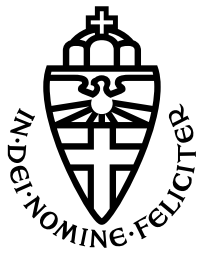
Radboud University Nijmegen



Faculty of Science

Scaling Online Planning for MPOMDPs with Many Agents

New Algorithm Variants and Addressing Scaling Issues

Thesis MSc Computing Science

Data Science Specialisation

Author:
Maris F.L. Galesloot BSc

Supervisor:
Dr. Nils Jansen

Second reader:
Dr. Tal Kachman

Second supervisors:
Dr. Sebastian Junges

Dr. Thiago D. Simão

August 12, 2024

Acknowledgements

First and foremost, I would like to thank Nils, Sebastian, and Thiago for the fruitful discussions throughout the process of this thesis. I am sincerely thankful for the the time you invested in our meetings, and the opportunity of being actively involved in the research group. I thoroughly enjoyed the inspirational working environment. Additionally, I would like to thank Nils for his daily supervision and guidance during the last months of my master's degree. I would also like to thank the ELLIS Unit Nijmegen for supporting this research as part of the ELLIS Excellence Fellowship, and Tal Kachman for being the second reader of this thesis. Finally, I would like to thank my parents, my girlfriend, my sisters, and their (new) families for their guidance along this journey, and their countless attempts at making sense of what I was doing, consequently putting the matter of this thesis into interesting perspectives. More importantly, I would like to thank them for their unconditional support.

Abstract

Online planning in single-agent partially observable environments scales to realistic environments with billions of states. However, in centralised multi-agent systems, often modelled as multi-agent partially observable Markov decision processes (MPOMDPs), the action and observation spaces grow exponentially with the number of agents. In these models, conventional online planning is largely intractable. Prior work partially mitigates this issue by capturing the inherent structure of multi-agent settings in graphs. These graphs exploit the occurrence of interactions between subsets of agents and thereby approximate the value function by a mixture of local estimates. We propose the following additions: First, we bring the locality of interactions to an online planning algorithm for MPOMDPs operating on a so-called sparse particle filter belief tree. Next, we further exploit the graph and propose general improvements to online planners in MPOMDPs with (1) a scalable approximation of the belief and (2) a scalable extension to the graphical action selection algorithms. Our algorithms show competitive performance for settings with only a few agents and outperform the state-of-the-art on benchmarks with many agents.

Contents

Chapter 1

Introduction

Sequential decision-making under uncertainty is a field of study concerning decision-making over time in stochastic environments. These stochastic scenarios encompass a wide variety of systems where the transitions of these systems are influenced by random chance, such as nature itself. Often, the system is assigned a goal, which is formulated either by introducing *costs* or *rewards*. In the latter case, the aim is to maximise the accumulated rewards given some time span. The evolution of a stochastic system can be modelled by so-called Markov chains, which are probabilistic transition systems. As we consider decision-making in stochastic environments, non-determinism is added to these systems as actions. The actions then decide which transition probabilities are incurred on the system. These models are called Markov decision processes (MDPs, Puterman (1994)). It is often considered unrealistic to assume that a decision-making agent has complete access to the full state. The state might then be considered partially observable, where the agent receives *observations* instead of the true state of the system. This enables these so-called partially observable Markov decision processes (POMDPs) to more accurately reflect the partial knowledge of the agent in the environment it is operating in.

POMDPs (Kaelbling et al., 1998) are able to model many problems, such as airborne collision avoidance (Kochenderfer et al., 2015), conservation biology (Memarzadeh and Boettiger, 2018), automated driving (Ulbrich and Maurer, 2013), and robust maintenance planning in railway systems (Arcieri et al., 2022). However, this model is intractable in general (Madani et al., 2003). Online planning is a practical approach to tackle POMDPs under limited computational and time budgets (Silver and Veness, 2010), focusing the available resources on the reachable parts of the problem and on the most promising parts of the solution space (Kocsis and Szepesvári, 2006). This has brought online planning to multiple applications, such as the defence of cyber-networks (Miehling et al., 2018), autonomous driving (Sunberg et al., 2017), and, UAV navigation (Sandino et al., 2020).

Sequential decision-making problems with multiple agents under partial observability (Messias et al., 2011), such as teams of mobile robots or autonomous surveillance systems, can be modelled by multi-agent partially observable Markov decision processes (MPOMDPs). A particular challenge that makes solving MPOMDPs even harder than POMDPs is the combinatorial number of actions and local observations that grow exponentially with the number of agents (Pynadath and Tambe, 2002). Namely, in centralised multi-agent systems, to act optimally, one must consider the observations and actions of all agents. This increased complexity makes a naive application of online planning algo-

rithms ineffective as the reachable solution space increases drastically.

To mitigate this issue, we can exploit the locality of agent interactions, often captured by so-called *coordination graphs* (Guestrin et al., 2002a). Considering a graphical structure can increase the tractability of finding solutions by considering sub-spaces of the problem (Kuyer et al., 2008; Castellini et al., 2021). In particular, Amato and Oliehoek (2015) estimate the value for the actions of pairs of agents instead of the value of the actions of all the agents. By themselves, these methods do not alleviate the difficulty of estimating the underlying state of the MPOMDP when many agents are involved. The state space of an MPOMDP typically grows very large when numerous agents are involved. Furthermore, it needs to be inferred from the observations that are returned by the environment. In an MPOMDP, this set of possible observations grows increasingly large with the number of agents involved. Additionally, determining which action to execute is complicated by the combination of local sub-problems that must be reasoned over.

Therefore, to fully scale existing planning methods, we must ensure that we

(1) scale the state estimation procedures, and,

(2) decrease the computational complexity of selecting actions in a coordination graph.

We identify these problems as two requirements for scaling up. In this thesis, we aim to develop solutions to these two issues by exploiting the given structure as much as possible.

Contributions. This thesis provides an algorithmic framework for online planning that overcomes scaling issues caused by encompassing many agents. Our work can be identified as two distinct contributions.

- First, we introduce **new algorithm variants**. We start by casting the MPOMDP to an approximation where the underlying state is fully observable. Then, we introduce two variants of the so-called *sparse particle filter tree* (PFT, Lim et al. (2020)) algorithm that exploit graphical structure in a similar fashion to Amato and Oliehoek (2015), namely FS-PFT and FT-PFT (Sect. 4.3.2). These algorithms operate on the possible set of beliefs of the agents, which makes the branching factor insensitive to the number of possible observations. Additionally, we improve on an existing algorithm by gradually increasing the allowed search space, namely FS-POMCPOW (Sect. 4.2.1).
- Second, we **address the scaling problems** by the following two sub-contributions:
    1. For the first scaling problem (chapter 5), we consider two solutions: (a) We adopt the work of Ng et al. (2002) for factored Bayesian networks to the setting of MPOMDPs (Sect. 5.1). (b) We consider a graphical structure for the MPOMDP and propose to maintain a set of approximate state estimators based on the local observations received by the agents (Sect. 5.2).
    2. For the second (chapter 6), we provide an extension to existing procedures for selecting actions to improve their scalability. Existing methods fail to find the best actions within a reasonable time when the graph is large and dense. Our method specifically aims at settings with dense coordination structures and many agents by extracting a spanning tree from the coordination graph (Sect. 6.1.3).

Our empirical evaluation shows that the various contributions improve the state-of-the-art, particularly on problems with many agents. For example, on two well-known benchmarks, we can scale up to 64 instead of 10 and 6 instead of 2 agents, respectively. Furthermore, we show that exploiting local interactions between agents can also work in arbitrary MPOMDP environments that do not necessarily contain a desired structure, making our methods applicable to general MPOMDPs.

Outline. Chapter 2 introduces the formal concepts that encompass the context of this thesis. Chapter 3 introduces the notion of *online planning* and empirical state estimation. In Chapter 4, we (1) formalise coordination graphs, (2) describe existing methods by Amato and Oliehoek (2015), and (3) introduce new algorithm variants (FS-PFT and FT-PFT) of the particle filter tree that can handle the combinatorial explosion of the problem space. Chapter 5 contains methods to combat the difficulty of state estimation in high-dimensional settings. In chapter 6, we give two methods to select actions in the algorithms introduced in chapter 4 and introduce an extension based on a *maximum spanning tree* that scales these methods to problems with many agents. Chapter 7 contains an elaborate description of our benchmark set-up (Sect. 7.1 and 7.2) and the analysis of our experimental evaluation (Sect. 7.3). In chapter 8, we discuss the relevance of this thesis in the scientific context of the related work, particularly in the field of online planning and MPOMDPs. Chapter 9 introduces possible avenues of new research and subsequently concludes this work.

Chapter 2

Preliminaries

In this chapter, we describe and define the required theoretic concepts that will be used in the later chapters of this thesis. Below we briefly introduce the notation that is used throughout this thesis.

Sets and sequences. For a finite set $\mathcal{X}$, we denote the number of elements, i.e., *cardinality* of $\mathcal{X}$ as $|\mathcal{X}|$. The empty set is $\emptyset$. The set of integers and reals are denoted as $\mathbb{Z}$ and $\mathbb{R}$, respectively. We assume the set of natural numbers $\mathbb{N}$ consists of the non-negative integers $1, 2, \ldots$. The power-set $\mathcal{P}(\mathcal{X})$ of a set $\mathcal{X}$ is the union of all possible subsets of $\mathcal{X}$, including $\mathcal{X}$ itself and the empty set $\emptyset$. A sequence $(x_0, x_1, \ldots, x_t)$ is an ordered collection of elements, where $x \prec y$ indicates that x precedes y, i.e., $x_0 \prec x_1 \prec \ldots \prec x_t$. We often compress a sequence into a subscript, writing $x_{0:t} = (x_0, x_1, \ldots, x_t)$.

Probabilities. A discrete probability distribution over a finite set $\mathcal{X}$ is a function $p\colon \mathcal{X} \to [0, 1]$. A *proper* probability distribution satisfies the condition $\sum_{x \in \mathcal{X}} p(x) = 1$. The *simplex*, i.e., set of probability distributions over $\mathcal{X}$, is denoted as $\Delta(\mathcal{X})$. We write the expectation of a random variable $\mathcal{X}$ under variable y as $\mathbb{E}_y[\mathcal{X}]$. The Dirac delta distribution δ is a distribution with its mass centred around a single point. Its value is zero everywhere except said point, and its integration satisfies $\int_{\infty}^{-\infty} \delta(x)dx = 1$. The Kronecker delta function δ_{ij} maps two integers i, j to one if and only if they are equal, and zero otherwise:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

Vectors. We denote a vector of elements as $\vec{x} = \langle x_0, x_1, \ldots, x_{|\vec{x}|} \rangle$, where $|\vec{x}|$ is the number of elements of the vector. To retrieve an individual element x_i, we index the vector as in $\vec{x}_i = x_i$, with consequently, $x_i \in \vec{x}$.

2.1 Decision Making Under Uncertainty

Automated decision-making is a subject of study that is considered by many researchers. In order to do so, a set of formal frameworks have been defined that allow us to model the concept of decision-making over time. When we write time, we often imply the abstraction of real-time to discrete intervals or time steps. Commonly, we denote time with the letter $t \in \mathbb{N}$. Then, the order of the running process or system is defined by

a discrete-time clock $t = 1, 2, \ldots$. Although we do not study continuous-time models in this thesis, extensions to model real-time systems exist (Bradtke and Duff, 1994; Qiu and Pedram, 1999).

2.1.1 Single-Agent Sequential Decision Making

We start with a standard model for sequential decision-making with a single agent. Even though this model is the simplest form of reasoning about decision-making, its capabilities for modelling decision-making is large enough to cover a wide range of problems in which the state can be fully observed (Puterman, 1994).

Definition 1 (MDP). The Markov decision process (MDP) is defined by the tuple $\mathcal{M} = \langle \mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$, with:

- $\mathcal{S}$, a finite set of states with initial state distribution b_0,
- $\mathcal{A}$, a finite set of actions,
- $\mathcal{T}$, transition probability function defined as $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ specifying the probability of transitioning to new state $s' \in \mathcal{S}$ given the previous state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$,
- r, reward function $r \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ specifying the immediate reward given the state s and action a, and,
- γ, the discount factor balancing the effect of short and long-term rewards.

States and transitions. In order to model the dynamics of a system or environment, we need a definition of a state. A state s_t at time t from the set of states $s_t \in \mathcal{S}$ entails everything that forms the essence of a system at some perceived moment in time. As time passes, the dynamics influence the state of the model. These transitions can be subject to stochastic influences, such as nature itself. To reason about decision-making in environments with aleatoric uncertainty, we define that the state transitions of the model are subject to probability distributions (Badings et al., 2023). In MDPs, which are transition models with non-determinism in the form of actions, the transition probabilities are dependent on the resolving actions.

Markov property. Markov models describe how the state of a stochastic system evolves over time. One of the key principles of these models is the Markov property. A system is *Markovian* if it is defined from the current state in time and does not depend on the history of the system. Intuitively, the current state is a sufficient representation of the system at that perceived moment in time. An MDP is Markovian if the probability distribution of a next state s_{t+1} satisfies $\Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = \Pr(s_{t+1} \mid s_t, a_t) = \mathcal{T}(s_{t+1} \mid s_t, a_t)$. The Markov property is useful because it implies that the current state s_t entails enough information to pick an optimal action a_t, as the transition probability remains equivalent regardless of the history of the system (Wiering and Van Otterlo, 2012).

Accumulating rewards. In an MDP, we aim to maximise the accumulated rewards as given by the reward function r over time. More formally, we aim to maximise the cumulative discounted reward from time t onwards, commonly denoted as the *return* R_t:

$$R_t = \sum_{t}^{H} \gamma^t r_t, \tag{2.2}$$

where $\gamma \in (0, 1]$ is the discount factor that gives greater weight to short-term rewards, r_t the reward at time t given by the state and action at time t, and H is the planning horizon (Sutton and Barto, 1998). Typically, we evaluate algorithms with the return $R = \sum_{t=0}^{H} \gamma^t r_t$ achieved, starting at time $t = 0$. We distinguish two types of tasks based on the planning horizon and discount factor. For *finite horizon* optimisation tasks, we have a planning horizon $H \in \mathbb{N}$ and a discount factor of $\gamma = 1$ for *undiscounted* and $\gamma \in (0, 1)$ for *discounted* problems, respectively. For *infinite horizon* tasks, the planning horizon $H = \infty$ is infinite. In this case, discounting $\gamma \in (0, 1)$ is employed to ensure that optimising the return remains tractable, i.e., does not reach infinity.

Policies and values. We maximise the return by finding an optimal mapping from states to actions. We call this mapping a policy $\pi(a \mid s)$ and indicate the policy that maximises the value, the *optimal policy* by $\pi^*(a \mid s)$. A *deterministic policy* is a function $\pi \colon \mathcal{S} \to \mathcal{A}$ that outputs an action given an input state. Conversely, a *stochastic policy* is a function $\pi \colon \mathcal{S} \to \Delta(\mathcal{A})$ that gives a distribution over the action space given an input state. The *value function* $V \colon \mathcal{S} \to \mathbb{R}$ indicates quality of a state in the MDP in term of expected return. The value V^π of a state $s \in \mathcal{S}$ under a policy π is the expected return under policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[R_t \mid s \right] = \mathbb{E}_\pi \left[\sum_{t}^{H} \gamma^t r(s_t, \pi(s_t) \mid s_t = s \right]. \tag{2.3}$$

Additionally, the *action-value function* $Q(s, a)$ or Q-function $Q \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of a state-action pair is the value, i.e., expected return, of state $s \in \mathcal{S}$ under policy π given that the action $a \in \mathcal{A}$ is taken: $Q^\pi(s, a) = \mathbb{E}\left[V^\pi(s') \mid s' \sim \mathcal{T}(s, a) \right]$, by sampling a transition from the transition probability function. More formally:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t}^{H} \gamma^t r(s_t, a_t) \mid s_t = s, a_t = a, a_{t+1} = \pi(s_{t+1}) \right]. \tag{2.4}$$

The relation between Q and V is bidirectional, and we can find V from Q by $V(s) = \max_a Q(s, a)$. The optimal value V^* of the optimal policy π^* satisfies the Bellman optimality equation:

$$V^* = \mathcal{B} V^*, \tag{2.5}$$

where $\mathcal{B}$ is the Bellman backup operator as given by:

$$\mathcal{B} V^*(s) = \max_{a \in \mathcal{A}} \left[\sum_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, a) \left(r(s, a) + \gamma V^*(s') \right) \right], \tag{2.6}$$

Subsequently, the optimal policy π^* can be recovered from the optimal value function as:

$$\pi^*(s) = \arg\max_a V^*(s) = \arg\max_{a \in \mathcal{A}} \left[\sum_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, a) \left(r(s, a) + \gamma V^*(s') \right) \right]. \tag{2.7}$$

The optimal value V^{π^*} under the optimal policy π^* then satisfies:

$$V^{\pi^*}(s) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, \pi^*(s)) \left(r(s, \pi(s), s') + \gamma V^{\pi^*}(s) \right). \tag{2.8}$$

Additionally, from the Q-function, we can find the optimal policy by extracting the maximal action at every state by:

$$\pi^*(s) = \arg\max_a Q^*(s, a).$$

The benefit of the Q-function over the value function is, on top of giving value to individual actions, that it does not require a summation over the transition dynamics to find the optimal action.

We can determine optimal policies by applying various methods, such as value or policy iteration (Puterman, 1994). Because these methods rely on the exact transition probabilities, they require an explicit version of the true model and are aptly named *model-based*. On the other side, we can use Q-learning, which is *model-free*. It is a method for the *reinforcement learning* problem, in which a policy is to be found purely from experience. It does not require the specifics of the transition dynamics:

$$\begin{aligned} Q^+(s, a) &\leftarrow Q(s, a) + \alpha \cdot \left(r + \gamma \cdot \max_a Q(s', a) - Q(s, a) \right) \\ &= (1 - \alpha) \cdot Q(s, a) + \alpha \left(r + \gamma \cdot \max_a Q(s', a) \right) \end{aligned} \tag{2.9}$$

where $\alpha \in [0, 1]$ is the learning rate that gives weight to the updates, and s' is a possible next state given that s was the previous state. Thus, this update considers the *temporal difference* between the value of the current and next state. Note that in deterministic models, where the transition function $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is deterministic, this is a much easier learning target as the difference in value is not subject to stochasticity, and thus the target is stationary. Q-learning in the tabular setting is guaranteed to converge to the optimal policy given that every state-action pair is visited infinitely in the limit and α is decreased appropriately (Wiering and Van Otterlo, 2012). In general, finding optimal policies for MDPs that maximise the expected return in either the finite or discounted infinite horizon is considered tractable in general as it requires polynomial time (Papadimitriou and Tsitsiklis, 1987).

2.1.2 Partial Observability

In the real world, the information supplied to the agent may be noisy or incomplete. Examples include sensors with limited capacity or noisy signals, latency, and natural stochastic tendencies of the components of the agent or the perception of the environment.

In order to accurately model these systems, we assume that the agent cannot directly observe the full state. We call these systems partially observable and extend the MDP with an observation model and the probabilities of these observations occurring. An observation can be seen as a noisy representation of the true state, a part of the whole state,

or any statistic that represents some information on the true state. What information the agent actually does and does not observe is a modelling or environment design choice.

We define the partially observable model as an extension of the MDP (Def. 1).

Definition 2 (**POMDP**)**.** The partially observable Markov decision process (POMDP) as an extension of the MDP in Def. 1 is defined by the tuple $\mathcal{M} = \langle \mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma, \Omega, \mathcal{O} \rangle$, with:

- $\mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma$ as defined in the MDP in Def. 1,
- Ω, a finite set of observations, and,
- $\mathcal{O}$, the observation model $\mathcal{O}(o \mid s', a) \colon \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ that specifies the probability of observing o in state s' given action a.

Observations. In a POMDP, we receive observations of the state with some probability. As the state is no longer fully observable, we cannot define a policy as a mapping from states to actions. Since we only receive observations of the state, we have to reason about arbitrary-length histories of observations in order to plan optimally. An action-observation history $\vec{h}_t = (\vec{a}_0, \vec{o}_1, \vec{a}_1, \vec{o}_2, \ldots, \vec{a}_{t-1}, \vec{o}_t) \in (\Omega \times \mathcal{A})^{t-1} \times \Omega$ is the recorded history of previous actions taken and observations received at time $t \in \mathbb{N}$. As one can imagine, in infinite horizons, this history can grow infinitely large. Fortunately, we can capture the history of actions and observations as a probability distribution over the set of states.

Definition 3 (**Belief**)**.** A belief is a probability distribution over the set of states induced by the initial state distribution b_0 and the action-observation history:

$$b(s) \triangleq \Pr(s \mid b_0, a_0, o_1, a_1, \ldots, a_{t-1}, o_t). \tag{2.10}$$

Additionally, the set of beliefs $\mathcal{B} \triangleq \Delta(\mathcal{S})$ is the probability simplex over the set of states.

We denote the probability that state s is assigned by belief b as $b(s)$. The belief distribution follows the standard rules for probabilities, i.e., $\sum_{s \in S} b(s) = 1$ and $0 \leq b(s) \leq 1$. Given Def. 3, a history h can be compressed into the belief distribution $b \in \Delta(\mathcal{S})$, which is a sufficient statistic of the history (Kaelbling et al., 1998). That is, the belief $b(s)$ over a state s is $\Pr(s \mid b_0, h)$, i.e., the probability of that state given the initial belief b_0 and recorded history h. The belief distribution is aptly named a belief state.

Values and policies for POMDPs. As in MDPs, we can maximise the infinite-horizon cumulative discounted reward or *return* $R_t = \sum_t^\infty \gamma^t r_t$, where r_t is the reward given by the state and action at time $t \in \mathbb{N}$. Because we cannot rely on observing the state, optimal action choices depend on the action-observation history, which can be compressed into the belief.

Then, a deterministic policy $\pi(a \mid b) \colon \mathcal{B} \to \mathcal{A}$ is a mapping from history, i.e., the belief, to actions. Additionally, a stochastic policy $\pi(a \mid b) \colon \mathcal{B} \to \Delta(\mathcal{A})$ is defined in a similar fashion. The Q-value of a history h_t with corresponding belief b_t at time t under a policy π, i.e., the Q-function $Q \colon \mathcal{B} \times \mathcal{A} \to \mathbb{R}$ is the *expected return* given action a_t:

$$Q^\pi(h_t, a_t) = Q^\pi(b_t, a_t) = \mathbb{E}_\pi \left[\sum_t^\infty \gamma^t r(s_t, a_t) \mid a_{t+1} = \pi(b_t) \right]. \tag{2.11}$$

Similarly to the fully observable case, the value function $V\colon \mathcal{B} \to \mathbb{R}$ for a history can be recovered from the Q-function for that history by retrieving the value of the maximal action.

Belief Update

We have a prior distribution for the belief given by the initial state distribution $b_0 \in \Delta(S)$ of the model. The belief evolves over time, given that the system changes state and returns observations for each action taken. The belief distribution b_t can be updated by applying Bayes' rule, resulting in the successor belief state b_{t+1} as the posterior distribution given the prior belief b_t and the likelihood of the received observation o after taking action a:

$$\begin{aligned}
\mathcal{BU}(a, o, b)(s') &= \Pr(s' \,|\, a, o, b) \\
&= \frac{\Pr(o \,|\, s', a, b) \cdot \Pr(s' \,|\, a, b)}{\Pr(o \,|\, a, b)} \\
&= \frac{\Pr(o \,|\, s', a) \cdot \sum_{s \in \mathcal{S}} \Pr(s' \,|\, a, b, s) \cdot \Pr(s \,|\, a, b)}{\Pr(o \,|\, a, b)} \\
&= \frac{\mathcal{O}(o \,|\, s', a) \cdot \sum_{s \in \mathcal{S}} \mathcal{T}(s' \,|\, a, s) \cdot b(s)}{\Pr(o \,|\, a, b)},
\end{aligned} \tag{2.12}$$

where $\Pr(o \,|\, a, b)$ is a normalisation constant (Spaan, 2012) or marginal probability ensuring $\forall_{b \in \mathcal{B}} : \mathcal{BU}(a, o, b)(\cdot)$ are valid distributions, defined by:

$$\Pr(o \,|\, a, b) = \sum_{s \in \mathcal{S}} \mathcal{O}(o \,|\, s', a) \cdot \sum_{s \in \mathcal{S}} \mathcal{T}(s' \,|\, a, s) \cdot b(s). \tag{2.13}$$

The complexity of the belief update is dependent on the cardinality of the set of states. The denominator is quadratic in the number of states but only needs to be computed once. The nominator is of complexity $\mathcal{O}(|S|)$, but we call the belief update on the enumeration of all the possible successor states $s' \in \mathcal{S}$. Concluding, the belief update is quadratic in the number of states of the system $\mathcal{O}(|S|^2)$.

Belief-state MDP. Here, we show how a POMDP can be defined as an MDP with a state space that consists of the set of beliefs $\mathcal{B}$. We use the notion of a belief-state $b \in \mathcal{B}$ and the definition of the belief update $\mathcal{BU}$ to formalise the *belief-MDP* for a POMDP, which is an MDP with a continuous state-space.

Definition 4 (Belief-MDP). For a POMDP $P = \langle \mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma, \Omega, \mathcal{O} \rangle$; the belief-state Markov decision process (belief-MDP) (Cassandra et al., 1994) is defined by the tuple $\mathcal{M}_P = \langle \mathcal{B}, b_0, \mathcal{A}, \tau, \rho, \gamma \rangle$, with:

- $\gamma, \mathcal{A}$ as in the the POMDP P,
- $\mathcal{B}$, the set of belief states, i.e., the belief simplex $\mathcal{B} \triangleq \Delta(\mathcal{S})$ (Def. 3), with initial state b_0,
- τ, transition probability function $\tau(b' \,|\, b, a)\colon \mathcal{B} \times \mathcal{A} \to \Delta(\mathcal{B})$ over the belief-states, defined as:

$$\tau(b' \,|\, b, a) = \sum_{o \in \Omega} \mathbf{1}(b' = \mathcal{BU}(a, o, b)) \cdot \Pr(o \,|\, a, b), \tag{2.14}$$

  where $\mathcal{BU}(a, o, b)$, $\Pr(o \,|\, a, b)$ as in Eq. (2.12) and (2.13), respectively.

- ρ, belief-based reward function $\rho(b, a) \colon \mathcal{B} \times \mathcal{A} \to \mathbb{R}$ specifying the immediate reward given the belief-state b and action a defined by:

$$\rho(b, a) = \sum_{s \in \mathcal{S}} b(s) \cdot r(s, a). \tag{2.15}$$

Although we have removed the layer of partial observability from the problem by introducing Def. 4, we had to introduce a continuous state-space that has a potentially uncountably infinite size. POMDP planning is considered difficult. Infinite horizon planning in POMDPs is undecidable (Madani et al., 1999), and finite horizon return maximisation is PSPACE-complete (Papadimitriou and Tsitsiklis, 1987). Although finding solutions for (typically uncountably) infinite MDPs is non-trivial (Ahmadi et al., 2021), we will see later on in this thesis that this formulation can help to find suitable approximate algorithms for solving POMDPs.

2.2 Multi-Agent Systems

Multi-agent systems (MAS), or more specifically multi-agent decision problems (MADP), are a set of systems that encompasses models for decision-making under uncertainty for multiple agents (Pynadath and Tambe, 2002; Oliehoek and Amato, 2016). Most commonly, this set involves the extension of the earlier defined MDP and POMDP models for multiple decision-making agents.

2.2.1 Cooperation

In this thesis, we consider cooperative agents that aim to achieve a shared goal. Contrastively, various – often game-theoretic – frameworks exist for adversarial or competitive domains, such as stochastic games (Shapley, 1953; Mertens and Neyman, 1981). The clear distinction between cooperation and competition between agents is modelled by the reward function. In the cooperative case, the agents receive a single shared reward signal that gives an indication of the value of a state given a joint action. Contrarily, adversarial agents receive individual rewards that depend on the actions of other agents and aim to beat the other agents in the system in order to maximise individual return.

2.2.2 Centralised Agents

Centralised multi-agent systems are systems containing decision-making agents that can freely communicate their local state – or observations – and actions. The term "centralised" arises from the fact that there is, in theory, no distinction between the free communication of individual agent controllers and the deployment of a centralised controller that decides on actions and manages the individual components of the system. We extend the notion of fully and partially observable decision-making domains to multi-agent systems in the following definitions, respectively.

Definition 5 (**MMDP**)**.** The multi-agent Markov decision process (MMDP) is defined by the tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$, with:

- $\mathcal{I}$, a finite set of agents of size $n \in \mathbb{N}$,
- $\mathcal{S}$, a finite set of states with initial state distribution b_0,

- $\mathcal{A} = \times_{i \in \mathcal{I}} \mathcal{A}_i$, a finite set of actions consisting of the Cartesian product of individual action spaces for each agent $i \in \mathcal{I}$,
- $\mathcal{T}$, transition probability function $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ specifying the probability of new state s' given the previous state s and joint action $\vec{a} = \langle a_1, \ldots, a_n \rangle$,
- r, reward function $r \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ specifying the immediate total reward given state s and joint action $\vec{a}$, and,
- γ, the discount factor.

Note that the rare case of an MMDP with a single agent is synonymous with an MDP. Similarly to the multi-agent extension of the MDP, we extend the partially observable model to the multi-agent case as follows.

Definition 6 (**MPOMDP**)**.** The multi-agent partially observable Markov decision process (MPOMDP) is defined by the tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma, \Omega, \mathcal{O} \rangle$, with:

- $\mathcal{I}, \mathcal{S}, b_0, \mathcal{A}, \mathcal{T}, r, \gamma$ as defined in the MMDP in Def. 5,
- $\Omega = \times_{i \in \mathcal{I}} \Omega_i$, a finite set of observations consisting of the Cartesian product of the individual observation space for each agent $i \in \mathcal{I}$, and,
- $\mathcal{O}$, the observation model $\mathcal{O}(\vec{o} \,|\, s', \vec{a}) \colon \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ that specifies the probability of observing joint observation $\vec{o}$ in state s' given joint action $\vec{a}$.

The action space of an MMDP grows exponentially with the number of agents n. For an MPOMDP, both the size of the action and observation space suffer from this combinatorial explosion. Thus, solving these problems exactly becomes increasingly intractable for large n.

As communication is free and the components are identical, these multi-agent frameworks can be reduced to their single-agent counterparts (Pynadath and Tambe, 2002). In order to do so, the vector of per-agent actions and observations are concatenated and treated as a single action and a single observation in the set of actions and observations of the POMDP respectively. Consequently, solution methods for MDPs and POMDPs are also applicable to their centralised multi-agent variants. Additionally, the definitions of the value functions and policies are inherited from the single-agent models.

Therefore, we do not repeat the definitions of policies and value functions from Sect. 2.1.1 to 2.1.2, as they are the same, apart from that they range over a single action a and observation o instead of the joint versions $\vec{a}$ and $\vec{o}$ in the centralised multi-agent models, respectively. Furthermore, we note that there is overlap between the definitions of the single-agent and multi-agent models overlap, e.g., $\mathcal{T}$ is in both the (PO)MDP and M(PO)MDP. From this point forward, we override these symbols and let them only refer to their definitions in the multi-agent models, namely the MMDP and MPOMDP.

The distinction between the multi-agent and the single-agent case is useful as systems with multiple agents often exhibit structure, either in the coordination of actions, the decomposition value in the system, or a combination thereof. Additionally, the action and observation spaces are *factored*, as they are comprised of the product of individual actions and observations, respectively.

2.2.3 Decentralised Agents

If the agents cannot observe each other's observations, i.e., they are not able to share or communicate these, the problem becomes decentralised. For completeness's sake, we define these classes of multi-agent systems below.

Definition 7 (**Dec-POMDP**)**.** The decentralised partially observable Markov decision process (Dec-POMDP) is defined by the tuple $\mathcal{M} = \langle I, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, where the components are identical to those of the MPOMDP in Def. 6 but observations are only observed locally per agent.

Contrary to the centralised – shared – observations of a MPOMDP, observations are not shared in a Dec-POMDP. Consequentially, policies are mappings from individual action-observation histories, and the joint policy $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ is a vector of n individual policies $\pi_i \colon (\Omega_i \times \mathcal{A}_i)^* \times \Omega_i \to \mathcal{A}_i$. If partial observability is only in the form of decentralisation, such that each agent has full observability of its local state, the problem transforms to a decentralised MDP.

Definition 8 (**Dec-MDP**)**.** The decentralised Markov decision process (Dec-MDP) is a jointly observable Dec-POMDP (Oliehoek and Amato, 2016), where the state is perfectly observed by combining the agent-wise observations.

Each agent in a Dec-MDP can asses their local state perfectly but cannot observe the state of other agents.

2.2.4 Complexity Classes

In Table 2.1, we see the complexity classes of solving multi-agent decision problems with general (implicit) or free (explicit) communication given three levels of partial observability.

Observability	General Communication	Free Communication
Full	MMDP (P)	MMDP (P)
Joint Full	Dec-MDP (NEXP)	MMDP (P)
Partial	Dec-POMDP (NEXP)	**MPOMDP** (PSPACE)

Table 2.1: Effect of communication possibilities on model classes and complexity with varying observability, from Kochenderfer (2015). Full observability means that agents can see both the state and the other agents. Joint Full means only local states are observed.

In this thesis, we focus on systems with partial observability with free communication, which allow for a centralised controller paradigm, as in MPOMDPs.

Chapter 3

Simulators & Tree Search

In the previous section, we have seen various models for sequential decision-making. The drawback of these classes of models is in the complexity of the algorithms that find their exact solutions. In this section, we will see how we can approximate the optimal value of these models by Monte Carlo simulation. Although the methods outlined in this section are not guaranteed to find the exact optimal solution, they tend to have good convergence properties and can result in close approximations of the optimal value (Browne et al., 2012).

Algorithm 1 Online planning methodology

procedure EXECUTE(b_0)
  $\bar{b} \leftarrow b_0$
  repeat
    $\vec{a}^{\#} \leftarrow \text{SEARCH}(\bar{b})$ ▷ Find an action $\vec{a}^{\#}$ from belief $\bar{b}$
    $\vec{o}, r \leftarrow \mathcal{G}(\cdot \mid \vec{a}^{\#})$ ▷ Execute step in real environment.
    $\bar{b} \leftarrow \text{UPDATE}(\bar{b}, \vec{a}^{\#}, \vec{o})$ ▷ Update belief with $\vec{a}^{\#}$ and received $\vec{o}$.
  until INTERRUPTED
end procedure

Online planning. Online planners combat the complexity of large systems by interleaving planning and execution. Here, we introduce this concept in the context of MPOMDPs (Def. 6), but the framework is more general. Before executing an action in the true environment, an online planner performs a forward search in the set of states reachable from the current belief, incrementally building a look-ahead tree known as a *search tree*. It does so without explicit knowledge of the transition, reward, and, in some cases, the observation model by using a generative model $\mathcal{G}\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \Omega \times \mathbb{R}$ of the dynamics (Kearns et al., 2002), i.e., a *simulator*, with $s', \vec{o}, r \leftarrow \mathcal{G}(s, \vec{a})$. In that sense, online planning lies between the *model-free* and *model-based* paradigms. We do require a model in the form of a simulator, i.e., sample access to the state dynamics and observation function, but we do not require the exact probabilities.

After a predefined amount of computational budget is spent on the forward search, the planner executes an action $\vec{a}^{\#} \in \mathcal{A}$ and receives an observation $\vec{o} \in \Omega$, updates the global belief $\bar{b} \in \mathcal{B}$ given this action and received observation, and then repeats the process until some predefined end. Algorithm 1 summarizes the iterative online planning procedure. Planning starts from the initial belief b_0 for the model considered, which

is assigned to the belief $\bar{b}$ maintained during the episode. This belief is passed to the search algorithm, as searching is only performed in the reachable regions as given from the current belief. The simulator $\mathcal{G}$ is called without a state in order to demonstrate that states are not assumed observable. In practice, the episode is executed from some random initial state as defined by the environment, and this state is updated by the simulator sequentially. Moreover, note that we leave the update of the trees implicit here for generality. The computational budget is spent primarily in Line 4 of Algorithm 1, where in our experimental evaluation, we can restrict both the maximum time spent searching and the maximum number of SIMULATE calls, and the search is interrupted by violation of either constraint. In practice, the computational budget allowed would most probably be defined by the planning time allowed per step. An example of an interrupted signal can originate from an end-of-episode signal, e.g., receiving a terminal state from the simulator.

3.1 Monte Carlo Tree Search

In this section, we will introduce a Monte Carlo planning technique that incrementally builds a search tree. Monte Carlo planning techniques are a class of online planning algorithms that make use of the simulator $\mathcal{G}$ to search for the best actions given the reachable problem space defined by the current state of the system. In partially observable settings, the current state of the system is not observable. Thus, a search is defined on the current belief instead of the true state.

3.1.1 Bandit Algorithms in Monte Carlo Planning

The use of Monte-Carlo planning requires us to select actions that are either deemed optimal or enable us to explore the reachable regions of the problem space. The Monte Carlo tree search (MCTS) algorithm consists of building a search tree incrementally that contains the possible paths from the current state. The stages of the algorithm can be summarised as:

- The *selection* stage; where actions are selected to traverse and descend the tree,
- the *expansion* stage; where a new node of the tree is created for a newly visited state,
- the *simulation* stage; where the value of newly created nodes is estimated by Monte-Carlo *rollout*, and,
- the *back-propagation* stage; where, starting from the leaf to the root node, the visit counts and the running averages in the nodes are incremented and updated by the accumulated return, respectively.

It enables the application of multi-armed bandit algorithms to Monte Carlo planning. Bandit problems are stateless reinforcement learning problems in which the agent is tasked to find an optimal policy without access to a full specification of the underlying model. Thus, a great influence on the performance of these algorithms is their adequacy in balancing *exploration* and *exploitation*. That is, the algorithms need to simultaneously explore to find the most profitable actions and, at the same time, ensure high performance

by exploiting the current knowledge of actions by acting greedily as often as possible. Exploration concerns how well that agent is equipped to explore the possible sequences of actions. Conversely, exploitation of the best possible sequences is needed to achieve the best performance during learning. Discovering the best course of action sequences is a challenging task and requires repeated trial and error. The measure of success in these algorithms is the *regret*, which is the loss in return accumulated by following a policy that is not globally optimal. We can define regret in this context as the difference between the expected return, typically undiscounted, of the optimal policy π^* and the behaviour or learning policy $\hat{\pi}$ that actually picks the actions. Note that $\hat{\pi}$ might change over time (Moerland et al., 2023), for example, due to the learning of better estimates of the value function. The regret accumulated for the current time in the horizon $t \leq H$ is then defined as:

$$\text{Regret}_t = \mathbb{E}_{\pi^*}\left[\sum_{k=1}^{t} r_k\right] - \mathbb{E}_{\hat{\pi}}\left[\sum_{k=1}^{t} r_k\right], \tag{3.1}$$

where r_k denotes the reward achieves at time k, with the arguments left implicit. This concept of the trade-off between the practices of maximising exploration and minimising regret is commonly denoted as the *exploration-exploitation dilemma* and, as such, is heavily studied in the literature of both reinforcement learning (Sutton and Barto, 1998; Jaksch et al., 2010) and multi-armed bandits (Slivkins, 2019).

Upper confidence bounds. The upper confidence bound algorithm 1 (UCB1) by Auer et al. (2002) acts as a simple deterministic policy that provides a good balance in terms of regret by applying the principle of *optimism in the face of uncertainty* (OFU, Lai and Robbins (1985)). OFU policies consider actions with high upper confidence bounds as best. That is, actions for which the maximal return in expectation is highest are picked deterministically. More formally, the algorithm achieves optimal logarithmic regret for reward distributions with bounded support (Auer et al., 2002), i.e., reward functions with a finite number of possible rewards.

Stateful confidence bounds. In stateless problems, picking actions based on UCB1 achieves good performance in terms of regret. However, in our case, we are interested in the optimal policy for stateful problems such as MPOMDPs. Kocsis and Szepesvári (2006) introduced an algorithm that applied the UCB1 algorithm to build search trees in stateful domains, commonly denoted as upper confidence trees (UCT), combining Monte Carlo-based planning with UCB1 action selection. In order to do so, a set of statistics are maintained for every encountered state in the tree. These statistics include the visit counts of the state, the number of times an action was picked, and the maximum likelihood estimate (MLE) of the Q-value of all actions. The function is then defined as follows:

$$\mathcal{UCB}(V, N, n) = V + c\sqrt{\tfrac{\log(N+1)}{n+1}}, \tag{3.2}$$

where V is the value of an action $a \in \mathcal{A}$ as given by the MLE Q-value for a, e.g., $Q(s, a)$ in a fully observable setting, and c is the exploration constant. Then, at every node in the tree during a simulation, the UCB1 algorithm decides the actions. Subsequently, a tree is built incrementally from the traces generated by executing said actions in a simulator.

Algorithm 2 POMCP's SEARCH procedure.

```
1: procedure SEARCH(h)
2:     repeat
3:         if h = ∅ then
4:             s ∼ b_0
5:         else
6:             s ∼ b(h)                          ▷ Root sampling from belief.
7:         end if
8:         SIMULATE(s, h, 0)
9:     until TIMEOUT                  ▷ Search until time or simulation limit.
10: end procedure
```

Algorithm 3 POMCP's ROLLOUT procedure.

```
1: procedure ROLLOUT(s, h, d)                          ▷ d is the current depth.
2:     if γ^d < ε then                ▷ Return if the maximum depth is reached.
3:         return 0
4:     end if
5:     a ∼ π_rollout(h | ·)                          ▷ Rollout (e.g. random) policy.
6:     s', o, r ∼ G(s, a)                ▷ Generative transition using simulator.
7:     return r + γ · ROLLOUT(s', hao, d + 1)
8: end procedure
```

3.1.2 Partial Observability

Silver and Veness (2010) introduced partially observable Monte Carlo planning (POMCP), a well-known online algorithm for partially observable environments. The partially observable upper confidence trees (POUCT) algorithm is considered a subclass of POMCP employed with an exact belief update, i.e., as in Eq. (2.12). The algorithms plan with a look-ahead search tree comprised of paths of action and observation nodes, as is visible in Fig. 3.1. These paths of action-observation sequences, i.e., histories, aggregated together represent a sparse belief tree. These sequences are gathered from the traces left by the states sampled at the root during their simulation.

Essentially, it performs MCTS, in the form of UCT, on the belief-MDP. It starts sampling a state from the current belief. Then, it expands a trajectory of actions and observations using a generative model $\mathcal{G}$ until it reaches a new node in the tree, after which a (random) ROLLOUT (Algorithm 3) estimates the value (Kocsis and Szepesvári, 2006).

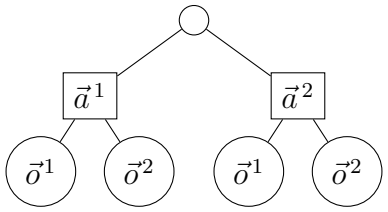


Figure 3.1: POMCP search tree.

The return of a trajectory is used to update a set of statistics for history h that include visit counts for the action $N(h, a)$ and observation nodes $N(h)$. Additionally, the

Algorithm 4 POMCP's MCTS SIMULATE procedure.

procedure SIMULATE(s, h, d)
  if $\gamma^d < \epsilon$ **then**
    return 0 ▷ Return if the maximum depth is reached.
  end if
  if $h \notin T$ **then** ▷ If tree does not contain history.
    for all $a \in \mathcal{A}$ **do**
      $T(h, a) \leftarrow (N_0(h, a), V_0(h, a), \emptyset)$ ▷ Construct nodes.
    end for
    return ROLLOUT(s, h, d) ▷ Get value estimate.
  end if
  $a \leftarrow \arg\max_{a'} \mathcal{UCB}(Q(h, a'), N(h), N(h, a'))$ ▷ UCB (Eq. (3.2)).
  $s',\ o,\ r \sim \mathcal{G}(s, a)$ ▷ Generative transition using simulator.
  $R \leftarrow r + \gamma \cdot \text{SIMULATE}(s', hao, d + 1)$
  $B(h) \leftarrow B(h) \cup \{s\}$ ▷ Add simulation state.
  $N(h) \leftarrow N(h) + 1$
  $N(h, a) \leftarrow N(h, a) + 1$
  $Q(h, a) \leftarrow Q(h, a) + \dfrac{R - Q(h, a)}{N(h, a)}$
  return R
end procedure

observation nodes include maximum likelihood estimates of the Q-values $Q(h, a)$ of all actions that are updated by a running average of the return. UCB1 decides which actions are most promising during the search, balancing exploration and exploitation, which is computed from the set of statistics (number of visits to observation node N, number of visits to action node n, and value of action node Q) and the exploration constant c by using Eq. (3.2):

$$\operatorname*{argmax}_a \left\{ \mathcal{UCB}(Q(h, a), N(h, a), n(h, a)) \right\}. \tag{3.3}$$

As with most online planning algorithms summarised in Algorithm 1, the entry point of POMCP is the SEARCH function (Algorithm 2). The MCTS selection, expansion, and back-propagation stages are combined in a SIMULATE function (Algorithm 4). The simulation stage is represented by the ROLLOUT function (Algorithm 3).

3.2 Particle filters

In systems with large state spaces, the belief update operation is infeasible as it depends on a quadratic enumeration over the whole set of states. In this subsection, we outline a common method to approximate beliefs empirically in online planning. A *Particle Filter* (Thrun, 1999) is a way to empirically approximate the posterior distribution over a state space.

3.2.1 Unweighted

The simplest form of this filter is the unweighted particle filter (PF). This filter consists of a collection of K samples of the state space $\{s_i\}_{i=1}^{K}$ where every individual particle represents the probability of the state by one over the total number of particles. Formally, the belief state is the sum of all particles in the filter:

$$b(s) \approx \hat{b}(s) = \frac{1}{K} \sum_{i=1}^{K} \delta_{s,s_i}, \tag{3.4}$$

where δ_{s,s_i} is the Kronecker delta function $\delta_{ij} = [i = j]$, as in Eq. (2.1). We can find state s_i, with $1 \leq i \leq K$ in the filter $\hat{b}$ by indexing $s_i \leftarrow \hat{b}^{(i)}$. In the case that the state space would be continuous, we would replace δ_{s,s_i} with $\delta'(s - s_i)$ where δ' denotes the Dirac delta function. In POMDPs, the belief state is initialised by sampling K particles from the initial state distribution b_0. The particle-belief state can be seen as the posterior belief state and represents an estimate of the current state $\hat{\Pr}(s_t \mid b_0, h_t)$ given the history and the initial belief.

Intuitively, in an unweighted particle filter, one can think of this as follows:

Example 1. *Consider an unweighted particle filter with K particles $\{(s_i)\}_{i=1}^{K}$. If some state s is represented n times in this particle filter, then the posterior density of this state particle, as given by the filter, is $\hat{b}(s) = \frac{n}{K}$.*

Updating The particle filter is updated based on the chosen action and received observation of the system. The most straightforward method of this is by using rejection sampling. Given the particle filter of size K, $k \leq K$ random particles are sampled from that filter, and a step is made in the environment for each until the sampled observation matches the true observation. Then, this next state s' is added to the new particle set. All other sampled states are rejected. A pseudo-code version of the rejection sampling procedure is given in Sect. 3.2.1, from Kochenderfer (2015).

As the rejection sampling update method relies on the assumption that the given observation will be sampled with some probability, this method is not suited for systems with large discrete or continuous observation spaces. In continuous spaces, it is set to fail as the probability of sampling two truly equal observations in the domain of reals is zero. In large discrete sizes, this probability can get very small, resulting in a potentially extremely large number of calls to the simulator before a particle can be accepted. This greatly diminishes the computational efficiency of the filtering procedure.

Algorithm 5 Rejection sampling algorithm for updating particle filters.

procedure REJECTIONSAMPLING(b, o_{real}, a)
  $b' \leftarrow \emptyset$
  for 1 to $|b|$ **do**
    $s \sim b$ ▷ Sample random state from particle filter.
    repeat
      $s',\ o,\ r \sim \mathcal{G}(s, a)$ ▷ Query the generative model.
    until $o = o_{real}$
    $b' \leftarrow b' \cup \{s'\}$
  end for
  return b'
end procedure

3.2.2 Weighted

A weighted particle filter (WPF) is a Bayes filter that includes the individual probability of the particle. The belief is then approximated by the set of particles and their associated weights: $\{(s_i, w_i)\}_{i=1}^{K}$. Similarly to Eq. (3.4), the weighted particle filter approximates the posterior belief state:

$$b(s) \approx \hat{b}(s) = \sum_{i=1}^{K} w_i \delta_{s,s_i}. \tag{3.5}$$

Similar to the unweighted case, in order to form the approximate belief posterior, the particles are propagated recursively given the previous set of particles and the incoming observations. Ideally, the next set of particles would be sampled from the conditional distribution $\Pr(s_t \mid s_{t-1}, h_t\}$. However, it is often difficult or intractable to sample from this distribution; thus, another method is required. Most commonly, an *importance sampling* method is applied, where the proposal distribution is the dynamics function $\Pr(s_t \mid s_{t-1})$ and the propagated particles are weighted according to their importance as given by the observation or evidence probability $\Pr(o_t \mid s_t)$. The transition probability and observation probability translate to the transition probability function $\mathcal{T}(s_t \mid s_{t-1}, \vec{a}_{t-1})$ and observation model $\mathcal{O}(\vec{o}_t \mid s_t, \vec{a}_{t-1})$ in a MPOMDP[1] respectively, which include the action taken at time $t \in \mathbb{N}$.

As is evident from the paragraph above, in the case of a POMDP, a weighted particle filter requires an observation model specification. Although this might be difficult to specify exactly, an approximation of this model can suffice (Sunberg and Kochenderfer, 2018).

3.2.3 Sequential Monte Carlo

A weighted particle filter is essentially a sequential Monte Carlo (SMC) algorithm where instead of sampling from the true distribution, we use a proposal and importance distribution that is contained within the model, e.g., the MPOMDP, itself. Abstractly, we reason about a random variable, i.e., states, $s \sim \mathcal{T}$ based on the probabilities of related random variables, i.e., observations, $o \sim \mathcal{O}$.

[1] or in an POMDP, then with $a \equiv \vec{a}, o \equiv \vec{o}$

Very generally, SMC aims to approximate a sequence of target probability density functions (PDFs), where the PDFs might be known up to a normalisation constant. The dimension is increasing with t as the support of every function in this sequence is defined as $\mathbb{R}^{d^t}$ where d is the original dimension of the variables.

In order to do so, we make use of importance sampling. Importance sampling is a variance reduction method that can be used in a Monte Carlo sampling approach. We can approximate a target distribution $\mathcal{P}$ by sampling from a proposal distribution $\mathcal{Q}$ and computing the importance weight given by the ratio between the distributions. The importance weight ensures an unbiased distribution by correcting the biased proposal distribution.

Monte Carlo estimates and importance sampling. Let us denote the target distribution as $\mathcal{P}(x)$. Given any function f, it might even be the identity function $f(x) = x$, we can approximate the expectation of the distribution $\mu = \mathbb{E}_{\mathcal{P}}[f(x)]$ with a Monte Carlo estimate as follows:

$$\mathbb{E}_{\mathcal{P}}[f(x)] = \int f(x)\mathcal{P}(x)dx \simeq^{\text{a.s.}}_{K\to\infty} \frac{1}{K}\sum_{i=1}^{K} f(x_i), \tag{3.6}$$

where $x_i \sim \mathcal{P}$ and $\simeq^{\text{a.s.}}_{K\to\infty}$ denotes "equals almost surely in the limit $K \to \infty$". We write integrals here for generality, but these can be replaced by (Riemann) summations in the discrete case that we consider. The target distribution might be a Bayes distribution, which gives the following general equation from Bayes' theorem:

$$\mathcal{P}(x) = p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}, \tag{3.7}$$

where y might be the incoming data or observations. $\mathcal{P}$ might be difficult to sample from. In order to estimate the target distribution $\mathcal{P}$, we introduce the use of a proposal distribution $\mathcal{Q}$. The importance weight w or likelihood is then given by the relative importance:

$$w(x) = \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}, \tag{3.8}$$

We require $f(x)\mathcal{P}(x) > 0 \to \mathcal{Q}(x) > 0$. The weights correct the bias introduced by sampling from a different distribution (Bishop, 2006). Using this, we make the Monte Carlo estimate based on samples from the proposal distribution:

$$\mathbb{E}_{\mathcal{Q}}[f(x)] = \int f(x)\frac{\mathcal{P}(x)}{\mathcal{Q}(x)}\mathcal{Q}(x) \simeq^{\text{a.s.}}_{K\to\infty} \frac{1}{K}\sum_{i=1}^{K} f(x_i)w(x_i) = \hat{\mu}_{\mathcal{Q}}, \tag{3.9}$$

with $x_i \sim \mathcal{Q}$ sampled from the proposal distribution. Here we are estimating in expectation, i.e., taking samples of the proposal distribution $\mathcal{Q}$ instead of $\mathcal{P}$. Note that $\hat{\mu}_{\mathcal{Q}}$ is unbiased estimator of μ, as $\mathbb{E}_{\mathcal{Q}}[\hat{\mu}_{\mathcal{Q}}] = \mu$ (Kennedy, 2016). In an ideal scenario, if the proposal distribution is $\mathcal{Q}(x) = \frac{f(x)\mathcal{P}(x)}{\mu}$, then the variance of the estimator is zero. Thus, as we do not know μ, we generally look for proposal distributions $\hat{\mathcal{Q}}(x)$ that are close to proportional to $f(x)\mathcal{P}(x)$, i.e., $\hat{\mathcal{Q}}(x) \approx \mathcal{Q}(x) \propto f(x)\mathcal{P}(x)$.

Self-normalised. The target distribution $\mathcal{P}(x) = \eta_{\mathcal{P}}\tilde{\mathcal{P}}(x)$ might be known up to a normalisation constant and similarly $\mathcal{Q}(x) = \eta_{\mathcal{Q}}\tilde{\mathcal{Q}}(x)$ may be found to have the same property. We then arrive at the following[2]:

$$\mathbb{E}_{\mathcal{Q}}\left[f(x)\right] \simeq^{\text{a.s.}}_{K\to\infty} \sum_{i=1}^{K} \tilde{w}_i f(x_i),\ \tilde{w}_i = \frac{w_i}{\sum_j w_j} = \frac{\frac{\tilde{\mathcal{P}}(x_i)}{\mathcal{Q}(x_i)}}{\sum_j \frac{\tilde{\mathcal{P}}(x_j)}{\mathcal{Q}(x_j)}} \tag{3.10}$$

The perceptive reader might have noticed that we have now arrived at a form that is similar to the estimate of the belief distribution given by the weighted particle filter in Eq. (3.5). For the normalised importance weights, we have $\forall_i \colon \tilde{w}_i \geq 0$ and $\sum_i \tilde{w}_i = 1$. The estimator in Eq. (3.10) requires $\mathcal{P}(x) > 0 \to \mathcal{Q}(x) > 0$ and is biased, but often has less variance in practice (Luo et al., 2019). Moreover, it is asymptotically unbiased in the limit of the number of samples (Owen, 2013).

Sequential filtering. As we are interested in a consecutive estimate of the belief distribution, we require a sequential importance sampling (SIS) approach. Our target distribution might look like $\mathcal{P}(x) = p(x_{0:t} \,|\, y_{0:t}, x_{0:t-1}) = p(x_t \,|\, y_{0:t})$, where we denote the sequence of t observations with $y_{0:t} = (y_0, y_1, \ldots, y_t)$, with naturally $y_t \prec y_{t+1}$.

Non-linear filtering. From Bayes' rule for conditional probabilities, we have in the general filtering setting:

$$p(x_{0:t} \,|\, y_{0:t}) = \frac{p(y_{0:t} \,|\, x_{0:t})p(x_{0:t})}{p(y_{0:t})}, \tag{3.11}$$

where:

$$p(y_{0:t} \,|\, x_{0:t}) = \prod_{i=0}^{t} p(y_i \,|\, x_i), \tag{3.12}$$

$$p(x_{0:t}) = \prod_{i=0}^{t} p(x_i \,|\, x_{i-1}), \tag{3.13}$$

$$p(y_{0:t}) = \int p(y_{0:t} \,|\, x_{0:t})p(x_{0:t})dx_{0:t}. \tag{3.14}$$

Since we are only interested in the filtering density $p(x_t \,|\, y_{o:t})$, we can proceed as follows. For the non-linear filtering update equation, we have a recursion, starting at $p(x_t \,|\, y_{0:t-1})$:

$$p(x_t \,|\, y_{0:t-1}) \to p(x_t \,|\, y_{0:t}) = \frac{p(y_t \,|\, x_t)p(x_t \,|\, y_{0:t-1})}{\int p(y_t \,|\, x'_t)p(x'_t \,|\, y_{0:t-1})dx'_t}, \tag{3.15}$$

with the convention that $p(x_0 \,|\, y_0) \equiv p(x_0)$. We aim to approximate the filtering distribution after the update in Eq. (3.15).

[2]Some derivations are omitted here for simplicity. For details we refer to Section 11.1.4, Bishop (2006)

Sequential importance sampling. The recursive version of the importance estimate in Eq. (3.10) is:

$$\int f(x_t)p(x_t \mid y_{0:t}) \approx \sum_{i=1}^{K} w_t^{(i)} f(x_t^{(i)}), \tag{3.16}$$

where the algorithm depends on the proposal distribution $\mathcal{Q}(x_t \mid x_{0:t-1}, y_{0:t})$. The optimal proposal is the target distribution $\mathcal{Q}(x_t \mid x_{0:t-1}, y_{0:t}) = p(x_t \mid x_{t-1}, y_t) = \dfrac{p(y_t \mid x_t)}{\int p(y_t \mid x_t)p(x_t \mid x_{t-1})dx_t}$. In practice, we often set the proposal distribution as the transition dynamics distribution $p(x_t \mid x_{t-1})$, i.e., $\mathcal{T}$.

A step in sequential importance sampling looks like the following:

1. Draw N samples from the proposal distribution $x_t^{(i)} \sim \mathcal{Q}(x_t \mid x_{0:t-1}^{(i)}, y_{0:t})$.

2. Updated the weights up to a normalisation constant $\tilde{w}_t^{(i)} = w_{t-1}^{(i)} \dfrac{p(y_t \mid x_t^{(i)})p(x_t^{(i)} \mid x_{t-1}^{(i)})}{\mathcal{Q}(x_t \mid x_{0:t-1}^{(i)}, y_{0:t})}$.

   In the case that the proposal distribution $\mathcal{Q}(x_t \mid x_{0:t-1}^{(i)}, y_{0:t}) = p(x_t^{(i)} \mid x_{t-1}^{(i)})$ is the prior transition probability distribution, as is the case in the well-known *bootstrap filter* (Gordon et al., 1993), the weight update simplifies as $\tilde{w}_t^{(i)} = w_{t-1}^{(i)} p(y_t \mid x_t^{(i)})$.

3. Self-normalise the weights $w_t^{(i)} = \dfrac{\tilde{w}_t^{(i)}}{\sum_j \tilde{w}_t^{(j)}}$.

This works because Bayes' theorem can be implemented as a weighted bootstrap (Gordon et al., 1993). We are trying to acquire samples from the PDF proportional to $\mathcal{L}(x)\hat{\mathcal{P}}(x)$, by sampling an empirical distribution $\hat{p}(x_0) = \{x_i\}_{i=0}^K$ from the prior of $\hat{\mathcal{P}}(x)$, and where $\mathcal{L}(x)$ is the known likelihood. We identify $\hat{\mathcal{P}}(x) = p(x_t \mid y_{0:t})$ as the prior and $\mathcal{L}(x) = p(y_t \mid x_t)$ as the likelihood. Subsequent empirical distributions $\{(x_i, w_i)\}_{i=0}^K$ with probability mass $w_i = \frac{\mathcal{L}(x_i)}{\sum_j \mathcal{L}(x_j)}$ tends in distribution proportional to $\mathcal{L}(x)\hat{\mathcal{P}}(x)$ almost surely as $K \to \infty$ (Smith and Gelfand, 1992).

Re-sampling. Because the weights of an importance sampling estimate are unequal, one can become much larger than the others. In this scenario, there is effectively only a single observation. This phenomenon is called *weight disparity*. In the extreme case, all weights might be zero, in which case importance sampling has obviously failed. One statistic to diagnose weight disparity is the effective sample size (ESS), where imbalanced weights give a result that is similar to averaging only $k \ll$ ESS observations (Owen, 2013). An approximation, as the true diagnostic depends on the intractable variance, of the ESS can be computed from the importance weights:

$$\mathrm{ESS}(\{w_i\}_{i=1}^K) \approx \frac{(\sum_i w_i)^2}{\sum_i w_i^2}, \tag{3.17}$$

where the simplification $\mathrm{ESS}(\{\tilde{w}_i\}_{i=1}^K) \approx \frac{1}{\sum_i \tilde{w}_i^2}$ holds if the weights $\tilde{w}$ are normalised. Re-sampling then involves sampling K samples from the posterior estimate and setting the weights to $\frac{1}{K}$. Given the ESS statistic, we can determine whether to re-sample. The threshold of determining a re-sample is application-specific, and in practice re-sampling can be executed at every step. Another method to determine whether to re-sample is

based on the relative disparity with respect to the number of samples in the estimate (Wu et al., 2021).

3.2.4 Particle Filtering

In summary, the use of particle filtering renders the belief update tractable by maintaining a recursive empirical distribution (Thrun et al., 2005). In an *unweighted* filter, the approximate belief contains only states $\tilde{b} = \{(s_i)\}_{i=1}^{K}$ and is updated using rejection sampling on the real observation $\vec{o}$; $\tilde{b} = \{s_i' \colon \vec{o} = \vec{o}_i\}$, where s_i', $\vec{o}_i$ are the next state and observation as sampled from the generative model (Kochenderfer et al., 2015). *Weighted* particle filters approximate the belief posterior by a set of K particles $\tilde{b} = \{(s_i, w_i)\}_{i=1}^{K}$, where s_i is a state particle with index i and w_i the associated weight. For weighted filters, we can update the belief using importance sampling techniques such as sequential importance re-sampling (SIR) (Gordon et al., 1993). The posterior belief is computed at every time-step by propagating the particles through the proposal distribution, i.e., the transition function $s_i' \sim \mathcal{T}(\cdot \,|\, s_i, \vec{a})$, and re-weighting according to its importance weight, i.e., by the observation function $w_i' \propto w_i \mathcal{O}(\vec{o} \,|\, s_i', \vec{a})$. Commonly, the posterior belief is re-sampled to alleviate sample degeneracy, after which the weights are set to $\frac{1}{K}$. Whether to re-sample can be decided by the threshold of the effective sample size (ESS) of the particle filter with respect to the number of particles therein (Septier and Peters, 2016; Wu et al., 2021). The effective sample size can be computed for a empirical distribution to quantify weight disparity by the following equation:

$$\mathrm{ESS}(\{(s_i, w_i)\}_{i=1}^{K}) \approx \frac{1}{\sum_{i=1}^{K} w_i^2}. \tag{3.18}$$

Using the ESS as a threshold is an alternative to re-sampling after every filtering step (Thrun et al., 2005). The likelihood of a belief update represents the probability of the new belief given the observation, action, and previous belief. It is a statistic on the quality of the approximate belief update. It is given by the sum of all updated weights $\mathcal{L}_t = \sum_i w_i'$ with which the likelihood $\mathcal{L}_{h_t}$ of history h_t can be propagated over time by $\mathcal{L}_{h_t} \leftarrow \mathcal{L}_{h_{t-1}} \mathcal{L}_t$ (Katt et al., 2019). The full procedure of a SIR filter at time t, including the use of ESS and propagating the likelihood, is given in Algorithm 6. Note that the SIS filter is a simplified version of this algorithm, in which the ESS and re-sampling steps are skipped.

Algorithm 6 A weighted particle filter (WPF) with SIR.

procedure SIR($a_t, o_t, \{(\vec{s}^i_{t-1}, w^i_{t-1})\}_{i=1}^K, \mathcal{L}_{t-1}, \mathcal{O}, \mathcal{T}, \tau$)
  $w \leftarrow 0$ ▷ Keep track of total weight.
  $\hat{b}(\cdot) \leftarrow \emptyset$ ▷ Initialise new intermediate belief.
  for $i \leftarrow 1$ to K **do**
    $s_t^{(i)} \sim \mathcal{T}(\cdot \mid s_{t-1}^{(i)}, \vec{a})$
    $w_t^{(i)} \leftarrow w_{t-1}^{(i)} \cdot \mathcal{O}(\vec{o}_t \mid s_t^{(i)}, \vec{a}_t)$
    $\hat{b}(\cdot) \leftarrow \hat{b}(\cdot) \cup \{\langle s_t^{(i)}, w_t^{(i)} \rangle\}$
    $w \leftarrow w + w_t^{(i)}$
  end for
  $\hat{b}(\cdot) \leftarrow \{\langle s_t^{(i)}, \frac{w_t^{(i)}}{w_t} \rangle\}_{i=1}^K$ ▷ Normalisation.
  if $ESS(\hat{b}(\cdot)) \leq \tau$ **then** ▷ ESS threshold, Eq. (3.18).
    $b(\cdot) \leftarrow \hat{b}(\cdot)$ ▷ Insignificant weight disparity.
  else
    $b(\cdot) \leftarrow \emptyset$ ▷ Create final belief by re-sampling.
    for $i \leftarrow 1$ to K **do**
      $s_t^{(i)} \sim \hat{b}(\cdot)$
      $b(\cdot) \leftarrow b(\cdot) \cup \{\langle s_t^{(i)}, \frac{1}{K} \rangle\}$
    end for
  end if
  $\mathcal{L}_t \leftarrow \mathcal{L}_{t-1} \cdot w$
  return $b(\cdot), \mathcal{L}_t$
end procedure

Chapter 4

Tree Search for Many-Agent POMDPs

Previously, we considered finding approximate solutions by searching the set of states that are reachable from the current belief. Additionally, we learned how to represent such a belief empirically. In this chapter, we firstly consider a method from the literature that provides a method to improve the generalisation of the algorithm across the solution space induced by an environment with many agents. In particular, we consider a graphical decomposition of the problem into several smaller connected sub-problems. Then, we proceed to introduce novel algorithm variants, and subsequently apply a similar technique to these variants.

4.1 Coordination Graphs

A coordination graph (CG) (Guestrin et al., 2002b; Oliehoek and Amato, 2016) is a framework to model the interaction schemes of groups of agents in a network or graph. In most scenarios with a large number of agents, there is a locality of interaction that translates to naturally formed subsets or cliques of agents that attempt to achieve the same goal. A coordination graph compactly represents the local interactions between agents. In particular, the graph represents the inherent structure of multi-agent systems by decomposing the global value function into a mixture of local functions over subsets of agents (Amato and Oliehoek, 2015).

Coordination. Formally, a CG is a graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$ where each vertex $v \in \mathcal{V}$ corresponds to an agent and each edge $(i, j) \in e \in \mathcal{E}$ indicates that agents $i \in \mathcal{V}$ and $j \in \mathcal{V}$ interact locally. For a component $e \in \mathcal{E}$, the local action $\vec{a}_e$ and observation spaces $\vec{o}_e$ range over the product of the individual agent action $\times_{i \in e} \mathcal{A}_i$ and observation spaces $\times_{i \in e} \Omega_i$. These edges can also be hyper-edges, connecting multiple nodes (i.e., agents). In this thesis henceforth, we assume that an edge $e \in \mathcal{E}$ in a coordination graph connects **two** agents $(i, j) \in e$. In theory, any CG containing one or more hyper-edges can be morphed into a graph with only pairwise dependencies (Kok and Vlassis, 2006a).

Using a given or inferred coordination graph, we can use the natural factorisation of value that occurs from the interaction structure. In a coordination graph, the value function is factorised in the sum of local per-agent utility functions and edge-based payoff functions. For interpretability, we introduce it to stateless problems here, without history. In a stateless setting the Q-function is decomposed as:

$$Q(\vec{a}) = \sum_{i \in \mathcal{V}} Q_i(a_i) + \sum_{(i,j) \in \mathcal{E}} Q_{ij}(a_i, a_j), \tag{4.1}$$

with $Q_i(a_i)$ the individual utility function of agent i and $Q_{ij}(a_i, a_j) = Q_e(\vec{a}_e)$ the pairwise payoff function for every edge connecting two agents i and j in the graph, with $a_i \in \mathcal{A}_i, a_j \in \mathcal{A}_j$. In the subsequent paragraph, we do not consider the explicit individual utility $Q_i(a_i)$ of the agents. Incorporating this estimate can increase performance marginally (Choudhury et al., 2022) but consequently increases the space complexity.

A mixture of experts. To predict the Q-value $Q(\vec{a})$ based on the local actions, we define a local payoff function $Q_e(\vec{a}_e)$ for each edge $e \in \mathcal{E}$, where $\vec{a}_e \in \times_{i \in e} \mathcal{A}_i$ is the local joint action of the agents in e assigned by the joint action $\vec{a} \in \mathcal{A}$. This way, with these local estimates of the global Q-value, we can use a mixture of experts (MoE) (Peng et al., 2019) approach to estimate the Q-value of the joint action:

$$Q(\vec{a}) \approx \hat{Q}(\vec{a}) = \sum_e \alpha_e \hat{Q}_e(\vec{a}_e), \tag{4.2}$$

where $\alpha_e \geq 0$ is the weight of the expert for component e and $\sum_{e \in \mathcal{E}} \alpha_e = 1$. In order to pick the estimated maximising joint action $\vec{a}^{\#}$ we maximise over the sum of restricted-scope functions:

$$\arg\max_{\vec{a}^{\#}} \sum_e \alpha_e \hat{Q}_e(\vec{a}_e^{\#}). \tag{4.3}$$

More intuitively, we aim to find local actions (for the edges of the graph) that maximise the estimated joint value function. Notice that any agent i might belong to multiple edges. Therefore agent i must be assigned the same action $a_i^{\#} \in \mathcal{A}_i$ in all edges $e \in \mathcal{E}$ where $i \in e$. This maximisation can be computed efficiently with graphical inference algorithms, which are discussed in detail in chapter 6.

Analysis. Amato and Oliehoek (2015) prove that the MoE optimisation introduces a policy-dependent bias term $B_\pi(\vec{a})$ given sample policy π. This bias is introduced by the overlap of the components, where agent i might be part of both component e and e'. We denote the neighbouring components that overlap in the agents of e as $\Gamma(e) = \{e' \in \mathcal{E} \setminus \{e\} \mid e \cap e' \neq \emptyset\}$, where $e \cap e'$ is non-empty if there exists an agent i with $i \in e$ and $i \in e'$. Furthermore, we assume the experts are weighted uniformly, i.e., $\alpha_e = \frac{1}{|\mathcal{E}|}$ such that these weights can be omitted. We summarise their analysis here.

The MoE estimate approaches the true value plus the bias term $\hat{Q}(\vec{a}) \approx Q(\vec{a}) + B_\pi(\vec{a})$. Let $\vec{a}_{e' \setminus e}$, specified by $\vec{a}$, be the actions of agents in e' that are not in e, and, conversely, let $\vec{a}_{e' \cap e}$ be the actions of the agents that participate in both e and e'. Then, the bias term is defined as:

$$B_\pi(\vec{a}) \triangleq \sum_e \sum_{e' \neq e} \sum_{\vec{a}_{e' \setminus e}} \pi(\vec{a}_{e' \setminus e} \mid \vec{a}_e) Q_{e'}(\vec{a}_{e' \setminus e}, \vec{a}_{e' \cap e}). \tag{4.4}$$

Bias by itself is not an issue, but differing biases per joint action can result in non-optimal action selection. Fortunately, their proof also encompasses that the bias terms between actions are bounded by $\forall_{\vec{a},\vec{a}'} : |B_\pi(\vec{a}) - B_\pi(\vec{a}')| \leq \epsilon$ if the Q-function is sufficiently structured. In the case that value functions do not overlap, MoE recovers the maximal joint action. If they do overlap, then we have the following.

Theorem 4.1.1. *If for components with overlap e, e', and any two $\vec{a}_{e'\cap e}, \vec{a}'_{e'\cap e} \in \mathcal{A}_{e'\setminus e}$, with $\mathcal{A}_{e'\setminus e}$ the set of actions with overlap, the true value function Q satisfies:*

$$\forall_{\vec{a}_{e'\setminus e}} : Q_{e'}(\vec{a}_{e'\setminus e}, \vec{a}_{e'\cap e}) - Q_{e'}(\vec{a}_{e'\setminus e}, \vec{a}'_{e'\cap e}) \leq \frac{\epsilon}{|\mathcal{E}| \cdot |\Gamma(e)| \cdot |\mathcal{A}_{e'\setminus e}| \cdot \pi(\vec{a}_{e'\setminus e})}, \tag{4.5}$$

then MoE optimisation will return an ϵ-optimal joint action in the limit.

Proof. See the proof for Theorem 6 in the appendix of Amato and Oliehoek (2015). □

4.2 Factored-Value POMCP

Factored-Value POMCP (Amato and Oliehoek, 2015) consists of two techniques that exploit the structure of a CG to scale POMCP to problems with large action and observation spaces. These techniques factor the action space to introduce statistics for each component $e \in \mathcal{E}$ of a graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$ to compute the UCB1 values of the local joint action space $\vec{a}_e$. The two variants are outlined next.

4.2.1 Factored Statistics

factored-statistics (FS) POMCP (FS-POMCP) reduces the number of statistics maintained and improves the efficiency of action selection in large action spaces. The tree structure remains the same as in Fig. 3.1 for the joint history $\vec{h}$ and associated visit count $N(\vec{h})$, but the value nodes (circles in Fig. 3.1) maintain a set of statistics $Q_e(\vec{h}, \vec{a}_e)$, $N(\vec{h}, \vec{a}_e)$ for each edge $e \in \mathcal{E}$, independently, instead of for every $Q(\vec{h}, \vec{a})$. Thus, the MoE optimisation from Sect. 4.1 is applied directly in each node of the search tree. The improvement to vanilla POMCP is that fewer statistics are maintained, as the combination of local action spaces $\vec{a}_e$ of each component $e \in \mathcal{E}$ is smaller than the joint action space $\vec{a}$. A graphical inference algorithm then selects the maximal joint action $\vec{a}^{\#}$ by maximising over the set of local UCB1 values of the Q-values induced by the CG (using Eq. (3.2)):

$$\arg\max_{\vec{a}^{\#}} \sum_{e\in\mathcal{E}} \mathcal{UCB}(Q_e(\vec{h}, \vec{a}_e), N(\vec{h}), N(\vec{h}, \vec{a}_e)). \tag{4.6}$$

Progressive widening. Sunberg and Kochenderfer (2018) introduce POMCPOW, POMCP with observation widening and weighted particle filtering to tackle POMDPs with continuous states and observations. Progressive widening (Browne et al., 2012) consists of limiting the number of children action and observation nodes can have based on the number of times said nodes were visited in the past. For increasing number of visits, the number of allowed children is increased according to a formula $N_C = kN^{\beta}$, where N_C is the number of children, N the number of times the nodes was visited, and k and β the hyper-parameters that balance the weight of the allowed children with respect to the number of visits. The same formula can be used for the action and observation nodes in the three, with possibly different k_a, k_o and β_a, β_o for each type of node. We introduce FS-POMCPOW as the direct application of FS to POMCPOW's SIMULATE function. The large action space is then addressed by the FS paradigm, and the large observation space by progressively widening the growth in breadth of the search tree. This results in a versatile algorithm that is well suited for MPOMDPs.

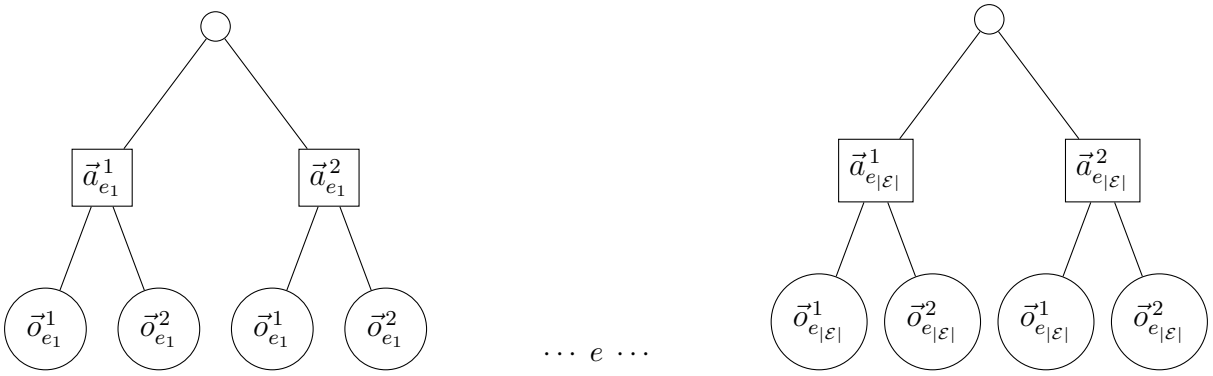


Figure 4.1: Factored POMCP, with a separate search tree for every $e \in \mathcal{E}$. In this example, we show a possible tree for two e, where we branch on two actions and two observations each.

4.2.2 Factored Trees

factored-trees (FT) POMCP (FT-POMCP) builds a tree for every $e \in \mathcal{E}$ for the factored history $\vec{h}_e$, which consists of a sequence of factored actions and observations $\vec{h}_{e,t} = (\vec{a}_{e,0}, \vec{o}_{e,1}, \ldots, \vec{a}_{e,t-1}, \vec{o}_{e,t})$. This further reduces the scope of $Q_e(\vec{h}, \vec{a}_e)$ to $Q_e(\vec{h}_e, \vec{a}_e)$ by introducing an expert for every $\vec{h}_e, \vec{a}_e$ pair. Fig. 5 in the appendix shows the first and last tree out of the $|\mathcal{E}|$ trees built in parallel, where both trees are depicted with a single expansion level. In each tree, the nodes maintain MLE statistics $Q_e(\vec{h}_e, \vec{a}_e)$, $N(\vec{h}_e)$, and $N(\vec{h}_e, \vec{a}_e)$, according to the factored history $\vec{h}_e$. Again, an inference algorithm selects the maximal joint action $\vec{a}^{\#}$ by maximising over the set of factored actions with respect to their UCB1 value (using Eq. (3.2)):

$$\arg\max_{\vec{a}^{\#}} \left[\sum_{e \in \mathcal{E}} \mathcal{UCB}(Q_e(\vec{h}_e, \vec{a}_e), N(\vec{h}_e), N(\vec{h}_e, \vec{a}_e)) \right]. \tag{4.7}$$

Let $T_e(\vec{h}_{e,t-1}, \vec{a}_{e,t-1}, \vec{o}_{e,t})$ denote the tree for factor e that represents the history $\vec{h}_{e,t-1}\vec{a}_{e,t-1}\vec{o}_{e,t}$. After a step in the true environment executing action $\vec{a}_{e,t-1}$ and receiving observation $\vec{o}_{e,t}$, we prune the search tree as follows. For all factors $e \in \mathcal{E}$, the tree node $T_e(\vec{h}_{e,t-1}, \vec{a}_{e,t-1}, \vec{o}_{e,t})$ becomes the respective new root. If the tree node $T_e(\vec{h}_{e,t-1}, \vec{a}_{e,t-1}, \vec{o}_{e,t})$ does not exist for any $e \in \mathcal{E}$, a new tree is built from scratch for that particular component e.

The pseudo-code of the FT version of the original POMCP simulate function is outlined in Algorithm 7.

Analysis. Amato and Oliehoek (2015) introduce a strong result that the FT variant might diverge when UCB1 with exploration constant $c = 0$ is employed. In that case, the algorithm corresponds with a class of Monte-Carlo control, such as SARSA(1), with linear regression. These settings divert in general (Fairbank and Alonso, 2012).

This can be explained by the fact that instead of the full history $\vec{h}$, the local history $\vec{h}_e$ is not Markov if the factors are not independent. In practice, the exploration constant c of the UCB1 policy during the search is never set to zero, except for when the final action is picked for execution in the true environment. Although UCB1 is a greedy policy with respect to the upper confidence bounds of the Q-values, it is not clear how the exploration bonus of the UCB1 policy affects the convergence of the algorithm. Additionally, UCB1 is

Algorithm 7 Factored-Trees-POMCP's SIMULATE procedure.

procedure SIMULATE(s, h, d)
  if $\gamma^d < \epsilon$ **then**
    return 0 ▷ Return if the maximum depth is reached.
  end if
  for all $e \in E$ **do** ▷ Enumerate every component/factor tree.
    if $\vec{h}_e \notin T_e$ **then** ▷ If tree does not contain history.
      for all $\vec{a}_e \in \vec{\mathcal{A}}_e$ **do** ▷ Enumerate component actions.
        $T_e(\vec{h}_e, \vec{a}_e) \leftarrow (N_0(\vec{h}_e, \vec{a}_e), V_0(\vec{h}_e, \vec{a}_e), \emptyset)$ ▷ Construct nodes.
      end for
    end if
  end for
  if $\exists_{e \in E} : \vec{h}_e \notin T_e$ **then**
    return ROLLOUT($s, \vec{h}, d$) ▷ Get value estimate.
  end if
  $\vec{a} \leftarrow \arg\max'_{\vec{a}} \left[\sum_e \mathcal{UCB}(Q_e(\vec{h}_e, \vec{a}_e), N(\vec{h}_e), N(\vec{h}_e, \vec{a}'_e))\right]$ ▷ VE or MP.
  $s',\ \vec{o},\ r \sim \mathcal{G}(s, \vec{a})$ ▷ Generative transition using simulator $\mathcal{G}$.
  $R \leftarrow r + \gamma \cdot \text{SIMULATE}(s', \vec{h}\vec{a}\vec{o}, d+1)$
  for all $e \in E$ **do** ▷ Update statistics for every tree.
    $B(\vec{h}_e) \leftarrow B(\vec{h}_e) \cup \{s\}$ ▷ Add simulation state.
    $N(\vec{h}_e) \leftarrow N(\vec{h}_e) + 1$
    $N(\vec{h}_e, \vec{a}_e) \leftarrow N(\vec{h}_e, \vec{a}_e) + 1$
    $Q_e(\vec{h}_e, \vec{a}_e) \leftarrow Q_e(\vec{h}_e, \vec{a}_e) + \dfrac{R - Q_e(\vec{h}_e, \vec{a}_e)}{N(\vec{h}_e, \vec{a}_e)}$ ▷ Update expert.
  end for
  return R
end procedure

not a component-wise policy as it is defined for the full action space. However, FT variants might produce high-quality results in practice, especially when the problem benefits from an increased search depth. Such search depth is made possible due to the additional generalisation offered by considering local histories.

4.3 Particle Belief-Space Planning for MPOMDPs

In this section, we lift the sparse particle filter tree (PFT) algorithm (Sunberg and Kochenderfer, 2018; Fischer and Tas, 2020a; Lim et al., 2020, 2023) to MPOMDPs. As before, we propose extensions that exploit the factorization of the action space as induced by coordination graphs. Firstly, we introduce particle belief approximation, and subsequently, PFT for MPOMDPs. Then, we introduce factored-statistics (FS) PFT (FS-PFT) and factored-trees (FT) PFT (FT-PFT) to combat large action spaces.

4.3.1 Particle-Belief MMDP

As introduced in Sect. 2.1.2, POMDPs can be represented by fully observable belief-MDP with the same action space as the POMDP and the beliefs as states (Cassandra et al., 1994). Similarly, an MPOMDP has a belief-MMDP. The belief-MMDP for a MPOMDP is similar to how MDPs relate to MPOMDPs and can be deduced from Def. 4 by aptly replacing the single-agent spaces with joint spaces from the MPOMDP. Particle-belief MDPs approximate belief-MDPs (Lim et al., 2020, 2023), by representing the posterior belief distribution by a finite number of samples in a particle filter.

We introduce the particle-belief-MMDP as an approximation of a belief-MMDP:

Definition 9 (**PB-MMDP**)**.** The particle-belief-MMDP corresponding to an MPOMDP $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, \mathcal{T}, r, \{\Omega_i\}, \mathcal{O}, \gamma \rangle$ as in Def. 6 is a tuple $\langle \mathcal{I}, \Sigma, \{\mathcal{A}_i\}, \tau, \rho, \gamma \rangle$, with:

- $\mathcal{I}$, $\mathcal{A}_i$, γ as in the original MPOMDP $\mathcal{M}$ (Def. 6);
- Σ, the state space over particle beliefs $\tilde{b} = \{(s_i, w_i)\}_{i=1}^{C}$, where $s_i \in \mathcal{S}$, $w_i \in \mathbb{R}^+$ is the associated weight of state particle s_i, and $C = |\tilde{b}|$ the number of particles;
- τ, the transition density function $\tau(\tilde{b}' \,|\, \tilde{b}, \vec{a})\colon \Sigma \times \mathcal{A} \to \Delta(\Sigma)$. Given that the importance weights are updated according to the observation density:

$$w_i' \propto w_i \mathcal{O}(\vec{o} \,|\, \vec{a}, s_i), \tag{4.8}$$

  that transition density is defined as:

$$\tau(\tilde{b}' \,|\, \tilde{b}, \vec{a}) \equiv \sum_{\vec{o} \in \Omega} \Pr(b' \,|\, b, \vec{a}, \vec{o}) \Pr(\vec{o} \,|\, b, \vec{a}) \tag{4.9}$$

- ρ, the reward function $\rho(\tilde{b}, \vec{a})\colon \Sigma \times \mathcal{A} \to \mathbb{R}$ defined by $\rho(\tilde{b}, \vec{a}) = \frac{\sum_i w_i r(s_i, \vec{a})}{\sum_i w_i}$, with r the reward function of the MPOMDP. If the original reward function r is bounded by $r \leq r_{max}$ then ρ is bounded by $||\rho||_\infty \leq r_{max}$ as the importance weights sum to $\sum_i w_i = 1$.

Defining the transition density. The first term $\Pr(b' \,|\, b, \vec{a}, \vec{o})$ in Eq. (4.9) is the conditional transition density given observation $\vec{o}$. This probability is synonymous with the belief update function in Eq. (2.12), albeit for a particle belief. The state transition updates for each particle in the particle belief are independent of each other, and the update of the likelihood weight is deterministic given $s_i, s_i', \vec{a}, \vec{o}$. Furthermore, the integral is only non-zero when $b' = \{(s_i', w_i')\})_{i=1}^{C}$, such that the calculation simplifies to the

product of the individual transition densities given that the likelihood update matches (Lim et al., 2023):

$$\Pr(b' \mid b, \vec{a}, \vec{o}) = \begin{cases} \prod_{i=1}^{C} \mathcal{T}(s'_i \mid s_i, \vec{a}) & \text{if } \forall_i \colon w'_i = w_i \mathcal{O}(\vec{o} \mid \vec{a}, s_i), \text{ and,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

The second term $\Pr(\vec{o} \mid b, \vec{a})$ is the observation likelihood given a particle belief and a joint action. Again, we need to define this probability, which is given for an exact belief in Eq. (2.13), for a particle belief. The likelihood is equivalent to the weighted sum of the observation likelihoods conditioned on the probability that this observation was generated from state particle s_i:

$$\Pr(\vec{o} \mid b, \vec{a}) = \frac{\sum_{i=1}^{C} w_i \cdot [\sum_{s \in \mathcal{S}} \mathcal{O}(\vec{o} \mid s'_i, \vec{a}) \mathcal{T}(s'_i \mid s_i, \vec{a})]}{\sum_{i=1}^{C} w_i}. \tag{4.11}$$

Although this transition density is difficult to calculate exactly, it is possible to sample from it. Given that we do not require true probabilities of the dynamics but mere simulation access (chapter 3), we can use this model to plan with in an online fashion.

Advantage of conversion. PB-MMDPs are special MDPs and thus allow the direct application of MDP methods, e.g., MCTS. This requires us to update the associated generative model: We simulate particle-beliefs instead of individual states and thus change $\mathcal{G}$ to $\mathcal{G}_{PF} : \Sigma \times \mathcal{A} \to \Sigma \times \mathcal{P}(\mathbb{R})$. This extension increases the complexity of the generative model by a factor $\mathcal{O}(C)$ and is outlined in Algorithm 8. A tree for PFT (Fig. 4.2) represents a sparse particle-belief tree. Tree nodes are constructed for actions and subsequent belief nodes, where each action node can have up to C children. More specifically, the belief nodes contain particles, representing a particle-belief state.

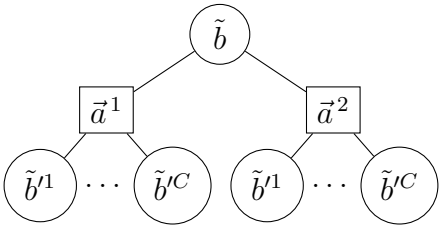


Figure 4.2: Sparse particle-belief tree.

Approximation quality. Lim et al. (2023) show that, under a set of assumptions, the approximation error of the optimal value of a PB-MDP to a POMDP is small with high probability. In addition to a finite horizon H, and a finite action space $\mathcal{A}$, the proof relies on a reward function r of the POMDP (MPOMDP in our case) that is bounded and Borel measurable, such that $||r||_\infty \leq r_{\max} < +\infty$, and the following holds for the maximum value $V_{\max} \equiv \dfrac{r_{\max}}{1-\gamma} < +\infty$. Finally, their proof assumes the state and observation spaces are continuous, but note that it also holds for discrete spaces as long as the Rényi divergence $d_\infty(\mathcal{P}^d \,||\, \mathcal{Q}^d)$ between the target $\mathcal{P}^d$ and the proposal $\mathcal{Q}^d$ distributions is bounded above almost surely by $d_\infty^{\max}$, up to a given maximum depth D, with $0 \leq d \leq D$:

Algorithm 8 Generative step for a particle belief-state.

procedure $\mathcal{G}_{PF}(\tilde{b}, \vec{a})$
  $s_0 \sim \tilde{b}$
  $\vec{o} \leftarrow \mathcal{G}(s_0, \vec{a})$
  $\tilde{b}' \leftarrow \emptyset, C \leftarrow |\tilde{b}|$
  for $i \leftarrow 1$ to C **do**
    $s'_i, r \leftarrow \mathcal{G}(s_i, \vec{a})$
    $w'_i \leftarrow \mathcal{O}(\vec{o} \,|\, \vec{a}, s'_i)$
    $\tilde{b}' \leftarrow \tilde{b}' \cup \{(s'_i, w'_i)\}$
  end for
  $\rho \leftarrow \frac{\sum_i w_i r_i}{\sum_i w_i}$
  return $\tilde{b}', \rho$
end procedure

$$d_\infty(\mathcal{P}^d || \mathcal{Q}^d) = \operatorname{ess\,sup}_{x \sim \mathcal{Q}^d} \tilde{w}(x) \leq d_\infty^{max} < +\infty, \tag{4.12}$$

Where $\tilde{w}(x)$ is the self-normalised weight as in typical sequential Monte Carlo (Sect. 3.2.3). More intuitively, this means that the ratio between the marginal $p(y)$ and conditional $p(y \,|\, x)$ observation probability cannot become infinitely large. Within these assumptions, the proof also holds in our discrete setting by replacing the integrals to (Riemann) sums, but we do not give it here as it is rather involved and beyond the point of this thesis. For more details on the proof for the continuous setting, we refer to Lim et al. (2020, 2023).

Sparse particle filter tree. Sparse-PFT is equivalent to UCT with particle-belief states. While it was designed for continuous state spaces, the fact that the tree branches on a fixed amount of belief nodes instead of the number of (joint) observations is also beneficial in our setting. Sparse-PFT constructs a sparse look-ahead particle-belief tree, depicted in Fig. 4.3, incrementally during a forward search by allowing each action node to expand up to C particle-belief nodes. The particle-belief nodes correspond to the states of the particle-belief MMDP (Def. 9). The root particle-belief $\tilde{b} \leftarrow \left\{(s_i, \frac{1}{C})\right\}_{i=1}^{C} \sim \bar{b}$ is sampled at every simulation iteration from the current belief $\bar{b}$, where $\bar{b}$ is maintained during the episode as in Algorithm 1. During forward search, UCB1 selects the actions $\vec{a}$. If the number of children of the action node $\vec{a}$ is less than C, then the particle-belief is simulated through $\mathcal{G}_{PF}$ (Algorithm 8) to obtain the next particle-belief $\tilde{b}'$ and belief-based reward ρ. Otherwise, $\tilde{b}'$ and ρ are sampled uniformly from one of its children. We continue traversing the particle-belief tree until we reach a leaf node or a predetermined maximum depth. The MLE $Q(\tilde{b}, \vec{a})$ is updated according to the return of a particle belief, i.e., the expected cumulative discounted belief-based reward, $P_t = \sum_t^H \gamma^t \rho_t$, which is used synonymously to the return R_t the update the expert predictions $Q(\tilde{b}, \vec{a}) \leftarrow Q(\tilde{b}, \vec{a}) + \frac{P_t - Q(\tilde{b}, \vec{a})}{N(\tilde{b}, \vec{a})}$ when the statistics are updated in the simulate procedure.

In MPOMDPs, PFT suffers from a similar problem as POMCP, namely the exponential number of joint actions that need to be considered via UCB1. To improve on the algorithm's weakness in large action spaces, we introduce two extensions.

4.3.2 Coordination Graph Particle Filter Tree

We alleviate the algorithm's weakness to large action spaces by incorporating the CG approximations from factored POMCP to the sparse particle filter tree. This combination produces two new algorithm variants.

Factored Statistics

For the first variant, we propose *factored statistics* PFT (FS-PFT), in which we keep factored UCB1 action statistics in the nodes of the particle filter tree. In addition to the belief-node count $N(\tilde{b})$, we maintain sets of statistics $Q_e(\tilde{b}, \vec{a}_e)$, $N(\tilde{b}, \vec{a}_e)$ in every particle filter belief node that predicts the Q-function for every component $e \in \mathcal{E}$, applying MoE optimisation (Eq. (4.2)) in the particle belief-states. Then, in a similar fashion to algorithms of Sect. 4.2, a graphical inference algorithm selects the maximal joint action $\vec{a}^{\#}$ by maximising over the sets of upper confidence bounds: induced by the coordination graph:

$$\arg\max_{\vec{a}^{\#}} \sum_{e \in \mathcal{E}} \mathcal{UCB}(Q_e(\tilde{b}, \vec{a}_e), N(\tilde{b}), N(\tilde{b}, \vec{a}_e)). \tag{4.13}$$

Factored Trees

For the second variant, we introduce *factored trees* PFT (FT-PFT), where we construct multiple particle filter trees in parallel for each $e \in \mathcal{E}$. Each tree has a local action space $\vec{a}_e$ and maintains individual statistics $Q_e(\tilde{b}, \vec{a}_e)$, $N_e(\tilde{b})$, $N_e(\tilde{b}, \vec{a}_e)$ for every $e \in \mathcal{E}$. The pseudo-code of the SIMULATE procedure is available in Algorithm 9 Since particle filter trees do not branch on observations, only the action space is factored in the trees. We use a single particle belief step in each layer and build the trees in parallel. Thus, every tree is constructed from the same simulated particle filter beliefs. Although the belief nodes might consist of the same particle filter belief, we maintain independent visit count statistics N_e for each belief node and associated local joint actions per tree, respectively. We suspect the improvement of this variant is an increase in node re-use and search depth due to the smaller factored action space in each tree. An overview of these trees with one layer of depth can be seen in Fig. 4.3. The inference equation for picking the maximal UCB action (using Eq. (3.2)) is given by:

$$\arg\max_{\vec{a}^{\#}} \sum_{e \in \mathcal{E}} \mathcal{UCB}(Q_e(\tilde{b}, \vec{a}_e), N_e(\tilde{b}), , N_e(\tilde{b}, \vec{a}_e)). \tag{4.14}$$

Figure 4.3: Factored sparse particle-belief trees.

Algorithm 9 FT-PFT's SIMULATE procedure.

procedure SIMULATE($\hat{b}, d$)
  if $\gamma^d < \epsilon$ **then**
    return 0
  end if
  for all $e \in \mathcal{E}$ **do**
    if $\hat{b} \notin T_e$ **then**
      for all $\vec{a}_e \in \vec{\mathcal{A}}_e$ **do**
        $T_e(\hat{b}, \vec{a}_e) \leftarrow (N_0(\hat{b}, \vec{a}_e), V_0(\hat{b}, \vec{a}_e), \emptyset)$
      end for
    end if
  end for
  $\vec{a} \leftarrow \max_{\vec{a}^\#} \sum_{e \in \mathcal{E}} \mathcal{UCB}(Q_e(\hat{b}, \vec{a}_e), N_e(\hat{b}), N_e(\hat{b}, \vec{a}_e))$ ▷ VE or MP (Eq. (3.2)).
  if $\exists_{e \in \mathcal{E}} \colon |Ch_e(\hat{b}, \vec{a}_e)| \neq C$ **then** ▷ If any node can be expanded.
    $\hat{b}', \rho \leftarrow \mathcal{G}_{PF}(\hat{b}, \vec{a})$ ▷ Generative transition (Algorithm 8).
    for $e \in \mathcal{E}$ **do**
      $Ch_e(\hat{b}, \vec{a}_e) \leftarrow Ch_e(\hat{b}, \vec{a}_e) \cup \{(\hat{b}', \rho)\}$
    end for
    $P \leftarrow \rho + \gamma \cdot \text{ROLLOUT}(\hat{b}', d+1)$
  else
    $\hat{b}', \rho \sim \{Ch_e(\hat{b}, \vec{a}_e) \mid e \in \mathcal{E}\}$
    $P \leftarrow \rho + \gamma \cdot \text{SIMULATE}(\hat{b}', d+1)$
  end if
  for all $e \in \mathcal{E}$ **do** ▷ Update statistics for every tree.
    $N_e(\hat{b}) \leftarrow N_e(\hat{b}) + 1$
    $N_e(\hat{b}, \vec{a}_e) \leftarrow N_e(\hat{b}, \vec{a}_e) + 1$
    $Q_e(\hat{b}, \vec{a}_e) \leftarrow Q_e(\hat{b}, \vec{a}_e) + \dfrac{P - Q_e(\hat{b}, \vec{a}_e)}{N_e(\hat{b}, \vec{a}_e)}$
  end for
  return P
end procedure

Chapter 5

Scalable Particle Filtering

As we have seen previously, the Bayesian filter belief update requires $\mathcal{O}(|\mathcal{S}|^2)$ computations. In MPOMDPs with large state spaces, which often contain millions to billions of states, this becomes intractable. To ensure we can scale to large problems, we represent this belief with *particle filters*. Due to the large size of these models, when many agents are involved, we run into trouble when updating the filter. In this chapter, we briefly describe why and then proceed to the proposed resolutions. We consider two filtering procedures, which each have their benefits and drawbacks.

- In the first method presented, we adopt a well-known monitoring method by Ng et al. (2002) to represent the belief by an empirical distribution that ranges over the product of a set of clusters.
- In our method, introduced secondly, there are multiple particle filters in an ensemble with differing importance weight update functions.

Particle filtering in MPOMDPs. In MPOMDPs, the observation signal becomes increasingly sparse as the number of agents increases, as it commonly depends on the probability of all individual observations. This can result in an impoverishment of the particles. Comparably, for unweighted filters in larger observation spaces, the likelihood to match the received joint observation in the rejection update is small. If the particle filter reaches a deprived state where no particles remain, the planner has to rebuild the belief from the recorded history or default to a baseline policy.

5.1 Factored Particle Filtering

In this section, we present an approach to increase the scalability of the particle filters. We adopt a method from the literature to improve the belief approximation in an MPOMDPs online planning setting with many agents.

Before giving the method, we introduce a required assumption.

Assumption 1 (Factored State Space)**.** *The state space $\mathcal{S}$ of an MPOMDP can be **factored** by $\mathcal{S} = \times_{\mathcal{X} \in \chi}$ if it can be defined by the product of a set of variables $\chi = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_{|\chi|}\}$, also known as factors or features. Furthermore, the observations Ω_i of each agent $i \in \mathcal{I}$ of the MPOMDP are influenced by a subset of the set of variables $\zeta_i \subseteq \chi$.*

Even though the state space might be factored, we do not assume that we have access to a factored simulator or that the value function is exactly factored. Leveraging the structure of state spaces is known to be effective in the filtering problem for large state spaces (Chen et al., 2022). We exploit Assumption 1 in a filtering method from Ng et al. (2002). We summarise and introduce its direct application to MPOMDPs here. For simplicity, we only consider weighted filtering, as it is unclear how to properly attribute relative importance to the particles in an unweighted setting.

Factored particle filtering (FPF, Ng et al. (2002)) works as follows. We define a set of clusters $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$, where each cluster $c \in \mathcal{C}$ maintains a set of *factored particles* $s_c^{(1)}, \ldots, s_c^{(N_c)}$, with K_c the number of particles in c, which may vary over time. In our case, the set of clusters $\mathcal{C}$ can be defined to be synonymous with the set of edges $\mathcal{E}$ from the CG. Let $\mathcal{I}_c \subset \mathcal{I}$ be the set of agents associated with cluster c and let ζ_i be the state variables associated with agent $i \in \mathcal{I}$. Then, $\chi_c \triangleq \cup_{i \in \mathcal{I}_c} \zeta_i$ are the state variables of the agents $\mathcal{I}_c$ associated with cluster c We define the set of factored particles $\mathcal{S}_c$ for cluster c, i.e., the cluster state space, as the product of the state variables $\mathcal{S}_c \triangleq \times_{\mathcal{X} \in \chi_c}$. The belief state is then approximated as the product of the cluster distributions, where particle filtering is used as a non-parametric density function, as given by:

$$\overline{b}(s) \approx \prod_{c \in \mathcal{C}} \frac{1}{K_c} \sum_{i=1}^{K_c} \delta_{s_c, s_c^{(i)}}, \tag{5.1}$$

where δ is the Kronecker delta function, s_c indexes the cluster variables of s as given by the cluster c and $s_c^{(i)}$ is the same but for particle $s_i \in \overline{b}$ in the filter.

5.1.1 Factored Filtering

Filtering consists of two main methods. Intuitively, the process is outlined as follows. We maintain full-state particles and update these with a standard filtering method, such as SIR (Algorithm 6). Then, we distribute the full-state particles over the clusters and construct full-state particles by sampling from these local subsets. This set of joined factored particles can be larger than the original set of full-state particles.

Projection. Firstly, we introduce the notion of the *projection* of a full state particle $s \in \mathcal{S}$ to a factored particle $s_c \in \mathcal{S}_c$. For every cluster $c \in \mathcal{C}$, we can find the factored particle s_c by projecting s to the subset of state variables of s that are in $\mathcal{S}_c$. In doing so, s is projected to a factored particle s_c for every cluster $c \in \mathcal{C}$.

Example 2. *Consider the following example of a projection, adopted from Ng et al. (2002), for clusters $c_1 = \{\mathcal{X}_1, \mathcal{X}_2\}$, $c_2 = \{\mathcal{X}_2, \mathcal{X}_3\}$:*

	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$
s_1	1	1	1
s_2	2	1	2
s_3	2	2	2

$\Rightarrow$

	$\mathcal{X}_1$	$\mathcal{X}_2$
$s_{1,1}$	1	1
$s_{2,1}$	2	1
$s_{3,1}$	2	2

and

	$\mathcal{X}_2$	$\mathcal{X}_3$
$s_{1,2}$	1	1
$s_{2,2}$	1	2
$s_{3,2}$	2	2

Algorithm 10 Sample-based join procedure for FPF.

1: **procedure** SAMPLEJOIN($\mathcal{C}, N$)
2: $\quad \bar{b} \leftarrow \emptyset$
3: $\quad$ **for** $i \leftarrow 1$ to N **do**
4: $\quad\quad w_i \leftarrow 1$, $\mathcal{Z} \leftarrow \emptyset$, $s_i \leftarrow \emptyset$
5: $\quad\quad$ **for** $c \in \mathcal{C}$ **do**
6: $\quad\quad\quad \mathcal{Y} \leftarrow \chi_c \cap \mathcal{Z}$
7: $\quad\quad\quad s_{i,c} \sim P_c^y$ ▷ P_c^y is consistent with assignment $y \in \mathcal{Y}$.
8: $\quad\quad\quad w_i \leftarrow w_i \cdot \frac{|P_c^{\mathcal{Y}}|}{|P_c|}$ ▷ Relative importance weight.
9: $\quad\quad\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \chi_c$
10: $\quad\quad$ **end for**
11: $\quad\quad \bar{b} \leftarrow \bar{b} \cup \{(s_i, w_i)\}$
12: $\quad$ **end for**
13: $\quad$ **return** $\bar{b}$
14: **end procedure**

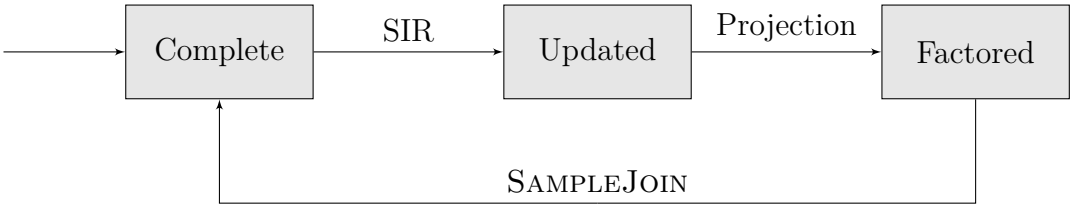


Figure 5.1: Schematic overview of FPF, adopted from Ng et al. (2002). The process is as follows. From a set of complete particles at time t, we update and re-sample these particles to arrive at a subsequent filtering distribution with uniform weights. Then, we project (Sect. 5.1.1) the particles to their factored representation. From these sets of factored particles, we construct a (larger) new set of particles by the sample-join procedure (Sect. 5.1.1), resulting in a set of complete particles at time $t+1$ with weights defined by the relative importance of the sample in the SAMPLEJOIN operation.

Joining particles. Secondly, we run the SAMPLEJOIN algorithm to build N full-state particles from the collection of projected particles, where N is the desired number of constructed particles. Let P_c be the set of particles of cluster c, and let $P_c^y \subseteq P_c$ be the set of particles that agree with the previously chosen variables $y \in \mathcal{Y}$. Then, the update procedure is given by Algorithm 10. Note that in practice, instead of using a fixed order at every iteration, we shuffle the order of enumerating every cluster in line 5.

5.1.2 Drawbacks

By introducing a reversible transition to a factored sub-space, this technique is able to produce a larger number of particles to sample from and can give a better approximation to the belief posterior in multi-dimensional systems. Unsurprisingly, the combination of these transformations introduces computational overhead. Although the projection operation is relatively straightforward, it can take more time when the number of clusters becomes larger. A more substantial problem with computational overhead and a large number of clusters is in joining the projected particles. Recall that we define the number of clusters to depend on the number of edges in the CG. With a large number of clusters,

the joining procedures arrive at a bottleneck when the factored spaces are coupled. As with every factored particle chosen, the constraint is introduced that particles sampled from a subsequent cluster must adhere to the previously assigned variables. It can thus be the case that even after significant iterations, there are no complete particles found. There are two possible solutions. Either the clusters can be merged arbitrarily to a predefined maximum number of clusters. Or, the clusters must be defined such that they are disjoint. Both are not optimal, as either the clustered representation is shrunk to fewer clusters, giving a lesser size of possible combinations of particles to sample from. Additionally, even with larger clusters, the same problem might occur. Or, by arbitrarily decoupling the clusters, we introduce independence between the state variables that might not exist in the actual system. Although this would be possible in theory, it would not be straightforward to design such a disjoint partition in an environment that contains state variables that are not assigned to a particular agent. Additionally, inter-dependencies between state variables over time are ignored. This might result in the construction of complete particles that do not accurately reflect the posterior distribution as there are no variables defined for the factored particles to 'agree' on. Finally, it does not adhere to the decomposition as given by the CG.

5.2 Locality-based filtering

To increase the scalability of the filter and decrease the chance of deprivation of the particle filter in large observation spaces, we introduce a general filtering approach independent of the online planning algorithm. This method is applicable to both factored variants of POMCP (Sect. 4.2) as well as PFT (introduced in Sect. 4.3.2).

Our method relies on the following assumption:

Assumption 2 (Individual Observation Model)**.** *Given an MPOMDP $\mathcal{M}$ with observation model $\mathcal{O}$, the observation model decomposes into individual observation probability functions $\mathcal{O}_i \colon \mathcal{S} \times \mathcal{A} \to \Delta(\{\Omega_i\})$ for each agent $i \in \mathcal{I}$. Individual observations probabilities are conditionally independent, given the successor state and the previous joint action. Therefore, we can rewrite the observation function as the product of individual observation probabilities:*

$$\mathcal{O}(\vec{o} \,|\, s', \vec{a}) = \prod_{i \in \mathcal{I}} \mathcal{O}_i(o_i \,|\, s', \vec{a}). \tag{5.2}$$

Note that this differs from the assumption in Messias et al. (2013), where they assume the local observation for each agent $o_i \equiv \vec{o}$ is synonymous with the joint observation.

We exploit the structure of a coordination graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$ in the following way. For every component $e \in \mathcal{E}$, we introduce a separate particle filter with K_e particles.[1] We choose K_e to be a number smaller than the original size K, such that $K = \sum_e K_e$. We distinguish an unweighted Monte Carlo filter as in POMCP and a bootstrapped weighted particle filter (with SIR). Given Assumption 2, we update the particle filters for the filters by the local part of the observation space $\vec{o}_e \in \times_{i \in e} \Omega_i$. We can retrieve the factored observation $\vec{o}_e$ from the joint observation $\vec{o}$ by retrieving the individual observations o_i, o_j for each agent $i, j \in e$.

Both the real environment observation as well as the sampled observation can be mapped to its factored version. Then, we can change the filtering procedure to be based

[1] Amato and Oliehoek (2015) mention the use of separate particle filters for each component in the factored trees pseudo-code in their appendix but do not give details in the main paper.

on the local component observation space of the corresponding component e for particle filter $\bar{b}_e$. We distinguish the weighted and unweighted cases:

- For a *weighted* filter $\bar{b}_e = \{(s_i, w_i)\}_{i=1}^{K_e}$, we compute the new importance sampling weight w_i' for particle s_i by $w_i' \propto w_i' \mathcal{O}_e(\vec{o}_e \mid s', \vec{a})$, where $\mathcal{O}_e(\vec{o}_e \mid s', \vec{a}) \equiv \prod_{i \in e} \mathcal{O}_i(o_i \mid s', \vec{a})$, and $o_i \in \Omega_i$.
- For the *unweighted* filter, $\bar{b}_e = \{(s_i)\}_{i=1}^{K_e}$, we reject based on the local part of the observation $\tilde{b}_e = \{s_i' \colon \vec{o}_e = \vec{o}_{e,i}\}$. In practice, the rejection sampling algorithm in Sect. 3.2.1 is adjusted to reject only based on the elements o_i, o_j of the observation $\vec{o}$ associated with the agents $i, j \in e$ in the edge.

With the combination of these locally biased particle filters, we can build an estimate of the belief posterior while requiring fewer particles on average. We analyse what this means in terms of the estimator of the belief in the subsequent subsection.

Since we run multiple particle filters in parallel, we must decide which filter to sample from at every simulation iteration. In the unweighted case, we use the naive approach of sampling from the filters with uniform probability. With the weighted filter, we can acquire more information to base our choice of filter to sample from. We use the likelihood of the weighted update as a statistic for the quality of the belief approximation (Katt et al., 2019). Intuitively, we sample more often from higher-quality filters. More precisely, we give filters that contain particles with a higher probability of generating the true observation a higher chance of getting sampled.

Given the likelihood of a belief update $\mathcal{L}_t$ at time t, we can sample a state s from a set $\{\bar{b}_e \mid e \in \mathcal{E}\}$ consisting of $|\mathcal{E}|$ particle filters $\bar{b}$ maintained during an episode with probability proportional to the likelihood of the individual particle filters:

$$s \sim \bar{b}_e \text{ w.p. } \frac{\mathcal{L}(\bar{b}_e)}{\sum_{e' \in \mathcal{E}} \mathcal{L}(\bar{b}_e)}. \tag{5.3}$$

Root sampling in Eq. (5.3) would occur in a slightly adapted version of Algorithm 2, where the state is sampled from the collection of particle filters. The general update procedure of multiple weighted particle filters is given in Sect. 5.2.

Algorithm 11 Locality-based multiple particle filters update using the SIR procedure.

```
1: procedure UpdatePF({⟨b_e, L_e⟩ | e ∈ E}, a, {o_e | e ∈ E}, τ)
2:     for e ∈ E do
3:         b'_e, L'_e ← SIR(a, o_e, b_e, L_e, O_e, T, τ)        ▷ Algorithm 6
4:     end for
5:     return {⟨b'_e, L'_e⟩ | e ∈ E}
6: end procedure
```

5.2.1 Multiple Estimators

In the previous, we introduced multiple particle filters conditioned on a subset of the observation space. Although not explicitly, our approach shares characteristics with methods that maintain an estimate of the posterior belief distribution by a combination of multiple estimators. We will consider the weighted variant in more detail here.

We essentially define the proposal distribution $\mathcal{Q}$ as a mixture distribution of M proposal distributions $\mathcal{Q}_\alpha = \{\langle \mathcal{Q}_i, \alpha_i \rangle\}_{i=1}^{M}$ (Owen, 2013), where $\mathcal{Q}_i$ are individual proposal distributions and α_i the (uniform) weights. In our case, $M \equiv |\mathcal{E}|$, and $\mathcal{Q}_e$ corresponds to the proposal distribution that contains the likelihood of observing the factored observation $p(\vec{o}_e \,|\, s, \vec{a})$ instead of the full likelihood $p(\vec{o} \,|\, s, \vec{a})$.

Multiple importance sampling. Elvira and Martino (2021) define a general framework of multiple importance sampling (MIS), we summarise them for analysing our filtering approach here. They define MIS as a method of taking a predefined K_m from M proposal distributions. Similarly, mixture importance sampling is a subset of MIS, where K_m is a random variable. A valid mixing scheme for simulating N samples from a mixture must satisfy $\Phi(x) = \frac{1}{N} \sum_i \mathcal{Q}_i$. Consider the case where, $n = 1, \ldots, N$ and we sample once per proposal distribution, $x_n \sim \mathcal{Q}_n(x)$. Then, consider the following two weighting schemes:

- Option 1. Standard MIS:

$$w_n = \frac{\mathcal{P}(x_n)}{\mathcal{Q}_n(x_n)} \tag{5.4}$$

- Option 2. Deterministic MIS:

$$w_n = \frac{\mathcal{P}(x_n)}{\Phi(x_n)} = \frac{\mathcal{P}(x_n)}{\frac{1}{N} \sum_{i=1}^{N} \mathcal{Q}_i(x_n)} \tag{5.5}$$

Option 2 generally has less variance than Option 1 (Elvira et al., 2019) but requires more proposal evaluations, which can become a computational bottleneck. Our approach is more in line with Option 1, as we do not consider the other densities when evaluating the local condition probability.

An ensemble approach. Our method does not exactly adhere to the formalisations of MIS. Essentially, our method consists of the approximation of the posterior distribution by a set of importance sampling estimators. The target distributions are then the filtering densities $p(x_t \,|\, x_{t-1}, y_t^e)$, where $y^e \equiv o_e$, and the proposal distribution is the transition dynamics function, which is the same across the estimators. The importance weights are then computed as the local observation conditional $w_t \propto p(x_t \,|\, y_t^e)$ as given by the relative importance between the target and proposal distribution $\approx \dfrac{p(x_t \,|\, x_{t-1}, y_t^e)}{p(x_t \,|\, x_{t-1})}$.

5.2.2 Limitations

Unfortunately, we did not succeed in finding an appropriate framework within the field of particle filtering and importance sampling to capture our approach. One of the big limitations of this method is we do not have any guarantees on the quality of the belief approximation. Additionally, there are some inherent limitations to introducing multiple locally biased particle filters. Because each filter is only based on information contained locally, it might be impossible to represent a joint belief that requires the knowledge of all agents, even when we sample states from all filters. One solution would be to combine the locally biased beliefs into a joint belief, which is known as the *assimilation* of beliefs, generally considered a difficult topic. However, earlier work on *factored beliefs*

in MPOMDPs has shown its effectiveness in approximating the global value function Messias et al. (2011).

Unfortunately, we must conclude that we cannot alleviate these concerns. But, in the context of our experimental evaluation, we find that the method works well in practice, achieving better results than the existing method of Sect. 5.1 while being less computationally expensive.

Chapter 6

Action Selection

Action selection in a coordination graph involves finding the maximising action over the set of conditional payoff functions.

We are tasked to find the joint optimal action $\vec{a}^*$ by proxy via our MoE estimate $\hat{Q}$ of the true Q-function. Thus, we try to find a deemed maximal action $\vec{a}^{\#}$:

$$\vec{a}^{\#} = \arg\max_{\vec{a}} \hat{Q}(\vec{a}). \tag{6.1}$$

We can decompose the global maximisation in terms of consequent local maximisation (Van Der Pol, 2016) by making use of the factored Q-functions from Sect. 4.1 and the distributive laws of maximisation and summation (Bishop, 2006):

$$\begin{aligned}
\max_{\vec{a}} \hat{Q}(\vec{a}) &= \max_{a_1} \cdots \max_{a_n} \hat{Q}(\vec{a}) \\
&= \max_{a_1} \cdots \max_{a_n} \left[\sum_{e \in \mathcal{E}} \hat{Q}_e(\vec{a}_e) \right] \\
&= \max_{a_1} \cdots \max_{a_n} \left[\hat{Q}_{e_1}(\vec{a}_1) + \cdots + \hat{Q}_{e_{|\mathcal{E}|}}(\vec{a}_{|\mathcal{E}|}) \right] \\
&= \max_{a_1} \left[\max_{a_2} \left[\hat{Q}_{e_1}(\vec{a}_1) + \left[\cdots \max_{a_n} \hat{Q}_{e_{|\mathcal{E}|}}(\vec{a}_{|\mathcal{E}|}) \right] \right] \right].
\end{aligned} \tag{6.2}$$

In the final representation of Eq. (6.2), the ordering, which can be permuted, determines the computational efficiency of the procedure. We compute this maximisation efficiently with algorithms adapted from the graphical inference literature. In the following section, we will disseminate two of such algorithms and introduce a technique that can alleviate the computational complexity by introducing a bounded error.

6.1 Algorithms

Given that we have a factorisation of the action space as induced by the coordination graph, we have to pick the best UCB1 action at every simulation step during the forward search with an inference algorithm. This introduces overhead during the search.

6.1.1 Exact Algorithm

variable elimination (VE) is an exact algorithm for inference in probabilistic graphical models. The global best action is computed by eliminating agents from the graph one by one, collecting their optimal conditioned action, and adding the resulting conditional

Algorithm 12 VE-based exact action selection algorithm for CGs.

Require: $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$, set of edge Q-functions $\mathcal{F} = \{Q_e \mid e \in \mathcal{E}\}$, elimination order κ

procedure VariableElimination
  for $i \in \mathcal{V}$ with loop-order κ **do**
    $\Phi \leftarrow$ all $f \in \mathcal{F}$ for which $\mathcal{A}_i \subset \text{Scope}[f]$
    Set $g = \max_{a_i} \sum_{f \in \Phi} f$, with $\text{Scope}[g] = \bigcup_{f \in \Phi} \text{Scope}[f] - \{a_i\}$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{g\}$
  end for
  $\mathcal{K} \leftarrow \emptyset$
  for $j \in \mathcal{V}$ with *reverse* loop-order κ **do**
    $\vec{a}_j^{\#} \leftarrow \text{argmax}_{a_j} \mathcal{F}$ subject to previous assignments $\forall_{k \in \mathcal{K}} \colon \vec{a}_k$ of $\vec{a}$
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{j\}$
  end for
  return $\vec{a}^{\#}$
end procedure

payoff to the graph as new edges until the last agent computes the action that maximises the final conditional strategy. Finally, a reverse pass decides the optimal action for all eliminated agents based on the fixed actions of other agents.

More formally, the algorithm maintains a set $\mathcal{F}$ of functions, with $\mathcal{F} = \{Q_e \mid e \in \mathcal{E}\}$ initially, where Q_e is the component value-function indexed by the edge $e \in \mathcal{E}$ in the graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$. Following Guestrin et al. (2001), the algorithm then proceeds as:

1. Pick any non-eliminated agent $i \in \mathcal{V} - \mathcal{X}$ ($\mathcal{X} = \emptyset$ initially).
2. Aggregate all $f \in \mathcal{F}$ for which $a_i \in \text{Scope}[f]$ into Φ.
3. Add function $g = \max_{a_i} \sum_{f \in \Phi} f$ to $\mathcal{F}$, with $\text{Scope}[g] = \bigcup_{f \in \Phi} \text{Scope}[f] - \{a_i\}$.
4. Add agent i to the set of eliminated agents $\mathcal{X} \cup \{i\}$.

The maximal action is then recovered by a reverse pass, using reverse ordering, picking the maximal action for every action over the set of functions. Pseudo-code for the procedure is given in Algorithm 12.

Elimination order. Even though the optimal action is found regardless of elimination order, the complexity of the procedure is heavily dependent on this ordering. Finding the optimal ordering is NP-complete (Kok and Vlassis, 2005), but good heuristics, e.g., sorting by degree, exist.

In order to find a sufficient ordering within a reasonable time, we employ an order induced by sorting by degree.

The degree $\deg \colon \mathcal{V} \to \mathbb{N}$ is a function that maps vertex $i \in \mathcal{V}$ to the local degree of vertex i in the graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$, representing the number of edges that *touch* vertex i, with $\sum_{i \in \mathcal{V}} \deg(i) = 2 \cdot |\mathcal{E}|$. We then define κ as an ordering that sorts elements $v \in \mathcal{V}$ by degree in ascending order.

6.1.2 Anytime Algorithm

The max-plus (MP) algorithm for computing the maximum a posteriori (MAP) configuration is analogous to belief propagation in graphical models (Pearl, 1989). Finding the MAP configuration is equivalent to finding the optimal joint action in a CG (Vlassis et al., 2004). Messages are sent between nodes in an iterative fashion. The messages μ_{ij} represent the running updates of the locally optimised payoff functions between agent i and j over the edges of the CG (Kok and Vlassis, 2005). Messages are computed by the following equation:

$$\mu_{ij}(a_j) = \max_{a_i} \left[Q_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(a_i) \right], \tag{6.3}$$

where $\Gamma(i)$ denotes the neighbours of i in the graph. In practice, the messages are normalised after the maximisation by subtracting a normalisation constant,

$$c_{ij} = \frac{1}{|\mathcal{A}_j|} \sum_{a_j \in \mathcal{A}_j} \mu_{ij}(a_j),$$

from Eq. (6.3) to prevent the messages from continuously increasing, which is especially prevalent in graphs with cycles (Wainwright et al., 2004). Messages are passed until they converge, or an anytime signal is received. The MP algorithm is anytime since messages are approximations of the exact value, meaning performance and computational efficiency can be interchanged. Furthermore, messages need only to sum over the received messages from its neighbours defined over the actions of the receiving instance instead of enumerating all possible action combinations as in VE. Additionally, MP requires no change in the graph topology as opposed to the eliminations of VE. Pearl (1989) proves that MP converges in finitely many iterations in acyclic graphs and Wainwright et al. (2005) prove the existence of fixed points for MAP in general graphs by making use of the spanning tree properties of the graph. The convergence speed of MP can be increased by using the same ordering of message passing as the elimination ordering heuristic of VE (Loeliger, 2004), such as the ordering by degree described in Sect. 6.1.1. In optimal cases with the right ordering, MP can converge in just one forward and backward pass, giving the same result as the exact counterpart. Kok and Vlassis (2005) study the performance of MP and VE in a static CG action selection without confounding influences. We refer to their extensive empirical analysis for an objective comparison of the selection procedures.

After reaching convergence or the computational budget, we find the maximal action by selecting the individual best actions locally:

$$a_i^{\#} = \operatorname*{argmax}_{a_i} \sum_{j \in \Gamma(i)} \mu_{ji}(a_i). \tag{6.4}$$

Finally, we note that Max-Plus can be computed in a distributed manner (Kok and Vlassis, 2005), in theory enabling the use of the combined computational power of the agents.

6.1.3 Eliminating Cycles

A large factor in deciding the run-time complexity of both classes of algorithms is the topology of the coordination graph. In MP, the convergence guarantee does not hold in

Algorithm 13 MP-based anytime action selection algorithm for CGs.

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Q_{ij} for $(i,j) \sim e \in \mathcal{E}$, $q = -\infty$, Message order κ

procedure MaxPlus(c, M)
  $\mu_{ij}^0(a_j) = 0$ for $(i,j) \in \mathcal{E}$, $a_j \in \mathcal{A}_j$
  for $t \leftarrow 1$ to M **do**
    for $i \in \mathcal{V}$ with order κ **do**
      for $j \in \Gamma(i)$ **do**
        Compute μ_{ij} using Eq. (6.3)
        $\mu_{ij}^t(a_j) \leftarrow \mu_{ij}^{t-1}(a_j) - \frac{1}{|\mathcal{A}_j|} \sum_j \mu_{ij}(a_j)$
      end for
      Compute a_i using Eq. (6.4) and set $\vec{a}_i \leftarrow a_i$
    end for
    if $u(\vec{a}) > q$ **then** ▷ Anytime extension.
      $\vec{a}^* \leftarrow \vec{a}$
      $q \leftarrow u(\vec{a})$
    end if
    if $||\mu_{ij}^t - \mu_{ij}^{t-1}||_1 \approx 0$ **then** ▷ Convergence heuristic.
      break
    end if
  end for
  return $\vec{a}^*$
end procedure

graphs with cycles. Thus, the algorithm requires an unknown number of iterations before converging to a—possibly sub-optimal—value. In VE, a graph with cycles incurs a large induced width of the elimination tree during the execution of the algorithm. We build a spanning tree for a CG to eliminate cycles. To compute a maximum spanning tree (MST), e.g., with Kruskal's algorithm (Cormen et al., 2009), we require edge weights. We find these by taking the maximum value of a local Q-function corresponding to an edge $e \in \mathcal{E}$ as the sum of the maximum value with respect to both agents in the edge (Rogers et al., 2011):

$$w_e = \max_{a_i} \left[\max_{a_j} Q_e(a_i) - \min_{a_j} Q_e(a_i) \right] + \max_{a_j} \left[\max_{a_i} Q_e(a_j) - \min_{a_i} Q_e(a_j) \right], \tag{6.5}$$

where $i, j \in e$ and $a_i \in \mathcal{A}_i, a_j \in \mathcal{A}_j$. Note that in practice, we negate the weights w_e as we are looking to find the MST from a *minimum* spanning tree algorithm. We compute the spanning tree before action selection from the approximate Q-values at that time during the search. Although this introduces an approximation error into the value recovered, this error is bounded by the sum of the edges removed by the spanning tree. More formally, if we define the maximum spanning tree for the graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$ as $\mathcal{MST} = (\mathcal{V}, \hat{\mathcal{E}})$ with $\hat{\mathcal{E}} \subseteq \mathcal{E}$, and we define the set of edges not in the tree as $\mathcal{R} \triangleq \mathcal{E} \setminus \hat{\mathcal{E}}$ the introduced error $E_{\mathcal{MST}}$ is bounded by $E_{\mathcal{MST}} \leq \sum_{e \in \mathcal{R}} w_e$. Despite this error, the cycle elimination makes the action selection procedure tractable in problems with *many* agents.

Algorithm 14 coordination graph (CG) maximum spanning tree (MST)

Require: Coordination Graph $\mathcal{CG} = (\mathcal{V}, \mathcal{E})$, Edge Q-functions Q_e for $e \in \mathcal{E}$

procedure CGSpanningTree
  Let $\mathcal{W}$ be an array to hold the edge weights for each $e \in \mathcal{E}$ initialised to $\mathbf{0}$.
  for $e \in \mathcal{E}$ **do**
    $w_{ij} \leftarrow \max_{a_i} \left[\max_{a_j} Q_e(a_i) - \min_{a_j} Q_e(a_i)\right]$
    $w_{ji} \leftarrow \max_{a_j} \left[\max_{a_i} Q_e(a_j) - \min_{a_i} Q_e(a_j)\right]$
    $\mathcal{W}_e \leftarrow -(w_{ij} + w_{ji})$ ▷ Negative weights.
  end for
  return Kruskal$(\mathcal{CG}, \mathcal{W})$
end procedure

6.2 Comparison

In order to determine which algorithm best suits what settings, we relate some characteristics of both algorithms here.

6.2.1 Computational Complexity

Variable Elimination has a run-time complexity of $\mathcal{O}(|\mathcal{E}| \cdot |\mathcal{A}_{\max}|^w)$ Amato and Oliehoek (2015), where $|\mathcal{E}|$ is the number of agents, $|\mathcal{A}_{\max}|$ is the cardinality of the largest action-set, and w is the induced width of the coordination graph. The induced width is synonymous to the largest clique to be computed during node elimination.

Max-Plus has a run-time complexity of $\mathcal{O}(|\mathcal{V}| \cdot |\mathcal{E}|)$ that is linear in the number of nodes and edges in the graph. Thus, the run-time of Max-Plus is polynomial in the size of the graph.

VE can be the better choice in settings where the action and observation spaces of multiple connected agents can be merged in a hyper-graph. In terms of scalability, VE is a very efficient approach to finding the exact maximal action in loosely connected or sparse graphs, but in comparison to MP it fails to scale well to coordination schemes with a large number of agents and interactions due to its exponential worst-case complexity. However, MP can also suffer from dense graphs in practice. However, MP is an anytime algorithm, which makes it sufficient for settings in which the computational budget is tight or where a call to VE might cost too much time. In the factored algorithms variants of Sect. 4.2, a large part that decides the number of simulation calls in the case of a time-constrained computational budget depends on the time spent in these graphical action selection procedures.

6.2.2 Exploration

As we saw in Sect. 3.1.1, UCT underlies the POMCP algorithm, which in turn relies on the UCB1 value of actions to decide on the balance between exploration and exploitation. The pseudo-code of Algorithm 12 and 13 does not contain these exploration bonuses explicitly. For VE, we implement the algorithm as defined in the paragraph above. We proceed similarly to its introduction to UCB1 action selection by Amato and Oliehoek (2015) and add the exploration bonus component-wise during the eliminations of the nodes of the graph as in Eq. (3.2). For MP, we follow the edge-wise exploration methodology

as coined by Choudhury et al. (2022). Similarly, edge exploration is added only after the final iteration of message passing in order to avoid divergent behaviour in cyclic interaction graphs. Instead of state-based statistics in the MMDP setting, we maintain history-based statistics in the belief tree. Note that we ignore the individual node-level utility and exploration bonus for generalisation purposes, as we do not assume a factored or local reward vector and solely rely on the global joint reward signal of the simulator. If we use factored trees, we can factorise the joint-action exploration bonus, similar to edge-wise exploration, by using the factored statistics of the individual trees. The formula in Eq. (6.3) is augmented with the exploration bonus $\mathcal{UCB}(0, N(\vec{h}_e), N(\vec{h}_e, \vec{a}_e))$ during a final additional message passing round.

Chapter 7

Experimental Evaluation

As our algorithms compute solutions *online*, we require a thorough empirical evaluation in order to adequately assess their performance. This chapter presents a description of the benchmarks and the details of the computational infrastructure employed in the empirical evaluation. We evaluate our methods on a set of four benchmarks and compare average performance across multiple episodes with random initialisation. The following section contains our analysis and interpretation of the results of this procedure.

7.1 Benchmark descriptions

Here, we outline the descriptions of the four benchmarks.

7.1.1 Firefighting in a Line

In a similar fashion to the introduction of the factored POMCP algorithms (Amato and Oliehoek, 2015), we adopt the FireFightingGraph (FFG) environment (Oliehoek et al., 2008). Agents are standing in a line, and houses are located to the left and right of each agent. Given n agents, this gives $n_h = n + 1$ houses in total. A problem instance with three agents i, j, k, thus giving four houses, is drawn in Fig. 7.1. This drawing includes the CG, which in this case, consists of two edges. Each house has an associated fire level in $[0, n_f)$, with $n_f = 3$, which indicates the severity of the fire. A state is comprised of the fire levels of each house, giving a state space of $|\mathcal{S}| = n_f^{n_h}$. Fire spreads more quickly between adjacent burning houses. The agents can move to either the left or right house in a deterministic fashion and receive a binary "flames" observation that indicates noisily whether the house the agent visited was burning.[1] The state, action $|\mathcal{A}| = 2^n$, and observation space $|\Omega| = 2^n$ grow exponentially with the number of agents. Unless specified otherwise, we limit the maximum number of steps in the environment to $H = 10$.

7.1.2 System Administration

We follow Choudhury et al. (2022) in their definition of SysAdmin as an MMDP. The environment consists of a set of computers connected in a graph, where each vertex represents a computer. Each individual computer i has two state variables; status $S_i \in$

[1] Note that Amato and Oliehoek (2015) introduced a third non-specified action, which we leave out.

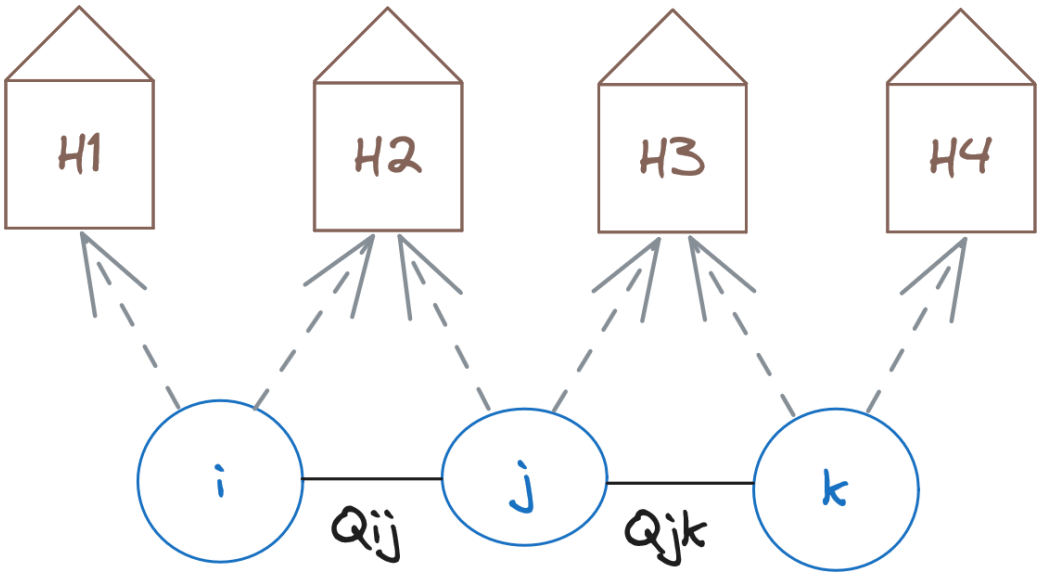


Figure 7.1: FFG with $n = 3$ agents i, j, k and consequently $n_h = n + 1 = 4$ houses.. Agents are connected in a line formation, resulting in two expert Q-function predictions, indicated by Q_{ij}, and Q_{jk}. These could also be indicated by the edge indices Q_{e_1} and Q_{e_2}, respectively.

$\{\texttt{GOOD}, \texttt{FAULTY}, \texttt{DEAD}\}$ and load $L_i \in \{\texttt{IDLE}, \texttt{LOADED}, \texttt{SUCCESS}\}$. Dead machines increase the chance of their neighbouring machines going faulty. Actions for each agent are either a `NOOP` or a reboot. A reboot sets the status to GOOD, but any progress in loading is lost. The system receives a reward of $+1$ if a process completes successfully. Although the environment is an MMDP, we run our algorithms on this benchmark as if it were an MPOMDP. We define the initial belief over all possible initial states and perform particle filtering as in an MPOMDP, albeit with unweighted particle filters. Unless specified otherwise, we limit the maximum number of steps in the environment to $H = 50$. In SysAdmin, there is no way for an episode to end prematurely.

7.1.3 Rocksampling

RockSample (Smith and Simmons, 2004) is a traditional benchmark for online planners (Silver and Veness, 2010). Similarly to Cai et al. (2021), we introduce multi-agent RockSample (MARS) by increasing the number of agents that need to be controlled. Particularly, we model the problem as an MPOMDP. MARS environments are defined by their size m, the number of agents n, and the number of rocks k. We write MARS(m, k) to indicate the MARS environment with size m and k rocks. The agents are spread out on the left-most x-coordinate on the $m \times m$ grid and are tasked with sampling rocks before leaving the area on the right-most x-coordinate. The action space is $|\mathcal{A}| = (5 + k)^n$, where k is the number rock sample actions. The observation space $|\Omega| = 3^n$ is relatively small. The agents receive a reward of ± 10 depending on whether the rock they sampled was good and $+10$ upon leaving the area on the right side. During the search, there is a -100 penalty for leaving the grid at any other side (i.e., west, north, or south) of the map. This -100 penalty can be found in the code of POMCP for the single-agent RockSample problem (Silver and Veness, 2010) and in HyP-DESPOT for MARS (Cai

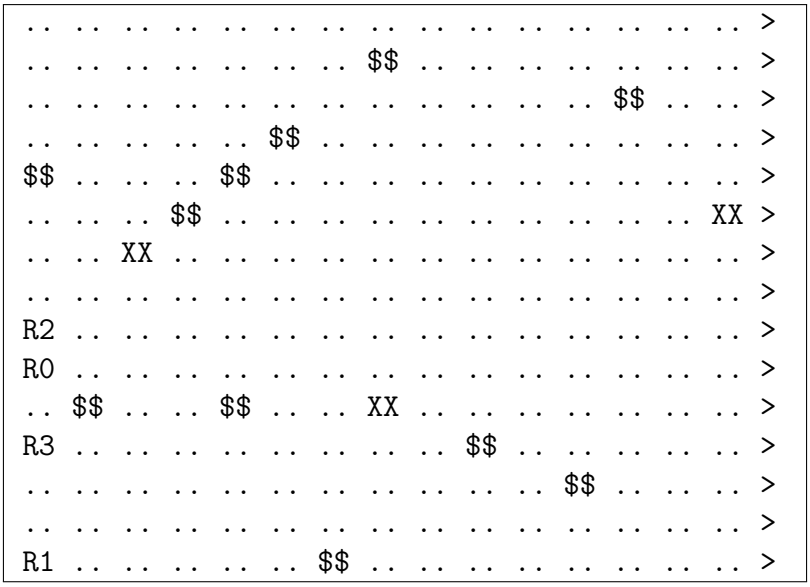


Figure 7.2: Random initialisation of MARS$(15, 15)$ with $n = 4$ agents, which are indicated with Rn ($n \in \{0, 1, 2, 3\}$). Greater-than signs (>) indicate the points in the grid the agents can leave the area. The symbol pairs `$$`'s and `XX`'s indicate good and bad rocks, respectively. Unsurprisingly, the double dots (`..`) indicate empty grid points.

et al., 2021), but it is not mentioned in either of their benchmark descriptions or in the original introduction of RockSample (Smith and Simmons, 2004). We do not inflict this penalty during episode execution to ensure results remain legible. Unless specified otherwise, we limit the maximum number of steps in the environment to $H = 40$. We present an ASCII-based render of a random initialisation in Fig. 7.2.

7.1.4 Capturing a Target

Inspired by Xiao et al. (2021), we introduce CaptureTarget (CT) as an MPOMDP problem. n agents move around and try to capture a single target in a $m \times m$ grid world. States consist of the (x, y) coordinates of each agent and the target, giving a state space of $\mathcal{S} = m^{2n+2}$. The target moves to the position which is the furthest away from all agents. Unlike the toroidal world of Xiao et al. (2021), our grid is confined by walls. Rewards are sparse; +1 if one or more agents capture the target and 0 otherwise. The episode ends if the target is captured or the maximum horizon is reached. Observations are binary and indicate whether the agent can 'see' the target either horizontally or vertically in the grid. However, they are very noisy, as there is a 30% chance of the target flickering, i.e., it can't be seen, and a 10% sensor measurement imprecision for every agent. Note that we report the averaged undiscounted rewards over a set of episodes, which can be summarised as a *capture rate*, i.e., the percentage of episodes in which the target was captured. Unless specified otherwise, we limit the maximum number of steps in the environment to $H = 50$. A static render of the environment is depicted in Fig. 7.3.

7.2 Set-up

All POMCP and PFT algorithm variants are based on the same Python 3 code, available in this Gitlab repository. We used the DESPOT and MARS C++ implementation from

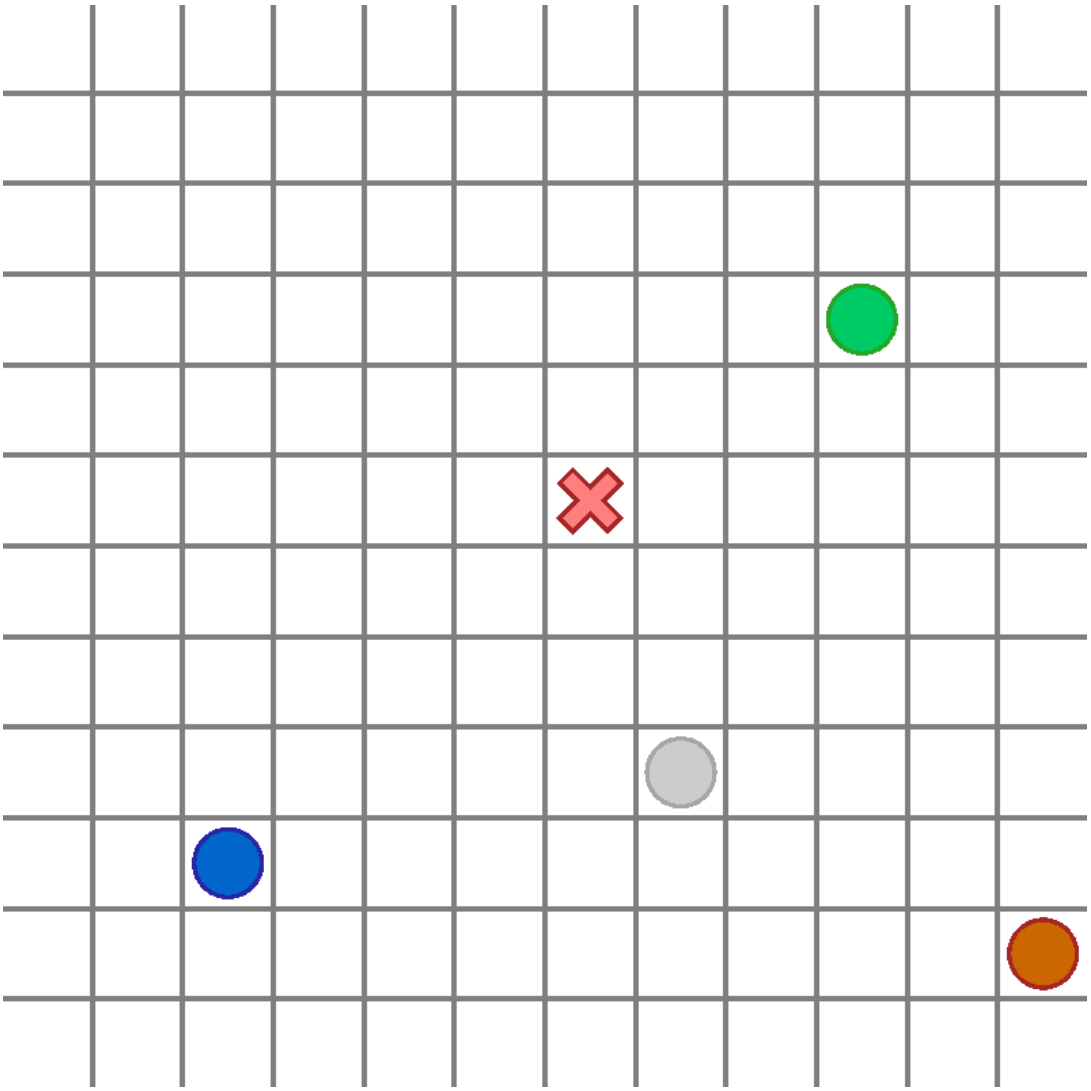


Figure 7.3: Random initialisation of CaptureTarget(12×12) with $n = 4$ agents. The target is indicated by the red cross and moves away from the closest agents (Xiao et al., 2021). Both the agents and the target can take steps in any of the four Manhattan-based directions.

the code of HyP-DESPOT (Cai et al., 2021). All code runs only on a single core of an Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz and 256 GB RAM (8 x 32GB DDR4-3200). Our Python code ran episodes in parallel with 34 threads such that each episode had access to $256/34 \approx 7.5$GB of RAM. The DESPOT experiments ran single-threaded. All results reported are averaged cumulative discounted rewards over 100 episodes and are reported with 95% confidence intervals. An OOR entry in a table indicates that the algorithm ran out of resources, i.e., time or memory. We did not run an extensive hyper-parameter optimisation for any algorithm. As such, the discount factor γ and UCB1 exploration constant c were set depending on the environment.

Particles. In chapter 5, we introduced multiple particle filters in parallel. In order to maintain equal comparison in the evaluation, we set the number of particles in the flat counters K to equal the sum of particles $\sum_e K_e$ in the factored filters. For example, if we have three factors with $K_e = 100$, then the flat counterpart has $K = 300$. In CT and MARS, we had 100 particles per filter. In FFG, the base number of particles was set to 20. If the particle filter belief is deprived at any point in time during the episode, the policy defaults to a random policy. For PFT, the particle number C was fixed to 20.

Graphs. In line with Castellini et al. (2021), this thesis studies large problems without natural structure in the experimental evaluation, such as MARS and CaptureTarget, without a natural graph structure. We aim to show that an arbitrary CG can act as a heuristic to make these problems tractable. However, choosing such a heuristic structure might be difficult. This problem has been considered in the literature, e.g., by learning the graph (Kok and Vlassis, 2006b). In our experiments, we chose the decomposition based on the objective. The "line" composition is a tree-like decomposition inspired by the FireFightingGraph domain. The "team" composition increases locality by disconnecting teams of agents from the remaining agents.

7.3 Results

This section contains the results and analysis thereof from the experimental evaluation described in the previous part of the chapter. Our experimental evaluation is, apart from studying numerous algorithm variants, primarily aimed at evaluating our contributions, which we summarise one by one below:

- We extend the PFT algorithm to MPOMDPs and introduce FS-PFT and FT-PFT as its factored extensions.
- We consider the notion of progressive widening on the observation space of the POMCP algorithm with factored UCB1 statistics (FS-POMCPOW).
- We consider weighted particle filtering for (factored) POMCP.
- We introduce a general method for scaling both weighted and unweighted filtering in a coordination graph setting. We employ this method on FT-POMCP and FT-PFT.
- We consider anytime action selection for factored POMCP.
- We introduce a method, namely MST, to scale graphical action selection algorithms in coordination graphs with many nodes and edges.

FT	factored-trees	constructs a separate tree per factor in the CG
FS	factored-statistics	retains UCB1 statistics per factor in the CG
PFT	sparse particle filter tree	operates on the particle-belief MMDP states
WPF	weighted particle filter	weighted particle filtering via SIR.
FPF	factored particle filtering	Method by Ng et al. (2002) we adopted for MPOMDPs
MP	max-plus	anytime action selection via message passing
VE	variable elimination	exact action selection via node elimination
MST	maximum spanning tree	algorithm that extracts trees from cyclic graphs

Table 7.1: Summary of the acronyms commonly used in the experimental evaluation. For a full list, we refer to the glossary at the end of this thesis.

7.3.1 Overview

In Table 7.1, we refresh some of the acronyms that are related to our contributions summarised above. Considering that this thesis introduced numerous algorithm variants and extensions related to online planning in MPOMDP, we supply an overview of the algorithm variants under test in Table 7.2. We go over them shortly here. POMCP (Silver and Veness, 2010) was introduced in Sect. 3.1.2, we also consider a version with a weighted particle filter. They serve as a baseline in our experiments. In Sect. 4.2.1, we extend POMCPOW (Sunberg and Kochenderfer, 2018) to the coordination graph setting, resulting in FS-POMCPOW. Factored POMCP (Amato and Oliehoek, 2015) was introduced in Sect. 4.2. FS-POMCP-VE and FT-POMCP-VE are the variants that were originally introduced, it is unclear if they were implemented with factored or multiple particle filters. We test them with our weighted and unweighted locality-based filter

and with the anytime MP algorithm for action selection. PFT (Lim et al., 2020) was introduced in Sect. 4.3.1. We extended it to Coordinated PFT in Sect. 4.3.2.

	Variants	WPF	Locality-Based Filtering	Action Selection
POMCP	Flat	**Yes**/No	No	N.A.
POMCPOW	Flat/**FS**	Yes	No	N.A.
PFT	Flat	Yes	No	N.A.
Factored POMCP	FS/FT	**Yes**/No	**Only FT-POMCP**	VE/**MP ± MST**
Coordinated PFT	**FS**/**FT**	Yes	**Only FT-PFT**	**VE**/**MP ± MST**

Table 7.2: Overview of the algorithm variants that we consider in the experimental evaluation. "Yes/No" implies that the algorithm can be implemented with and without this method, similarly, "VE/MP" implies it can run with both. For simplicity, we only consider locality-based filtering in variants with *factored trees*. Variants in bold are either our improvements or variants considered for the first time by us.

7.3.2 Analysis

We empirically evaluate the effectiveness of the proposed solutions on benchmarks with many agents. The key question is **Q1**:

Does the use of CGs accelerate online planners for MPOMDPs?

We evaluate this on two types of benchmarks: **Q1a**: on benchmarks with a natural coordination graph (FFG and SysAdmin) and on **Q1b**: on artificially chosen CGs (MARS and CT). Furthermore, we are interested in **Q2**: the sensitivity of the performance with respect to the number of particles, **Q3**: the sensitivity of the performance with respect to the action selection algorithm, and **Q4**: the performance comparison of PFT and POMCP variants.

Q1a: the factored algorithms out-scale flattened approaches. We can see in Fig. 7.6a for FFG that the factored approaches out-scale their flat counterparts, even with fewer simulations. In the relatively simple setting with four agents, all algorithms approach the value achieved by a full-width solver with 30 minutes of offline solving time. However, if the number of agents increases, only the factored approach manages to achieve good results. This shows clearly that the factored-value (FS/FT) variants of POMCP and PFT outperform their flat counterparts. This can be explained by the fact that the domain has a structure that has a nearly factored Q-function, such that the MoE in Eq. (4.2) approximately holds (see Theorem 4.1.1 and Amato and Oliehoek (2015)).

To study the best-performing algorithms in FFG, ours, and FT-POMCP, we plot the comparison in the 64-agent setting in more detail in Fig. 7.4. Our algorithms (solid) perform just as well or better, but there is a slight overlap in confidence intervals between our best and FT-POMCP (dotted).

For MARS, on the small map in Table 7.3a, we see similar results for five agents. For three agents, regular POMCP works well, and for four agents POMCP(OW) with weighted particle filtering achieves superior performance within the fifteen seconds time frame. DESPOT achieves a good result in the larger map with three agents and the

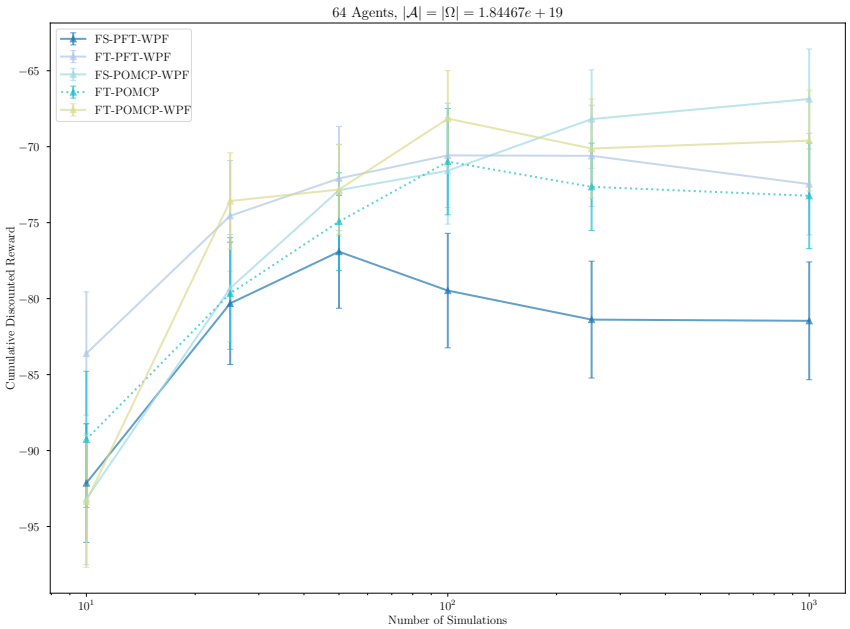


Figure 7.4: FireFightingGraph ($H = 10, c = 5, \gamma = 0.99$), like Fig. 7.6a, zoomed on the 64 agent setting for various numbers of allowed simulations. The x-axis is logarithmic.

smaller map with four agents. For the latter result, the algorithm exceeded the allowed time, requiring 20 seconds on average. Overall, DESPOT performs well but struggles with the computational budget and often runs out of resources (time or memory). Our factored approaches were able to scale well to multiple agents but achieved worse performance on average for fewer agents.

Q1b: the CG-induced factorisation can be employed as a heuristic in problems without natural factorisation. Not all benchmarks contain Q-functions that factor as well as in the FFG benchmark. In these cases, one may still assume a CG as a chosen heuristic to make the problem tractable for online planning algorithms. CT and MARS are such benchmarks. With n agents, the *line* connects all agents in a pairwise line, resulting in $n-1$ factors. The *teams* formation connects two agents in a factor, giving $\frac{n}{2}$ factors. From our results in Table 7.3 to 7.5, we conclude that the coordination graph can act as a heuristic to make unfactored problems with many agents tractable. The chosen graph topology influences the performance, as is visible in the results of CT (Table 7.4). Here, the team coordination functions much better, as it allows the two agents to act independently from the other two in their attempt to capture the target. In MARS, we see less difference between the graph topologies in terms of performance. However, we do have a worse performance when introducing factorisation as compared to flat POMCP.

Q2: the number of particles has a marginal effect on the performance. The number of particles has a small effect on the performance of the algorithms, which further diminishes when using our technique from Sect. 5.2. For example, in FFG with 16 agents, *flat* POMCP enjoys $\approx 6\%$ increase in performance between $K = 150$ and $K = 1500$ particles, which is within the confidence interval. FT-POMCP-VE reaches comparable performance with both $K_e = 10$ particles and $K_e = 100$ particles in each filter. In Fig. 7.5, we plot the performance on FFG with 4, 8, and 16 agents with respect to various numbers of particles in the filters on the x-axis. We compare the performance of

(a) Small map: MARS($m = 7, k = 8$), $\gamma = 0.95, c = 1.25$

Environment	MARS(7, 8)			
Nr. of Agents	3	4	5	6
FS-PFT-VE	3.9 ± 1.5	-3.9 ± 1.6	-6.9 ± 1.7	-9.4 ± 2.0
FS-POMCP-MP-WPF	0.3 ± 1.2	0.9 ± 1.8	-1.2 ± 2.6	-1.8 ± 2.8
FS-POMCP-VE	0.7 ± 2.0	2.8 ± 3.0	$\mathbf{2.8} \pm 2.8$	-0.8 ± 3.8
FS-POMCP-VE-WPF	-1.0 ± 1.1	0.8 ± 1.3	$\mathbf{2.7} \pm 1.2$	2.1 ± 1.4
FS-POMCPOW-MP	-0.2 ± 1.2	0.5 ± 1.0	0.7 ± 2.3	0.0 ± 2.7
FS-POMCPOW-VE	1.2 ± 1.1	4.5 ± 1.6	$\mathbf{1.7} \pm 1.3$	$\mathbf{8.9} \pm 0.8$
FT-PFT-VE-WPF	0.4 ± 1.6	0.2 ± 1.4	1.0 ± 1.6	-11.5 ± 2.5
FT-POMCP-MP	2.5 ± 1.8	0.4 ± 1.1	$\mathbf{1.8} \pm 1.9$	-0.4 ± 3.6
FT-POMCP-MP-WPF	1.7 ± 1.1	1.7 ± 0.7	-0.1 ± 2.1	1.0 ± 0.8
FT-POMCP-VE	5.8 ± 1.1	3.0 ± 1.3	$\mathbf{2.7} \pm 1.2$	1.1 ± 1.2
FT-POMCP-VE-WPF	1.8 ± 1.1	2.9 ± 1.1	-1.3 ± 1.3	3.4 ± 1.5
DESPOT	OOR	OOR	OOR	OOR
PFT	-6.0 ± 2.5	-9.1 ± 2.8	-0.0 ± 0.7	OOR
POMCP	12.8 ± 2.8	7.5 ± 3.7	-4.6 ± 2.3	OOR
POMCP-WPF	$\mathbf{24.6} \pm 2.0$	6.4 ± 1.9	-1.9 ± 1.5	OOR
POMCPOW	14.7 ± 1.9	$\mathbf{11.1} \pm 1.8$	-0.2 ± 0.6	OOR

(b) Medium map: MARS(11, 11), $\gamma = 0.95, c = 1.25$

Environment	MARS(11, 11)			
Nr. of Agents	3	4	5	6
FS-POMCP-MP-WPF	0.4 ± 0.8	-0.2 ± 0.4	-1.7 ± 0.7	-0.3 ± 0.3
FS-POMCP-VE-WPF	-1.1 ± 0.6	1.5 ± 1.2	-3.7 ± 1.2	3.9 ± 1.0
FS-POMCPOW-MP	-0.3 ± 0.7	0.1 ± 0.3	0.3 ± 0.9	-1.5 ± 1.3
FS-POMCPOW-VE	2.2 ± 0.9	2.1 ± 1.0	-2.4 ± 1.2	-2.1 ± 1.2
FT-POMCP-MP	0.0 ± 0.9	-0.0 ± 1.4	-0.0 ± 0.7	-1.2 ± 1.2
FT-POMCP-MP-WPF	0.5 ± 0.4	0.1 ± 0.2	0.0 ± 0.2	-0.1 ± 0.2
FT-POMCP-VE	1.5 ± 1.0	1.7 ± 1.3	$\mathbf{4.3} \pm 1.1$	4.3 ± 0.9
FT-POMCP-VE-WPF	5.2 ± 0.9	-3.3 ± 0.8	-3.6 ± 0.9	$\mathbf{6.8} \pm 1.0$
DESPOT	OOR	OOR	OOR	OOR
PFT	-17.7 ± 2.2	1.7 ± 1.6	-0.1 ± 0.5	OOR
POMCP	$\mathbf{17.0} \pm 1.9$	-3.6 ± 2.9	-0.1 ± 0.2	OOR
POMCP-WPF	$\mathbf{15.5} \pm 1.9$	$\mathbf{8.5} \pm 1.3$	0.0 ± 0.0	OOR
POMCPOW	2.4 ± 1.0	-0.8 ± 1.3	-0.0 ± 0.5	OOR

Table 7.3: MARS experiments with three maps and various numbers of agents. Maximum time per step is **five** seconds.

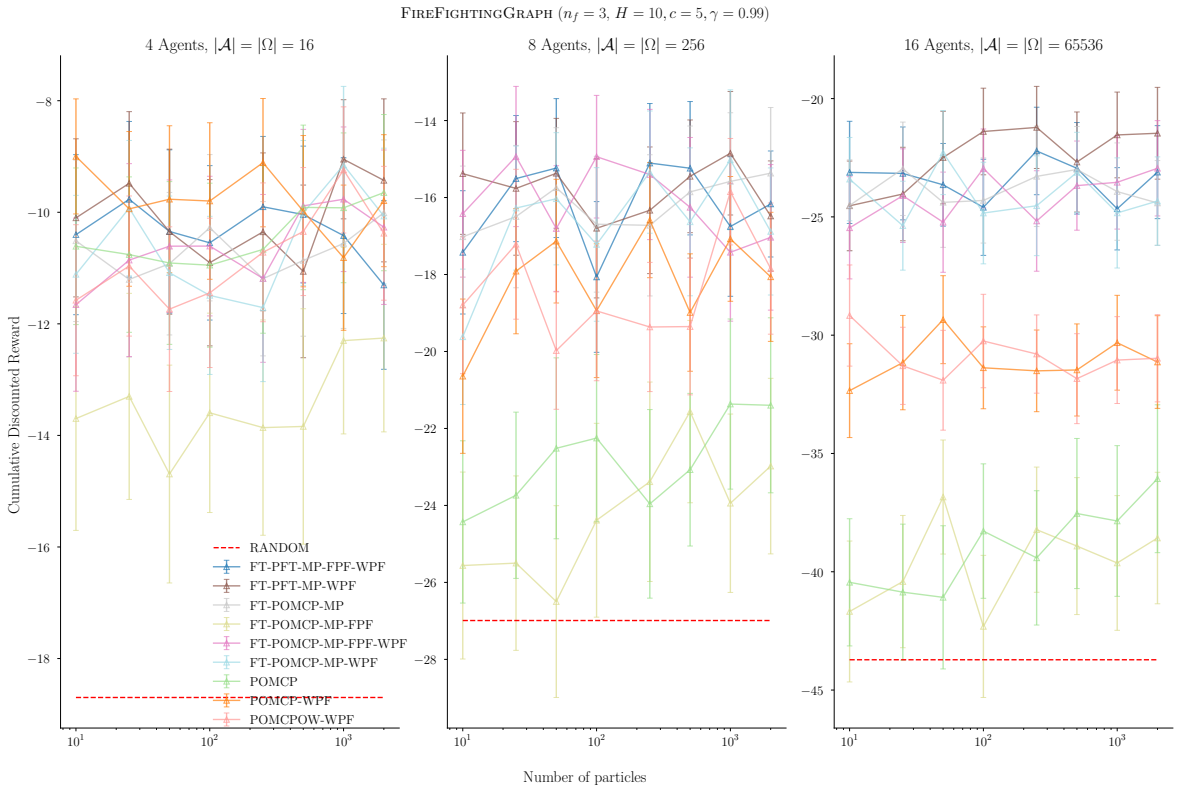


Figure 7.5: Algorithm performance on FireFightingGraph($H = 10, c = 5, \gamma = 0.99, n_f = 3$) with various numbers of particles. This experiment is similar to Fig. 7.6a but differs in the fact that the logarithmic x-axis shows a varying number of particles in the particle filters maintained during the episode. All experiments were constrained by a maximum of either 250 simulations or **five** seconds of computational budget per search.

(c) Large map: MARS(15, 15), $\gamma = 0.95, c = 1.25$

Environment	MARS(15, 15)			
Nr. of Agents	3	4	5	6
FS-PFT-VE	-7.1 ± 1.6	-3.3 ± 1.0	-0.6 ± 0.5	-7.6 ± 1.9
FS-POMCP-MP	0.2 ± 0.4	0.0 ± 0.0	0.4 ± 0.5	-0.2 ± 0.3
FS-POMCP-VE	-3.9 ± 1.3	0.8 ± 1.9	-2.5 ± 3.2	-4.4 ± 2.3
FS-POMCP-VE-WPF	-0.6 ± 0.6	$\mathbf{3.0} \pm 0.8$	2.2 ± 1.1	-3.9 ± 0.8
FS-POMCPOW-VE	1.9 ± 0.5	2.1 ± 1.3	-3.2 ± 1.1	-2.6 ± 1.8
FT-PFT-VE	-2.5 ± 1.1	$\mathbf{2.0} \pm 1.1$	-0.6 ± 0.6	-10.3 ± 1.6
FT-POMCP-MP	0.3 ± 0.9	-0.9 ± 1.2	1.0 ± 1.2	-0.9 ± 0.6
FT-POMCP-MP-WPF	0.8 ± 0.6	0.0 ± 0.0	0.6 ± 0.4	-0.9 ± 1.7
FT-POMCP-VE	2.3 ± 0.8	0.6 ± 0.8	1.1 ± 0.9	$\mathbf{3.6} \pm 0.8$
FT-POMCP-VE-WPF	-0.6 ± 0.3	1.1 ± 1.0	$\mathbf{5.6} \pm 1.1$	-1.7 ± 0.6
DESPOT	OOR	OOR	OOR	OOR
PFT	-2.5 ± 1.4	0.5 ± 0.7	-0.1 ± 0.1	OOR
POMCP	$\mathbf{9.1} \pm 1.5$	-0.6 ± 1.6	0.0 ± 0.0	OOR
POMCP-WPF	-14.9 ± 1.6	-0.0 ± 0.5	0.0 ± 0.0	OOR
POMCPOW	4.3 ± 1.3	-1.3 ± 1.2	-0.4 ± 0.5	OOR

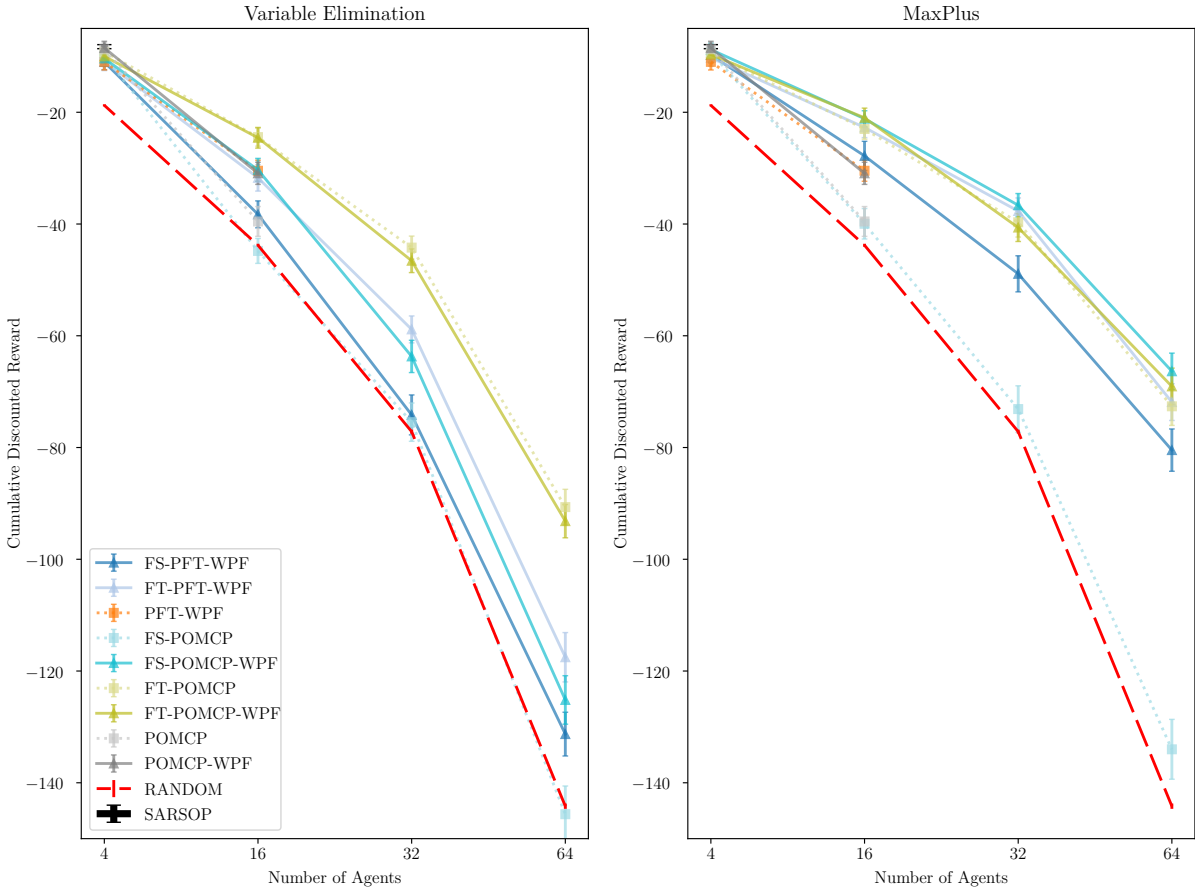
Table 7.3: MARS experiments with three maps and various numbers of agents. Maximum time per step is **five** seconds.

the flat algorithms to the FT algorithm variants, where the factored algorithms use the locality-based filtering approach (chapter 5). We run flat algorithms and factored trees in this comparison to demonstrate the difference in particle filtering methodology. We also consider the FPF method described in Sect. 5.1. Performance increases marginally for the flat filters, while the results of the factored filters do not exhibit such an increase. This increase is less distinguishable in the algorithms with weighted particle filters. Moreover, our locality-based filters achieve good performance with a small number of particles per filter. Some of the FPF variants perform equally well. However, this SampleJoin procedure of this method is much more computationally expensive than our locality-based filtering solution.

Nr. of agents	3	4	5	6
POMCP	0.08 ± 0.05	0.05 ± 0.04	0.08 ± 0.05	0.19 ± 0.08
POMCP-WPF	$\mathbf{0.28} \pm 0.09$	$\mathbf{0.62} \pm 0.10$	$\mathbf{0.69} \pm 0.09$	0.54 ± 0.10
FT-POMCP-MP	0.18 ± 0.08	$\mathbf{0.69} \pm 0.09$	0.18 ± 0.08	$\mathbf{0.78} \pm 0.08$
FT-POMCP-MP-WPF	0.06 ± 0.05	0.32 ± 0.09	0.05 ± 0.04	0.44 ± 0.10
FT-POMCP-VE	$\mathbf{0.26} \pm 0.09$	0.36 ± 0.09	0.33 ± 0.09	0.51 ± 0.10
FT-POMCP-VE-WPF	0.15 ± 0.07	0.41 ± 0.10	0.20 ± 0.08	0.55 ± 0.10

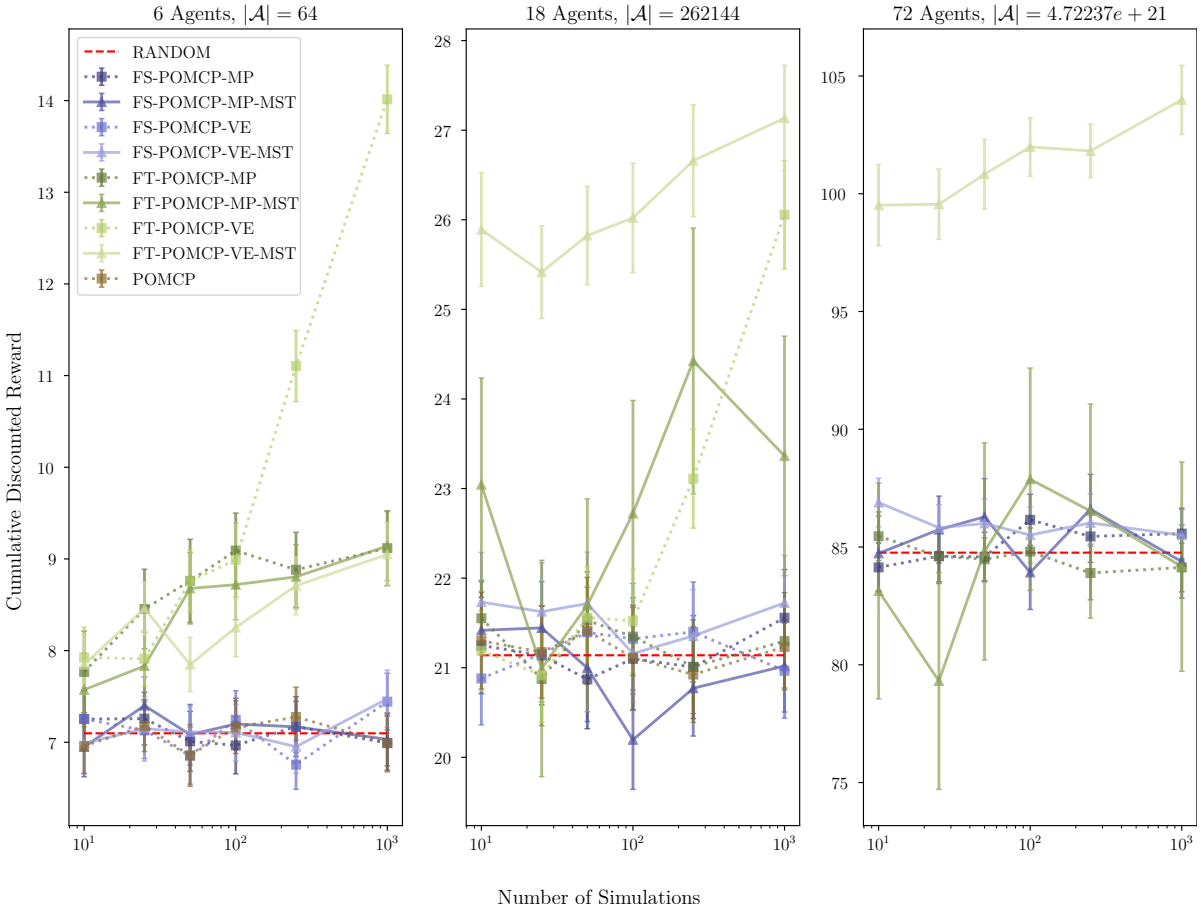
Table 7.4: CaptureTarget(12×12). Maximum time per step of **fifteen** seconds. We only show results above a certain threshold, full table is located in the appendix.

Q3: the action selection algorithm noticeably influences performance In CT (Table 7.4) and MARS (Table 7.3), we see when comparing VE and MP that there is a noticeable influence on the performance. This difference is also pronounced in the results for FFG (Fig. 7.6a), where MP outperforms the returns achieved by VE. Contrary to the benchmarks above, SysAdmin (Fig. 7.6b) has a cyclic topology (*ring-of-ring*, where subsets of agents in a ring of three are connected in a larger ring (Choudhury et al., 2022).) Both action selection algorithms are considered, and we distinguish MST (solid) and regular (dotted) variants. Here, MP especially does not work well, and the majority of simulation time is spent in the action selection procedures. The MST extension from Sect. 6.1.3 alleviates this issue, scaling our algorithm to a dense coordination structure with 72 agents. It is clear that when many agents are involved, the MST approximation is essential to achieving good returns.



(a) FireFightingGraph($H = 10, c = 5, \gamma = 0.99, n_f = 3$). We compare our algorithm variants (solid) to the state-of-the-art (dotted). For the four-agent setting, we ran the full-width planner SARSOP (Kurniawati et al., 2008) for 30 minutes before running the same episodic evaluation. The flat variants of POMCP and PFT were not able to run settings with 32 or more agents due to memory issues.

Q4: expensive simulations can prohibit PFT's success. PFT is very competitive in FFG (Fig. 7.6a) and MARS with an increased search duration (Table 7.5), but achieves mediocre results in MARS with five seconds per step (Table 7.3). In CT (Table 7.4), the PFT algorithm variants do not work well, as the simulation calls for this environment are expensive. Thus, the algorithm does not achieve the required number of



(b) SysAdmin($H = 50, c = 5, \gamma = 0.95$). We compare algorithms with the MST extension (solid lines) and without (dotted lines). As SysAdmin is an MMDP, we do not consider weighted filters due to the observation model requirement. POMCP was not able to run on the 72-agent setting due to memory issues.

Figure 7.6: The x-axes are logarithmic and show the permitted number of simulations per search. The time-out for the search call at each step was five seconds in both experiments. Flat algorithms are not run for setting with more than 60 agents because of memory issues. The error bars indicate 95% confidence intervals.

iterations to reach a sufficient search depth within the time limit. Additionally, it is not clear how to run PFT variants on SysAdmin in its MMDP format. From our results, we cannot claim that PFT variants outperform POMCP variants. In order to further evaluate these considerations, we run MARS with a larger computational budget in the subsection below.

7.3.3 Additional experiments

This section presents additional results of the empirical evaluation.

Increased search duration for MARS. In addition to the results in Table 7.3, we ran a full experiment on three maps with 15 seconds of maximum searching time before executing a step in the true environment. All environments had a horizon, i.e., maximum episode duration, of 40. The results for the small map are visible in Table 7.5a. This is the same map as in Table 7.3a, albeit the algorithms ran with a larger computational budget. The *flat* algorithms perform best on the settings with fewer agents but either go out of resources or do not achieve good results on the setting with 5 and 6 agents. A noticeable exception is regular POMCP with weighted particle filtering (POMCP-WPF), which achieves good results with 15 seconds of searching even when there are $n = 5$ agents and $k = 8$ rocks, which results in a large action space of $|\mathcal{A}| = 371293$. For the larger maps, in Table 7.5b and 7.5c, we see that the flat algorithms fail to perform well even though there is a reasonable computational budget. This can be explained by the fact that as the number of rocks k increases, the action space increases drastically due to the exponential factor of the number of agents. DESPOT achieves the best score in the medium map with 3 agents (Table 7.5b). POMCP and PFT variants both perform well in large maps with many agents. In particular, the factored statistics algorithms achieve good performance, especially in the largest map (Table 7.5c). Moreover, FS-PFT-VE-WPF is the only algorithm that achieves comparable performance to a flat counterpart in the version of Table 7.5b with 3 agents. In conclusion, there is no clear winner between the factored algorithms, but weighted particle filtering variants outperform unweighted particle filters. Additionally, from the difference in performance between Table 7.3 and 7.5, we can conclude that the PFT variants perform much better with a larger computational budget of 15 seconds as compared to 5 seconds.

(a) Small map: MARS(7, 8), $\gamma = 0.95, c = 1.25$

Environment	MARS(7, 8)			
Nr. of Agents	3	4	5	6
FS-PFT-VE	4.1 ± 1.7	2.0 ± 2.0	-3.4 ± 1.7	-6.4 ± 0.0
FS-POMCP-MP-WPF	-1.1 ± 1.1	1.6 ± 1.0	-2.9 ± 1.5	-2.3 ± 2.1
FS-POMCP-VE	9.9 ± 2.1	7.9 ± 2.6	-1.1 ± 2.5	0.0 ± 2.8
FS-POMCP-VE-WPF	3.4 ± 1.1	4.9 ± 1.3	0.3 ± 1.4	$\mathbf{6.6} \pm 1.5$
FS-POMCPOW-MP	-0.9 ± 0.8	3.6 ± 1.0	3.0 ± 2.8	3.2 ± 1.3
FS-POMCPOW-VE	1.0 ± 1.3	-2.3 ± 1.6	-1.5 ± 0.6	$\mathbf{6.2} \pm 0.5$
FT-PFT-MP	-6.2 ± 1.9	-7.2 ± 3.1	-6.4 ± 3.4	-4.7 ± 3.2
FT-POMCP-MP	2.4 ± 1.4	1.4 ± 1.7	4.7 ± 1.8	-1.3 ± 3.4
FT-POMCP-MP-WPF	1.9 ± 1.1	2.2 ± 0.8	1.9 ± 1.0	1.1 ± 1.2
FT-POMCP-VE	5.1 ± 1.0	4.6 ± 1.4	$\mathbf{7.0} \pm 2.5$	2.9 ± 1.1
FT-POMCP-VE-WPF	2.9 ± 1.2	4.3 ± 1.2	0.6 ± 1.6	2.4 ± 1.3
DESPOT	OOR	$20.8^{\dagger} \pm 0.3$	OOR	OOR
PFT	2.1 ± 1.9	-10.6 ± 2.3	-2.3 ± 1.9	-0.7 ± 0.8
POMCP	20.0 ± 2.1	18.3 ± 4.3	-0.8 ± 2.1	0.0 ± 0.0
POMCP-WPF	11.1 ± 2.4	$\mathbf{26.6} \pm 2.4$	$\mathbf{8.5} \pm 1.5$	0.0 ± 0.0
POMCPOW	$\mathbf{32.6} \pm 2.1$	12.3 ± 1.8	-4.9 ± 1.6	OOR

†: exceeded the time bound of 15 seconds per step.

Table 7.5: MARS experiments with three maps and various numbers of agents. Maximum time per step is **fifteen** seconds.

(b) Medium map: MARS(11, 11), $\gamma = 0.95, c = 1.25$

Environment	MARS(11, 11)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	4.3 ± 1.3	-4.1 ± 1.3	-4.6 ± 1.7	-2.6 ± 4.1
FS-PFT-VE	15.9 ± 1.4	-5.9 ± 1.3	-2.0 ± 1.9	$\mathbf{7.2} \pm 1.3$
FS-POMCP-MP-WPF	-1.4 ± 0.7	0.2 ± 0.4	2.1 ± 1.2	0.9 ± 1.7
FS-POMCP-VE	5.7 ± 2.1	1.5 ± 2.0	-6.4 ± 2.7	-2.8 ± 5.5
FS-POMCP-VE-WPF	3.4 ± 1.0	1.0 ± 0.8	-0.7 ± 1.1	-1.4 ± 1.3
FS-POMCPOW-MP	2.0 ± 1.0	-1.6 ± 0.7	-2.3 ± 1.0	-1.6 ± 2.4
FS-POMCPOW-VE	-2.1 ± 0.8	1.8 ± 1.2	$\mathbf{2.9} \pm 1.5$	-5.0 ± 1.5
FT-PFT-MP	1.9 ± 0.8	0.1 ± 1.4	-4.5 ± 2.3	-5.6 ± 2.5
FT-PFT-VE	3.1 ± 1.0	-7.8 ± 1.1	-9.5 ± 1.4	-5.4 ± 1.4
FT-POMCP-MP-WPF	-1.1 ± 1.7	0.0 ± 0.0	1.1 ± 0.6	-0.7 ± 1.7
FT-POMCP-VE	4.0 ± 1.1	$\mathbf{5.5} \pm 1.4$	$\mathbf{3.4} \pm 1.8$	1.5 ± 0.9
FT-POMCP-VE-WPF	5.9 ± 1.0	-2.3 ± 1.0	$\mathbf{3.1} \pm 1.1$	-2.1 ± 0.9
DESPOT	$\mathbf{25.1} \pm 0.7$	OOR	OOR	OOR
PFT	-9.1 ± 2.0	-6.8 ± 2.5	-0.2 ± 0.4	OOR
POMCP	18.0 ± 2.4	-6.4 ± 1.9	-6.4 ± 4.2	OOR
POMCP-WPF	6.9 ± 1.0	-9.3 ± 2.1	-4.5 ± 1.3	OOR
POMCPOW	14.2 ± 1.1	-4.1 ± 1.6	-1.1 ± 0.6	OOR

(c) Large map: MARS(15, 15), $\gamma = 0.95, c = 1.25$

Environment	MARS(15, 15)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	2.3 ± 0.8	-2.4 ± 2.8	-3.0 ± 1.8	-2.6 ± 3.7
FS-PFT-VE	-5.8 ± 1.3	-5.2 ± 1.2	$\mathbf{4.2} \pm 0.9$	2.6 ± 0.6
FS-POMCP-VE-WPF	-2.9 ± 0.8	0.5 ± 0.5	0.1 ± 0.8	$\mathbf{6.9} \pm 1.1$
FS-POMCPOW-MP	1.3 ± 2.2	-0.5 ± 1.6	-0.3 ± 2.0	-1.7 ± 1.8
FS-POMCPOW-VE	-2.9 ± 0.9	$\mathbf{4.9} \pm 0.8$	$\mathbf{3.3} \pm 1.1$	0.4 ± 1.4
FT-PFT-VE	-3.8 ± 0.9	-3.6 ± 1.0	1.9 ± 0.6	3.0 ± 0.9
FT-POMCP-MP	-1.0 ± 0.6	-0.3 ± 0.4	0.1 ± 0.1	-0.6 ± 1.1
FT-POMCP-MP-WPF	-0.1 ± 0.6	0.1 ± 0.2	0.2 ± 0.2	0.4 ± 0.3
FT-POMCP-VE	-1.3 ± 0.6	0.4 ± 0.6	-0.9 ± 0.6	3.4 ± 0.8
FT-POMCP-VE-WPF	1.7 ± 0.6	$\mathbf{3.6} \pm 0.7$	-0.2 ± 0.4	-1.5 ± 0.6
DESPOT	N.A.	N.A.	N.A.	N.A.
PFT	-1.7 ± 1.2	0.2 ± 0.7	-0.2 ± 0.2	OOR
POMCP	4.1 ± 2.0	0.1 ± 1.9	0.0 ± 0.0	OOR
POMCP-WPF	$\mathbf{8.4} \pm 1.3$	-1.5 ± 1.1	0.0 ± 0.0	OOR
POMCPOW	-13.5 ± 1.5	0.2 ± 1.1	-0.1 ± 0.5	OOR

Table 7.5: MARS experiments with three maps and various numbers of agents. Maximum time per step is **fifteen** seconds.

Chapter 8

Contributions, Related Work & Discussion

In this thesis, we introduced several extensions to the state-of-the-art in online planning for Multi-Agent POMDPs. Specifically, we outline the following contributions:

- By mapping an MPOMDP to a particle-belief MMDP in chapter 4, we were set to introduce a new version of this online planning algorithm which builds trees that are insensitive to the size of the observation space (Sect. 4.3.2). Additionally, we introduced the notion of progressive widening to limit the growth of the breadth of the search tree progressively in MPOMDPs (Sect. 4.2). In doing so, we address the problem of the factored statistics algorithm variant of POMCP in large observation spaces.
- In chapter 5, we introduced two ways to combat the possibility of a deprived state of a particle filter in domains with many agents. Both of these methods introduce an assumption on the model by either considering a local observation function or a factored state space. In our experiments, the locality-based filter proves the most efficient out of the two methods.
- We outlined and compared two of the ways one can select actions in accordance with the UCB1 algorithm in chapter 6. Furthermore, we developed an extension that enables these algorithms to scale to problems with many agents by extracting a tree from the graph based on the current estimated local values.
- In chapter 7, we ran an extensive experimental evaluation to assess the performance of these algorithmic approaches, comparing numerous combinations of algorithm variants.

8.1 Related Work

In our overview of the previous work related to this thesis, we consider three associated fields and discuss these below.

MPOMDP planning. Our main point of reference for this thesis is the work by Amato and Oliehoek (2015). Our work can be seen as an extension of theirs. To the best of our knowledge, it is one of the very few papers published on online planning in MPOMDPs.

Further work on multi-agent online planning considered action selection. Originally, Amato and Oliehoek (2015) only considered an exact method to compute maximal actions. Pfrommer (2016) introduced graphical POMCP with a message passing algorithm but only considered CGs that are trees with local reward functions. Choudhury et al. (2022) introduce an anytime selection algorithm for online planning. However, they only consider MCTS in the fully observable setting with factored statistics. Because they assume full state observability, they can support dynamic coordination graphs that partition the problem based on the current state. In partially observable settings, achieving such support for dynamic coordination within the tree search is much less trivial. However, a straightforward solution comes to mind. The algorithm could determine the coordination structure before every search iteration by considering the graph structure as part of the state space and, thus, in the particle filter. It would proceed in a consecutive fashion, determining a — possibly new — coordination structure before each call. In this case, the search tree from previous calls cannot be re-used out of the box.

In the literature, MPOMDPs are often studied either in multi-agent communicative decision problems (Pynadath and Tambe, 2002) or as a more tractable representation of Dec-POMDPs by introducing a form of communication (Spaan et al., 2008; Messias et al., 2011, 2013; Oliehoek and Amato, 2016). Messias et al. (2013) consider MPOMDPs with asynchronous execution. Therefore, the assumption that every agent needs to pick an action simultaneously is removed. They show that this event-driven model is more scalable than traditional synchronous methods. Apart from Amato and Oliehoek (2015), these studies often consider finding optimal policies *offline*. Cai et al. (2021) consider MPOMDPs implicitly by their extension of DESPOT to large action spaces and experiment on MARS with two agents, but do not consider the tractability problems of planning in MPOMDPs with many agents.

Online planning in large domains. As solutions for MPOMDPs can be found by representing the problem as a POMDP, we also compare to the relevant online planning literature of single-agent decision-making in a partially observable environment. In particular, we focus on algorithms that are designed to find good solutions to problems with large action and observation spaces. DESPOT (Ye et al., 2017) is an online planner that depends on a deterministic simulator to create a sparse belief tree from a set of sampled belief trajectories. However, it suffers from a large action space in particular. DESPOT-α (Garg et al., 2019) and HyP-DESPOT (Cai et al., 2021) are extensions that are aimed at addressing DESPOT's scalability issues. The variants employ alpha-vectors to fuse similar paths in the tree or (GPU) parallelism that relies on a factored simulator.

AdaOPS (Wu et al., 2021) and LABECOP (Hörger and Kurniawati, 2021) are online planners that show good results in continuous domains but do not necessarily outperform the state-of-the-art on discrete problems. AdaOPS achieves good performance by using adaptive particle filtering and fusing similar observation branches. POMCPOW (Sunberg and Kochenderfer, 2018), which we extend in Sect. 4.2.1, utilises double progressive widening and an explicit observation model with weighted particle filtering to combat large (continuous) state and observation spaces.

Learning Q-functions in multi-agent decision problems. Boehmer et al. (2020) employ a coordination graph in a multi-agent reinforcement learning setting. Similar techniques in the RL setting have been considered by others (Guestrin et al., 2002a; Kuyer et al., 2008; Van der Pol and Oliehoek, 2016; Li et al., 2021). These approaches

are characterised in that they learn neural pay-off functions, grouping network outputs by the graph structure. Van der Pol and Oliehoek (2016) and Boehmer et al. (2020) use a message-passing scheme in the graph-based neural network that is based on the MP algorithm. In essence, our approaches also learn MLEs of the component Q-values but online, i.e., without function approximation, which would require off-line training. Those works also differ in the state representation, some assume fully observable states (Van der Pol and Oliehoek, 2016) c.q. represent the belief approximation by a latent vector in a recurrent neural network (Boehmer et al., 2020). The experimental comparison to Böhmer et al. (2020) was not made in this thesis but is certainly interesting and relates to one of our ideas for future work. A multitude of other work considers learning neural network value and policy functions in multi-agent systems with decentralisation, i.e., Dec-POMDPs (Rashid et al., 2018; Xiao et al., 2021). Therein, training is often centralised, i.e., with communication, while guaranteeing that in deployment, the policy can be executed in a decentralised fashion (Oliehoek and Amato, 2016; Gronauer and Diepold, 2022).

8.2 Discussion

First, in this section, we discuss the position of our contributions with respect to the related work described previously. Secondly, we discuss the considerations and effects of the assumptions our methods entail.

Contributions. The contributions of this thesis were outlined in detail previously. Our approach is an algorithmic contribution to scale online planning in MPOMDPs to environments with many agents. It thus lies in the intersection of the fields of online and MPOMDP planning. Relatively few papers are published in this domain. We built on the initial work of Amato and Oliehoek (2015). On the one side, we introduced a partially novel set of algorithms that take a similar approach to tackling the problems that arise in the combinatorial explosion of the solution space when many agents are involved. On the other side, we enhanced the state-of-the-art by further increasing the use of structure to scale to settings with many agents. Therein, our approach tackles the components that fail in these situations, such as the belief estimator, i.e., particle filters, and action selection algorithms. Our proposed methods aim to solve these problems independently of the encompassing online planning algorithm. In doing so, our methods aim to be applicable to a multitude of online planning algorithms, such as the two main base algorithms, POMCPs and PFT contained in this thesis. Finally, we do not think our methods are necessarily improvements in settings with few agents where existing algorithms, e.g., DESPOT or POMCPOW, can still find suitable solutions.

Assumptions. Theorem 4.1.1 considers the locality of the factored Q-functions as a requirement for the algorithm to perform equally well to regular POMCP in the limit. It concerns the fact that the components of the CG must exhibit strict locality for the method to perform as expected, thereby limiting the introduced bias. If the value is not sufficiently structured, the representational capacity of the search tree is limited and prone to underfitting. In natural problems, there might always be interdependence or confounding that, as time evolves, in the long run, influence far-away components. This can even occur if the problem seems sufficiently decoupled (Oliehoek et al., 2021).

In practice, this assumption is not a direct requirement since the algorithms can still perform better than traditional methods without these assumptions.[1] Castellini et al. (2021) report similar findings in their experimental study on the factorisation of value in stateless settings.

Chapter 5 considers two methods to improve the efficiency of belief approximation. Both of these methods introduce an assumption.

Factored particle filtering relies on the assumption that the state can be factored into a vector of state variables. Furthermore, it requires that we can identify clusters of state variables of sufficient size, ideally with respect to the structure given by the CG. In practice, this method does not scale as well as desired, as the sampling procedure is prone to failure when there is sufficient overlap between the variables in the clusters. In order for the technique to work well, further assumptions are required, such that there exists a minimal inter-dependency between state variables. In our experiments, locality-based filtering performs much more reliably.

Locality-based filtering assumes that there exists an individual observation function that is independent given the joint action and state. Furthermore, because we lack a formal guarantee on the estimator performance of the ensemble, the method relies on whether the combination of target distributions based on a subset of observations can function as a good proxy for the posterior distribution. In practice, this seems to perform well. The assumption of the individual observation probability is arguably realistic, as the joint probability often relies on the probability of the observation of each agent. However, there likely are settings in which this does not hold. If the joint observation cannot be factored into individual observations, this would not be a realistic assumption. Furthermore, if the joint belief can only be reconstructed from the information distributed to *all* agents, our method, considering only local information, would fail to give a good approximation.

[1] For example, in the limit, POMCP converges to an ϵ-optimal result, but in practice, for large MPOMDPs, this is typically infeasible given the exponential size of the solution space.

Chapter 9

Future Work & Conclusion

In the final chapter of this thesis, we sketch new ideas for future research and give a conclusion to this work.

9.1 Future Work

Optimal decision-making in stochastic environments with partial observability is non-trivial. In the case that these systems consist of many agents, this problem becomes even harder. Although this thesis might be a step forward, we enumerate possible avenues for further extension here.

Factored beliefs. Messias et al. (2011) consider factored Dec-POMDPs with free communication, resulting in a factored MPOMDP. They show how the belief can be factored according to subsets of the state space. However, these beliefs are still conditioned on the joint history. In line with Djuric and Bugallo (2013), one might consider representing a set of independent beliefs over sets of the factored state-space for which the intersection is the empty set. One could sample from these sets to construct a full-state particle, propagate this through the joint dynamics function (simulator), project the full particle to a local particle, and update the weight according to our method in chapter 5. Although it is unsure how this would affect the quality of the belief approximation, it might require updating even fewer particles, thereby improving the computational complexity of the filtering procedure.

Decentralisation. Amato and Oliehoek (2015) rightly note that the FT algorithm variants might be able to reduce the communication required during planning. The MP action selection algorithm can also be implemented in a distributed manner. Cai et al. (2021) use a factorised simulator to enable parallelism during the search. Combining these ideas, one could create a concurrent approach to running the FT algorithm variants, where subsets of agents are grouped together as defined by a CG, and steps in the environment are set independently between the groups. Centralised action selection is then executed by decentralised message passing (Rogers et al., 2011). However, it is unclear how synchronisation should be handled in the case of a factored simulator and how this strict decoupling would affect planning performance.

KLD-sampling. AdaOPS (Wu et al., 2021) uses a Kullback-Leibner divergence (KLD) based sampling scheme by Fox (2001) to determine the number of particles propagated dynamically. They partition the state into buckets and let the sampler determine the required number of particles to ensure some predefined approximation error with a given confidence level. In our adopted version of the particle filter tree algorithm by Lim et al. (2023), the statically set number of particles C has a large influence on both the time complexity and performance of the algorithm. We are interested to see if determining C dynamically with a KLD sampler improves performance or can reduce the computational complexity.

Neural network policy improvement. The recent success of combining online planning algorithms with neural function approximation by Silver et al. (2017) (concurrently by Anthony et al. (2017)), resulting in a model-based reinforcement learning algorithm, is also applicable to our setting. One could combine the algorithmic approaches of this thesis with neural network predictions in an iterative learning loop, using the structure of the problem to reduce the scope of the output predictions (Van der Pol and Oliehoek, 2016; Boehmer et al., 2020). This setup could enjoy parameter sharing and increased sample efficiency. We expect this combination to work well and aim to investigate this further.

Continuous domains. Finally, we note that inspiration for some of the algorithms, specifically FS-POMCPOW and PFT variants, introduced in this thesis (chapter 4, mainly) partly originates from the recent developments in developing online algorithms for POMDPs with continuous spaces (Sunberg and Kochenderfer, 2018; Fischer and Tas, 2020b; Couëtoux et al., 2011; Lim et al., 2023, 2020). This does not mean that these algorithms will already perform well in continuous domains, as the factorisation of a continuous space still leads to an infinite number of possibilities, i.e., actions and observations, in the sub-domains. Therefore, an enumeration of a factored action or observation space remains infeasible. However, our weighted particle filter can support continuous state, action, and observation spaces, and both FS-POMCPOW and PFT consist of techniques that can handle such infinitely large observation spaces by progressive observation widening and insensitivity to observation space size, respectively. Currently, these algorithms lack a sophisticated approach to handle continuous actions. However, they could be extended to handle such domains by adapting existing techniques. For example, by using a Bayesian optimisation method (Mern et al., 2021) or a Voronoi partition of the action space in the form of a Voronoi optimistic optimisation approach (Kim et al., 2020; Lim et al., 2021).

We hypothesise that, if slightly adapted for these continuous spaces, these algorithms can also perform reasonably well in centralised multi-agent domains with continuous spaces.

9.2 Conclusion

Finding optimal policies for partially observable stochastic problems with solution spaces that grow exponentially, such as those modelled by MPOMDPs, is generally difficult. In this thesis, we have introduced several extensions to the state-of-the-art that allow us to scale to problems with *many* agents by exploiting the structure that these systems exhibit. In particular, we proposed solutions based on this structure to resolve the key issues that start appearing when the number of agents grows large.

A drawback to our work is the lack of theoretical guarantees these improvements come with. However, we conclude that within the context of our experimental evaluation, our algorithms perform well. Specifically, they show competitive performance in settings with few agents and outperform the state-of-the-art in settings with many agents. Furthermore, by our experimental evaluation, we find that an artificial structure in the form of a coordination graph can act as a heuristic. This heuristic decomposition can scale our online planning algorithms to problems that do not exhibit natural decomposition. This finding, in which decomposing the value of problems that do not necessarily factor can still allow for good estimates, aligns with the work of Castellini et al. (2021). Therefore, factorisation can make online planning in a general class of MPOMDPs with many agents feasible. However, the introduction of such a heuristic can come at a price, as planners that consider the full representation of the problem generally achieve better performance as long as the solution space remains of tractable size.

Glossary

FS-PFT factored-statistics (FS) PFT. 33

FS-POMCP factored-statistics (FS) POMCP. 29

FT-PFT factored-trees (FT) PFT. 33

FT-POMCP factored-trees (FT) POMCP. 30

CT CaptureTarget. 53, 55, 57, 62, 84

FFG FireFightingGraph. 51, 52, 55, 57, 58, 62, 84

MARS multi-agent RockSample. 52, 53, 55, 57, 62, 64, 68, 84

belief-MDP belief-state Markov decision process. 11, 18, 33

CG coordination graph. 27, 29, 36, 40, 46–49, 51, 56, 57, 68, 69, 71

Dec-MDP decentralised Markov decision process. 14

Dec-POMDP decentralised partially observable Markov decision process. 14, 68, 69, 71

ESS effective sample size. 24, 25

FPF factored particle filtering. 39, 56, 61

FS factored-statistics. 29, 33, 56, 57, 74

FT factored-trees. 30, 32, 33, 56, 57, 71, 74

MAP maximum a posteriori. 47

MCTS Monte Carlo tree search. 16, 18, 19, 68

MDP Markov decision process. 3, 7–14, 33

MIS multiple importance sampling. 43

MLE maximum likelihood estimate. 17, 30, 35, 69

MMDP multi-agent Markov decision process. 12, 13, 50–52, 67

MoE mixture of experts. 28, 29, 45, 57

MP max-plus. 47–49, 56, 57, 69, 71

MPOMDP multi-agent partially observable Markov decision process. 3–5, 13, 14, 17, 21, 29, 33, 34, 38, 39, 41, 52, 57, 67–71, 73

MST maximum spanning tree. 48, 49, 56

PDF probability density function. 22

PF unweighted particle filter. 20

PFT sparse particle filter tree. 33, 34, 41, 53, 55–57, 62, 64, 69, 74

POMCP partially observable Monte Carlo planning. 18, 29, 30, 41, 49, 52, 53, 57, 58, 64, 69, 70, 74

POMDP partially observable Markov decision process. 3, 10–13, 20, 21, 29, 33, 34, 67, 68, 72

RL reinforcement learning. 68

SIR sequential importance re-sampling. 25, 26, 56

SIS sequential importance sampling. 23, 25

UCB1 upper confidence bound algorithm 1. 17, 29, 30, 35, 36, 45, 49, 55, 56, 67

UCT upper confidence trees. 17, 18, 35, 49

VE variable elimination. 45–49, 56

WPF weighted particle filter. 21, 26, 56, 64

Bibliography

Ahmadi, M., Jansen, N., Wu, B., and Topcu, U. (2021). Control theory meets pomdps: A hybrid systems approach. *IEEE Trans. Autom. Control.*, 66(11):5191–5204.

Amato, C. and Oliehoek, F. A. (2015). Scalable planning and learning for multiagent pomdps. In Bonet, B. and Koenig, S., editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 1995–2002. AAAI Press.

Anthony, T., Tian, Z., and Barber, D. (2017). Thinking fast and slow with deep learning and tree search. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5360–5370.

Arcieri, G., Hoelzl, C., Schwery, O., Straub, D., Papakonstantinou, K. G., and Chatzi, E. N. (2022). Bridging pomdps and bayesian decision making for robust maintenance planning under model uncertainty: An application to railway systems. *CoRR*, abs/2212.07933.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256.

Badings, T. S., Simão, T. D., Suilen, M., and Jansen, N. (2023). Decision-making under uncertainty: Beyond probabilities. *CoRR*, abs/2303.05848.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Boehmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 980–991. PMLR.

Bradtke, S. J. and Duff, M. O. (1994). Reinforcement learning methods for continuous-time markov decision problems. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, pages 393–400. MIT Press.

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

Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep Coordination Graphs. arXiv:1910.00091 [cs].

Cai, P., Luo, Y., Hsu, D., and Lee, W. S. (2021). Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *Int. J. Robotics Res.*, 40(2-3).

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In Hayes-Roth, B. and Korf, R. E., editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2*, pages 1023–1028. AAAI Press / The MIT Press.

Castellini, J., Oliehoek, F. A., Savani, R., and Whiteson, S. (2021). Analysing factorizations of action-value networks for cooperative multi-agent reinforcement learning. *Auton. Agents Multi Agent Syst.*, 35(2):25.

Chen, D., Yang-Zhao, S., Lloyd, J. W., and Ng, K. S. (2022). Factored conditional filtering: Tracking states and estimating parameters in high-dimensional spaces. *CoRR*, abs/2206.02178.

Choudhury, S., Gupta, J. K., Morales, P., and Kochenderfer, M. J. (2022). Scalable online planning for multi-agent MDPs. *J. Artif. Intell. Res.*, 73:821–846.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press.

Couëtoux, A., Hoock, J., Sokolovska, N., Teytaud, O., and Bonnard, N. (2011). Continuous upper confidence trees. In Coello, C. A. C., editor, *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pages 433–445. Springer.

Djuric, P. M. and Bugallo, M. F. (2013). Particle filtering for high-dimensional systems. In *5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, CAMSAP 2013, St. Martin, France, December 15-18, 2013*, pages 352–355. IEEE.

Elvira, V. and Martino, L. (2021). *Advances in Importance Sampling*, pages 1–14. John Wiley & Sons, Ltd.

Elvira, V., Martino, L., Luengo, D., and Bugallo, M. F. (2019). Generalized Multiple Importance Sampling. *Statistical Science*, 34(1):129 – 155.

Fairbank, M. and Alonso, E. (2012). The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012*, pages 1–8. IEEE.

Fischer, J. and Tas, Ö. S. (2020a). Information particle filter tree: An online algorithm for pomdps with belief-based rewards on continuous domains. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3177–3187. PMLR.

Fischer, J. and Tas, Ö. S. (2020b). Information particle filter tree: An online algorithm for POMDPs with belief-based rewards on continuous domains. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3177–3187. PMLR.

Fox, D. (2001). Kld-sampling: Adaptive particle filters. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 713–720. MIT Press.

Garg, N. P., Hsu, D., and Lee, W. S. (2019). Despot-alpha: Online POMDP planning with large state and observation spaces. In Bicchi, A., Kress-Gazit, H., and Hutchinson, S., editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140(2):107–113(6).

Gronauer, S. and Diepold, K. (2022). Multi-agent deep reinforcement learning: a survey. *Artif. Intell. Rev.*, 55(2):895–943.

Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored MDPs. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1523–1530. MIT Press.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002a). Coordinated reinforcement learning. In Sammut, C. and Hoffmann, A. G., editors, *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, pages 227–234. Morgan Kaufmann.

Guestrin, C., Venkataraman, S., and Koller, D. (2002b). Context-specific multiagent coordination and planning with factored mdps. In Dechter, R., Kearns, M. J., and Sutton, R. S., editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 253–259. AAAI Press / The MIT Press.

Hörger, M. and Kurniawati, H. (2021). An on-line POMDP solver for continuous observation spaces. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 7643–7649. IEEE.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134.

Katt, S., Oliehoek, F. A., and Amato, C. (2019). Bayesian reinforcement learning in factored pomdps. In Elkind, E., Veloso, M., Agmon, N., and Taylor, M. E., editors, *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 7–15. International Foundation for Autonomous Agents and Multiagent Systems.

Kearns, M. J., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Mach. Learn.*, 49(2-3):193–208.

Kennedy, T. (2016). *Monte Carlo Methods-a special topics course.*

Kim, B., Lee, K., Lim, S., Kaelbling, L. P., and Lozano-Pérez, T. (2020). Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9916–9924. AAAI Press.

Kochenderfer, M., Amato, C., Chowdhary, G., How, J., and Reynolds, H. (2015). *Decision Making Under Uncertainty: Theory and Application.* MIT Lincoln Laboratory Series. MIT Press.

Kochenderfer, M. J. (2015). *Decision making under uncertainty: theory and application.* MIT press.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer.

Kok, J. R. and Vlassis, N. (2005). Using the max-plus algorithm for multiagent decision making in coordination graphs. In Bredenfeld, A., Jacoff, A., Noda, I., and Takahashi, Y., editors, *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 1–12. Springer.

Kok, J. R. and Vlassis, N. (2006a). Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *The Journal of Machine Learning Research*, 7:1789–1828.

Kok, J. R. and Vlassis, N. (2006b). Collaborative multiagent reinforcement learning by payoff propagation. *J. Mach. Learn. Res.*, 7:1789–1828.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In Brock, O., Trinkle, J., and Ramos, F., editors, *Robotics: Science and Systems IV, Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland, June 25-28, 2008.* The MIT Press.

Kuyer, L., Whiteson, S., Bakker, B., and Vlassis, N. (2008). Multiagent reinforcement learning for urban traffic control using coordination graphs. In Daelemans, W., Goethals, B., and Morik, K., editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, volume 5211 of *Lecture Notes in Computer Science*, pages 656–671. Springer.

Lai, T. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6(1):4–22.

Li, S., Gupta, J. K., Morales, P., Allen, R. E., and Kochenderfer, M. J. (2021). Deep implicit coordination graphs for multi-agent reinforcement learning. In Dignum, F., Lomuscio, A., Endriss, U., and Nowé, A., editors, *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 764–772. ACM.

Lim, M. H., Becker, T. J., Kochenderfer, M. J., Tomlin, C. J., and Sunberg, Z. N. (2023). Optimality guarantees for particle belief approximation of POMDPs.

Lim, M. H., Tomlin, C. J., and Sunberg, Z. N. (2020). Sparse tree search optimality guarantees in POMDPs with continuous observation spaces. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4135–4142. ijcai.org.

Lim, M. H., Tomlin, C. J., and Sunberg, Z. N. (2021). Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation pomdps. In *2021 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA, December 14-17, 2021*, pages 4493–4500. IEEE.

Loeliger, H. (2004). An introduction to factor graphs. *IEEE Signal Process. Mag.*, 21(1):28–41.

Luo, Y., Bai, H., Hsu, D., and Lee, W. S. (2019). Importance sampling for online planning under uncertainty. *Int. J. Robotics Res.*, 38(2-3).

Madani, O., Hanks, S., and Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In Hendler, J. and Subramanian, D., editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*, pages 541–548. AAAI Press / The MIT Press.

Madani, O., Hanks, S., and Condon, A. (2003). On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34.

Memarzadeh, M. and Boettiger, C. (2018). Adaptive management of ecological systems under partial observability. *Biological Conservation*, 224:9–15.

Mern, J., Yildiz, A., Sunberg, Z., Mukerji, T., and Kochenderfer, M. J. (2021). Bayesian optimized monte carlo planning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11880–11887. AAAI Press.

Mertens, J. F. and Neyman, A. (1981). Stochastic games. *International Journal of Game Theory*, 10:53–66.

Messias, J. V., Spaan, M. T. J., and Lima, P. U. (2011). Efficient offline communication policies for factored multiagent POMDPs. In *NIPS*, pages 1917–1925.

Messias, J. V., Spaan, M. T. J., and Lima, P. U. (2013). Multiagent pomdps with asynchronous execution. In Gini, M. L., Shehory, O., Ito, T., and Jonker, C. M., editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1273–1274. IFAAMAS.

Miehling, E., Rasouli, M., and Teneketzis, D. (2018). A POMDP approach to the dynamic defense of large-scale cyber networks. *IEEE Trans. Inf. Forensics Secur.*, 13(10):2490–2505.

Moerland, T. M., Broekens, J., Plaat, A., and Jonker, C. M. (2023). Model-based reinforcement learning: A survey. *Found. Trends Mach. Learn.*, 16(1):1–118.

Ng, B., Peshkin, L., and Pfeffer, A. (2002). Factored particles for scalable monitoring. In Darwiche, A. and Friedman, N., editors, *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, pages 370–377. Morgan Kaufmann.

Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer.

Oliehoek, F. A., Spaan, M. T. J., Whiteson, S., and Vlassis, N. (2008). Exploiting locality of interaction in factored dec-pomdps. In Padgham, L., Parkes, D. C., Müller, J. P., and Parsons, S., editors, *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 1*, pages 517–524. IFAAMAS.

Oliehoek, F. A., Witwicki, S. J., and Kaelbling, L. P. (2021). A sufficient statistic for influence in structured multiagent environments. *J. Artif. Intell. Res.*, 70:789–870.

Owen, A. B. (2013). *Monte Carlo theory, methods and examples (book draft)*.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Math. Oper. Res.*, 12(3):441–450.

Pearl, J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

Peng, X. B., Chang, M., Zhang, G., Abbeel, P., and Levine, S. (2019). MCP: Learning composable hierarchical control with multiplicative compositional policies. In *NeurIPS*, pages 3681–3692. Curran Associates, Inc.

Pfrommer, J. (2016). Graphical partially observable monte-carlo planning. *Learning, Inference and Control of Multi-Agent Systems*.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley.

Pynadath, D. V. and Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *J. Artif. Intell. Res.*, 16:389–423.

Qiu, Q. and Pedram, M. (1999). Dynamic power management based on continuous-time markov decision processes. In Irwin, M. J., editor, *Proceedings of the 36th Conference on Design Automation, New Orleans, LA, USA, June 21-25, 1999*, pages 555–561. ACM Press.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. (2018). QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR.

Rogers, A., Farinelli, A., Stranders, R., and Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artif. Intell.*, 175(2):730–759.

Sandino, J., Vanegas, F., Gonzalez, F., and Maire, F. (2020). Autonomous uav navigation for active perception of targets in uncertain and cluttered environments. In *2020 IEEE Aerospace Conference*, pages 1–12. IEEE.

Septier, F. and Peters, G. W. (2016). Langevin and hamiltonian based sequential MCMC for efficient bayesian filtering in high-dimensional spaces. *IEEE J. Sel. Top. Signal Process.*, 10(2):312–327.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large POMDPs. In *NIPS*, pages 2164–2172. Curran Associates, Inc.

Slivkins, A. (2019). Introduction to multi-armed bandits. *Found. Trends Mach. Learn.*, 12(1-2):1–286.

Smith, A. F. and Gelfand, A. E. (1992). Bayesian statistics without tears: a sampling–resampling perspective. *The American Statistician*, 46(2):84–88.

Smith, T. and Simmons, R. G. (2004). Heuristic search value iteration for POMDPs. In Chickering, D. M. and Halpern, J. Y., editors, *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, Banff, Canada, July 7-11, 2004*, pages 520–527. AUAI Press.

Spaan, M. T. (2012). Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer.

Spaan, M. T. J., Oliehoek, F. A., and Vlassis, N. (2008). Multiagent planning under uncertainty with stochastic communication delays. In Rintanen, J., Nebel, B., Beck, J. C., and Hansen, E. A., editors, *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pages 338–345. AAAI.

Sunberg, Z. N., Ho, C. J., and Kochenderfer, M. J. (2017). The value of inferring the internal state of traffic participants for autonomous freeway driving. In *ACC*, pages 3004–3010. IEEE.

Sunberg, Z. N. and Kochenderfer, M. J. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. In de Weerdt, M., Koenig, S., Röger, G., and Spaan, M. T. J., editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, pages 259–263. AAAI Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass.

Thrun, S. (1999). Monte carlo pomdps. *Advances in neural information processing systems*, 12.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press.

Ulbrich, S. and Maurer, M. (2013). Probabilistic online POMDP decision making for lane changes in fully automated driving. In *ITSC*, pages 2063–2067. IEEE.

Van Der Pol, E. (2016). Deep reinforcement learning for coordination in traffic light control. *Master's thesis, University of Amsterdam*.

Van der Pol, E. and 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)*, 8:21–38.

Vlassis, N., Elhorst, R., and Kok, J. R. (2004). Anytime algorithms for multiagent decision making using coordination graphs. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: The Hague, Netherlands, 10-13 October 2004*, pages 953–957. IEEE.

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2004). Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Stat. Comput.*, 14(2):143–166.

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2005). MAP estimation via agreement on (hyper)trees: Message-passing and linear programming. *CoRR*, abs/cs/0508070.

Wiering, M. and Van Otterlo, M. (2012). *Reinforcement Learning: State of the Art*. Springer.

Wu, C., Yang, G., Zhang, Z., Yu, Y., Li, D., Liu, W., and Hao, J. (2021). Adaptive online packing-guided search for pomdps. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 28419–28430.

Xiao, Y., Lyu, X., and Amato, C. (2021). Local advantage actor-critic for robust multi-agent deep reinforcement learning. In *International Symposium on Multi-Robot and Multi-Agent Systems, MRS 2021, Cambridge, United Kingdom, November 4-5, 2021*, pages 155–163. IEEE.

Ye, N., Somani, A., Hsu, D., and Lee, W. S. (2017). DESPOT: online POMDP planning with regularization. *J. Artif. Intell. Res.*, 58:231–266.

Chapter 10

Experimental Evaluation

10.1 Full Results

For completeness, we report the full results of FFG (Fig. 10.1), CT (Table 10.1), and MARS (Table 10.2 and 10.3).

10.1.1 Firefighting

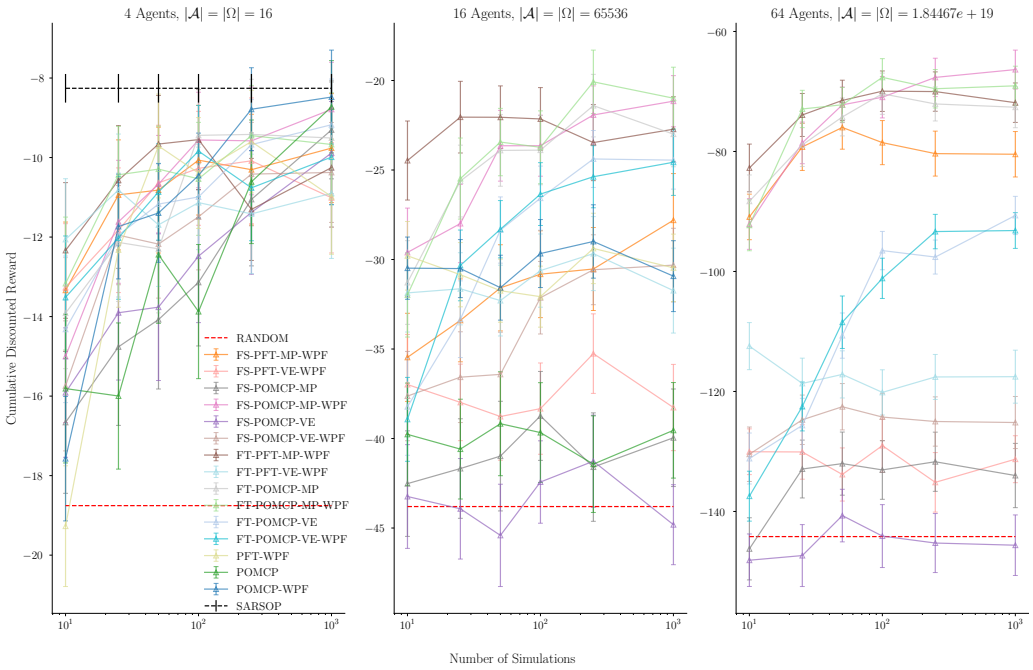


Figure 10.1: Extended results for FireFightingGraph($H = 10, c = 5, \gamma = 0.99, n_f = 3$). These results were compressed into Fig. 7.4 and 7.6a.

10.1.2 CaptureTarget

Nr. of agents	3	4	5	6
POMCP	0.08 ± 0.05	0.05 ± 0.04	0.08 ± 0.05	0.19 ± 0.08
POMCP-WPF	$\mathbf{0.28} \pm 0.09$	$\mathbf{0.62} \pm 0.10$	$\mathbf{0.69} \pm 0.09$	0.54 ± 0.10
FS-PFT-MP-WPF	0.01 ± 0.02	0.01 ± 0.02	0.04 ± 0.04	0.09 ± 0.06
FS-PFT-VE-WPF	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.03	0.02 ± 0.03
FS-POMCP-MP	0.00 ± 0.00	0.04 ± 0.04	0.03 ± 0.03	0.05 ± 0.04
FS-POMCP-MP-WPF	0.01 ± 0.02	0.03 ± 0.03	0.03 ± 0.03	0.06 ± 0.05
FS-POMCP-VE	0.01 ± 0.02	0.02 ± 0.03	0.02 ± 0.03	0.06 ± 0.05
FS-POMCP-VE-WPF	0.01 ± 0.02	0.02 ± 0.03	0.02 ± 0.03	0.02 ± 0.03
FT-PFT-MP-WPF	0.00 ± 0.00	0.02 ± 0.03	0.04 ± 0.04	0.02 ± 0.03
FT-PFT-VE-WPF	0.00 ± 0.00	0.00 ± 0.00	0.04 ± 0.04	0.02 ± 0.03
FT-POMCP-MP	0.18 ± 0.08	$\mathbf{0.69} \pm 0.09$	0.18 ± 0.08	$\mathbf{0.78} \pm 0.08$
FT-POMCP-MP-WPF	0.06 ± 0.05	0.32 ± 0.09	0.05 ± 0.04	0.44 ± 0.10
FT-POMCP-VE	$\mathbf{0.26} \pm 0.09$	0.36 ± 0.09	0.33 ± 0.09	0.51 ± 0.10
FT-POMCP-VE-WPF	0.15 ± 0.07	0.41 ± 0.10	0.20 ± 0.08	0.55 ± 0.10
PFT-WPF	0.00 ± 0.00	0.01 ± 0.02	0.02 ± 0.03	0.02 ± 0.03

Table 10.1: CaptureTarget(12×12). Maximum time per step of **fifteen** seconds. For completeness, we report full results from Table 7.4.

10.1.3 MARS

Small computational budget

Larger computational budget

(a) Small map: MARS($m = 7, k = 8$), $\gamma = 0.95, c = 1.25$

Environment	MARS(7, 8)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	-8.7 ± 2.4	-3.5 ± 2.5	-10.3 ± 4.6	-11.9 ± 4.8
FS-PFT-VE	3.9 ± 1.5	-3.9 ± 1.6	-6.9 ± 1.7	-9.4 ± 2.0
FS-POMCP-MP	-0.5 ± 2.1	-3.5 ± 2.4	-2.3 ± 3.4	-2.3 ± 2.1
FS-POMCP-MP-WPF	0.3 ± 1.2	0.9 ± 1.8	-1.2 ± 2.6	-1.8 ± 2.8
FS-POMCP-VE	0.7 ± 2.0	2.8 ± 3.0	$\mathbf{2.8} \pm 2.8$	-0.8 ± 3.8
FS-POMCP-VE-WPF	-1.0 ± 1.1	0.8 ± 1.3	$\mathbf{2.7} \pm 1.2$	2.1 ± 1.4
FS-POMCPOW-MP	-0.2 ± 1.2	0.5 ± 1.0	0.7 ± 2.3	0.0 ± 2.7
FS-POMCPOW-VE	1.2 ± 1.1	4.5 ± 1.6	$\mathbf{1.7} \pm 1.3$	$\mathbf{8.9} \pm 0.8$
FT-PFT-MP	-3.6 ± 2.1	-6.5 ± 3.1	-9.7 ± 1.9	-9.1 ± 4.5
FT-PFT-VE	0.4 ± 1.6	0.2 ± 1.4	1.0 ± 1.6	-11.5 ± 2.5
FT-POMCP-MP	2.5 ± 1.8	0.4 ± 1.1	$\mathbf{1.8} \pm 1.9$	-0.4 ± 3.6
FT-POMCP-MP-WPF	1.7 ± 1.1	1.7 ± 0.7	-0.1 ± 2.1	1.0 ± 0.8
FT-POMCP-VE	5.8 ± 1.1	3.0 ± 1.3	$\mathbf{2.7} \pm 1.2$	1.1 ± 1.2
FT-POMCP-VE-WPF	1.8 ± 1.1	2.9 ± 1.1	-1.3 ± 1.3	3.4 ± 1.5
DESPOT	OOR	OOR	OOR	OOR
PFT	-6.0 ± 2.5	-9.1 ± 2.8	-0.0 ± 0.7	OOR
POMCP	12.8 ± 2.8	7.5 ± 3.7	-4.6 ± 2.3	OOR
POMCP-WPF	$\mathbf{24.6} \pm 2.0$	6.4 ± 1.9	-1.9 ± 1.5	OOR
POMCPOW	14.7 ± 1.9	$\mathbf{11.1} \pm 1.8$	-0.2 ± 0.6	OOR

(b) Medium map: MARS(11, 11), $\gamma = 0.95, c = 1.25$

Environment	MARS(11, 11)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	-3.6 ± 1.2	-5.7 ± 1.5	-1.0 ± 3.1	-11.7 ± 5.9
FS-PFT-VE	-1.6 ± 1.3	-2.7 ± 1.9	-4.6 ± 2.7	-1.3 ± 1.6
FS-POMCP-MP	-0.5 ± 1.5	-4.1 ± 3.0	-5.1 ± 3.4	-10.7 ± 4.5
FS-POMCP-MP-WPF	0.4 ± 0.8	-0.2 ± 0.4	-1.7 ± 0.7	-0.3 ± 0.3
FS-POMCP-VE	1.8 ± 2.9	-1.0 ± 3.0	-7.3 ± 4.4	-6.8 ± 2.7
FS-POMCP-VE-WPF	-1.1 ± 0.6	1.5 ± 1.2	-3.7 ± 1.2	3.9 ± 1.0
FS-POMCPOW-MP	-0.3 ± 0.7	0.1 ± 0.3	0.3 ± 0.9	-1.5 ± 1.3
FS-POMCPOW-VE	2.2 ± 0.9	2.1 ± 1.0	-2.4 ± 1.2	-2.1 ± 1.2
FT-PFT-MP	1.0 ± 1.8	-4.8 ± 1.2	-4.7 ± 3.0	-7.4 ± 5.0
FT-PFT-VE	-5.3 ± 1.6	-8.9 ± 1.6	-1.9 ± 0.9	-0.8 ± 1.5
FT-POMCP-MP	0.0 ± 0.9	-0.0 ± 1.4	-0.0 ± 0.7	-1.2 ± 1.2
FT-POMCP-MP-WPF	0.5 ± 0.4	0.1 ± 0.2	0.0 ± 0.2	-0.1 ± 0.2
FT-POMCP-VE	1.5 ± 1.0	1.7 ± 1.3	$\mathbf{4.3} \pm 1.1$	4.3 ± 0.9
FT-POMCP-VE-WPF	5.2 ± 0.9	-3.3 ± 0.8	-3.6 ± 0.9	$\mathbf{6.8} \pm 1.0$
DESPOT	OOR	OOR	OOR	OOR
PFT	-17.7 ± 2.2	1.7 ± 1.6	-0.1 ± 0.5	OOR
POMCP	$\mathbf{17.0} \pm 1.9$	-3.6 ± 2.9	-0.1 ± 0.2	OOR
POMCP-WPF	$\mathbf{15.5} \pm 1.9$	$\mathbf{8.5} \pm 1.3$	0.0 ± 0.0	OOR
POMCPOW	2.4 ± 1.0	-0.8 ± 1.3	-0.0 ± 0.5	OOR

Table 10.2: MARS experiments with three maps and various numbers of agents. Maximum time per step is **five** seconds.

(c) Large map: MARS(15, 15), $\gamma = 0.95, c = 1.25$

Environment	MARS(15, 15)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	-5.8 ± 1.4	-1.4 ± 0.7	-3.9 ± 1.1	-2.1 ± 2.1
FS-PFT-VE	-7.1 ± 1.6	-3.3 ± 1.0	-0.6 ± 0.5	-7.6 ± 1.9
FS-POMCP-MP	0.9 ± 1.9	-1.7 ± 1.6	-3.1 ± 1.3	-7.8 ± 3.5
FS-POMCP-MP-WPF	0.2 ± 0.4	0.0 ± 0.0	0.4 ± 0.5	-0.2 ± 0.3
FS-POMCP-VE	-3.9 ± 1.3	0.8 ± 1.9	-2.5 ± 3.2	-4.4 ± 2.3
FS-POMCP-VE-WPF	-0.6 ± 0.6	$\mathbf{3.0} \pm 0.8$	2.2 ± 1.1	-3.9 ± 0.8
FS-POMCPOW-MP	-0.9 ± 0.4	-1.0 ± 1.6	-4.3 ± 1.2	0.3 ± 0.3
FS-POMCPOW-VE	1.9 ± 0.5	2.1 ± 1.3	-3.2 ± 1.1	-2.6 ± 1.8
FT-PFT-MP	-1.8 ± 0.7	-2.1 ± 1.7	-3.6 ± 1.0	-2.7 ± 1.7
FT-PFT-VE	-2.5 ± 1.1	$\mathbf{2.0} \pm 1.1$	-0.6 ± 0.6	-10.3 ± 1.6
FT-POMCP-MP	0.3 ± 0.9	-0.9 ± 1.2	1.0 ± 1.2	-0.9 ± 0.6
FT-POMCP-MP-WPF	0.8 ± 0.6	0.0 ± 0.0	0.6 ± 0.4	-0.9 ± 1.7
FT-POMCP-VE	2.3 ± 0.8	0.6 ± 0.8	1.1 ± 0.9	$\mathbf{3.6} \pm 0.8$
FT-POMCP-VE-WPF	-0.6 ± 0.3	1.1 ± 1.0	$\mathbf{5.6} \pm 1.1$	-1.7 ± 0.6
DESPOT	OOR	OOR	OOR	OOR
PFT	-2.5 ± 1.4	0.5 ± 0.7	-0.1 ± 0.1	OOR
POMCP	$\mathbf{9.1} \pm 1.5$	-0.6 ± 1.6	0.0 ± 0.0	OOR
POMCP-WPF	-14.9 ± 1.6	-0.0 ± 0.5	0.0 ± 0.0	OOR
POMCPOW	4.3 ± 1.3	-1.3 ± 1.2	-0.4 ± 0.5	OOR

Table 10.2: MARS experiments with three maps and various numbers of agents. Maximum time per step is **five** seconds.

(a) Small map: MARS($m = 7, k = 8$), $\gamma = 0.95, c = 1.25$

Environment	MARS(7, 8)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	-17.1 ± 2.5	-8.8 ± 2.9	-9.1 ± 2.3	-8.5 ± 4.6
FS-PFT-VE	4.1 ± 1.7	2.0 ± 2.0	-3.4 ± 1.7	-6.4 ± 0.0
FS-POMCP-MP	0.9 ± 3.0	1.3 ± 2.8	-5.3 ± 4.3	-7.1 ± 3.3
FS-POMCP-MP-WPF	-1.1 ± 1.1	1.6 ± 1.0	-2.9 ± 1.5	-2.3 ± 2.1
FS-POMCP-VE	9.9 ± 2.1	7.9 ± 2.6	-1.1 ± 2.5	0.0 ± 2.8
FS-POMCP-VE-WPF	3.4 ± 1.1	4.9 ± 1.3	0.3 ± 1.4	$\mathbf{6.6} \pm 1.5$
FS-POMCPOW-MP	-0.9 ± 0.8	3.6 ± 1.0	3.0 ± 2.8	3.2 ± 1.3
FS-POMCPOW-VE	1.0 ± 1.3	-2.3 ± 1.6	-1.5 ± 0.6	$\mathbf{6.2} \pm 0.5$
FT-PFT-MP	-6.2 ± 1.9	-7.2 ± 3.1	-6.4 ± 3.4	-4.7 ± 3.2
FT-PFT-VE	-9.4 ± 1.8	-6.3 ± 1.0	-14.2 ± 2.3	-2.8 ± 2.1
FT-POMCP-MP	2.4 ± 1.4	1.4 ± 1.7	4.7 ± 1.8	-1.3 ± 3.4
FT-POMCP-MP-WPF	1.9 ± 1.1	2.2 ± 0.8	1.9 ± 1.0	1.1 ± 1.2
FT-POMCP-VE	5.1 ± 1.0	4.6 ± 1.4	$\mathbf{7.0} \pm 2.5$	2.9 ± 1.1
FT-POMCP-VE-WPF	2.9 ± 1.2	4.3 ± 1.2	0.6 ± 1.6	2.4 ± 1.3
DESPOT	OOR	$20.8^{\dagger} \pm 0.3$	OOR	OOR
PFT	2.1 ± 1.9	-10.6 ± 2.3	-2.3 ± 1.9	-0.7 ± 0.8
POMCP	20.0 ± 2.1	18.3 ± 4.3	-0.8 ± 2.1	0.0 ± 0.0
POMCP-WPF	11.1 ± 2.4	$\mathbf{26.6} \pm 2.4$	$\mathbf{8.5} \pm 1.5$	0.0 ± 0.0
POMCPOW	$\mathbf{32.6} \pm 2.1$	12.3 ± 1.8	-4.9 ± 1.6	OOR

(b) Medium map: MARS(11, 11), $\gamma = 0.95, c = 1.25$

Environment	MARS(11, 11)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	4.3 ± 1.3	-4.1 ± 1.3	-4.6 ± 1.7	-2.6 ± 4.1
FS-PFT-VE	15.9 ± 1.4	-5.9 ± 1.3	-2.0 ± 1.9	$\mathbf{7.2} \pm 1.3$
FS-POMCP-MP	-2.5 ± 1.1	-1.6 ± 1.5	-5.1 ± 4.2	-4.8 ± 2.6
FS-POMCP-MP-WPF	-1.4 ± 0.7	0.2 ± 0.4	2.1 ± 1.2	0.9 ± 1.7
FS-POMCP-VE	5.7 ± 2.1	1.5 ± 2.0	-6.4 ± 2.7	-2.8 ± 5.5
FS-POMCP-VE-WPF	3.4 ± 1.0	1.0 ± 0.8	-0.7 ± 1.1	-1.4 ± 1.3
FS-POMCPOW-MP	2.0 ± 1.0	-1.6 ± 0.7	-2.3 ± 1.0	-1.6 ± 2.4
FS-POMCPOW-VE	-2.1 ± 0.8	1.8 ± 1.2	$\mathbf{2.9} \pm 1.5$	-5.0 ± 1.5
FT-PFT-MP	1.9 ± 0.8	0.1 ± 1.4	-4.5 ± 2.3	-5.6 ± 2.5
FT-PFT-VE	3.1 ± 1.0	-7.8 ± 1.1	-9.5 ± 1.4	-5.4 ± 1.4
FT-POMCP-MP	-2.7 ± 1.5	-1.4 ± 1.4	-0.5 ± 0.6	-0.5 ± 1.9
FT-POMCP-MP-WPF	-1.1 ± 1.7	0.0 ± 0.0	1.1 ± 0.6	-0.7 ± 1.7
FT-POMCP-VE	4.0 ± 1.1	$\mathbf{5.5} \pm 1.4$	$\mathbf{3.4} \pm 1.8$	1.5 ± 0.9
FT-POMCP-VE-WPF	5.9 ± 1.0	-2.3 ± 1.0	$\mathbf{3.1} \pm 1.1$	-2.1 ± 0.9
DESPOT	$\mathbf{25.1} \pm 0.7$	OOR	OOR	OOR
PFT	-9.1 ± 2.0	-6.8 ± 2.5	-0.2 ± 0.4	OOR
POMCP	18.0 ± 2.4	-6.4 ± 1.9	-6.4 ± 4.2	OOR
POMCP-WPF	6.9 ± 1.0	-9.3 ± 2.1	-4.5 ± 1.3	OOR
POMCPOW	14.2 ± 1.1	-4.1 ± 1.6	-1.1 ± 0.6	OOR

Table 10.3: MARS experiments with three maps and various numbers of agents. Maximum time per step is **five** seconds.

(c) Large map: MARS(15, 15), $\gamma = 0.95, c = 1.25$

Environment	MARS(15, 15)			
Nr. of Agents	3	4	5	6
FS-PFT-MP	2.3 ± 0.8	-2.4 ± 2.8	-3.0 ± 1.8	-2.6 ± 3.7
FS-PFT-VE-WPF	-5.8 ± 1.3	-5.2 ± 1.2	$\mathbf{4.2} \pm 0.9$	2.6 ± 0.6
FS-POMCP-MP	-1.2 ± 0.5	-1.4 ± 2.3	-1.5 ± 1.6	-2.9 ± 2.3
FS-POMCP-MP-WPF	-1.3 ± 0.5	-0.1 ± 0.7	-0.2 ± 0.2	-0.0 ± 0.1
FS-POMCP-VE	3.4 ± 2.2	-3.2 ± 2.5	-2.1 ± 2.1	-3.8 ± 2.3
FS-POMCP-VE-WPF	-2.9 ± 0.8	0.5 ± 0.5	0.1 ± 0.8	$\mathbf{6.9} \pm 1.1$
FS-POMCPOW-MP-WPF	1.3 ± 2.2	-0.5 ± 1.6	-0.3 ± 2.0	-1.7 ± 1.8
FS-POMCPOW-VE-WPF	-2.9 ± 0.9	$\mathbf{4.9} \pm 0.8$	$\mathbf{3.3} \pm 1.1$	0.4 ± 1.4
FT-PFT-MP-WPF	-6.5 ± 1.3	-0.4 ± 0.6	-0.1 ± 0.7	-2.1 ± 0.8
FT-PFT-VE-WPF	-3.8 ± 0.9	-3.6 ± 1.0	1.9 ± 0.6	3.0 ± 0.9
FT-POMCP-MP	-1.0 ± 0.6	-0.3 ± 0.4	0.1 ± 0.1	-0.6 ± 1.1
FT-POMCP-MP-WPF	-0.1 ± 0.6	0.1 ± 0.2	0.2 ± 0.2	0.4 ± 0.3
FT-POMCP-VE	-1.3 ± 0.6	0.4 ± 0.6	-0.9 ± 0.6	3.4 ± 0.8
FT-POMCP-VE-WPF	1.7 ± 0.6	$\mathbf{3.6} \pm 0.7$	-0.2 ± 0.4	-1.5 ± 0.6
DESPOT	N.A.	N.A.	N.A.	N.A.
PFT-WPF	-1.7 ± 1.2	0.2 ± 0.7	-0.2 ± 0.2	OOR
POMCP	4.1 ± 2.0	0.1 ± 1.9	0.0 ± 0.0	OOR
POMCP-WPF	$\mathbf{8.4} \pm 1.3$	-1.5 ± 1.1	0.0 ± 0.0	OOR
POMCPOW-WPF	-13.5 ± 1.5	0.2 ± 1.1	-0.1 ± 0.5	OOR

Table 10.3: MARS experiments with three maps and various numbers of agents. Maximum time per step is **five** seconds.