Journal of Artificial Intelligence Research 73 (2022) 821-846 Submitted 08/2021; published 03/2022

Scalable Online Planning for Multi-Agent MDPs

Shushman Choudhury SHUSHMAN@CS.STANFORD.EDU
Stanford University

Jayesh K. Gupta JAYESH.GUPTA@MICROSOFT.COM
Microsoft

Peter Morales IO.PETER.MORALES@GMAIL.COM

Mykel J. Kochenderfer MYKEL@STANFORD.EDU
Stanford University

Abstract

We present a scalable tree search planning algorithm for large multi-agent sequential decision problems that require dynamic collaboration. Teams of agents need to coordinate decisions in many domains, but naive approaches fail due to the exponential growth of the joint action space with the number of agents. We circumvent this complexity through an approach that allows us to trade computation for approximation quality and dynamically coordinate actions. Our algorithm comprises three elements: online planning with Monte Carlo Tree Search (MCTS), factored representations of local agent interactions with coordination graphs, and the iterative Max-Plus method for joint action selection. We evaluate our approach on the benchmark SysAdmin domain with static coordination graphs and achieve comparable performance with much lower computation cost than our MCTS baselines. We also introduce a multi-drone delivery domain with dynamic coordination graphs, and demonstrate how our approach scales to large problems on this domain that are intractable for other MCTS methods. We provide an open-source implementation of our algorithm at https://github.com/JuliaPOMDP/FactoredValueMCTS.jl.

1. Introduction

Coordination is crucial for effective decision-making in cooperative multi-agent systems with a shared objective. Various real-world problems like formation control (Oh et al., 2015), package delivery (Choudhury et al., 2021), and firefighting (Oliehoek et al., 2008) require a team of autonomous agents to perform a common task. Such cooperative sequential decision-making problems can be modeled as a multi-agent Markov decision process (MMDP) (Boutilier, 1996), an extension of the Markov decision process (MDP) (Bellman, 1957). MMDPs can be reduced to centralized single-agent MDPs with a joint action space for all agents. Such reductions often make large problems intractable because the action space grows exponentially with the number of agents. Solving independent MDPs for all agents yields suboptimal behavior in problems where reasoning about the effects of joint actions is necessary for better performance (Matignon et al., 2012).

Many previous MMDP approaches have tried to balance these extremes of optimality and efficiency. In the *offline* setting, these include ad hoc function decomposition approaches, such as Value Decomposition Networks (Sunehag et al., 2018) and QMIX (Rashid et al., 2018), or parameter sharing in decentralized policy optimization (Gupta et al., 2017). Guestrin et al. (2002) introduced the concept of a coordination graph to reason about joint

value estimates from a factored representation, while Kok and Vlassis (2004) used approximations to scale these ideas to larger problems. Monte Carlo Tree Search (MCTS) (Browne et al., 2012), a common approach to *online* planning, has been combined with coordination graphs in Factored Value MCTS (Amato & Oliehoek, 2015). However, Factored Value MCTS coordinates actions with an exact Variable Elimination (Var-El) step, which slightly negates the anytime nature of MCTS planning.

The key idea of this paper is to *build a scalable planning algorithm for MMDPs that require coordination*, while fully retaining the anytime properties of Monte Carlo Tree Search. To that end, we propose combining Max-Plus action selection, introduced by Vlassis et al. (2004), with the Factored Value MCTS of Amato and Oliehoek (2015). We do so for several reasons. Unlike Var-El, which is exact, Max-Plus is an iterative procedure and allows for truly anytime behavior that can trade computation for approximation quality. The representation of Max-Plus is much more efficient than that of Var-El for using dynamic, i.e., state-dependent, coordination graphs (state-dependent data-structures are a key benefit of online planning for MDPs). Finally, Max-Plus can scale to much larger multiagent teams and denser agent interactions than Var-El, and can be distributed for even more scalability (Kok & Vlassis, 2005).

We present a scalable MMDP planning algorithm called Factored Value MCTS with Max-Plus. On the standard SysAdmin benchmark domain with static coordination graphs, we demonstrate that our approach performs as well as or better than Factored Value MCTS with Var-El and is much faster for the same tree search hyperparameters. We also introduce a new MMDP domain, Multi-Drone Delivery, with dynamic coordination graphs. On the second domain, we show how our approach scales to problem sizes that are entirely intractable for other MCTS variants, while also achieving better performance on smaller problem sizes.

2. Background and Related Work

We first review MDPs and their multi-agent formulation. We then describe how coordination graphs can efficiently exploit the locality of interactions in multi-agent problems. Finally, we discuss how to use coordination graphs to solve multi-agent MDPs.

2.1 Multi-Agent Markov Decision Processes

An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. The objective for solving an MDP is to obtain a *policy*, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that specifies a probability distribution over actions for the agent to take from its current state to maximize its *value*, i.e. its expected cumulative reward. An action-value function $Q(s, a)$ defines the expected cumulative reward after taking action a in state s before following the specified policy.

We focus on decision-making settings where multiple agents cooperate to achieve a shared objective (Boutilier, 1996). Such problems are multi-agent Markov decision processes (MMDPs), where each agent takes an individual action and the controller policy observes the states of all agents. In principle, we can solve an MMDP as a standard MDP with a

joint action space $\mathcal{A} = \prod_i \mathcal{A}_i$ (Pynadath & Tambe, 2002). There exist both offline and online methods for computing such MDP policies (Bertsekas, 2005).

Offline methods pre-compute a policy over the entire state space (exactly or approximately) and query the policy during execution. Various exact offline methods exist, but reinforcement learning has emerged as an attractive solution technique due to the complexity of planning in large MMDPs (Sutton & Barto, 1998). Reinforcement learning approaches attempt to compute an effective value function $Q(s, a)$ or a policy $\pi(a \mid s)$ through repeated interaction with the environments, either by directly learning those quantities, or by learning transition and reward models to then plan with.

Most reinforcement learning approaches still struggle in large multi-agent problems, where the size of the joint action space is exponential in the number of agents. A common strategy is to *decentralize* the policy or value function, such that each agent's performance only depends on its own actions (Gupta et al., 2017; Rashid et al., 2018; Son et al., 2019; Sunehag et al., 2018). Unfortunately, such decentralized approaches are often sub-optimal for coordination and encounter exploration bottlenecks due to uncooperative random actions from the agents (Böhmer et al., 2020). Zhang et al. (2021) summarizes several of these approaches.

Online methods use an alternative strategy to handle the complexity of multi-agent planning; they interleave planning and execution by focusing only on states that are reachable for the current state, while computing the next action to take. Monte Carlo Tree Search (MCTS) is a common framework for online planning and has succeeded in a variety of domains from game-playing to scheduling to vehicle routing (Browne et al., 2012; Świechowski et al., 2021), including in multi-player contexts (Nijssen & Winands, 2011; Zerbel & Yliniemi, 2019).

The *anytime* nature of MCTS (search depth and number of simulations) allows us to trade computation time for approximation quality. However, the exponentially large action space of MMDPs can still be a bottleneck for the naive application of MCTS techniques (Chaslot et al., 2008). Dec-MCTS tries to work around this bottleneck by allowing robots to optimize their own actions and communicate their compressed trees (Best et al., 2019). However, it directly chooses the next state, and thus does not apply to action-dependent stochastic transitions of an MDP. A more recent work by Ma et al. (2020) uses an offline heuristic obtained from centralized imitation learning to bias the online tree search that is distributed across agents.

2.2 Monte Carlo Tree Search for MDPs

Because MCTS is the foundation of our proposed approach, we describe it in further detail, following the presentation in Kochenderfer (2015). This description assumes a single-agent MDP; we will discuss how it handles MMDPs in Section 2.1. MCTS runs simulations from the current state to estimate the state-action value function. The estimate is updated by gathering relevant statistics through interactions with a simulated generative model of the environment (Silver & Veness, 2010). These statistics typically track the average simulated reward obtained for trying an action, the frequency of action attempts, and the number of occurrences of each state in the tree.

Algorithm 1 Monte Carlo Tree Search

Require: time limit, depth, exploration constant c, state s

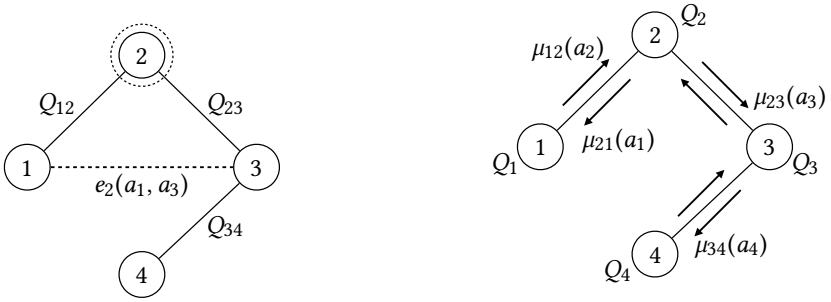
1: **function** MCTS(s, depth)
2: **while** time limit not reached
3: SIMULATE(s, depth)
4: $a^* \leftarrow \underset{a}{\text{argmax}}\ Q(s,a)$ ▷ No exploration here
5: **return** a^* ▷ Best action

6: **function** SIMULATE(s, depth)
7: **if** depth $= 0$
8: **return** 0
9: **if** $s \notin T$ ▷ state not in tree
10: Initialize $N(s,a)$ and $Q(s,a)$ for all $a \in \mathcal{A}$
11: $T \leftarrow T \cup s$
12: **return** ROLLOUT(s, depth)
13: $a \leftarrow \underset{a}{\text{argmax}}\ Q(s,a) + c \cdot \sqrt{\frac{\log N(s)}{N(s,a)}}$ ▷ UCB Exploration
14: $s', r \sim T(s,a), R(s,a)$ ▷ Generative model
15: $q \leftarrow r + \gamma \cdot$ SIMULATE(s', depth $- 1$)
16: $N(s,a) \leftarrow N(s,a) + 1$
17: $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$

Algorithm 1 provides the pseudocode of MCTS for online planning in MDPs. The algorithm runs a number of simulations from the current state s up to some time limit, which typically depends on the real-time computation constraints (line 2). The simulations conduct a lookahead search up to some depth (line 3); after running them, we compute and return the action that maximizes the estimated state action value function (lines 4–5). Each individual simulation consists of three conceptual stages, which we now describe.

The *expansion* stage (lines 9–11) is triggered when the simulation reaches a state not in the tree (either the root node in an empty tree or a leaf node in a non-empty one). This stage initializes the N and Q statistics for the frequency and estimated value respectively, of each action at that state. It then adds the state node to the tree. After expansion, we have the *rollout* stage, where we select actions according to some default rollout policy for the remaining depth and compute the accumulated return. Rollouts can allow domain experts to encode heuristic knowledge and bias the search.

The third stage is the so-called *search* stage, where we choose a branch of the tree to search further from the current state node. We choose the branch corresponding to the action that maximizes the equation in line 13; the second term of that equation is the Upper Confidence Bound exploration bonus, which encourages choosing actions that have not been tried as much, i.e., that have low $N(s,a)$. With this exploration strategy, MCTS is called Upper Confidence Trees (Kocsis & Szepesvári, 2006). We simulate trying this action and obtain a Monte Carlo sample of the next state s' and reward from the generative model



(a) Eliminating agent 2 in Var-El introduces an edge between nodes (agents) 1 and 3 and a new payoff function e_2.

(b) Max-Plus passes messages along the edges. Messages are functions of the actions of the receiving agent; e.g., agent 1 sends $\mu_{12}(a_2)$ to agent 2.

Figure 1: A coordination graph for an MMDP with 4 agents.

of the MDP (line 14). Finally, we recurse the simulation from s' and update the Q and N statistics at (s, a).

2.3 Coordination Graphs and Variable Elimination

Several real-world multi-agent systems demonstrate *locality of interaction*, i.e. the outcome of an agent's action depends only on the actions of a subset of other agents. The coordination graph (CG) structure is often used to encode such interactions (Guestrin et al., 2003; Guestrin et al., 2002). A CG for a multi-agent system has one node per agent, and edges connect agents if their payoffs depend on each other. For now, we assume a stateless or single-shot decision setting (rather than a sequential one). The CG structure induces a set of payoff components, *where each component is associated with a clique*, i.e., a subset of agents that are all mutually connected.

For CGs in multi-agent settings, we assume that we can factor the global payoff for a joint action as the sum of local component payoffs, i.e. $Q(\overline{a}) = \sum_c Q_c(\overline{a}_c)$, where $\overline{a}$ is the global joint action, and $\overline{a}_c$ is the local component action (the projection of $\overline{a}$ on component c). Given this factored representation and the local component payoffs , we can compute the best joint action, $\text{argmax}_{\overline{a}}\ Q(\overline{a})$, with the Variable Elimination (Var-El) algorithm originating from the probabilistic inference literature (Guestrin et al., 2003). Computing the optimal joint action in a CG is equivalent to obtaining the maximum a posteriori configuration in an undirected probabilistic graphical model.

Consider the 4-agent CG in Figure 1a. Here, $Q(\overline{a}) = Q_{12}(a_1, a_2) + Q_{23}(a_2, a_3) + Q_{34}(a_3, a_4)$, where a_i is the action variable for agent i. In Var-El, we *eliminate*, i.e., maximize over variables one at a time by collecting the local payoffs that depend on them. For instance, if we start with agent 2, then the first elimination is

$$\max_{a_1,a_3,a_4} Q_{34}(a_3, a_4) + \max_{a_2} \left[Q_{12}(a_1, a_2) + Q_{23}(a_2, a_3)\right]. \tag{1}$$

The optimal choice for agent 4 depends on a_2 and a_3. The internal max expression is summarized by a new intermediate payoff function $e_2(a_1, a_3) = \max_{a_2}[Q_{12}(a_1, a_2) + Q_{23}(a_2, a_3)]$

and a new edge between 1 and 3, after which the algorithm continues with Q_{34} and e_2. After all eliminations, we recover the action for each agent by maximizing the conditional functions in reverse, finally obtaining the optimal joint action. Var-El is exponential in the induced width of the CG, which depends on the elimination order (Dechter, 1999).

Although most works in the literature assume a domain-dependent static coordination graph structure, some incorporate state-dependent or dynamic CGs (Yu et al., 2020), including learning the most suitable CG structure from interactions with the environment (Kok et al., 2005; Li et al., 2021).

2.4 Scalable MMDP Methods with Coordination Graphs

In the *offline* context of tabular reinforcement learning methods, Kok and Vlassis (2004) explored action inference with predefined static coordination graphs over factorized value functions; Van der Pol and Oliehoek (2016) and Böhmer et al. (2020) extended these ideas to the neural network function approximation regime. We focus on *online* planning approaches to solving MMDPs. Amato and Oliehoek (2015) provide an online planning solution by extending the basic MCTS of Algorithm 1 to the regime of coordination graphs and factored values. Although they apply their algorithm to partially observed MDPs, the key ideas are the same for the fully observed case, which we describe now.

Recall that the CG topology induces payoff components, each associated with a graph clique. Thus, they maintain *local component statistics*, i.e. the mean payoff of each local component action $\overline{a}_e$ and the number of times it was attempted in that component; they call this *mixture of experts optimization*, albeit with a simple maximum likelihood estimator expert. For instance, during tree search from the current joint state $\overline{s} \equiv \{s_i\}$ (where s_i is the state of agent i), suppose the system simulates a joint action $\overline{a}$ and obtains a reward vector $\overline{r}$. Then, in a particular CG component e and the corresponding local subset of the joint action $\overline{a}_e$, they augment the local component action frequency statistic $N(\overline{s}, \overline{a}_e)$ by 1 and update the local component payoff statistic $Q_e(\overline{s}, \overline{a}_e)$ as

$$Q_e(\overline{s}, \overline{a}_e) \doteq Q_e(\overline{s}, \overline{a}_e) + \frac{\overline{r}_e - Q_e(\overline{s}, \overline{a}_e)}{N(\overline{s}, \overline{a}_e)}, \tag{2}$$

which is a standard running average update. The UCB exploration step uses the current statistics to select joint actions, i.e.,

$$\max_{\overline{a}} \sum_e U_e(\overline{s}, \overline{a}_e) = \max_{\overline{a}} \sum_e Q_e(\overline{s}, \overline{a}_e) + c \cdot \sqrt{\frac{\log N(\overline{s})}{N(\overline{s}, \overline{a}_e)}}, \tag{3}$$

where $N(\overline{s})$ is the visit frequency for state $\overline{s}$. Given these local component payoffs, i.e., the Q_e functions, their method computes the best joint action at the next time-step through Variable Elimination over the CG, as in Section 2.3. *Consequently, it loses the anytime property of MCTS because exact variable elimination cannot be stopped at an intermediate step.* Although Vlassis et al. (2004) explored various anytime algorithms for action selection with coordination graphs, they did not investigate their interaction with online planning algorithms like MCTS.

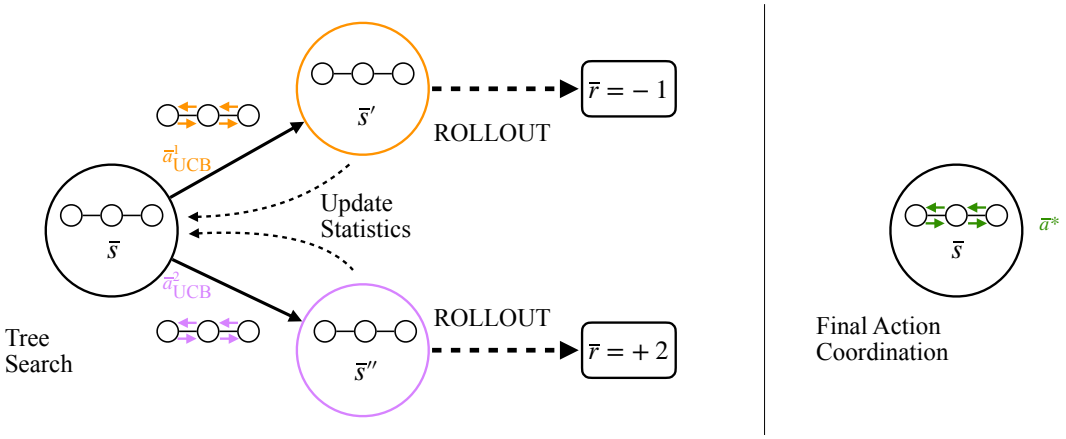


Figure 2: Our MMDP planning algorithm, Factored Value MCTS with Max-Plus, computes the best joint action $\overline{a}^*$ for the current joint state $\overline{s}$. The tree search uses an Upper Confidence Bound (UCB) exploration bonus during action selection, while the final action coordination does not.

3. Factored-Value Monte Carlo Tree Search with Max-Plus

We now discuss our method for online multi-agent MDP planning with coordination graphs, *Factored-Value Monte Carlo Tree Search with Max-Plus*. To apply the mixture of experts optimization to each node of the search tree, we must define the factored statistics to maintain for each node. Given a potentially state-dependent undirected *coordination graph* (CG), $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, we factor the CG-induced global payoff at the current state, $\overline{s}$, as follows:

$$Q(\overline{a}) = \sum_{i \in \mathcal{V}} Q_i(a_i) + \sum_{(i,j) \in \mathcal{E}} Q_{ij}(a_i, a_j). \tag{4}$$

Here, Q_{ij} is a local payoff function for agents i and j connected by edge (i, j), while Q_i is an individual utility function for agent i, if applicable to the domain. *All state-dependent quantities in this section's equations are defined implicitly for the current joint state* $\overline{s}$.

Exploiting the duality between computing the maximum a posteriori configuration in a probabilistic graphical model and the optimal joint action in a CG, Vlassis et al. (2004) introduced the Max-Plus algorithm for computing the joint action via message passing. Each node, i.e., agent, iteratively dispatches messages to its neighbours $j \in \Gamma(i)$ in the CG (Figure 1b). A message from agent i is a scalar-valued function of the action space of *receiving agent* j, i.e.,

$$\mu_{ij}(a_j) = \max_{a_i} \Big\{ Q_i(a_i) + Q_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus \{j\}} \mu_{ki}(a_i) \Big\}, \tag{5}$$

where $\Gamma(i)$ is the set of neighbors of i. Agents exchange messages until convergence or for a maximum number of rounds. Finally, each agent computes its optimal action individually, i.e.,

$$a_i^* = \operatorname*{argmax}_{a_i} \Big\{ Q_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i) \Big\}. \tag{6}$$

Max-Plus is equivalent to belief propagation in graphical models (Pearl, 1989) and its time complexity scales linearly with the CG size (the number of edges); it is more suitable for real-time systems and more tractable for large numbers of agents than Var-El.

Similar to Factored Value MCTS with Var-El, our method with Max-Plus (that we illustrate in Figure 2) is more efficient than a naive application of MCTS with the joint action space, since it maintains fewer statistics and performs efficient action selection. For the rest of this section, we will discuss the key differences from the prior work of Amato and Oliehoek (2015), which underscore how our approach is more suitable than theirs for large multi-agent MDPs, especially with dynamic coordination.

3.1 UCB Exploration with Max-Plus

The key implementation issue for extending MCTS to factored value functions and coordination graphs is that of action exploration as per the Upper Confidence Bound (UCB) strategy. In the Var-El case, Amato and Oliehoek (2015) added the exploration bonus using component-wise statistics during each elimination step in Equation (3). We cannot apply this strategy with Max-Plus as it does not use components. In contrast to Var-El, Max-Plus has two distinct phases of computation: message passing per edge in Equation (5), followed by action selection per node in Equation (4). We use these two phases to define how our algorithm explores.

Edge Exploration

Analogous to the edge payoff statistics Q_{ij}, we keep track of corresponding frequency statistics N_{a_i,a_j} (for pairwise actions). The natural exploration strategy over edges is to add the bonus to Equation (5) as follows:

$$\mu_{ij}(a_j) \quad = \quad \max_{a_i} \left\{ Q_i(a_i) \; + \; Q_{ij}(a_i, a_j) \; + \sum_{k \in \Gamma(i) \setminus \{j\}} \mu_{ki}(a_i) \; + \; c\sqrt{\frac{\log(N+1)}{N_{a_i,a_j}}} \right\}. \quad (7)$$

Adding this bonus during the message passing rounds can cause divergence for cyclic graphs with any cycle of length less than the number of rounds. Figure 3 illustrates intuition for this divergent behavior with a simple triangle graph. The bonuses accumulate in successive rounds for messages in either direction along the cycle, making the effective bonus proportional to the total number of rounds (divided by cycle length). Therefore, we only augment each message once *after the final round of message passing.*

Node Exploration

We maintain individual action frequency statistics N_{a_i} and modify Equation (4) to add a node exploration bonus during the action selection:

$$a_i^* = \operatorname*{argmax}_{a_i} \left\{ Q_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i) + c\sqrt{\frac{\log(N+1)}{N_{a_i}}} \right\}. \quad (8)$$

Note that the joint-action payoff $Q(\overline{a})$ can be factorized over the CG nodes and edges as in Equation (4), but the joint-action exploration bonus $c\sqrt{\frac{\log N(\overline{s})}{N(\overline{s},\overline{a})}}$ cannot. Therefore,

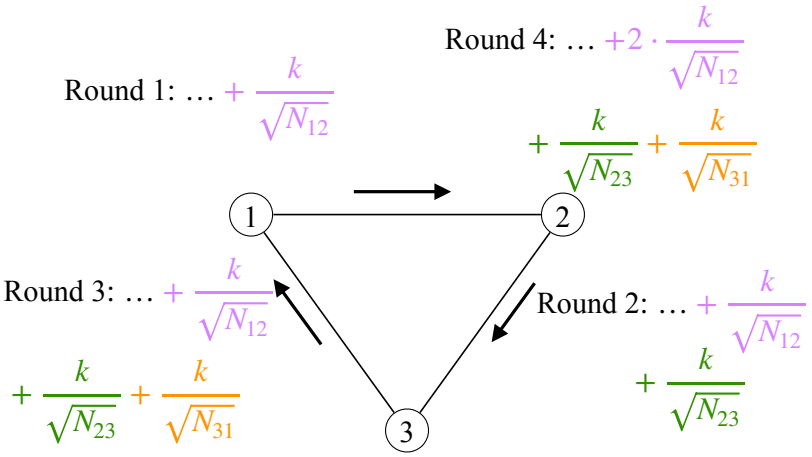


Figure 3: For coordination graphs with cycles, adding an edge exploration bonus to the messages *at every round* can lead to divergent behavior. The exploration bonuses accumulate over rounds from one node to the next (clockwise or anti-clockwise along the cycle). If the number of rounds is greater than the length of a cycle (usually true), the effective relevant exploration bonus for each edge gets compounded each time the messages loop back around. We simplify some notation for convenience, i.e., N_{ij} is a counting function for the pairwise actions of agents i and j.

the node and edge exploration strategies we have defined here are heuristic choices that we make and will evaluate empirically through an ablation. Our exploration strategies differ from the component-wise exploration of Equation (3) in the previous work that uses Var-El, because we do not consider cliques/components in the CG, only nodes and edges. We outline our approach in Algorithm 2 as well as the Max-Plus routine in Algorithm 3.

3.2 Other Differences from FV-MCTS with Variable Elimination

In this section, we briefly discuss major differences that arise from the choice of Max Plus over Variable Elimination in FV-MCTS.

3.2.1 Convergence

For graphs without cycles, Max-Plus converges to a fixed point in finitely many iterations (Pearl, 1989). For cyclic graphs, there are no such guarantees in general (Wainwright et al., 2004), but message passing on them can work well in practice (Murphy et al., 1999).

3.2.2 Agent Utilities

The Max-Plus global payoff in Equation (4) includes a utility function Q_i for each individual agent. The standard implementation of FV-MCTS with Var-El has no such individual utility (unless a node has degree 0 in the CG). If such agent utilities were known or learned *independent of the payoffs*, we would naturally use them during action coordination, with

Algorithm 2 Factored Value MCTS with Max-Plus

Require: time limit, depth, exploration constant c, state $\overline{s}$

Initialize N_i, Q_i ▷ Node statistics
Initialize N_{ij}, Q_{ij} ▷ Edge statistics
function FV-MCTS-MP($\overline{s}$, depth)
  while time limit not reached
    Simulate($\overline{s}$, depth)
  $\overline{a}^* \leftarrow$ MaxPlus(0) ▷ No exploration here
  return $\overline{a}^*$ ▷ Best joint action
function Simulate($\overline{s}$, depth)
  if depth $= 0$
    return 0
  $\overline{a} \leftarrow$ MaxPlus(c)
  $\overline{s}', \overline{r} \sim T(\overline{s}, \overline{a}), R(\overline{s}, \overline{a})$ ▷ Generative model
  $\overline{q} \leftarrow \overline{r} + \gamma \cdot$ Simulate($\overline{s}'$, depth $- 1$)
  UpdateStats($\overline{s}, \overline{a}, \overline{q}$)
function UpdateStats($\overline{s}, \overline{a}, \overline{q}$)
  for every agent i
    $N_i(\overline{s}, a_i)$ += 1
    $Q_i(\overline{s}, a_i)$ += $\frac{q_i - Q_i(\overline{s}, a_i)}{N_i(\overline{s}, a_i)}$
  for every edge $(i, j) \in \mathcal{G}(\overline{s})$
    $N_{ij}(\overline{s}, a_i, a_j)$ += 1
    $q_e \leftarrow q_i + q_j$
    $Q_{ij}(\overline{s}, a_i, a_j)$ += $\frac{q_e - Q_{ij}(\overline{s}, a_i, a_j)}{N_{ij}(\overline{s}, a_i, a_j)}$

minor modifications to the update rules of Q_i and Q_{ij}. However, in FV-MCTS we estimate all statistics from the rewards obtained during tree search with a simulated environment model; the environment model returns precisely one reward vector for each joint state-action pair.

We already account for the simulated rewards in tree search through the Q_{ij} local payoff statistics in Equation (2). We do not receive independent per-agent rewards, so utility statistics would be derived from the same information we use for the payoff statistics. Our experiments compare the benefit of these derived individual agent (node) utilities, in addition to local edge payoffs. We maintain separate statistics N_i and Q_i for the per-agent frequencies and utilities respectively and estimate them from the joint rewards during tree search; the corresponding updates are $N_i(\overline{s}, \overline{a}_i) = N_i(\overline{s}, \overline{a}_i) + 1$ and $Q_i(\overline{s}, \overline{a}_i) = Q_i(\overline{s}, \overline{a}_i) + \frac{\overline{r}_i - Q_i(\overline{s}, \overline{a}_i)}{N_{\overline{a}_i}}$ for an agent i. The results in Section 4.1 demonstrate how *including derived node utilities* enables better empirical performance.

Algorithm 3 MaxPlus Action Selection

Require: Coordination Graph $\mathcal{G}(s) = \langle \mathcal{V}, \mathcal{E} \rangle$; state node statistics N, Q; max iterations M; flags (exploration; normalization)

function MaxPlus(c)
  for $t \leftarrow 1$ to M
    $\mu_{ij}(a_j) = \mu_{ji} = 0$ for $(i,j) \in \mathcal{E}, a_i \in \mathcal{A}_i, a_j \in \mathcal{A}_j$
    for every agent i
      for all neighbors $j \in \Gamma(i)$
        Compute $\mu_{ij}(a_j)$ using Equation (5)
        if message normalization
          $\mu_{ij}(a_j) \mathrel{-}= \frac{1}{|\mathcal{A}_j|} \sum_{a_j \in \mathcal{A}_|} \mu_{ij}(a_j)$
        Send message $\mu_{ij}(a_j)$ to agent j
        if $\mu_{ij}(a_j)$ close to previous message
          break
    for every agent i
      if edge exploration
        for all neighbors $j \in \Gamma(i)$
          Compute $\mu_{ij}(a_j)$ using Equation (7)
      $q_i(a_i) = Q_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$
      if node exploration
        $q_i(a_i) \mathrel{+}= c\sqrt{\frac{\log(N+1)}{N_i(a_i)}}$
      $a_i' = \arg\max_{a_i} q_i(a_i)$
    if time limit reached
      break
  return $\overline{a}'$

3.2.3 Dynamic Coordination Graphs

As demonstrated by Yu et al. (2020), static non-adaptive coordination graphs can be quite ineffective for modeling realistic multi-agent situations like autonomous driving, where "agents' dependencies change continuously". The runtime complexity of Var-El depends on both the structure of the coordination graph and the elimination ordering; it is exponential in the induced width, which depends on the size of the largest clique during the elimination process). Recall that MCTS (and online MDP planning in general) can use computational structures that vary with the current state, even within the same tree search call. When the CG topology can change over the planning horizon for each encountered state, the likelihood of yielding CGs with larger induced width during Var-El is quite high for FV-MCTS, since no heuristic to minimize the induced width may work for all CGs. On the other hand, while approximate, Max-Plus is an anytime algorithm. It does not suffer from this issue because its complexity depends on the number of nodes and edges in the graph, which is polynomial in the CG size. Therefore, dynamic CGs can be used seamlessly in FV-MCTS with MaxPlus.

3.2.4 Memory Complexity of Statistics

Factored Value MCTS collects frequency and payoff statistics for each unique joint state encountered during tree search. Assume the same action set $\mathcal{A}$ for each agent, and a CG with $|\mathcal{V}|$ nodes (agents), $|\mathcal{E}|$ edges, and C local components or cliques. Then, the memory complexity of per-state statistics for Max-Plus is $\mathcal{O}(|\mathcal{V}||\mathcal{A}| + |\mathcal{E}||\mathcal{A}|^2)$; the first term only applies if we track per-node utilities. A popular instantiation of Var-El in the probabilistic inference literature represents the graph in terms of the cliques or local connected components C, which we follow in our implementation (Koller & Friedman, 2009). Such a representation implies that the per-state memory for Var-El statistics is $\mathcal{O}(\sum_{c \in C} |\mathcal{A}|^{|\mathcal{V}|_c} \cdot |\mathcal{V}|_c)$, where $|\mathcal{V}|_c$ is the size of local component c. For a CG that is connected (typically the case), the memory complexity for Var-El is at least $\mathcal{O}(|\mathcal{E}||\mathcal{A}|^2)$, which is the dominant term for Max-Plus, and more generally is exponential in the largest clique in the CG. *Therefore, as per our implementation, Max-Plus is more memory-efficient than Var-El.* Note that it is possible to use an alternative graph representation with Var-El (Vlassis et al., 2004). While we do not implement this version of Var-El with FV-MCTS, the issue of runtime complexity mentioned in the previous section would still hold.

3.2.5 Distributed Implementation

Unlike with Var-El, *we can execute Max-Plus in a distributed manner* by sending messages in parallel, albeit incurring additional communication complexity. Such an implementation can allow further scalability with available resources. Note that this is distinct from full decentralization wherein the agent actions can be computed independently.

3.3 On the Application to Multi-Agent POMDPs

The prior work of Amato and Oliehoek (2015) plans for multi-agent POMDPs (rather than MDPs), where the agents receive noisy observations of the true underlying state. Given that our work is for MMDPs, we briefly comment on how and to what extent our approach could be applied to an MPOMDP. We do so by referring separately to the two related but distinct ideas in the prior work: the *factored statistics* technique that applies the mixture of experts optimization to each node of the search tree and the *factored trees* technique that additionally splits joint observation histories into local histories and distributes them over the factors.

The factored statistics algorithm from Amato and Oliehoek (2015) is the primary reference point for our baseline approach of FV-MCTS with Var-El. It focuses entirely on the challenge of the exploding joint action space in a Monte Carlo Tree Search planning algorithm, which is just as pertinent for an MMDP as it is for an MPOMDP. To handle partial observability, the prior work uses POMCP, a partially observable extension of MCTS (specifically, Upper Confidence Trees) in which particle filters track and update the belief state with new observations (Silver & Veness, 2010). Our approach with Max-Plus could also be used straightforwardly with the POMCP of the factored statistics technique and applied to MPOMDPs.

On the other hand, the factored trees algorithm from the prior work seeks to also address the large number of joint observations in an MPOMDP. It does so by introducing an expert for the observation history and action component of each factor e in the coordi-

nation graph, which it implements by constructing multiple trees in parallel (one for each factor). In MMDPs there is no such thing as an observation history, since the current system state is fully observed; thus there is no equivalent variant to implement for the evaluation. Furthermore, in contrast to the prior work, our approach can accommodate dynamic or state-dependent coordination graphs where the factors can change between and within tree searches. The question of how to incorporate our unique changes from Max-Plus into factored trees to plan for MPOMDPs (and indeed, whether it is possible to do so with dynamic factors), besides the changes to action exploration and selection, is beyond the scope of our current work.

4. Experiments and Results

We used cumulative discounted return as the primary metric to evaluate our approach, Factored Value MCTS with Max-Plus (FV-MCTS-MP). Our most relevant baseline is Factored Value MCTS with Variable Elimination (FV-MCTS-Var-El). We also compared against standard MCTS (with no factorization), independent Q-learning (IQL), and a random policy. Besides measuring performance, we examined the effect of different exploration schemes on the performance of FV-MCTS-MP (as we discussed in Section 3.2), the effect of problem size on MCTS computation time, and various hyperparameter ablations. Both of our experimental domains represent a range of MMDP problems and underlying coordination graphs.[1]. All implementation and simulations are in Julia with the POMDPs.jl library (Bezanson et al., 2017; Egorov et al., 2017)

We have shown qualitatively and quantitatively that using Max-Plus rather than Var-El with MCTS results in a more scalable online planning algorithm. However, *there are many confounds in quantitatively evaluating the anytime property of our algorithm.* Our metric is the average discounted return over the episode, where the Max-Plus routine is called several times; typical anytime evaluation reports improving solution quality with more compute time for a single call to a method. MCTS itself has several hyper-parameters that affect the computation-vs-quality tradeoff, such as tree depth, exploration constant, and number of trials. With dynamic CGs as in our multi-drone delivery domain, the same Max-Plus hyper-parameters lead to different computation times. Moreover, due to the difference in how Var-El and Max-Plus represent CGs in our implementations, the statistical power of the action-value estimates for the two approaches can vary. Distinguishing this effect in performance from estimation and action selection can be a question for future work. Note that Vlassis et al. (2004) do evaluate the anytime property of Max-Plus in a one-shot decision-making domain that does not have any of the above confounds.

However, we do have some useful insights on the quantitative anytime-ness of FV-MCTS with Max-Plus. Our many ablations in this section evaluate the impact of various hyper-parameters on our approach, each of which independently influences the runtime (such as the MCTS hyper-parameters and the number of Max-Plus message passing rounds). Taken together, these individual ablations capture the anytime-ness well enough to serve as a useful starting point for practitioners. The extent to which the anytime property is ultimately useful in practice depends on the dynamics and reward structure of the MMDP,

1. Source code for experiments is available at https://sites.google.com/stanford.edu/fvmcts/

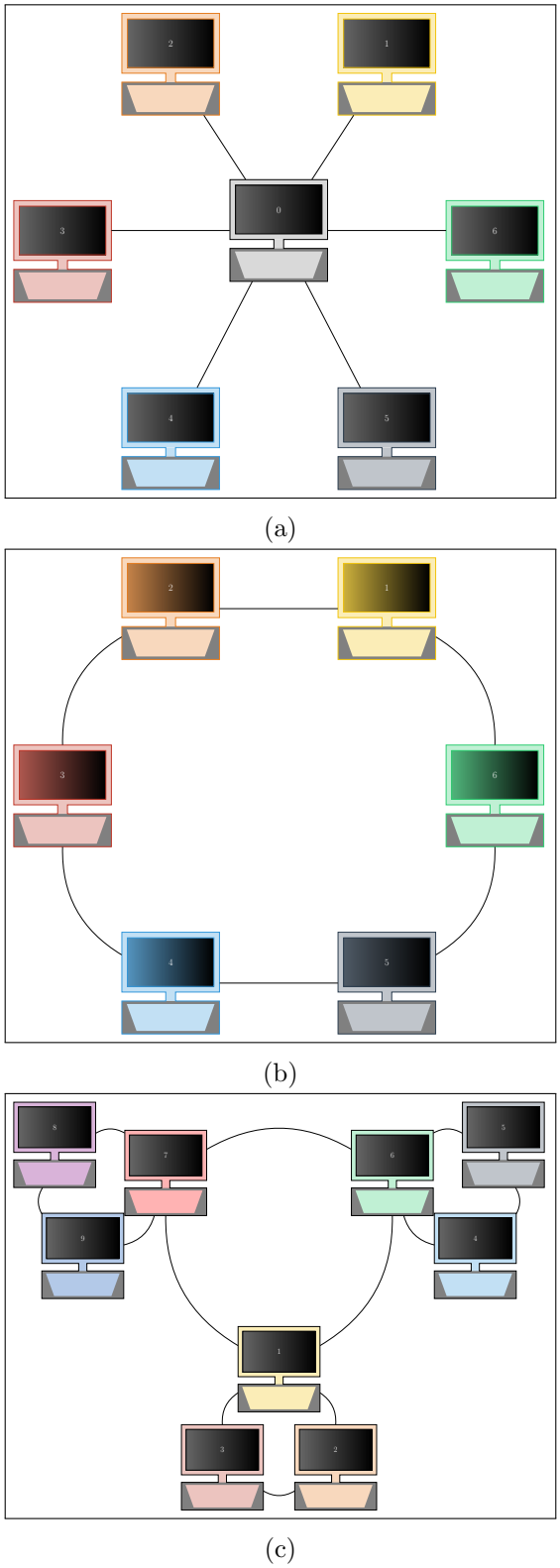


Figure 4: Our SysAdmin experimental domain with three different topologies: Star (top), Ring (middle) and Ring-of-Ring(bottom).

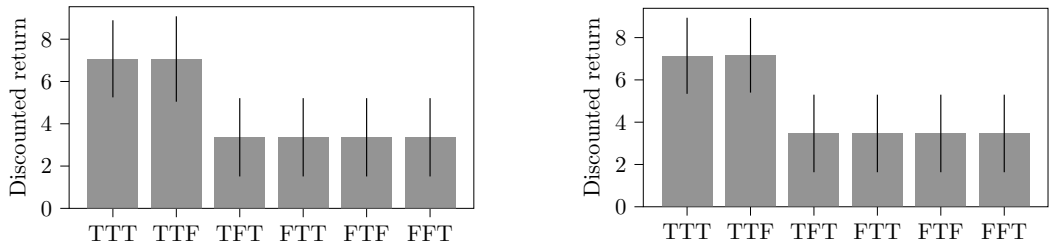


Figure 5: The performance of FV-MCTS-MP varies with different combinations of exploration strategies for the 4-agent Sysadmin on Ring (left) and Star (right) topologies. The True/False (T/F) labels correspond to Agent Utilities, Node Exploration and Edge Exploration in order, e.g. TTF implies agent utilities and only node (but not edge) exploration.

the real-time computational constraints of the application, and several other engineering design decisions.

4.1 SysAdmin Domain

Our first domain is a standard MMDP benchmark: SysAdmin (Guestrin et al., 2003). Each agent i represents a machine in a network with two state variables: Status $S_i \in \{\text{GOOD}, \text{FAULTY}, \text{DEAD}\}$, and Load $L_i \in \{\text{IDLE}, \text{LOADED}, \text{SUCCESS}\}$. A DEAD machine increases the probability that its neighbor also dies. The system gets a reward of 1 if a process terminates successfully, processes take longer when status is FAULTY, and a DEAD machine loses the process. Each agent must decide whether to reboot its machine, in which case the Status becomes GOOD and any running process is lost. The discount factor, γ used in all the experiments is 0.9. All evaluations have been averaged over 40 runs. Error bars indicate standard deviations. Figure 4 illustrates the three network topologies that we use for SysAdmin: Star, Ring, and Ring-of-Ring.

4.1.1 Exploration Schemes for FV-MCTS-MP

The three knobs affecting exploration in FV-MCTS-MP are per-agent utility, node bonus, and edge bonus. We compared the discounted return of variants that either use or ignore per-agent utilities, and use either or both bonuses. Figure 5 demonstrates that the combination of *agent utilities and only node exploration (TTF) is enough*; including the edge bonus as well (TTT) does not have much effect. All other schemes are poorer. Therefore, we used the TTF variant of FV-MCTS-MP to compare against the other baselines. The lack of a noticeable difference between some of the exploration strategies is largely to do with the small action space of the SysAdmin domain.

4.1.2 FV-MCTS-MP compared to baselines

For all three SysAdmin topologies (and corresponding fixed CGs), we varied the number of machines (agents) and compared the performance of all methods in Figure 6. With fewer agents, all MCTS methods perform similarly to each other and better than Q-Learning. However, with more agents, standard MCTS runs out of memory even on our 128 GiB RAM machine as expected for large joint action spaces. Both Factored Value MCTS variants

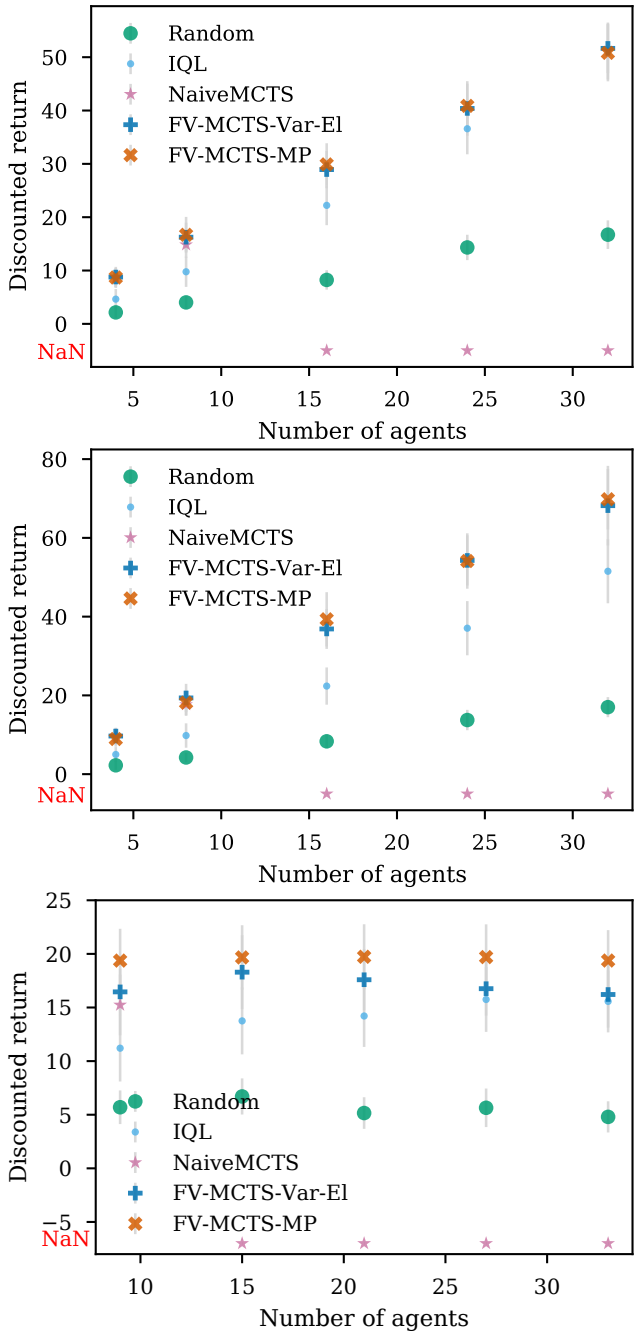


Figure 6: On SysAdmin topologies: Ring (top), Star (middle), and Ring-of-Rings (bottom), FV-MCTS with MaxPlus performs as well as or slightly better than Var-El, while being much more efficient for larger problems as in Figure 7. NaN indicates that the algorithm ran out of memory.

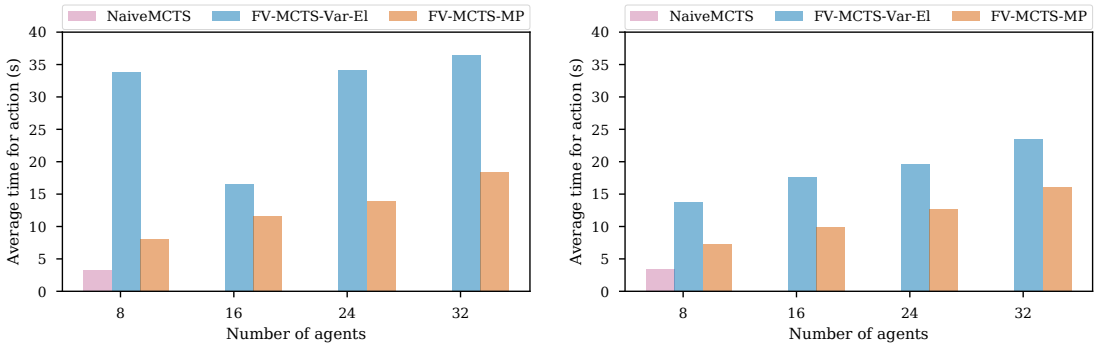


Figure 7: Runtime comparisons (lower is better) for the same tree search hyperparameters on SysAdmin with Ring (left) and Star (right) topologies. The NaiveMCTS baseline ran out of memory with more than 8 agents.

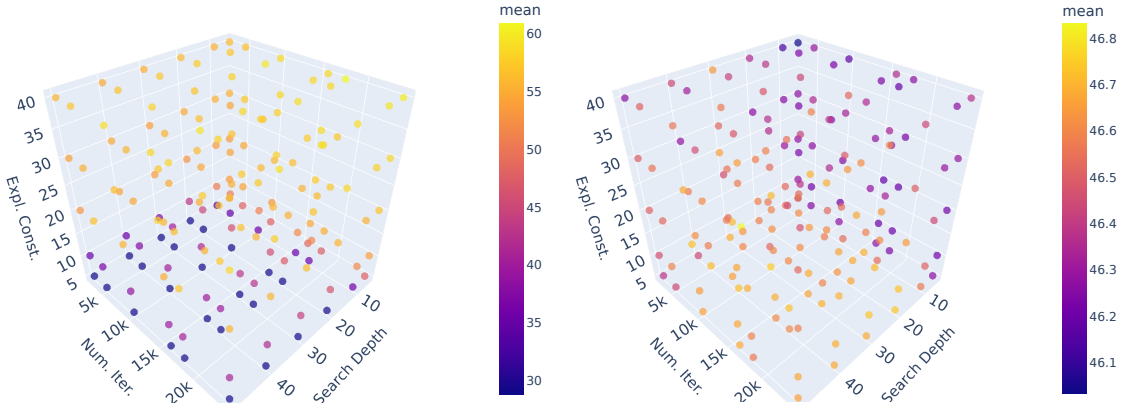
perform comparably on larger problems (on ring-of-rings, our Max-Plus variant was slightly better). However, as we discuss subsequently, FV-MCTS-Var-El is much slower than FV-MCTS-MP, e.g., taking approximately 35 s versus 16 s for 32 agents on a single-threaded implementation in the Ring topology.

4.1.3 Effect of Hyperparameters

We ran several ablation experiments by varying the exploration constant, tree search exploration depth, and number of Monte Carlo rollouts, for both FV-MCTS-Var-El and FV-MCTS-MP. We show 3D plots of the cumulative reward for both approaches on the three SysAdmin topologies: Ring (Figure 8), Star (Figure 9) and Ring-of-Ring (Figure 10). In general, the performance of either approach does not vary appreciably with tree depth d, and improves only slightly with increasing number of iterations. The exploration constant c appears to have a range of effects. For Ring SysAdmin, a higher c significantly improves Var-El but not Max-Plus, and the best combination for Var-El is significantly better than the best for Max-Plus. For Star SysAdmin, both Var-El and Max-Plus improve with increasing c. For Ring-of-Ring, Var-El shows minor improvement with increasing c, while Max-Plus does not. For both of the latter two topologies, the best combinations for both Var-El and Max-Plus are similar. These results underscore the complex effect of exploration on Factored-Value MCTS, and suggests a need for deeper analysis in future work.

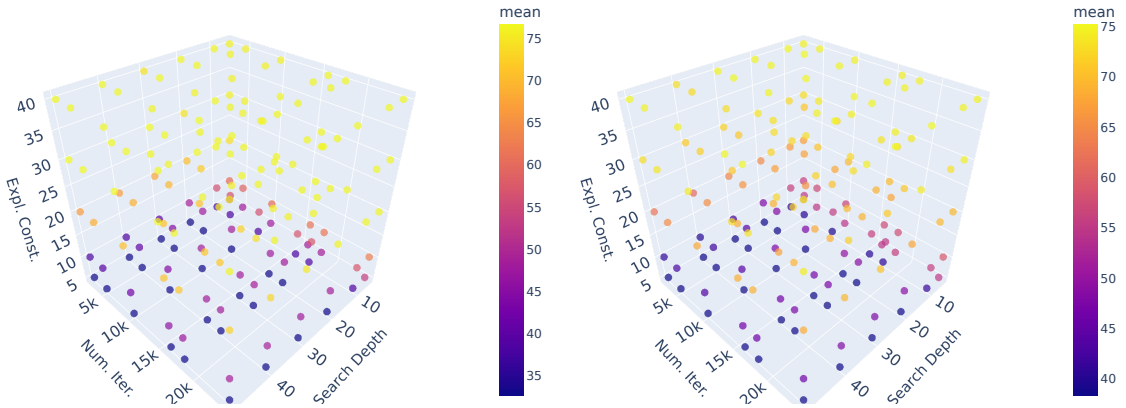
4.1.4 Computation Time

For the same tree search hyperparameters with number of iterations fixed as 16000, exploration constant as 20 and tree search depth as 20, we compared the average time taken for each action for different number of agents in the coordination graphs. For a fair comparison, we used a single threaded implementation. As demonstrated in Figure 7, we found FV-MCTS-MP to be consistently faster than FV-MCTS-Var-El. Although NaiveMCTS was faster when there were a small number of agents, it scales poorly and thus ran out of memory with an increasing number of agents.



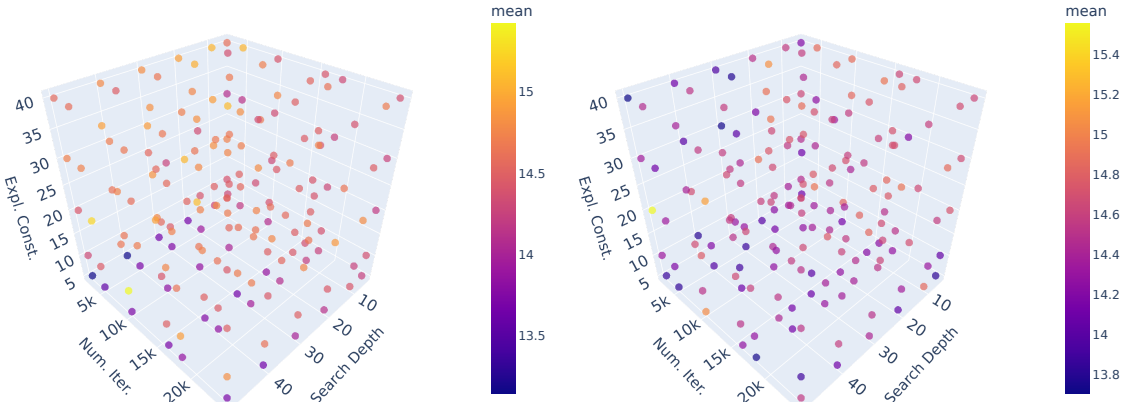
(a) VarEl (b) MaxPlus

Figure 8: Results of the hyper-parameter ablation study for 32 agents on Ring SysAdmin. For lower values of the exploration constant, Max-Plus consistently outperforms Var-El, but the relative performance is reversed for higher values. The range of mean rewards for Max-Plus is much lower than for Var-El. There is minor improvement for both with increasing number of iterations, and negligible change with increasing search depth.



(a) VarEl (b) MaxPlus

Figure 9: Results of the hyper-parameter ablation study for 32 agents on Star SysAdmin. Max-Plus slightly outperforms Var-El at lower values of the exploration constant, but the two are comparable at higher values. Both of the other hyper-parameters have negligible effects.



(a) VarEl (b) MaxPlus

Figure 10: Results of the hyper-parameter ablation study for 32 agents on Ring-of-Ring SysAdmin. The two approaches are largely comparable throughout the grid of hyper-parameter values.

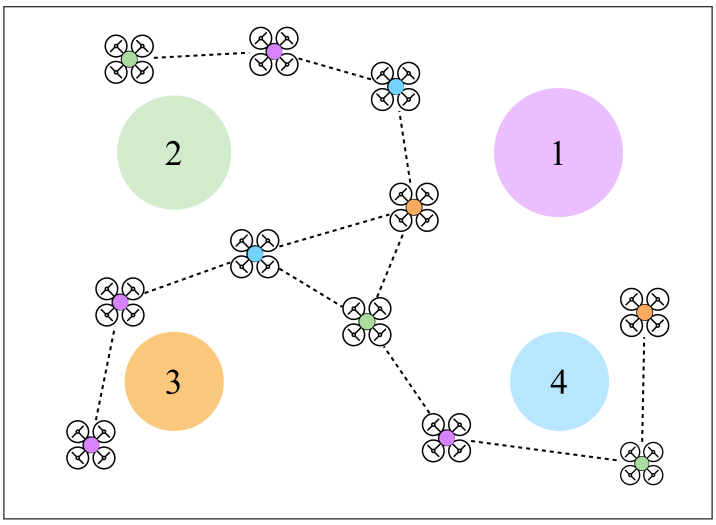


Figure 11: Multi-Drone Delivery. Dotted lines illustrate a subset of the Coordination Graph edges for the current state (for clarity, we omit some edges between drones of the same color, i.e., assigned to the same goal).

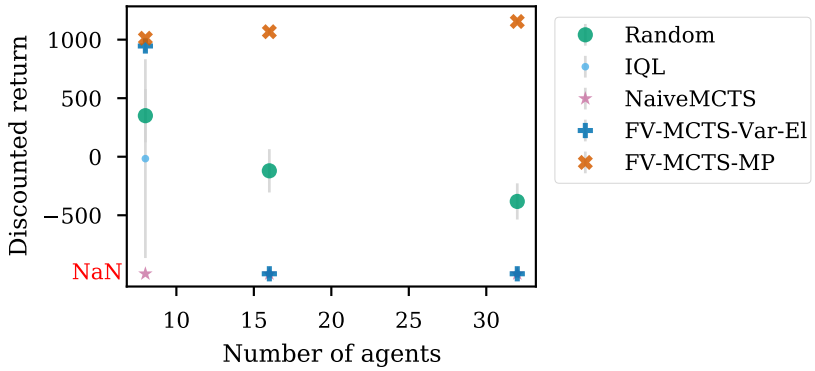


Figure 12: For Multi-Drone Delivery, FV-MCTS-MP vastly outperforms the baselines while effectively using dynamic CGs without any memory issues. NaN indicates that the algorithm ran out of memory.

4.2 Multi-Drone Delivery Domain

Besides the SysAdmin domain, previous multi-agent decision-making work has also used the Firefighter (Amato & Oliehoek, 2015) and Traffic Control (Kuyer et al., 2008) domains for benchmarking. Underneath the differing high-level descriptions, however, *the MMDP details of all three domains are very similar*: a small state space and binary action space, the degrees of most nodes in the coordination graph are independent of the total number of agents (except the hub node for Star SysAdmin), and there is no scope in any of them for dynamic or state-dependent CGs.

We introduce and use a truly distinct domain for our second set of experiments. It simulates *a team of delivery drones navigating a shared operation space to reach their assigned goal regions*. We are motivated by recent advances in drone delivery technology, from high-level routing to low-level control (Dorling et al., 2016; Lee, 2017); in particular, drones using ground vehicles as temporary modes of transit to save energy and increase effective travel range (Choudhury et al., 2019; Choudhury et al., 2021). Our domain models a key component of such drone-transit coordination: multiple drones assigned to board transit vehicles in close proximity to each other (within the same time window).

4.2.1 Domain Details

Figure 11 illustrates our Multi-Drone Delivery domain; for convenience and consistency with MMDP benchmarks we discretize everything, but MCTS could accommodate a continuous state space. Each drone starts in a randomly sampled unique cell in a grid (we use larger grids for more drones in our simulations). There are four circular goal regions, one in each quadrant, that represent a transit vehicle; each goal region has a radius and maximum capacity of drones it can accommodate, since no two drones can occupy the same grid cell (we also vary goal radius with grid size). We allocate the drones to the goal regions at random such that at least two drones target every region.

Each drone has 10 actions in total: one for moving to each of the 8-connected grid neighbors, a NO-OP action for staying in place, and a BOARD action that is only valid when the drone is inside its assigned goal region. The MMDP is episodic and terminates only when all drones have reached their goals and executed BOARD inside them, thus boarding

Agents	XY axis res.	Noise	Expl. const.	Expl. depth	Iterations
8	0.20	0.10	5	10	4000
16	0.10	0.05	10	10	8000
32	0.08	0.05	20	10	16000
48	0.05	0.02	30	10	24000

Table 1: Multi-Drone Delivery hyperparameters.

the transit vehicle and receiving a reward of 1000. Drones also receive an intermediate positive reward if they get closer to their assigned goals. Besides drone movement, the other sources of negative reward, i.e., cost, are penalties for two or more drones being too close to each other, attempting to enter the same cell (which makes them both stay in place), and attempting to board in the same goal region at the same time.

Unlike the typical MMDP domains used in prior work, *Multi-Drone Delivery motivates dynamic or state-dependent coordination graphs*; any two drones benefit from coordination only when they are close to each other. Therefore, at the current joint state, we assign a CG edge between any two drones whose mutual distance is lower than a resolution-dependent threshold. We also add edges apriori between all drones assigned to the same goal region, as they need to coordinate while boarding.

For all experiments, we set the discount factor γ to 1, the goal reaching reward to 1000.0 units, and the collision penalty to 10.0 units. Table 1 describes the full set of varying problem resolutions and MCTS hyperparameters. The average degree of the dynamic coordination graphs ranged from 2.4 for 8 agents to 11.8 for 48 agents. We averaged all evaluations over 20 runs. Unless otherwise stated, error bars indicate standard deviations.

4.2.2 FV-MCTS-MP Compared to Baselines

As with SysAdmin, we varied the number of drones (agents), discretizing the grid appropriately, and compared against all baselines (except Random) in Figure 12. We observed that FV-MCTS-Var-El and MCTS quickly ran out of memory, which is expected given the large action space per agent. Even on the problems where Var-El runs, its restriction to static CGs leads to slightly worse performance. On the other hand, FV-MCTS-MP can solve tasks even with 48 agents successfully. Moreover, even on the eight agent problem, FV-MCTS-MP is much faster, taking on average approximately 1 s instead of 40 s for FV-MCTS-Var-El for the same tree search hyperparameters. *FV-MCTS-MP scales to MMDP problem sizes that FV-MCTS-Var-El is unable to accommodate.*

4.2.3 Effect of Message Passing Rounds

In Section 4.1.3, we ran an extensive set of ablation experiments for the Monte Carlo Tree Search hyper-parameters on the SysAdmin domain. Given the computationally expensive nature of the Multi-Drone Delivery domain, here we examine only the effect of the number of message passing iterations on the performance of FV-MCTS-MP, keeping other hyper-parameters fixed. For the 8-agent instantiation of our problem, we vary the number of

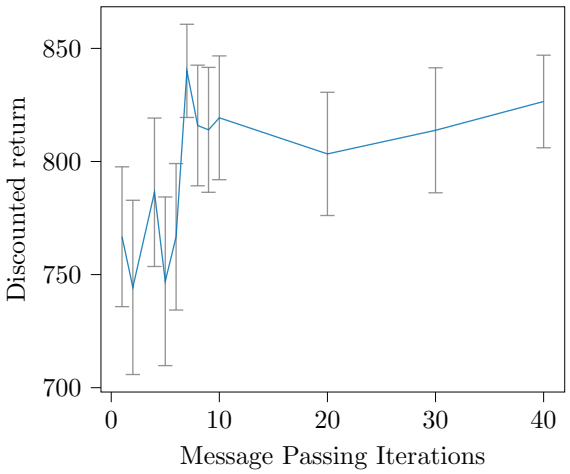


Figure 13: Effect of number of message passing iterations for 8 agents on Multi-Drone Delivery. Error bars refer to standard errors.

MaxPlus message passing iterations from 1 to 40, and compute the average discounted return of FV-MCTS-MP over 40 trials.

Figure 13 summarizes the results of our study. The performance generally improves with more iterations, which is to be expected with MaxPlus. However, the improvement is noisy and non-monotonic, which is likely due to the underlying complexity of the problem and that Coordination Graphs of the Multi-Drone Domain can have multiple cycles that change over time. We did not run similar ablations on message passing rounds with SysAdmin because the simple state and action spaces make it unlikely for there to be any variation in FV-MCTS-MP performance.

5. Conclusion

We introduced a scalable online planning algorithm for multi-agent MDPs with dynamic coordination graphs. Our approach, FV-MCTS-MP, uses Max-Plus for action coordination, in contrast to the previously introduced FV-MCTS with Variable Elimination. Over the standard SysAdmin and the custom Multi-Drone Delivery domains, we demonstrated that FV-MCTS-MP performs as well as Var-El on static CGs, outperforms it significantly on dynamic CGs, and is far more computationally efficient (enabling online MMDP planning on previously intractable problems).

Several interesting questions arise for future work. We have already summarized ideas for extensions to MPOMDPs in Section 3.3. We rely on a domain expert to pre-define the coordination graph for the problem; a predetermined CG can be particularly difficult with dynamic domains where the CG depends on the state. More work is required towards learning the dynamic coordination graph itself via interaction with the model. Similarly, extending ideas from Alpha-Zero (Anthony et al., 2017; Silver et al., 2018) would be particularly relevant for distilling the coordinated individual actions for the agents into decentralized policies (Phan et al., 2019) with FV-MCTS acting as a scalable policy improvement operator (Grill et al., 2020). Finally, a theoretical analysis of the exploration strategies and

their interaction with MaxPlus' convergence would improve our understanding of the performance. While upper confidence bound exploration as well its alternatives have been quite well studied in the context of single-agent MCTS (James et al., 2017; Kocsis & Szepesvári, 2006; Shah et al., 2020), there has not been nearly as much examination in the multi-agent case. Moreover, as investigated by Mern and Kochenderfer (2021), the empirical performance of MCTS with upper confidence bound exploration can be greatly improved even in the single agent case. Finding out similar bottlenecks for the factored case would be another important avenue for future work.

Acknowledgments

SC and JKG contributed equally in this work. PM contributed to this work during his affiliation with MIT Lincoln Laboratory. This research is supported by the Ford Motor Company and the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

References

Amato, C., & Oliehoek, F. (2015). Scalable planning and learning for multiagent pomdps. *AAAI Conference on Artificial Intelligence (AAAI)*.

Anthony, T., Tian, Z., & Barber, D. (2017). Thinking Fast and Slow with Deep Learning and Tree Search. *Advances in Neural Information Processing Systems (NeurIPS)*, 5360–5370.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control* (Vol. 1). Athena Scientific.

Best, G., Cliff, O. M., Patten, T., Mettu, R. R., & Fitch, R. (2019). Dec-MCTS: Decentralized Planning for Multi-Robot Active Perception. *International Journal of Robotics Research*, *38*(2-3), 316–337.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98.

Böhmer, W., Kurin, V., & Whiteson, S. (2020). Deep Coordination Graphs. *International Conference on Machine Learning (ICML)*, 980–991.

Boutilier, C. (1996). Planning, Learning and Coordination in Multi-Agent Decision Processes. *Conference on Theoretical Aspects of Rationality and Knowledge*, 195–210.

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Liebana, D. P., Samothrakis, S., & Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Chaslot, G. M. J.-B., Winands, M. H., van den Herik, H. J., Uiterwijk, J. W., & Bouzy, B. (2008). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, *4*(3), 343–357.

Choudhury, S., Knickerbocker, J. P., & Kochenderfer, M. J. (2019). Dynamic Real-time Multimodal Routing with Hierarchical Hybrid Planning. *IEEE Intelligent Vehicles Symposium (IV)*, 2397–2404.

Choudhury, S., Solovey, K., Kochenderfer, M. J., & Pavone, M. (2021). Efficient Large-scale Multi-drone Delivery using Transit Networks. *Journal of Artificial Intelligence Research*, *70*, 757–788.

Dechter, R. (1999). Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, *113*(1-2), 41–85.

Dorling, K., Heinrichs, J., Messier, G. G., & Magierowski, S. (2016). Vehicle Routing Problems for Drone Delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(1), 70–85.

Egorov, M., Sunberg, Z. N., Balaban, E., Wheeler, T. A., Gupta, J. K., & Kochenderfer, M. J. (2017). POMDPs. jl: A Framework for Sequential Decision Making under Uncertainty. *Journal of Machine Learning Research*, *18*(1), 831–835.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., & Munos, R. (2020). Monte-Carlo Tree Search as Regularized Policy Optimization. *International Conference on Machine Learning (ICML)*, 3769–3778.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Guestrin, C., Koller, D., & Parr, R. (2002). Multiagent Planning with Factored MDPs. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems (neurips)*.

Gupta, J. K., Egorov, M., & Kochenderfer, M. J. (2017). Cooperative Multi-Agent Control using Deep Reinforcement Learning. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 66–83.

James, S., Konidaris, G., & Rosman, B. (2017). An Analysis of Monte Carlo Tree Search. *AAAI Conference on Artificial Intelligence (AAAI)*.

Kochenderfer, M. J. (2015). *Decision Making under Uncertainty: Theory and Application*. MIT Press.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *European Conference on Machine Learning (ECML)*.

Kok, J. R., Hoen, E. J., Bakker, B., & Vlassis, N. (2005). Utile Coordination: Learning Interdependencies Among Cooperative Agents. *EEE Symposium on Computational Intelligence and Games, Colchester, Essex*, 29–36.

Kok, J. R., & Vlassis, N. (2004). Sparse Cooperative Q-Learning. *International Conference on Machine Learning (ICML)*.

Kok, J. R., & Vlassis, N. A. (2005). Using the Max-Plus Algorithm for Multi-Agent Decision Making in Coordination Graphs. *Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, 359–360.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques*. MIT press.

Kuyer, L., Whiteson, S., Bakker, B., & Vlassis, N. (2008). Multiagent Reinforcement Learning for Urban Traffic Control using Coordination Graphs. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 656–671.

Lee, J. (2017). Optimization of a Modular Drone Delivery System. *IEEE International Systems Conference*, 1–8.

Li, S., Gupta, J. K., Morales, P., Allen, R., & Kochenderfer, M. J. (2021). Deep Implicit Coordination Graphs for Multi-agent Reinforcement Learning. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Ma, A., Ouimet, M., & Cortés, J. (2020). Exploiting Bias for Cooperative Planning in Multi-agent Tree Search. *IEEE Robotics and Automation Letters*, *5*(2), 1819–1826.

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2012). Independent Reinforcement Learners in Cooperative Markov Games: A Survey Regarding Coordination Problems. *The Knowledge Engineering Review*, *27*(1), 1–31.

Mern, J., & Kochenderfer, M. J. (2021). Measurable Monte Carlo Search Error Bounds. *arXiv preprint arXiv:2106.04715*.

Murphy, K. P., Weiss, Y., & Jordan, M. I. (1999). Loopy Belief Propagation for Approximate Inference: An Empirical Study. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 467–475.

Nijssen, J. A. M., & Winands, M. H. M. (2011). Enhancements for Multi-Player Monte-Carlo Tree Search. *Computers and Games*.

Oh, K.-K., Park, M.-C., & Ahn, H.-S. (2015). A Survey of Multi-Agent Formation Control. *Automatica*, *53*, 424–440.

Oliehoek, F. A., Spaan, M. T. J., Whiteson, S., & Vlassis, N. A. (2008). Exploiting Locality of Interaction in Factored Dec-POMDPs. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 517–524.

Pearl, J. (1989). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Phan, T., Schmid, K., Belzner, L., Gabor, T., Feld, S., & Linnhoff-Popien, C. (2019). Distributed Policy Iteration for Scalable Approximation of Cooperative Multi-Agent Policies. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2162–2164.

Pynadath, D. V., & Tambe, M. (2002). The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research*, *16*, 389–423.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *International Conference on Machine Learning (ICML)*, 4295–4304.

Shah, D., Xie, Q., & Xu, Z. (2020). Non-asymptotic Analysis of Monte Carlo Tree Search. *ACM SIGMETRICS Performance Evaluation Review*, *48*(1), 31–32.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play. *Science*, *362*(6419), 1140–1144.

Silver, D., & Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2164–2172.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., & Yi, Y. (2019). QTRAN: Learning to Factorize with Transformation for Cooperative Multi-agent Reinforcement Learning. *International Conference on Machine Learning (ICML)*, 5887–5896.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2018). Value-Decomposition Networks for Cooperative Multi-Agent Learning based on Team Reward. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2085–2087.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Świechowski, M., Godlewski, K., Sawicki, B., & Mańdziuk, J. (2021). Monte Carlo Tree Search: A Review of Recent Modifications and Applications. *arXiv preprint arXiv:2103.04931*.

Van der Pol, E., & Oliehoek, F. A. (2016). Coordinated Deep Reinforcement Learners for Traffic Light Control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*.

Vlassis, N., Elhorst, R., & Kok, J. R. (2004). Anytime Algorithms for Multiagent Decision Making using Coordination Graphs. *IEEE International Conference on Systems, Man and Cybernetics*, *1*, 953–957 vol.1.

Wainwright, M. J., Jaakkola, T. S., & Willsky, A. S. (2004). Tree Consistency and Bounds on the Performance of the Max-Product Algorithm and its Generalizations. *Statistical Computing*, *14*(2), 143–166.

Yu, C., Wang, X., Xu, X., Zhang, M., Ge, H., Ren, J., Sun, L., Chen, B., & Tan, G. (2020). Distributed Multiagent Coordinated Learning for Autonomous Driving in Highways Based on Dynamic Coordination Graphs. *IEEE Transactions on Intelligent Transportation Systems*, *21*(2), 735–748.

Zerbel, N., & Yliniemi, L. (2019). Multiagent Monte Carlo Tree Search. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2309–2311.

Zhang, K., Yang, Z., & Başar, T. (2021). Multi-agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *Handbook of Reinforcement Learning and Control*, 321–384.