Leveraging Statistical Multi-Agent Online Planning with Emergent Value Function Approximation

Thomy Phan, Lenz Belzner, Thomas Gabor and Kyrill Schmid
Institute of Informatics
LMU Munich
Munich, Germany
{thomy.phan,belzner,thomas.gabor,kyrill.schmid}@ifi.lmu.de

ABSTRACT

Making decisions is a great challenge in distributed autonomous environments due to enormous state spaces and uncertainty. Many online planning algorithms rely on statistical sampling to avoid searching the whole state space, while still being able to make acceptable decisions. However, planning often has to be performed under strict computational constraints making online planning in multi-agent systems highly limited, which could lead to poor system performance, especially in stochastic domains.

In this paper, we propose *Emergent Value function Approximation for Distributed Environments (EVADE)*, an approach to integrate global experience into multi-agent online planning in stochastic domains to consider global effects during local planning. For this purpose, a value function is approximated online based on the emergent system behaviour by using methods of reinforcement learning.

We empirically evaluated EVADE with two statistical multi-agent online planning algorithms in a highly complex and stochastic smart factory environment, where multiple agents need to process various items at a shared set of machines. Our experiments show that EVADE can effectively improve the performance of multi-agent online planning while offering efficiency w.r.t. the breadth and depth of the planning process.

KEYWORDS

multi-agent planning; online planning; value function approximation

ACM Reference Format:
Thomy Phan, Lenz Belzner, Thomas Gabor and Kyrill Schmid. 2018. Leveraging Statistical Multi-Agent Online Planning with Emergent Value Function Approximation. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Decision making in complex and stochastic domains has been a major challenge in artificial intelligence for many decades due to intractable state spaces and uncertainty. Statistical approaches based on Monte-Carlo methods have become popular for planning under uncertainty by guiding the search for policies to more promising regions in the search space [1, 7, 14, 24, 36, 38, 47]. These methods can be combined with online planning to adapt to unexpected changes in the environment by interleaving planning and execution of actions [1, 7, 14, 36, 38].

However, online planning often has to meet strict real-time constraints limiting the planning process to local search. This makes the consideration of possible global effects difficult, which could lead to suboptimal policies, especially in stochastic domains. The problem is further intensified in multi-agent systems (MAS), where the search space grows exponentially w.r.t. the dimension and the number of agents, which is known as the *curse of dimensionality* [1, 8, 29]. Furthermore, one has to cope with the coordination of individual actions of all agents to avoid potential conflicts or suboptimal behaviour [8, 10].

Many multi-agent planning approaches assume the availability of a pre-computed value function of a more simplified model of the actual environment to consider possible global effects in the local planning process, which can be exploited to prune the search space or to further refine the policy [17, 31, 40, 43]. This might be insufficient for highly complex and uncertain domains, where the dynamics cannot be sufficiently specified beforehand [7]. Depending on the domain complexity, pre-computing such a value function might be even computationally infeasible [8, 38]. Thus, an adaptive and model-free approach is desirable for learning a value function at system runtime in MAS.

Recently, approaches to combine online planning and reinforcement learning (RL) have become popular to play games with high complexity like Go and Hex [2, 36, 37]. A tree search algorithm is used for planning, which is guided by a value function approximated with RL. These approaches were shown to outperform plain planning and RL, even achieving super-human level performance in Go without any prior knowledge about the game beyond its rules [37]. So far, these approaches have only been applied to deterministic domains with only one agent.

In this paper, we propose *Emergent Value function Approximation for Distributed Environments (EVADE)*, an approach to integrate global experience into multi-agent online planning in stochastic domains. For this purpose, a value function is approximated online based on the emergent system behaviour by using methods of RL. With that value function, global effects can be considered during local planning to improve the performance and efficiency of existing multi-agent online planning algorithms.

We also introduce a smart factory environment, where multiple agents need to process various items with different tasks at a shared set of machines in an automated and self-organizing way. Given a sufficient number of agents and stochasticity w.r.t. agent behaviour and the outcome of actions, we show that our environment has a significantly higher branching factor than the game of Go [36].

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

We empirically evaluate the effectiveness of EVADE in this stochastic and complex domain based on two existing multi-agent planning algorithms [5, 30].

The rest of the paper is organized as follows. Section 2 provides some background about decision making in general. Section 3 discusses related work. Section 4 describes EVADE for enhancing multi-agent planning algorithms. Section 5 presents and discusses experimental results achieved by two statistical multi-agent planning algorithms enhanced with EVADE in our smart factory environment. Finally, section 6 concludes and outlines a possible direction for future work.

2 BACKGROUND

2.1 Markov Decision Processes

We formulate our problem as *multi-agent Markov Decision Process (MMDP)* assuming a fully cooperative setting, where all agents share the same common goal [8, 29]. For simplicity, this work only focuses on fully observable problems as modeled in [8, 14, 44].

Although more realistic models exist for describing large-scale MAS like Dec-MDPs or Dec-POMDPs [29], the focus of this work is just to evaluate the possible performance and efficiency gain based on integrating global experience into the multi-agent online planning process. An extension of our approach to partially observable models is left for future work.

2.1.1 MDP. Decision-making problems with discrete time steps and a single agent can be formulated as *Markov Decision Process (MDP)* [8, 22, 34]. An 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 $\mathcal{R}(s_t, a_t)$ is the scalar reward function. In this work, it is always assumed that $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, $r_t = \mathcal{R}(s_t, a_t)$, where s_{t+1} is reached after executing a_t in s_t at time step t. Π is the policy space and $|\Pi|$ is the number of all possible policies.

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

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

where $\gamma \in [0, 1]$ is the discount factor. If $\gamma < 1$, then present rewards are weighted more than future rewards.

A policy π can be evaluated with a *state value function* $V^\pi = \mathbb{E}_\pi[G_t|s_t]$, which is defined by the expected return at state s_t [4, 8, 22]. π is optimal if $V^\pi(s_t) \geq V^{\pi'}(s_t)$ for all $s_t \in S$ and all policies $\pi' \in \Pi$. The optimal value function, which is the value function for any optimal policy π^*, is denoted as V^* and defined by [4, 8]:

$$V^*(s_t) = max_{a_t \in \mathcal{A}} \{ r_t + \gamma \sum_{s' \in \mathcal{S}} P(s'|s_t, a_t) \cdot V^*(s') \} \tag{2}$$

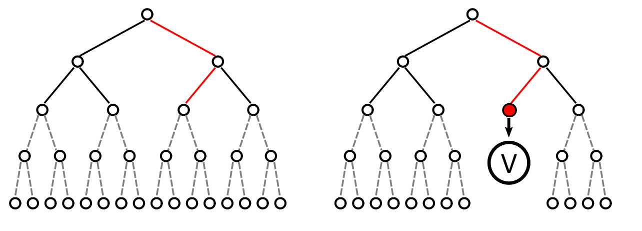
2.1.2 Multi-Agent MDP. An MMDP is defined by a tuple $M = \langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{D} = \{1, ..., n\}$ is a (finite) set of agents and $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$ is the (finite) set of joint actions. $\mathcal{S}$, $\mathcal{P}$ and $\mathcal{R}$ are defined analogously to an MDP, given joint actions instead of atomic actions [8].

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 individual policy of agent $i \in \mathcal{D}$. Given n agents in the MMDP, the number of possible joint policies is defined by $|\Pi| = \prod_{i=1}^{n} |\Pi_i|$. If all agents share the same individual policy space Π_i, then $|\Pi| = |\Pi_i|^n$.

Similarly to MDPs, a value function V^π can be used to evaluate the joint policy π.

2.2 Planning

Planning searches for a policy, given a generative model $\hat{M}$, which represents the actual environment M. $\hat{M}$ provides an approximation for $\mathcal{P}$ and $\mathcal{R}$ of the underlying MDP or MMDP [7, 8, 47]. We assume that $\hat{M}$ perfectly models the environment such that $\hat{M} = M$. *Global planning* methods search the whole state space to find π^* or V^*. An example is *value iteration*, which computes the optimal value function V^* by iteratively updating value estimates for each state according to eq. 2 [4, 8, 22]. *Local planning* methods only regard the current state and possible future states within a horizon of h to find a local policy π_{local} [7, 47]. An example for local planning is given in fig. 1a for a problem with a branching factor of two and a planning horizon of $h = 2$. The nodes in the search tree represent states and the links represent actions.



(a) local planning (b) local planning with value function

Figure 1: Illustration of local planning with a horizon of $h = 2$. The nodes in the search tree represent states and the links represent actions. The red path represents a sampled plan. The dashed gray links mark unreachable paths. (a) plain local planning. (b) local planning with a value function to consider global effects in the unreachable subtree.

In this paper, we only focus on local planning methods for *online planning*, where planning and execution of actions are performed alternately at each time step, given a fixed computation budget n_{budget} [1, 7, 14, 36, 38].

Local planning can be performed via *closed-loop* or *open-loop* search. Closed-loop search corresponds to a tree search, where a search tree is constructed and traversed guided by an action selection strategy π_{tree} [6, 33]. The nodes of the tree represent states and the links represent actions. The state values $V^{\pi_{tree}}(s_t)$ are computed recursively according to eq. 1 starting from the leaves of the search tree. *Monte Carlo Tree Search (MCTS)* is a popular closed-loop planning approach, which is applied to very large and complex domains [13, 24, 36–38]. MCTS can also be adapted to multi-agent planning [1, 14]. Open-loop planning searches for action sequences or plans of length h [6, 9, 33, 47]. These plans are typically sampled from a sequence of distributions $\Phi_1, ..., \Phi_h$ and simulated in $\hat{M}$. The resulting rewards are accumulated according to eq. 1 and used to update the distributions. Open-loop planning does not store any information about intermediate states, thus enabling efficient planning in large-scale domains [33, 47]. An approach to open-loop planning in MAS is proposed in [5].

2.3 Reinforcement Learning

Reinforcement Learning (RL) corresponds to a policy search for an unknown environment M. In general, an agent knows the state and action space $\mathcal{S}$ and $\mathcal{A}$ but it does not know the effect of executing $a_t \in \mathcal{A}$ in $s_t \in \mathcal{S}$ [8, 42]. *Model-based* RL methods learn a model $\hat{M} \approx M$ by approximating $\mathcal{P}$ and $\mathcal{R}$ [8, 20, 42]. $\hat{M}$ can be used for planning to find a policy. In this paper, we focus on *model-free* RL to approximate V^* based on experience samples $e_t = \langle s_t, a_t, s_{t+1}, r_t \rangle$ and a parametrized function approximator $\hat{V}_\theta$ with parameters θ without learning a model $\hat{M}$ [42]. A policy $\hat{\pi}$ can be derived by maximizing $\hat{V}_\theta$ such that $\hat{\pi}(s_t) = argmax_{a_t \in \mathcal{A}}(\hat{Q}_\theta(s_t, a_t))$, where $\hat{Q}_\theta(s_t, a_t) = \mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t)\hat{V}_\theta(s_{t+1})$ is the approximated *action value function* [8, 42]. The experience samples are obtained from interaction between the agent and the environment.

3 RELATED WORK

Online Planning with Evaluation Heuristics. Some work on online planning in partially observable domains provide an evaluation heuristic to compensate for highly limited planning horizons [16, 32, 35, 39]. Those heuristics are usually based on domain knowledge or computed in an offline phase, where a value function V_{MDP} is constructed based on a fully observable model of the actual environment by using variants of value iteration. V_{MDP} can be used to enhance online planning to search for a policy under the consideration of possible global effects. It was shown that V_{MDP} provides an upper bound to V^* of the actual environment [11, 31].

This can be exploited to prune the search space without loosing optimality of the solutions found. Many multi-agent planning algorithms use similar methods to enhance planning with such a pre-computed value function V_{MDP} [17, 31, 40, 43].

In our approach, V^* is approximated *online* based on *actual* experience without requiring a model. A generative model is only used for online planning to find a joint policy. We intend to apply our approach to highly complex and stochastic domains, where an offline computation is not feasible, since any change in the model would require the re-computation of V_{MDP}.

Online Planning and Deep RL. *AlphaGo* is a program introduced in [36], which is able to play Go at a super-human level. It recently defeated the currently best human Go players in various tournaments [36, 37]. AlphaGo uses MCTS for online planning and deep neural networks, which approximate π^* and V^* to guide the tree search. With this approach, AlphaGo is able to develop extremely complex strategies within given time constraints.

MCTS-based planning combined with an approximation of V^* was shown to improve the performance of plain online planning or RL in complex and deterministic games like Go and Hex [2, 36, 37]. The idea of these approaches is based on the human mind, which is able to think ahead into the future, while guiding the thoughts with intuition learned from experience. In the context of artificial intelligence, online planning represents the future thinking, while deep RL represents the integration of strong intuition [2, 18, 23].

Our approach extends these ideas to environments with *multiple* agents. We also focus on *stochastic* domains, where the outcome of actions and the behaviour of agents are not deterministic.

Distributed Value Function Approximation. In this paper, we focus on centralized learning of V^*, where all agents share the same parameters θ similarly to [19, 45]. Unlike previous work on multi-agent RL, we do not use the approximated value function to directly derive a policy. Instead, we use it to guide online planning in MAS.

Besides, there exist approaches to approximate the value function asynchronously and in parallel [25, 28]. In that case, multiple agents act independently of each other in different instances of the same domain. They share experience with each other in order to update the same value function approximation $\hat{V}_\theta$ in parallel to accelerate the learning process.

Our approach approximates V^* based on the global experience of multiple agents, which act in the *same* environment. Our approximation $\hat{V}_\theta$ is not meant to improve the performance of individual agents but to improve the behaviour of the MAS as a *whole*.

4 EVADE

We now describe *Emergent Value function Approximation for Distributed Environments (EVADE)* for leveraging statistical multi-agent online planning with a value function, which is approximated online at system runtime. EVADE is a framework for combining multi-agent online planning and RL to further improve the performance in MAS.

4.1 Combining Online Planning and RL

Given a perfect generative model $\hat{M} = M$, online planning can be used for decision making with high quality and accuracy w.r.t. the expected return. However, due to computational constraints, online planning is unable to make lookaheads for arbitrarily long horizons, which would be required for highly complex tasks that require much more time steps to solve than the actually feasible horizon as sketched in fig. 1a. In contrast, model-free RL with a parametrized function approximator $\hat{V}_\theta$ allows for potentially infinite future prediction but has approximation erros due to the compressing nature of $\hat{V}_\theta$.

By combining online planning and RL, a decision maker can benefit from both advantages [2, 35–37]. The limited lookahead of planning can be enhanced with $\hat{V}_\theta$ as shown in fig. 1b. Online planning can plan accurately for h initial time steps, which are weighted more than the outcome estimate $\hat{V}_\theta(s_{t+h})$, given a discount factor of $\gamma < 1$. The discount can also neglect possible approximation errors of $\hat{V}_\theta$. Especially in highly complex and stochastic domains with multiple agents, we believe that the integration of a value function approximation could improve the performance of otherwise limited multi-agent online planning.

4.2 Multi-Agent Planning with Experience

We focus on online settings, where there is an alternating *planning* and *learning* step for each time step t. In the planning step, the system searches for a joint policy π_{local}, which maximizes $G_{t,EVADE}$:

$$G_{t,EVADE} = G_t + \gamma^h \hat{V}_\theta(s_{t+h}) \quad (3)$$

$G_{t,EVADE}$ extends G_t from eq. 1 with $\hat{V}_\theta(s_{t+h})$ as the provided global outcome estimate to enhance local planning with a limited horizon of h as sketched in fig. 1b. The planning step can be implemented with an arbitrary multi-agent planning algorithm, depending on the concrete problem.

After the planning step, all agents execute the joint action $a_t = \pi_{local}(s_t)$ causing a state transition from s_t to s_{t+1} with a reward signal r_t. This emergent result is stored as experience sample $e_t = \langle s_t, a_t, s_{t+1}, r_t \rangle$ in an experience buffer D. A sequence of experience samples $e_1, ..., e_T$ is called *episode* of length T.

In the subsequent learning step, a parametrized function approximator $\hat{V}_\theta$ is used to minimize the one-step temporal difference (TD) error of all samples e_t in D w.r.t. θ. The TD error for e_t is defined by [41, 42]:

$$\delta_t = \hat{V}_\theta(s_t) - (r_t + \gamma \hat{V}_\theta(s_{t+1})) \quad (4)$$

It should be noted that the approximation only depends on the experience samples $e_t \in D$ and does not require a model like hybrid planning approaches explained in section 3. The updated value function $\hat{V}_\theta$ can then be used for the next planning step at $t + 1$.

The complete formulation of multi-agent online planning with EVADE is given in algorithm 1, where T is the length of an episode, $\hat{M}$ is the generative model used for planning, n is the number of agents in the MAS, h is the planning horizon, n_{budget} is the computation budget and $\hat{V}_\theta$ is the value function approximator. The parameter *MASPlan* can be an arbitrary multi-agent planning algorithm for searching a joint policy π_{local} by maximizing $G_{t,EVADE}$. Given that the computation budget n_{budget} is fixed and the time to update $\hat{V}_\theta$ at each time step is constant[1], EVADE is suitable for online planning and learning in real-time MAS.

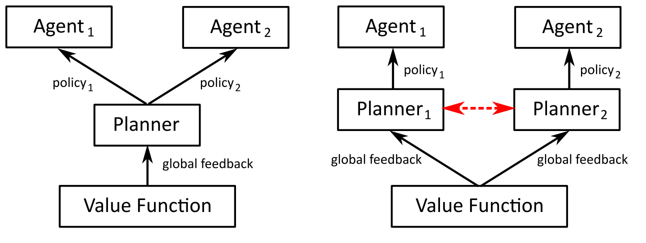
Algorithm 1 Multi-agent online planning with EVADE

```
1: procedure $EVADE(MASPlan, \hat{M}, n, h, n_{BUDGET}, \hat{V}_\theta)$
2:     Initialize $\theta$ of $\hat{V}_\theta$
3:     Observe $s_1$
4:     for $t = 1, T$ do
5:         Find $\pi_{local}$ using $MASPlan(s_t, \hat{M}, n, h, n_{budget}, \hat{V}_\theta)$
6:         Execute $a_t = \pi_{local}(s_t)$
7:         Observe reward $r_t$ and new state $s_{t+1}$
8:         Store new experience $e_t = \langle s_t, a_t, s_{t+1}, r_t \rangle$ in $D$
9:         Refine $\theta$ to minimize the TD error $\delta_t$ for all $e_t \in D$
```

4.3 Architecture

We focus on centralized learning, since we believe that V^* can be approximated faster if all agents share the same parameters θ [19, 45]. Online planning can be performed in a centralized or decentralized way by using a concrete MAS planning algorithm. In both cases, each planner uses the common value function approximation $\hat{V}_\theta$ to search for π_{local} by maximizing $G_{t,EVADE}$. A conceptual overview of the EVADE architecture is shown in fig. 2. Completely decentralized architectures, where all agents plan and learn independently of each other, are not considered here and left for future work.

[1]In practice, θ is updated w.r.t. experience batches of constant size, which are sampled from D [26, 27].



(a) Centralized planning **(b) Decentralized planning**

Figure 2: Illustration of the possible MAS planning architectures for EVADE. The planners get global feedback from a value function, which is approximated in a centralized way. The red dashed arrow between the planners in fig. 2b represents a coordination mechanism for decentralized planning.

Decentralized planning approaches require an explicit coordination mechanism to avoid convergence to suboptimal joint policies as shown in fig. 2b and in [8, 10]. This could be done by using a consensus mechanism to *synchronize* on time or on a common seed value to generate the same random numbers when sampling plans [17]. Agents could also exchange observations, experience, plans or policies via *communication* [45, 48]. Another way is to *predict* other agents' actions by using a policy function similarly to [36] or by maintaining a belief about other agents' behaviour [10, 29].

5 EXPERIMENTS

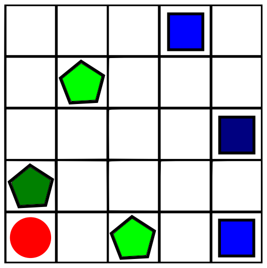
5.1 Evaluation Environment

5.1.1 Description. We implemented a smart factory environment to evaluate multi-agent online planning with EVADE. Our smart factory consists of a 5×5 grid of *machines* with 15 different machine types as shown in fig. 3a. Each *item* is carried by one agent i and needs to get processed at various machines according to its randomly assigned processing tasks $tasks_i = [\{a_{i,1}, b_{i,1}\}, \{a_{i,2}, b_{i,2}\}]$, where each task $a_{i,j}$,$b_{i,j}$ is contained in a *bucket*. While tasks in the same bucket can be processed in any order, buckets themselves have to be processed in a specific order. Fig. 3b shows an example for an agent i with $tasks_i = [\{9, 12\}, \{3, 10\}]$. It first needs to get processed by the machines marked as green pentagons before going to the machines marked as blue rectangles. Note that i can choose between two different machines for processing its requests $a_{i,1} = 9$ and $a_{i,2} = 3$, which are rendered as light green pentagons or light blue rectangles. In the presence of multiple agents, coordination is required to choose an appropriate machine to avoid conflicts.

All agents have a random initial position and can move along the machine grid or enqueue at their current position represented by a machine. Each machine can process exactly one item per time step with a cost of 0.25 but fails with a probability of 0.1 to do so. Enqueued agents are unable to perform any actions. If a task is processed, it is removed from its bucket. If a bucket is empty, it is removed from the item's tasks list. An item is *complete* if its tasks list is empty. The goal is to complete as many items as possible within 50 time steps, while avoiding any conflicts or enqueuing at wrong machines.

2	11	5	3	13
14	9	8	2	1
4	0	7	1	10
12	6	7	0	6
4	5	9	8	3

(a) machine grid



(b) an agent and its tasks

Figure 3: Illustration of the smart factory setup used in the experiments. (a) the 5×5 grid of machines. The numbers in each grid cell denote the machine type. (b) an agent i (red circle) in the factory with $tasks_i = [\{9, 12\}, \{3, 10\}]$. It should get processed at the green pentagonal machines first before going to the blue rectangular machines.

5.1.2 MMDP Formulation. The smart factory environment can be modeled as MMDP $M = \langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$. $\mathcal{D}$ is the set of n agents with $\mathcal{D}_{active} \cap \mathcal{D}_{complete} = \emptyset$ and $\mathcal{D} = \mathcal{D}_{active} \cup \mathcal{D}_{complete}$. $\mathcal{D}_{active}$ is the set of agents with incomplete items and $\mathcal{D}_{complete}$ is the set of agents with complete items. $\mathcal{S}$ is a set of system states described by the individual state variables of all agents, items and machines. $\mathcal{A}$ is the set of joint actions. Each agent $i \in \mathcal{D}$ has the same individual action space $\mathcal{A}_i$ enabling it to move north, south, west or east, to enqueue at its current machine $m = pos_i$ or to do nothing. Any attempt to move across the grid boundaries is treated the same as "do nothing". $\mathcal{P}$ is the transition probability function. $\mathcal{R}$ is the scalar reward function. $\mathcal{R}$ at time step t is defined by $R(s_t, a_t) = score_{t+1} - score_t$, where $score_t$ is the immediate evaluation function for the system state:

$$score_t = |\mathcal{D}_{complete}| - tasks_t - cost_t - tpen_t \tag{5}$$

where $tasks_t = \sum_{i \in \mathcal{D}_{active}} \sum_{c \in tasks_i} |c|$ is the total number of currently unprocessed tasks, $cost_t$ is the total sum of processing costs for each machine after processing an enqueued item and $tpen_t = tpen_{t-1} + \sum_{i \in \mathcal{D}_{active}} penalty$ is the total sum of time penalities with $penalty = 0.1$ for all incomplete items at time step t. Processing tasks and completing items increases $score_t$. Otherwise, $score_t$ decreases for each incomplete item or enqueuing at a wrong machine.

5.1.3 Complexity. Depending on the number of agents n, the number of possible joint actions is $|\mathcal{A}| = |\mathcal{A}_i|^n = 6^n$. The machine failure probability of 0.1 increases the branching factor of the problem even more. Given a planning horizon of h, the number of possible joint plans is defined by:

$$|\pi_{local}| = |\Pi_{local,i}|^n = (|\mathcal{A}_i|^h)^n = |\mathcal{A}_i|^{h \cdot n} = 6^{h \cdot n} \tag{6}$$

We tested EVADE in settings with 4 and 8 agents. In the 4-agent case, there exist $6^4 \approx 1300$ possible joint actions. In the 8-agent case, there exist $6^8 \approx 1.68 \cdot 10^6$ possible joint actions. In our stochastic smart factory setup, where machines can fail with a probability of 0.1 and where agents are not acting in a deterministic way, the environment has a significantly higher branching factor than the game of Go, which has a branching factor of 250 [36].

5.2 Methods

5.2.1 Online Open-Loop Planning. Due to the stochasticity and high complexity of our environment, we focus on open-loop planning because we think that current state-of-the-art algorithms based on closed-loop planning would not scale very well in our case [1, 33][2]. Also, we do not aim for optimal planning, since our goal is to enhance existing local planning algorithms, which might even perform suboptimal in the first place.

The individual policy π_i for each agent i is implemented as a stack or sequence of multi-armed bandits (MAB) of length h as proposed in [6]. Each MAB $\Phi_t = P(a_t|D_{a_t})$ represents a distribution, where D_{a_t} is a buffer of size 10 for storing local returns, which are observed when selecting arm $a_t \in \mathcal{A}$. Each buffer D_{a_t} is implemented in a sliding window fashion to consider only most recent observations to adapt to the non-stationary joint behaviour of all agents during the planning step.

Thompson Sampling is implemented as concrete MAB algorithm because of its effectiveness and robustness for making decisions under uncertainty [6, 12, 46]. The implementation is adopted from [3, 21], where the return values in D_{a_t} for each arm a_t are assumed to be normally distributed.

To optimize π_i, a plan of h actions is sampled from the MAB stack. The plan is evaluated in a simulation by using a generative model $\hat{M}$. The resulting rewards are accumulated to local returns according to eq. 3 and used to update the corresponding MABs of the MAB stack. This procedure is repeated $\lfloor \frac{n_{budget}}{h} \rfloor$ times. Afterwards, the action $a_t = argmax_{a_1 \in \mathcal{A}} \{\overline{D_{a_1}}\}$ is selected from the MAB Φ_1 for execution in the actual environment, where $\overline{D_{a_1}}$ is the mean of all local returns currently stored in D_{a_1}.

5.2.2 Multi-Agent Planning. We implemented two multi-agent planning algorithms to evaluate the performance achieved by using EVADE. All algorithms enhanced with EVADE were compared with their non-enhanced counterparts w.r.t. performance and efficiency.

Direct Cross Entropy (DICE) method for policy search in distributed models. DICE is a centralized planning algorithm proposed in [30] and uses stochastic optimization to search joint policies, which are optimal or close to optimal. In DICE a multivariate distribution $f_\xi(\pi) = \prod_{i=1}^n f_{\xi_i}(\pi_i)$ is maintained to sample candidate joint policies π. These candidates are evaluated in a simulation with a global model $\hat{M}$. The N_b best candidates are used to update f_ξ. This procedure is repeated until convergence is reached or n_{budget} has run out. Our implementation of DICE uses n MAB stacks representing $f_\xi(\pi)$ to sample joint plans of length h, which are simulated in $\hat{M}$. The resulting local returns are used to update all MAB stacks.

Distributed Online Open-Loop Planning (DOOLP). DOOLP is a decentralized version of DICE proposed in [5], where each agent is controlled by an individual planner with an individual model $\hat{M}_i = \hat{M}$ for simulation-based planning. At every time step each agent i iteratively optimizes its policy π_i by first sampling a plan

[2]We conducted experiments with MCTS on a joint action MDP formulation of our problem but always ran out of memory due to the large branching factor of our settings and the enormous size of the constructed search tree.

and then querying the sampled plans of its neighbours to construct a joint plan. The joint plan is simulated in $\hat{M}_i$ and the simulation result is used to update the individual policy π_i of agent i. The individual MAB stacks are assumed to be *private* for each agent i. Due to the stochasticity of the environment described in section 1 and 5.1, the planners can have different simulation outcomes leading to different updates to the individual MAB stacks.

As a decentralized approach, DOOLP requires an explicit coordination mechanism to avoid suboptimal joint policies (see section 4.3 and fig. 2b). We implemented a communication-based coordination mechanism, where each planner communicates its sampled plans to all other planners, while keeping its actual MAB stack private.

5.2.3 Value Function Approximation. We used a deep convolutional neural network as $\hat{V}_\theta$ to approximate the value function V^*. The weights of the neural network are denoted as θ. $\hat{V}_\theta$ was trained with TD learning by using methods of deep RL [26, 27]. An experience buffer D was implemented to uniformly sample minibatches to perform stochastic gradient descent on. D was initialized with 5000 experience samples generated from running smart factory episodes using multi-agent planning without EVADE.

An additional target network $\hat{V}_{\theta^-}$ was used to generate TD regression targets for $\hat{V}_\theta$ (see eq. 4) to stabilize the training [27]. All hyperparameters used for training $\hat{V}_\theta$ are listed in table 1.

hyperparameter	value
update rule for optimization	ADAM
learning rate	0.001
discount factor γ	0.95
minibatch size	64
replay memory size	10000
target network update frequency C	5000

Table 1: Hyperparameters for the value network $\hat{V}_\theta$.

The factory state is encoded as a stack of 5×5 feature planes, where each plane represents the spatial distribution of machines or agents w.r.t. some aspect. An informal description of all feature planes is given in table 2.

The input to $\hat{V}_\theta$ is a $5 \times 5 \times 35$ matrix stack consisting of 35 matrices. The first hidden layer convolves 128 filters of size 5×5 with stride 1. The next three hidden layer convolve 128 filters of size 3×3 with stride 1. The fifth hidden layer convolves one filter of size 1×1 with stride 1. The sixth hidden layer is a fully connected layer with 256 units. The output layer is a fully connected with a single linear unit. All hidden layers use exponential linear unit (ELU) activation as proposed in [15]. The architecture of $\hat{V}_\theta$ was inspired by the value network of [36].

5.3 Results

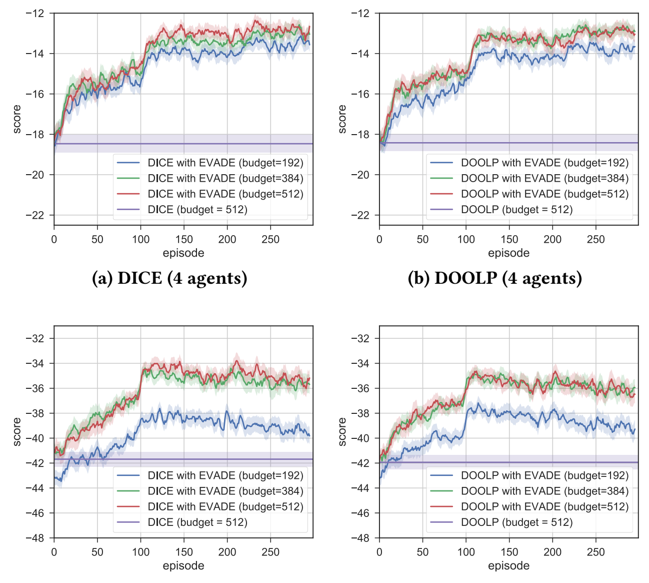
Various experiments with 4- and 8-agent settings were conducted to study the effectiveness and efficiency achieved by the multi-agent online planning algorithms from section 5.2.2 with EVADE.

An episode is reset after $T = 50$ time steps or when all items are complete such that $\mathcal{D}_{active} = \emptyset$. A run consists of 300 episodes and is repeated 100 times. Multi-agent online planning with EVADE searches for a joint policy π_{local} by maximizing $G_{t,EVADE}$ with a value function approximation $\hat{V}_\theta$ (see eq. 3). All baselines perform planning without EVADE by maximizing G_t instead (see eq. 1).

The performance of multi-agent online planning is evaluated with the value of $score_{50}$ at the end of each episode (see eq. 5) and the item completion rate $R_{completion} = \frac{|\mathcal{D}_{complete}|}{|\mathcal{D}|}$ at the end of the 300th episode, with $0 \leq R_{completion} \leq 1$. If all items are complete within 50 time steps, then $R_{completion} = 1$. If no item is complete within 50 time steps, then $R_{completion} = 0$. All baselines were run 500 times to determine the average of $score_{50}$ and $R_{completion}$.

5.3.1 Efficiency w.r.t. Computation Budget. The effect of EVADE w.r.t. the breadth of the policy search was evaluated. The experiments for each algorithm were run with different budgets $n_{budget} \in \{192, 384, 512\}$[3] and a fixed horizon of $h = 4$. The baselines represented by the corresponding non-enhanced planning algorithms had a computation budget of $n_{budget} = 512$.

Fig. 4 shows the average progress of $score_{50}$. In all cases, the EVADE enhanced versions outperform their corresponding baselines. There is a relatively large performance gain in the first hundred episodes. The average score increases slowly afterwards or stagnates as shown in the 8-agent case in fig. 4c and 4d. There are no significant differences between the enhanced versions with $n_{budget} \in \{384, 512\}$. Planning with a budget of $n_{budget} = 192$ leads to worse performance than the corresponding enhanced variants with a larger budget.



(a) DICE (4 agents) **(b) DOOLP (4 agents)**

(c) DICE (8 agents) **(d) DOOLP (8 agents)**

Figure 4: Average progress of $score_{50}$ of 100 runs shown as running mean over 5 episodes for different computation budgets $n_{budget} \in \{192, 384, 512\}$ and a horizon of $h = 4$. All baselines have a computation budget of $n_{budget} = 512$. Shaded areas show the 95% confidence interval.

[3] We also experimented with $n_{budget} = 256$ but there was no significant difference to planning with $n_{budget} = 384$.

Table 2: Description of all feature planes as input for $\hat{V}_\theta$.

Feature	# Planes	Description
Machine type	1	The type of each machine as a value between 0 and 14 (see fig. 3a)
Agent state	4	The number of agents standing at machines whose types are (not) contained in their current tasks and whether they are enqueued or not.
Tasks (1st bucket)	15	Spatial distribution of agents containing a particular machine type in their first bucket of tasks for each available machine type.
Tasks (2nd bucket)	15	Same as "Tasks (1st bucket)" but for the second bucket of tasks.

The average completion rates $R_{completion}$ at the end of the 300^{th} episode of all experiments are listed in table 3. In the 4-agent case, the completion rates of the baselines are about 63%, while the rates achieved by the EVADE enhanced versions range from 86 to 92%. In the 8-agent case, the completion rates of the baslines are about 54%, while the rates achieved by the EVADE enhanced versions range from 65 to 78%. EVADE enhanced planning with $n_{budget} \in \{384, 512\}$ always tends to achieve a higher completion rate than using a budget of $n_{budget} = 192$.

5.3.2 Efficiency w.r.t. Horizon. Next the effect of EVADE w.r.t. the depth of the policy search was evaluated. The experiments for each algorithm were run with different horizon lengths $h \in \{2, 4, 6\}$ and a fixed computation budget of $n_{budget} = 384$. The baselines represented by the corresponding non-enhanced planning algorithms had a horizon of $h = 6$.

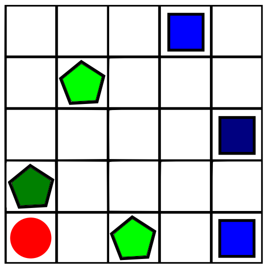
The planning horizon h influences the reachability of machines in each simulation step as shown in fig. 5. In this example, the agent can only reach about one fifth of the grid when planning with $h = 2$ (see fig. 5b), while it can theoretically reach almost any machine when planning with $h = 6$ (see fig. 5d).

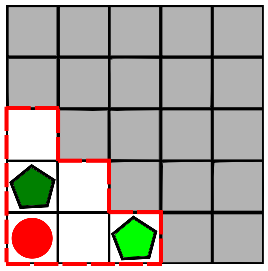
Fig. 6 shows the average progress of $score_{50}$. Planning with a horizon of $h = 2$ always had the worst initial average performance but the largest performance gain in the first hundred episodes, while planning with a horizon of $h = 6$ had the best initial average performance but the smallest performance gain. In the 8-agent case, planning with EVADE and a horizon of $h = 2$ even outperforms the planning variants with a longer horizon after about one hundred episodes as shown in fig. 6c and 6d. This phenomenon will be discussed in the next section.

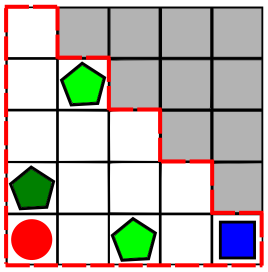
The average completion rates $R_{completion}$ at the end of the 300^{th} episode of all experiments are listed in table 4. In the 4-agent case, the completion rate of the baselines are about 70%, while the rate achieved by the EVADE enhanced versions range from about 82 to 92%. In the 8-agent case, the completion rates of the baslines are about 59%, while the rate achieved by the EVADE enhanced versions range from about 66 to 77%. Increasing the horizon from 2 to 6 in the 4-agent case tends to slightly increase $R_{completion}$, while in the 8-agent case it leads to a decrease of $R_{completion}$.

5.4 Discussion

Our experiments show that statistical multi-agent online planning can be effectively improved with EVADE, even when using a smaller computation budget n_{budget} than planning without any value function. However, n_{budget} must not be too small, since statistical online planning algorithms always require a minimum of computation to reach promising states with higher probability. This is shown in the experimental settings with $n_{budget} = 192$ in fig. 4 and table 3.


(a) an agent and its tasks


(b) horizon of $h = 2$


(c) horizon of $h = 4$

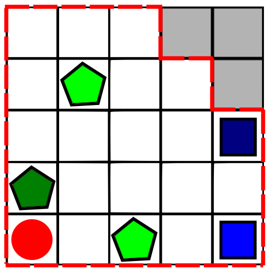

(d) horizon of $h = 6$

Figure 5: Reachability of machines for an agent (red circle) in a simulation step depending on the planning horizon h. Gray grid cells mark unreachable machines. (a) The example from fig. 3b. (b), (c) and (d) Reachable machines within the dashed red boundaries, given resp. horizons of h.

In the smart factory environment, planning with a sufficient horizon length is crucial to find joint policies with high quality as shown in fig. 5 and table 3 and 4 regarding the performance of the baselines. If a needed machine is unreachable in the simulation, it cannot be considered in the local planning process, thus possibly leading to poor solutions. In our experiments, the value function approximation could improve multi-agent planning with horizons which were too short to consider the entire factory.

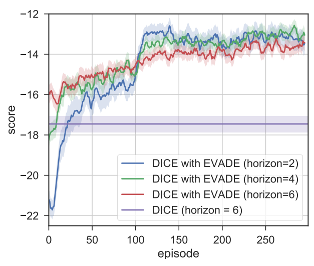
If the discount factor is $\gamma < 1$, then the value function influences planning with short horizons more than planning with a long horizon (see eq. 3). In our experiments, planning with a horizon of $h = 2$ was able to keep up with planning variants with a longer horizon, even outperforming them in the 8-agent case, given an equal computation budget of $n_{budget} = 384$. These are strong indications that

Table 3: Average rate of complete items $R_{completion}$ at the end of the 300^{th} episode of all experiments within a 95% confidence interval. Planning was performed with different computation budgets n_{budget} and a horizon of $h = 4$.

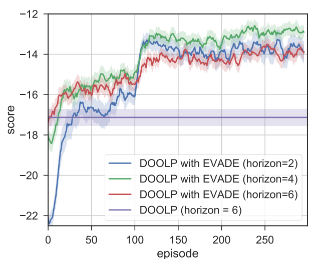
algorithm (# agents)	baseline ($n_{budget} = 512$)	EVADE ($n_{budget} = 192$)	EVADE ($n_{budget} = 384$)	EVADE ($n_{budget} = 512$)
DICE (4 agents)	62.5 ± 2.1%	86.8 ± 3.6%	89.3 ± 3.0%	**91.8 ± 3.0%**
DOOLP (4 agents)	63.7 ± 2.1%	88.5 ± 3.0%	**91.3 ± 2.9%**	91.0 ± 3.4%
DICE (8 agents)	55.2 ± 1.5%	65.0 ± 3.4%	73.1 ± 3.3%	**77.5 ± 3.2%**
DOOLP (8 agents)	53.9 ± 1.4%	65.8 ± 3.7%	72.8 ± 3.6%	**73.0 ± 3.5%**

Table 4: Average rate of complete items $R_{completion}$ at the end of the 300^{th} episode of all experiments within a 95% confidence interval. Planning was performed with different horizons h and a computation budget of $n_{budget} = 384$.

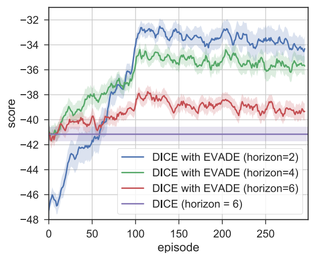
algorithm (# agents)	baseline ($h = 6$)	EVADE ($h = 2$)	EVADE ($h = 4$)	EVADE ($h = 6$)
DICE (4 agents)	69.7 ± 2.0%	87.0 ± 3.2%	89.3 ± 3.0%	**90.8 ± 3.2%**
DOOLP (4 agents)	71.6 ± 2.0%	82.3 ± 4.0%	**91.3 ± 2.9%**	88.5 ± 3.4%
DICE (8 agents)	58.3 ± 1.5%	**77.0 ± 3.8%**	73.1 ± 3.3%	66.1 ± 3.1%
DOOLP (8 agents)	60.0 ± 1.5%	**72.9 ± 3.5%**	72.8 ± 3.6%	67.6 ± 2.9%



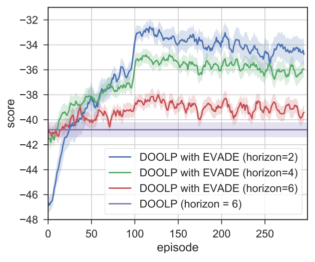
(a) DICE (4 agents)



(b) DOOLP (4 agents)



(c) DICE (8 agents)



(d) DOOLP (8 agents)

Figure 6: Average progress of $score_{50}$ of 100 runs shown as running mean over 5 episodes for different horizons $h \in \{2, 4, 6\}$ and a computation budget of $n_{budget} = 384$. All baselines have a horizon of $h = 6$. Shaded areas show the 95% confidence interval.

our approach offers planning efficiency w.r.t. the breadth and the depth of the policy search after a sufficient learning phase.

The performance stagnation in the 8-agent case after hundred episodes can be explained with the enormous policy space to be searched and the relatively small computation budget n_{budget}. This also explains the rather poor performance of online planning with a horizon of $h = 6$ compared to variants with $h = 2$ or $h = 4$ as shown in fig. 6c and 6d. Given $n_{budget} = 384$, the former only performs $\lfloor \frac{n_{budget}}{h} \rfloor = 64$ simulations per time step, while searching a much larger policy space ($|\pi_{local}| > 10^{37}$) than the latter ($|\pi_{local}| < 10^{25}$) according to eq. 6. When using the value function approximation $\hat{V}_\theta$, more simulations should lead to high quality results with a higher accuracy. Thus, a larger performance gain can be expected when increasing n_{budget}.

6 CONCLUSION & FUTURE WORK

In this paper, we presented EVADE, an approach to effectively improve the performance of statistical multi-agent online planning in stochastic domains by integrating global experience. For this purpose, a value function is approximated online based on the emergent system behaviour by using model-free RL. By considering global outcome estimates with that value function during the planning step, multi-agent online planning with EVADE is able to overcome the limitation of local planning as sketched in fig. 1.

We also introduced a smart factory environment, where multiple agents need to process various items with different tasks at a shared set of machines in an automated and self-organizing way. Unlike domains used in [2, 36, 37], our environment can have multiple agents, is stochastic and has a higher branching factor, given a sufficient number of agents.

EVADE was successfully tested with two existing statistical multi-agent planning algorithms in this highly complex and stochastic domain. EVADE offers planning efficiency w.r.t. the depth and the breadth of the joint policy search requiring less computational effort to find solutions with higher quality compared to multi-agent planning without any value function.

For now, EVADE has only been applied to fully observable settings. Decentralized partially observable problems can often be decomposed into smaller subproblems, which are fully observable themselves. This is common in distributed environments, where agents can sense and communicate with all neighbours within their range. EVADE could be directly applied to those subproblems. As a possible direction for future work, EVADE could be extended to partially observable domains without any problem decomposition.

REFERENCES

[1] Christopher Amato and Frans A Oliehoek. 2015. Scalable planning and learning for multiagent POMDPs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 1995–2002.

[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*. 5366–5376.

[3] Aijun Bai, Feng Wu, Zongzhang Zhang, and Xiaoping Chen. 2014. Thompson sampling based Monte-Carlo planning in POMDPs. In *Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling*. AAAI Press, 29–37.

[4] Richard Bellman. 1957. *Dynamic Programming* (1 ed.). Princeton University Press, Princeton, NJ, USA.

[5] Lenz Belzner and Thomas Gabor. 2017. Scalable Multiagent Coordination with Distributed Online Open Loop Planning. *arXiv preprint arXiv:1702.07544* (2017).

[6] Lenz Belzner and Thomas Gabor. 2017. Stacked Thompson Bandits. In *Proceedings of the 3rd International Workshop on Software Engineering for Smart Cyber-Physical Systems*. IEEE Press, 18–21.

[7] Lenz Belzner, Rolf Hennicker, and Martin Wirsing. 2015. OnPlan: A framework for simulation-based online planning. In *International Workshop on Formal Aspects of Component Software*. Springer, 1–30.

[8] 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., 195–210.

[9] S Bubeck and R Munos. 2010. Open Loop Optimistic Planning. In *Conference on Learning Theory*.

[10] 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, 183–221.

[11] Anthony R Cassandra and Leslie Pack Kaelbling. 2016. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995: Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, July 9-12 1995*. Morgan Kaufmann, 362.

[12] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems*. 2249–2257.

[13] Guillaume Chaslot. 2010. Monte-carlo tree search. *Maastricht: Universiteit Maastricht* (2010).

[14] 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*. International Foundation for Autonomous Agents and Multiagent Systems, 492–500.

[15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR* abs/1511.07289 (2015).

[16] Adam Eck, Leen-Kiat Soh, Sam Devlin, and Daniel Kudenko. 2016. Potential-based reward shaping for finite horizon online pomdp planning. *Autonomous Agents and Multi-Agent Systems* 30, 3 (2016), 403–445.

[17] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. 2004. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*. IEEE Computer Society, 136–143.

[18] Jonathan St BT Evans. 1984. Heuristic and analytic processes in reasoning. *British Journal of Psychology* 75, 4 (1984), 451–468.

[19] Jakob Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*. 2137–2145.

[20] Todd Hester and Peter Stone. 2013. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine learning* 90, 3 (2013), 385–429.

[21] Junya Honda and Akimichi Takemura. 2014. Optimality of thompson sampling for gaussian bandits depends on priors. In *Artificial Intelligence and Statistics*. 375–383.

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

[23] Daniel Kahneman. 2003. Maps of bounded rationality: Psychology for behavioral economics. *The American economic review* 93, 5 (2003), 1449–1475.

[24] Levente Kocsis and Csaba Szepesvári. [n. d.]. Bandit based monte-carlo planning. In *ECML*, Vol. 6. Springer, 282–293.

[25] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.

[27] 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), 529–533.

[28] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. 2015. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296* (2015).

[29] Frans A Oliehoek and Christopher Amato. 2016. *A concise introduction to decentralized POMDPs*. Springer.

[30] Frans A Oliehoek, Julian FP Kooij, and Nikos Vlassis. 2008. The cross-entropy method for policy search in decentralized POMDPs. *Informatica* 32, 4 (2008).

[31] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. 2008. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research* 32 (2008), 289–353.

[32] Sébastien Paquet, Brahim Chaib-draa, and Stéphane Ross. 2006. Hybrid POMDP algorithms. In *Proceedings of The Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM-06)*. 133–147.

[33] 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, 337–344.

[34] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[35] Stéphane Ross, Brahim Chaib-Draa, et al. 2007. AEMS: An Anytime Online Search Algorithm for Approximate Policy Refinement in Large POMDPs.. In *IJCAI*. 2592–2598.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[37] 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), 354–359.

[38] David Silver and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*. 2164–2172.

[39] Jonathan Sorg, Satinder P Singh, and Richard L Lewis. 2011. Optimal Rewards versus Leaf-Evaluation Heuristics in Planning Agents.. In *AAAI*.

[40] Matthijs TJ Spaan, Frans A Oliehoek, and Christopher Amato. 2011. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence-Volume Volume Three*. AAAI Press, 2027–2032.

[41] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3, 1 (1988), 9–44.

[42] Richard S Sutton and Andrew G Barto. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT Press Cambridge.

[43] Daniel Szer, Francois Charpillet, and Shlomo Zilberstein. 2005. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *21st Conference on Uncertainty in Artificial Intelligence-UAI'2005*.

[44] 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), e0172395.

[45] Ming Tan. 1997. Multi-agent reinforcement learning: independent vs. cooperative agents. In *Readings in agents*. Morgan Kaufmann Publishers Inc., 487–494.

[46] William R Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3/4 (1933), 285–294.

[47] Ari Weinstein and Michael L Littman. 2013. Open-Loop Planning in Large-Scale Stochastic Domains.. In *AAAI*.

[48] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. 2009. Multi-Agent Online Planning with Communication. In *Nineteenth International Conference on Automated Planning and Scheduling*.