7%	91.7 ± 5.6%	98.3 ± 2.5%	95.8 ± 2.1%	85.9 ± 4.0%
$DICE$ ($n_b = 128N$)	8.3 ± 6.7%	58.3 ± 9.2%	87.2 ± 3.9%	97.5 ± 3.3%	98.8 ± 1.7%	98.3 ± 1.7%
$STEP$ ($n_b = 256$)	**60.5 ± 4.1%**	**80.3 ± 1.2%**	81.0 ± 0.7%	**96.9 ± 0.9%**	**97.3 ± 0.7%**	92.5 ± 4.7%
$STEP$ ($n_b = 128$)	58.1 ± 3.2%	77.8 ± 1.9%	80.8 ± 1.6%	95.1 ± 0.9%	96.4 ± 0.7%	**95.5 ± 1.4%**
$STEP$ ($n_b = 64$)	59.5 ± 4.8%	76.4 ± 2.3%	79.4 ± 1.2%	91.2 ± 0.4%	94.7 ± 1.2%	89.2 ± 3.9%
DQL_{local}	6.9 ± 1.8%	69.4 ± 5.8%	**91.0 ± 1.4%**	72.8 ± 6.5%	72.5 ± 3.2%	70.0 ± 7.6%
DQL_{global}	4.3 ± 1.6%	19.0 ± 2.6%	31.5 ± 2.9%	16.7 ± 9.0%	0.6 ± 0.5%	0.0 ± 0.0%
DAC_{local}	17.2 ± 2.1%	63.3 ± 1.6%	81.9 ± 1.2%	47.8 ± 20.7%	49.0 ± 21.1%	47.9 ± 24.2%
DAC_{global}	14.8 ± 1.0%	62.5 ± 1.2%	77.9 ± 2.4%	0.0 ± 0.0%	0.0 ± 0.0%	0.0 ± 0.0%



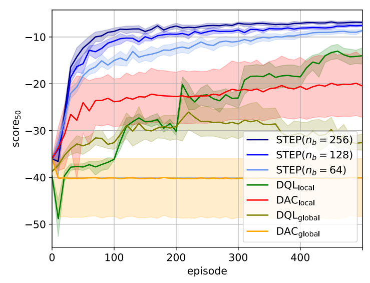
(a) Pursuit & Evasion (2 agents)



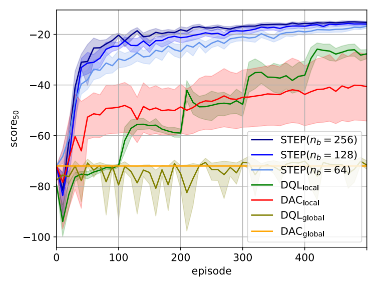
(b) Pursuit & Evasion (4 agents)



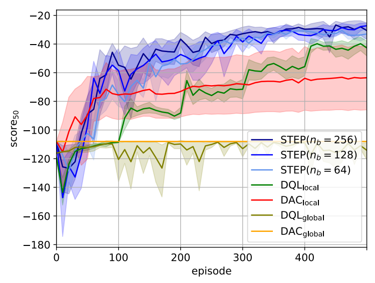
(c) Pursuit & Evasion (6 agents)



(d) Smart Factory (4 agents)



(e) Smart Factory (8 agents)



(f) Smart Factory (12 agents)

Figure 5: Average test progress of $R_{capture}$ (Pursuit & Evasion) and $score_{50}$ (Smart Factory) of 30 runs of the neural networks f_θ, $\hat{Q}_\omega$, and $\hat{\pi}_\phi$ trained with STEP (using DOLUCT), DQL, and DAC resp. Shaded areas show the 95 % confidence interval.

learn from action recommendations of a decentralized multi-agent planner. STEP combines decentralized multi-agent planning with centralized learning, where each agent is able to explicitly reason about emergent dependencies to make coordinated decisions, only requiring a generative model for distributed black box optimization without additional domain knowledge or reward decomposition.

We experimentally evaluated STEP in two challenging and stochastic domains with large state and joint action spaces. We demonstrated that multi-agent open-loop planning is especially suited for efficient training, when the problem is too large to provide sufficient computation budget for planning. Our experiments show that STEP is able to learn stronger decentralized policies than standard MARL algorithms, without any domain or reward decomposition. The policies learned with STEP are able to effectively coordinate on joint tasks and to avoid conflicts, thus can be reintegrated into the multi-agent planning process to further improve performance.

For the future, we plan to address partially observable domains by combining multi-agent planning with deep recurrent reinforcement learning for cooperative MAS [1, 13, 14, 16, 19].

REFERENCES

[1] Christopher Amato and Frans A Oliehoek. 2015. Scalable Planning and Learning for Multiagent POMDPs. In *29th AAAI Conference on Artificial Intelligence*.

[2] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking Fast and Slow with Deep Learning and Tree Search. In *Advances in Neural Information Processing Systems*.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine learning* 47, 2-3 (2002), 235–256.

[4] Daniel S Bernstein, Christopher Amato, Eric A Hansen, and Shlomo Zilberstein. 2009. Policy Iteration for Decentralized Control of Markov Decision Processes. *Journal of Artificial Intelligence Research* 34 (2009), 89–132.

[5] Daniel S Bernstein, Eric A Hansen, and Shlomo Zilberstein. 2005. Bounded Policy Iteration for Decentralized POMDPs. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI)*. 52–57.

[6] Craig Boutilier. 1996. 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.

[7] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. 2010. Multi-Agent Reinforcement Learning: An Overview. In *Innovations in multi-agent systems and applications-1*. Springer.

[8] Yu-Han Chang, Tracey Ho, and Leslie P Kaelbling. 2004. All learning is local: Multi-agent learning in global reward games. In *Advances in neural information processing systems*. 807–814.

[9] Daniel Claes, Frans Oliehoek, Hendrik Baier, and Karl Tuyls. 2017. Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS.

[10] Daniel Claes, Philipp Robbel, Frans A Oliehoek, Karl Tuyls, Daniel Hennes, and Wiebe Van der Hoek. 2015. Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS.

[11] Sam Devlin and Daniel Kudenko. 2011. Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. IFAAMAS, 225–232.

[12] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. 2014. Potential-based Difference Rewards for Multiagent Reinforcement Learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. IFAAMAS, 165–172.

[13] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. 2004. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proc. of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*. IEEE Computer Society.

[14] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

[15] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. 2018. Learning with Opponent-Learning Awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. IFAAMAS, 122–130.

[16] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual Multi-Agent Policy Gradients. *32th AAAI Conference on Artificial Intelligence* (2018).

[17] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. 2017. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*.

[18] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. 2014. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In *Advances in Neural Information Processing Systems*.

[19] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative Multi-Agent Control using Deep Reinforcement Learning. In *International Conference on Autonomous Agents and Multiagent Systems*. Springer.

[20] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the 19th national conference on Artifical intelligence*. AAAI Press, 709–715.

[21] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. 2016. Opponent Modeling in Deep Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*. 1804–1813.

[22] Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, and Chun-Yi Lee. 2018. A Deep Policy Inference Q-Network for Multi-Agent Systems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. IFAAMAS, 1388–1396.

[23] Ronald A. Howard. 1961. *Dynamic Programming and Markov Processes*. The MIT Press.

[24] Daniel Jiang, Emmanuel Ekwedike, and Han Liu. 2018. Feedback-Based Tree Search for Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR.

[25] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo Planning. In *European conference on machine learning*. Springer.

[26] Vijay R Konda and John N Tsitsiklis. 2000. Actor-Critic Algorithms. In *Advances in neural information processing systems*. 1008–1014.

[27] Guillaume J Laurent, Laëtitia Matignon, Le Fort-Piat, et al. 2011. The world of Independent Learners is not Markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems* 15, 1 (2011), 55–64.

[28] Erwan Lecarpentier, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson. 2018. Open Loop Execution of Tree-Search Algorithms. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI Organization, 2362–2368. https://doi.org/10.24963/ijcai.2018/327

[29] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. IFAAMAS, 464–473.

[30] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. 2018. Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:1802.06444* (2018).

[31] Laëtitia Matignon, Guillaume Laurent, and Nadine Le Fort-Piat. 2007. Hysteretic Q-Learning: An Algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'07*. 64–69.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature* 518, 7540 (2015).

[33] Frans A Oliehoek and Christopher Amato. 2016. *A Concise Introduction to Decentralized POMDPs*. Springer.

[34] Frans A Oliehoek, Julian FP Kooij, and Nikos Vlassis. 2008. The Cross-Entropy Method for Policy Search in Decentralized POMDPs. *Informatica* 32, 4 (2008), 341–357.

[35] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. 2017. Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability. In *International Conference on Machine Learning*.

[36] Liviu Panait and Sean Luke. 2005. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous agents and multi-agent systems* 11, 3 (2005), 387–434.

[37] Liviu Panait, Keith Sullivan, and Sean Luke. 2006. Lenient Learners in Cooperative Multiagent Systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 801–803.

[38] Diego Perez Liebana, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas. 2015. Open Loop Search for General Video Game Playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM.

[39] Thomy Phan, Lenz Belzner, Thomas Gabor, and Kyrill Schmid. 2018. Leveraging Statistical Multi-Agent Online Planning with Emergent Value Function Approximation. In *Proceedings of the 17th Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS.

[40] Martin L Puterman. 2014. *Markov Decision Processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[41] Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. 2018. Machine Theory of Mind. In *Proceedings of the 35th International Conference on Machine Learning*. 4218–4227.

[42] Sven Seuken and Shlomo Zilberstein. 2007. Memory-Bounded Dynamic Programming for DEC-POMDPs.. In *Proceedings of the 20th international joint conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 2009–2015.

[43] Yoav Shoham and Moshe Tennenholtz. 1995. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial intelligence* 73, 1-2 (1995), 231–252.

[44] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550, 7676 (2017).

[45] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2017. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv preprint arXiv:1706.05296* (2017).

[46] Richard S Sutton. 1988. Learning to Predict by the Methods of Temporal Differences. *Machine learning* 3, 1 (1988).

[47] Richard S Sutton and Andrew G Barto. 1998. *Introduction to Reinforcement Learning*. Vol. 135. MIT Press Cambridge.

[48] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in neural information processing systems*. 1057–1063.

[49] Daniel Szer and François Charpillet. 2006. Point-based Dynamic Programming for DEC-POMDPs. In *21st National Conference on Artificial Intelligence-AAAI'2006*.

[50] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent Cooperation and Competition with Deep Reinforcement Learning. *PloS one* 12, 4 (2017).

[51] Ming Tan. 1993. Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents. In *Proceedings of the 10th International Conference on International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc.
[52] Pradeep Varakantham, Shih-Fen Cheng, Geoff Gordon, and Asrar Ahmed. 2012. Decision Support for Agent Populations in Uncertain and Congested Environments. In *26th AAAI Conference on Artificial Intelligence*.
[53] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. 2002. Probabilistic Pursuit-Evasion Games: Theory, Implementation, and Experimental Evaluation. *IEEE transactions on robotics and automation* 18, 5 (2002).
[54] Christopher JCH Watkins and Peter Dayan. 1992. Q-Learning. *Machine learning* 8, 3-4 (1992), 279–292.
[55] Ari Weinstein and Michael L Littman. 2013. Open-Loop Planning in Large-Scale Stochastic Domains. In *27th AAAI Conference on Artificial Intelligence*.
[56] David H Wolpert and Kagan Tumer. 2002. Optimal Payoff Functions for Members of Collectives. In *Modeling complexity in economic and social systems*. World Scientific, 355–369.
[57] Feng Wu, Nicholas R Jennings, and Xiaoping Chen. 2012. Sample-based Policy Iteration for Constrained DEC-POMDPs. In *Proceedings of the 20th European Conference on Artificial Intelligence*. IOS Press, 858–863.
[58] Chongjie Zhang and Victor Lesser. 2010. Multi-Agent Learning with Policy Prediction. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press, 927–934.
[59] Martin Zinkevich and Tucker R. Balch. 2001. Symmetry in Markov Decision Processes and its Implications for Single Agent and Multiagent Learning. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 632–.