Master thesis
Computing Science

Radboud University

A* Algorithms for Dec-POMDPs

Author:
Wietze Koops

Supervisor:
dr. Sebastian Junges

Second Supervisor:
prof. dr. Nils Jansen

August 26, 2024

Abstract

Partially observable Markov decision processes (POMDPs) formalize sequential decision making in a stochastic environment, where the decision maker (agent) cannot fully observe the state of the environment. Decentralized POMDPs (Dec-POMDPs) generalize this to a multi-agent setting, where each agent locally decides which action to take, and each agent receives local observations of the state. As such, Dec-POMDPs provide a way to model settings where multiple agents cooperate to achieve a common goal, but where communication is costly or ineffective.

In this thesis, we present three algorithms that advance the state-of-the-art in exact and approximate Dec-POMDP solving. These algorithms build on multi-agent A* (MAA*), an algorithm that finds policies by exploring a search tree. In our algorithms, we use a novel small-step variant of the search tree, which has a constant rather than a double exponential outdegree. The first algorithm, recursive small-step multi-agent A* (RS-MAA*), is an exact algorithm, for which we introduce tight, recursive heuristics that can be computed on-the-fly. The second algorithm, policy-finding multi-agent A* (PF-MAA*), is designed to find good policies for high horizons fast. Key ingredients for PF-MAA* are (1) using clustered sliding window memory, (2) pruning the A* priority queue, and (3) using novel heuristics tailored to policy-finding. The third algorithm, terminal reward multi-agent A* (TR-MAA*), aims to find upper bounds on the optimal value for high horizons. For this, we propose a tight, scalable heuristic which periodically reveals the state.

Our experiments show excellent performance of all three algorithms on a wide range of standard benchmarks. We extend the horizon for which exact solving is possible on all hard benchmarks. In addition, PF-MAA* finds superior policies for several benchmarks for high horizons, while TR-MAA* certifies the near-optimality of these policies by finding close upper bounds.

Acknowledgements

I am grateful to my supervisors, dr. Sebastian Junges, prof. dr. Nils Jansen, and dr. Thiago D. Simão, for introducing me to the exciting topic of Dec-POMDPs and for giving me the freedom to explore my own research direction, while also guiding me in the process. I would like to thank Sebastian and Nils for their feedback over the past two years, which has taught me a lot on how to write an academic paper. I also really appreciated our discussions about research and academic life in general. I would like to thank Sebastian for the extensive feedback on the thesis. Finally, I would like to thank Thiago for his supervision throughout the first year.

The first part of this thesis was supported by an ELLIS Excellence Fellowship, funded by the ELLIS unit Nijmegen. I would like to thank the other ELLIS Excellence Fellows for the nice interactions and discussions.

I am also grateful to Nils and to the PhD students in Nils' research group: Christoph, Eline, Maris, Marnix, Merlijn, and Thom, for being able to feel welcome and actively participate in the research group and in both academic and social events. This experience has been very enjoyable, and I will always remember the past two years as an excellent start of my academic career.

Contents

1 Introduction

Sequential decision making studies agents that repeatedly make choices regarding how to act in a changing environment. With the rise of autonomous agents such as robots and drones, the influence of this subfield of artificial intelligence on daily life continues to grow. Proper functioning of such drones is sometimes a matter of life and death, for instance when they are deployed in a search and rescue operation [Lyu *et al.*, 2023].

There are several challenges that drones (also known as unmanned aerial vehicles, UAVs) face in a search and rescue operation [Waharte and Trigoni, 2010]. Firstly, when surveying an area, the UAVs use sensory data to detect whether there is a victim at that area. Of course, it is crucial to prevent false negatives, i.e. missing victims. At the same time, also false positives should be avoided as they lead to the costly and risky deployment of a human rescue team to a location without any victims. Secondly, battery sizes put limits on the amount of energy that UAVs can use. To save energy, a possibility is to make a landing and have the UAV remain idle for some time while searching. Thirdly, the UAVs have to avoid environmental hazards, such as trees and buildings, while taking into account (stochastic) weather conditions such as winds.

These aspects of a search and rescue operation using a UAV can be modeled using a partially observable Markov decision process (POMDP) [Chanel *et al.*, 2013; Eaton *et al.*, 2017]. In general, a POMDP models sequential decision making in a stochastic, partially observable environment [Kaelbling *et al.*, 1998]. The environment is modeled using a state. In the UAV context, the state for example describes the location of victims and the location and battery level of the UAV. The state changes stochastically as a result of an action that the UAV takes. For example, the UAV can fly in a particular direction, but its new location is stochastic due to the wind. In addition, the state is partially observable. In particular, victims can only be observed if the UAV is sufficiently close to them, and even then, there is a chance that the UAV misclassifies its sensory input as not containing a victim while there is a victim, or conversely. However, the UAV does observe its own location (at least, if the GPS is working) and its own battery level. Finally, there are rewards and costs (modeled as negative rewards), describing the performance of the UAV. For example, there is a large reward for finding victims, while there are costs for running out of battery, crashing into a tree, and reporting victims which are actually not there.

Further complications arise when multiple UAVs are used at the same time. Firstly, the UAVs have to coordinate their actions. For example, when two UAVs are close, they should be careful to not crash into each other or block each other's view. Secondly, communication between UAVs may be costly or even impossible, for instance due to battery usage or limited network connectivity.

A multi-UAV search and rescue operation can be modeled using a decentralized POMDP (Dec-POMDP) [Zhu *et al.*, 2021]. In general, Dec-POMDPs can model POMDPs with multiple agents (decision makers) where communication is difficult or lossy, but where the agents aim to achieve a common goal [Oliehoek and Amato, 2016]. In particular, agents receive their own local observations of the environment and have to decide locally which actions to take. Communication is only possible if this is an explicit action. In the UAV context, each of the UAVs takes actions based their own sensory input and the scarce messages they may receive from other UAVs.

However, before the operation starts, a (centralized) planner can coordinate the actions of the UAVs, for instance by dividing the search area into several regions and assigning a specific region to each UAV. In addition, the planner can decide on rules when the UAVs should communicate and how other UAVs should act upon receiving messages of other UAVs. In this way, the UAVs are each provided with a local policy describing how to act given a particular history of sensory inputs and received messages. Together, these local policies form a joint policy. The goal of the planner is to design a joint policy that yields the largest expected reward (also called value) over a fixed number of time steps, called the horizon. Such a joint policy is called optimal.

The goal of this thesis is to solve the problem of the planner, i.e., to find a joint policy for a Dec-POMDP that yields the largest (or at least, a large) value over a given finite horizon. In short, we search for good or optimal policies using an A* algorithm [Hart *et al.*, 1968], which is a search algorithm that uses heuristics to steer the search in the direction of good policies and to prove that particular groups of policies cannot be optimal.

Problem definition. We now define the problem that we aim to solve in this thesis a bit more precisely. In fact, we aim to solve two related problems. The first problem is solving Dec-POMDPs exactly over a finite horizon. An exact algorithm for solving Dec-POMDPs must guarantee that the policy that it provides is optimal, i.e. achieves the largest value among all policies. The second problem is solving Dec-POMDPs over a finite horizon approximately with guarantees, i.e. providing a policy and an upper bound on the optimal value. Using the upper bound, we can give a guarantee on the quality of the found policy in comparison to an optimal policy.

We note that the decision problem underlying solving Dec-POMDPs exactly over a finite horizon is NEXP-hard [Bernstein *et al.*, 2002]. This means that in practice, exact algorithms take double exponential time in the worst case. In addition, also solving Dec-POMDPs ϵ-optimally is NEXP-hard [Rabinovich *et al.*, 2003]. Hence, also providing a policy and an upper bound such that the value of the policy and the upper bound are always at most a factor $1 - \epsilon$ apart for some fixed $\epsilon > 0$, is intractable in the worst case. Nevertheless, in many practical cases it is possible to exploit structure to do better than these complexity results suggest.

Algorithms. To solve these problems, we introduce three algorithms for solving Dec-POMDPs in this thesis, building on multi-agent A* (MAA*) [Szer *et al.*, 2005]. The first algorithm, recursive small-step multi-agent A* (RS-MAA*), is designed to solve Dec-POMDPs exactly, i.e. to find an optimal policy. The second algorithm, policy-finding multi-agent A* (PF-MAA*), aims to find good (but not necessarily optimal) policies for high horizons fast. The third algorithm, terminal reward multi-agent A* (TR-MAA*), in turn aims to compute upper bounds on optimal values for Dec-POMDPs for high horizons. Together, the latter two algorithms form an approximate solver with guarantees. We proceed by introducing each of these algorithms in more detail.

Multi-agent A*. Classical multi-agent A* (MAA*) [Szer *et al.*, 2005] searches through the space of partially specified policies (hereafter: partial policies) by incrementally fixing actions for a particular time step (called a stage), for all agents and all observation histories at once. The search carried out by the A* algorithm can be represented using a search tree, whose root is the partial policy that does not specify any actions and where the children of a partial policy are the partial policies that specify actions for an additional stage. The leaves of the search tree are fully specified policies. Since the number of observation histories is exponential in the stage t, the outdegree of this search tree is double exponential in the stage t. In particular, the running time of classical MAA* is always double exponential in the horizon h.

Our first insight is that this double exponential outdegree can be avoided by only fixing one action of one agent at once. This leads to a constant outdegree, at the cost of increasing the height of the search tree from linear to exponential in the horizon h. We call this novel search tree the small-step search tree, and the corresponding algorithm small-step multi-agent A*. Small-step MAA* is the basis for all three of our algorithms.

Finally, we note that any MAA*-variant uses a heuristic that assigns a value to partial policies, to decide which partial policies to explore next. Such a heuristic for a partial policy should be an upper bound for the value of any fully specified policy agreeing with the partial policy. In that case, if a heuristic value of a partial policy is smaller than the value of a found policy, we are sure that that partial policy cannot have any descendants better than the found policy.

RS-MAA*. Recursive small-step multi-agent A* (RS-MAA*) is our exact algorithm. Besides the small-step search tree, the main ingredient for RS-MAA* is the use of tight, recursive heuristics. In the context of A*, heuristics should overestimate the value of the best policy among a group of policies. Existing heuristics in the literature overestimate the value by assuming that agents can use all joint observations or all but the last joint observation to decide how to act, rather than only their own local observations. Instead, our new heuristics correspond to revealing a small, fixed number of joint observations. We solve the corresponding problem by recursively solving Dec-POMDPs with a smaller horizon. To make this more scalable, we exploit the anytime nature of A* algorithms, which allows for returning an upper bound on the heuristic value at any point. In addition, we reuse the lossless clustering introduced by Oliehoek *et al.* [2009], which allows grouping together local observation histories without loss in policy value.

PF-MAA*. Policy-finding multi-agent A* (PF-MAA*) is our policy finding algorithm. A first challenge that PF-MAA* must address is that the size of policies, when presented as functions of the local observation history, grows exponentially with the horizon. To limit the size of the policies in our search space, we consider policies that only use the last k local observations to decide how to act. Our first contribution for PF-MAA* is the development of a lossless clustering of these windows of k observations. Secondly, to limit the time spent searching, we only expand the most promising partial policies, and prune other partial policies. Thirdly, we introduce novel heuristics, designed to guide the search towards the best policies.

TR-MAA*. Finally, terminal reward multi-agent A* (TR-MAA*) is our algorithm that finds upper bounds for high horizons. TR-MAA* also reuses lossless clustering [Oliehoek *et al.*, 2009]. However, the recursive heuristics used by RS-MAA* are often too expensive for high horizons. To make the heuristic more scalable, we periodically reveal the true state of the Dec-POMDP.

Applications. We have seen the application of Dec-POMDPs in the context of multi-UAV search and rescue operations. Other applications of Dec-POMDPs include wireless sensor networks [Zhang *et al.*, 2015], bandwidth allocation [Hemmati *et al.*, 2015], decentralized computation offloading [Tang *et al.*, 2020], and maintenance problems [Bhustali and Andriotis, 2023].

Although this thesis makes significant contributions in making Dec-POMDP solving more scalable (especially with respect to the horizon), real-world applications typically remain too large. The main reason for this is that the number of states of Dec-POMDPs used to model practical applications is usually very large.

1.1 Contributions

We now discuss the concrete contributions of this thesis. First, we note that the results presented in this thesis are based on the results reported already in two conference papers [Koops *et al.*, 2023; Koops *et al.*, 2024], of which I am the first author. However, the thesis provides a more thorough description, and also some new results. In Appendix C, we provide an overview of the relation between this thesis and the papers.

We can categorize the contributions in our algorithms into three main areas. Firstly, concise representation of policies, using clustering of observation histories. Secondly, the search algorithm used to explore the space of policies. Thirdly, heuristics used in the A* search.

Clustering. For clustering, our contribution lies in developing a lossless clustering for sliding window memory, similarly to the lossless clustering introduced by Oliehoek *et al.* [2009].

Search. Our primary contribution in terms of search lies in the small-step search tree. In addition, we developed two methods to optimize the search in the last stage of the Dec-POMDP. Finally, we present queue pruning methods for PF-MAA*.

Heuristics. We first provide a general framework for defining heuristics. The general idea of these heuristics is that agents receive additional information to decide how to act. In particular, we use this framework to define the recursive heuristics, which are the tighter than heuristics currently available in the literature, and terminal reward heuristics, which are tight and scalable. In addition, we develop heuristics tailored to policy-finding.

Empirical results. We implemented our algorithms in Python.[1] RS-MAA* achieves superior performance compared to existing exact solvers. In particular, to the best of our knowledge, RS-MAA* is the first exact algorithm to scale to horizon 12 on DecTiger (an improvement from 6) and to horizon 9 on Mars (an improvement from 4). In addition, PF-MAA* finds superior policies for several benchmarks, whose quality is certified by the upper bounds from TR-MAA*. In particular, for BoxPushing with horizons up to 100, we find policies with values that are only 0.5% smaller than our upper bounds.

[1] Source code and scripts to reproduce the results are available at https://zenodo.org/records/13374100.

1.2 Overview

We now provide a brief overview of the sections in this thesis. Section 2 introduces background and discusses related literature for (single-agent) POMDPs. Section 3 introduces Dec-POMDPs and formally defines the problems that we solve. Section 4 discusses clustering of observation histories. Section 5 explains how we search through the space of policies, and in particular the MAA* algorithm. Section 6 introduces a general framework for defining heuristics. Section 7, Section 8 and Section 9 respectively explain the algorithms RS-MAA*, PF-MAA* and TR-MAA*, and in particular the heuristics used by these algorithms. Section 10 provides an empirical evaluation of the algorithms. Finally, Section 11 discusses related work and Section 12 concludes.

2 Partially Observable MDPs (POMDPs)

In this section we introduce partially observable Markov decision processes (POMDPs), and give some background on algorithms for solving POMDPs. We assume that the reader is familiar with MDPs and the value iteration algorithm for MDPs, but provide a brief recap in Section 2.1. Section 2.2 introduces POMDPs, and Section 2.3.1 explains how to solve POMDPs using the belief MDP. This provides the background that is directly used later in this thesis. Section 2.3.2, 2.3.3, 2.3.4, and 2.4 provide additional background on POMDPs, which can be useful to appreciate the differences in algorithmic approaches and complexity between POMDPs and Dec-POMDPs. For more background on POMDPs, we also refer to the book chapter by Spaan [2012].

2.1 Markov Decision Processes (MDPs)

Markov decision processes (MDPs) [Bellman, 1957; Puterman, 1994] are the standard formalism for sequential decision making in a stochastically changing environment. As shown in Figure 1, an MDP models a decision making loop involving an agent and an environment. In each stage, the agent takes an action, which causes a state transition in the environment. The agent then learns the new state and also receives a reward based on the (old) state and the action taken.

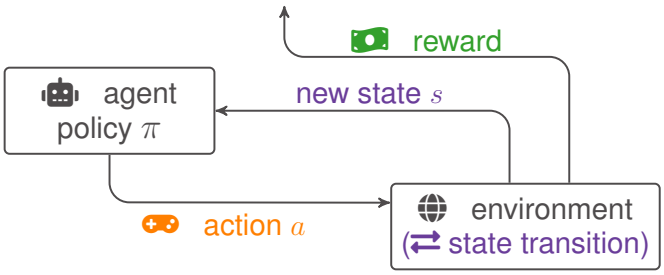


Figure 1: Schematic representation of a MDP

We now turn to the formal definition of an MDP. Let $\Delta(X)$ denote the set of probability distributions over a finite set X, i.e. the set of functions $p\colon X \to [0,1]$ satisfying $\sum_{x \in X} p(x) = 1$.

Definition 2.1 (MDP). *A Markov decision process (MDP) is a tuple* $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ *where*

- $\mathcal{S}$ *is a finite set of* states,
- $\mathcal{A}$ *is a finite set of* actions,
- $T\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ *is the* transition function, *and*
- $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the* reward *function.*

We write $T(s' \mid s, a)$ instead of $T(s,a)(s')$. An MDP describes a system whose state changes stochastically every stage. The MDP starts in some *initial state* s^0. In each stage t, the agent selects an action a^t. As a result, the state changes to s^{t+1} with probability $T(s^{t+1} \mid s^t, a^t)$. The agent is informed about the new state s^{t+1} and receives a reward $R(s^t, a^t)$.

Rewards based on the new state. In some problems, it is more natural to earn rewards based on the new state rather than the old state. However, when maximizing expected rewards, it is straightforward to transform rewards based on the new state to expected rewards based on the old state. Let $R'\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denote the reward function based on the new state. Then we can compute (expected) rewards based on the old state using the equation

$$R(s,a) = \sum_{s' \in \mathcal{S}} T(s' \mid a, s) R'(s', a).$$

Policies. We consider finding a policy for an MDP for a given finite number of stages (time steps) h, called the *horizon*. Since we consider MDPs with a finite set of states and a finite set of actions, there exists an optimal policy which is *Markovian* and *deterministic* [Puterman, 1994]. A Markovian (or *history-independent*) policy is a policy such that the action taken only depends on the current state $s \in \mathcal{S}$ and the stage $t \in \{0, \ldots, h-1\}$, rather than on the full history of states and actions taken previously. A deterministic policy is a policy such that there is only one action corresponding to each policy input, rather than a probabilistic choice over actions.

Formally, a *policy* for an MDP is a function $\pi \colon \mathcal{S} \times \{0, \ldots, h-1\} \to \mathcal{A}$, which specifies for each state $s \in \mathcal{S}$ and each stage $t \in \{0, \ldots, h-1\}$, which action $a \in \mathcal{A}$ the agent takes. The *value* of executing a policy π for an initial state s^0 over a horizon h is the expected reward:

$$V_\pi(s^0, h) = \mathbb{E}_\pi \left[\sum_{t=0}^{h-1} R(s^t, a^t) \right],$$

where s^t and a^t are the state and action at stage t. The goal is to find a policy π that maximizes the value $V_\pi(s^0, h)$, and to compute this value.

Value iteration. We now discuss how to compute (the value of) an optimal policy for an MDP using *value iteration* [Bellman, 1957]. This algorithm computes the value of an optimal policy for each initial state s and each horizon $0 \leq t \leq h$ using dynamic programming. For this, we use a *value function* $V^* \colon \mathcal{S} \times \{0, \ldots, h\} \to \mathbb{R}$, such that $V^*(s,t)$ is the value of an optimal policy of the MDP for initial state s over horizon t. We initialize $V^*(s,0) = 0$ and recursively define

$$V^*(s, t+1) = \max_{a \in \mathcal{A}} \Big[R(s,a) + \sum_{s' \in \mathcal{S}} T(s' \mid s, a) V^*(s', t) \Big]. \tag{2.1}$$

Note that $V^*(s,t)$ represents the optimal value when there are t stages to go. We now explain how to compute an optimal policy $\pi^* \colon \mathcal{S} \times \{0, \ldots, t-1\} \to \mathcal{A}$ after computing V^*. When computing $\pi^*(s,u)$, there are $t-u$ stages to go, corresponding to the computation of $V^*(s, t-u)$. Hence, we should select an action that attains the maximum in (2.1) when computing $V^*(s, t-u)$, i.e.

$$\pi^*(s,u) \in \arg\max_{a \in \mathcal{A}} \Big[R(s,a) + \sum_{s' \in \mathcal{S}} T(s' \mid s, a) V^*(s', t-u-1) \Big]. \tag{2.2}$$

Since we compute $O(|\mathcal{S}|h)$ values and each computation takes time $O(|\mathcal{A}||\mathcal{S}|)$, this shows that computing the value function V^* can be done in time $O(|\mathcal{A}||\mathcal{S}|^2 h)$, which is polynomial in $|\mathcal{A}|$, $|\mathcal{S}|$ and h. Since the total number of policies of an MDP $|\mathcal{A}|^{|\mathcal{S}|h}$ is exponential in $|\mathcal{S}|$ and h, this is an exponential speedup over brute force search. The main reason for this exponential speedup is that in each stage we can choose an optimal action $a \in \mathcal{A}$ for a given state $s \in \mathcal{S}$ independently from the choice for actions for other states in the same stage.

2.2 Partially Observable Markov Decision Processes (POMDPs)

Partially observable Markov decision processes (POMDPs) [Åström, 1965; Kaelbling *et al.*, 1998] capture the natural restriction that the agent does not (always) have full knowledge of the state of the environment, but instead only receives observations of the environment. As shown in Figure 2, a POMDP models a decision making loop similar to the loop in an MDP (Figure 1), but the agent is no longer informed about the new state (i.e., the state after the state transition).

Instead, the agent receives an observation, drawn from a distribution depending on the new state and the action taken, rather than being informed of the new state. The agent does not observe the reward while executing the policy, since this could 'leak' information about the state that the agent actually does not observe.

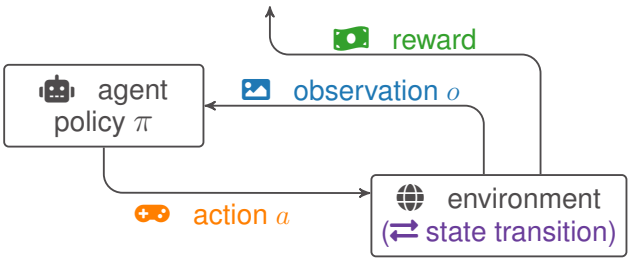


Figure 2: Schematic representation of a POMDP

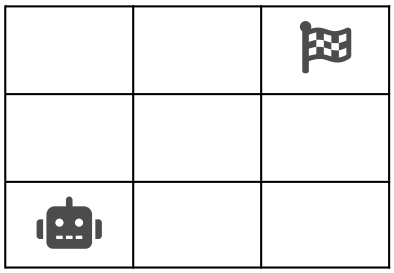


Figure 3: A POMDP on a 3×3 grid.

We now give the formal definition of a POMDP.

Definition 2.2 (POMDP)**.** *A partially observable Markov decision process (POMDP) is a tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R \rangle$ *where*

- $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ *is an MDP,*
- $\mathcal{O}$ *is a finite set of* observations*,*
- $O \colon \mathcal{A} \times \mathcal{S} \to \Delta(\mathcal{O})$ *is the* observation function*, and*
- $R \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the* reward *function.*

We write $T(s' \mid s, a)$ and $O(o \mid a, s')$ instead of $T(s, a)(s')$ and $O(a, s')(o)$, respectively. The POMDP starts in the state s^0 with probability $b(s^0)$ for some *initial belief* $b \in \Delta(\mathcal{S})$. In each stage t, the agent selects an action a^t. As a result, the state changes to s^{t+1} with probability $T(s^{t+1} \mid s^t, a^t)$. The agent earns, but does not observe, the reward $R(s^t, a^t)$. Finally, the agent receives the observation o^{t+1} with probability $O(o^{t+1} \mid a^t, s^{t+1})$. We call $\tau = o^1 o^2 \dots o^t$ an *observation history*, and denote the set of observation histories of length at most ℓ by $\mathcal{O}^{\leq \ell}$.

Policies. In a POMDP, a *policy* for the agent is a function $\pi \colon \mathcal{O}^{\leq h-1} \to \mathcal{A}$, which specifies for each observation history $\tau \in \mathcal{O}^{\leq h-1}$, which action $a \in \mathcal{A}$ the agent takes. The value of executing a policy π for an initial belief b over a horizon h is then

$$V_\pi(b, h) = \mathbb{E}_\pi \left[\sum_{t=0}^{h-1} R(s^t, a^t) \,\middle|\, s^0 \sim b \right],$$

where s^t and a^t are the state and action at stage t. The goal is to find a policy π that maximizes the expected reward $V_\pi(b, h)$.

Example. We now give an example of a POMDP.

Example 2.1 (3×3 Grid POMDP)**.** *We consider a* 3×3 *Grid POMDP. See Figure 3 for a schematic representation. This POMDP has 9 states, each representing that the agent is in a particular cell in the grid. The possible actions are staying in-place and moving in one of the four directions: up, down, left, right. Staying in place is successful with probability 1. However, the grid is slippery, and moving is only successful with probability 0.6, with there also being a probability of 0.1 for moving in each of the other three directions or staying in-place. If moving in a particular direction would cause the agent to go outside the grid, the probability for that direction is instead added to the probability for staying in-place. The goal is to navigate to the top right corner. Accordingly, the reward is 1 when the new state is the top right corner, and 0 otherwise. The initial belief assigns probability 1 to the bottom left corner.*

There are three observations, one for each column. Regardless of the action taken, the agent is always informed in which column it is, but not in which row it is.

To design a policy for this POMDP, also observations before the last one are potentially relevant for deciding which action to take. For example, if the second last and the last observation differ, then we know that the agent's column changed, and hence the agent's row did not change in that move. This illustrates the need to take the full observation history as input for the policy.

2.3 Algorithms for Solving POMDPs

We proceed by discussing how to solve (i.e., compute an optimal policy for) POMDPs. We first introduce beliefs and show how they can be used for policy evaluation. Then we introduce the belief MDP, which can be used to solve POMDPs in exponential time. Finally, we briefly discuss further (exact and approximate) algorithms for solving POMDPs.

Beliefs. In a POMDP, we can recursively compute joint probability distributions $\Pr(\tau^t, s^t)$ for a given policy $\pi\colon \mathcal{O}^{\leq h-1} \to \mathcal{A}$. For $t = 0$, it holds that τ^t is empty and that $\Pr(\tau^0, s^0) = b(s^0)$, where b is the initial belief. If $\tau^{t+1} = o^1 \dots o^t o^{t+1}$, then we can recursively compute

$$\Pr(o^1 \dots o^t o^{t+1}, s^{t+1}) = \sum_{s^t \in \mathcal{S}} O(o^{t+1} \mid a^t, s^{t+1}) T(s^{t+1} \mid s^t, a^t) \Pr(o^1 \dots o^t, s^t),$$

where $a^t = \pi(o^1 \dots o^t)$. For a given policy, there are $\sum_{t=0}^{h-1} |\mathcal{O}|^t |\mathcal{S}| \in O(|\mathcal{O}|^{h-1}|\mathcal{S}|)$ pairs (τ, s), so we can compute these probabilities in time $O(|\mathcal{O}|^{h-1}|\mathcal{S}|^2)$. Using these joint probabilities, we can also compute conditional distributions $\Pr(s^t \mid \tau^t)$, called *beliefs* (or *state beliefs*). Note that to compute the probability $\Pr(s^t \mid \tau^t)$, we implicitly also condition on the initial belief and on past policy (i.e., the policy up to stage t).

Belief updates. We now explain how to compute the belief $\Pr(s^{t+1} \mid \tau^{t+1})$ from the belief $\Pr(s^t \mid \tau^t)$, the action a^t taken in stage t and the observation o^{t+1}. We do this by showing in general how to compute a posterior belief $b_{a,o} \in \Delta(\mathcal{S})$ from a prior belief $b \in \Delta(\mathcal{S})$, the action $a \in \mathcal{A}$ taken and the observation $o \in \mathcal{O}$ received. We only define $b_{a,o}$ if it is possible to observe o when taking action a in a state in the support of b, i.e. if $\Pr(o \mid b, a) > 0$.

Using Bayes' rule, we can compute $b_{a,o}$:

$$\begin{aligned}
b_{a,o}(s') &= \Pr(s' \mid b, a, o) \\
&= \frac{\Pr(o \mid s', b, a) \Pr(s' \mid b, a)}{\Pr(o \mid b, a)} \\
&= \frac{\Pr(o \mid s', b, a) \sum_{s \in \mathcal{S}} \Pr(s' \mid a, s) b(s)}{\Pr(o \mid b, a)} \\
&= \frac{O(o \mid a, s') \sum_{s \in \mathcal{S}} T(s' \mid s, a) b(s)}{\Pr(o \mid b, a)} \\
&= \frac{O(o \mid a, s') \sum_{s \in \mathcal{S}} T(s' \mid s, a) b(s)}{\sum_{s'' \in \mathcal{S}} \left[O(o \mid a, s'') \sum_{s \in \mathcal{S}} T(s'' \mid s, a) b(s) \right]}.
\end{aligned}$$

We formalize this as the *belief update function*:

Definition 2.3 (Belief update function)**.** *The* belief update function *corresponding to a POMDP* $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R \rangle$ *is the function* $\{(b, a, o) \in \Delta(\mathcal{S}) \times \mathcal{A} \times \mathcal{O} \colon \Pr(o \mid b, a) > 0\} \to \Delta(\mathcal{S})$ *which maps a belief* $b \in \Delta(\mathcal{S})$*, an action* $a \in \mathcal{A}$ *and an observation* $o \in \mathcal{O}$ *to the belief* $b_{a,o} \in \Delta(\mathcal{S})$ *defined by*

$$b_{a,o}(s') = \frac{O(o \mid a, s') \sum_{s \in \mathcal{S}} T(s' \mid s, a) b(s)}{\sum_{s'' \in \mathcal{S}} \left[O(o \mid a, s'') \sum_{s \in \mathcal{S}} T(s'' \mid s, a) b(s) \right]}$$

if $\Pr(o \mid b, a) = \sum_{s'' \in \mathcal{S}} \left[O(o \mid a, s'') \sum_{s \in \mathcal{S}} T(s'' \mid s, a) b(s) \right] > 0$.

We call computing a single updated belief $b_{a,o}$ also a *belief update*. Computing a belief update can be done in time $O(|\mathcal{S}|^2)$: computing $O(o \mid a, s') \sum_{s \in \mathcal{S}} T(s' \mid s, a) b(s)$ can be done in time $O(|\mathcal{S}|)$ per state $s' \in \mathcal{S}$, so this takes time $O(|\mathcal{S}|^2)$ in total. Computing the normalization constant $\Pr(o \mid b, a)$ also takes time $O(|\mathcal{S}|^2)$, but this needs to be done only once per belief update.

Policy evaluation. Using the joint probabilities $\Pr(\tau^t, s^t)$, we can compute the value $V_\pi(b, h)$ of a given policy $\pi\colon \mathcal{O}^{\leq h-1} \to \mathcal{A}$ by iterating over all pairs (τ^t, s^t) for each stage t:

$$V_\pi(b, h) = \sum_{t=0}^{h-1} \sum_{\tau^t \in \mathcal{O}^t} \sum_{s^t \in \mathcal{S}} \Pr(\tau^t, s^t) R(s^t, \pi(\tau^t)).$$

This computation takes time $O(|\mathcal{O}|^{h-1}|\mathcal{S}|)$. Hence, this allows us to do policy evaluation in time $O(|\mathcal{O}|^{h-1}|\mathcal{S}|^2)$, including the time to compute the joint probabilities $\Pr(\tau^t, s^t)$.

Brute force search. We analyze the (time) complexity of brute force search, which enumerates all policies $\pi\colon \mathcal{O}^{\leq h-1} \to \mathcal{A}$ and returns a policy with the highest value. In total, there are $\sum_{t=0}^{h-1} |\mathcal{O}|^t = \frac{|\mathcal{O}|^h - 1}{|\mathcal{O}| - 1}$ observation histories, so a total of $|\mathcal{A}|^{\frac{|\mathcal{O}|^h - 1}{|\mathcal{O}| - 1}}$ policies. Hence, the time complexity of brute force search is $O\Big(|\mathcal{A}|^{\frac{|\mathcal{O}|^h - 1}{|\mathcal{O}| - 1}} |\mathcal{O}|^{h-1}|\mathcal{S}|^2\Big)$, which is double exponential in h.

2.3.1 Belief MDP

We now turn to the *belief MDP* [Åström, 1965; Kaelbling *et al.*, 1998] corresponding to a POMDP. Formally, the belief MDP is a continuous-state MDP, whose states are the probability distributions over the states of the POMDP. The actions are the actions of the POMDP. Given a distribution $b \in \Delta(\mathcal{S})$ and an action $a \in \mathcal{A}$, the possible distributions the belief MDP can transition to are the distributions $b_{a,o}$ (introduced in Definition 2.3) over the states conditional on observing the new observation $o \in \mathcal{O}$, if $\Pr(o \mid b, a) > 0$. However, since different observations can lead to the same belief, we should actually sum over all $o \in \mathcal{O}$ such that $b' = b_{a,o}$ when calculating the probability of transitioning to belief b' when taking action a in belief b.

The rewards of the belief MDP are the expected reward according to the distribution $b \in \Delta(\mathcal{S})$, i.e. $R(b, a) = \sum_{s \in \mathcal{S}} R(s, a) b(s)$. Together, this defines the belief MDP:

Definition 2.4 (Belief MDP)**.** *The* belief MDP *corresponding to a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R \rangle$ is the MDP $\langle \Delta(\mathcal{S}), \mathcal{A}, T', R' \rangle$, where $T'\colon \Delta(\mathcal{S}) \times \mathcal{A} \to \Delta(\Delta(\mathcal{S}))$ is defined by*

$$T'(b' \mid b, a) = \sum_{o \in \mathcal{O} : \Pr(o|b,a) > 0} \mathbf{1}[b' = b_{a,o}] \Pr(o \mid b, a),$$

and where $R'\colon \Delta(\mathcal{S}) \times \mathcal{A} \to \mathbb{R}$ is defined by $R'(b, a) = \sum_{s \in \mathcal{S}} R(s, a) b(s)$.

The main reason that the belief MDP is useful, is that its optimal value coincides with the optimal value of the POMDP [Åström, 1965].

Theorem 2.1 ([Åström, 1965])**.** *The optimal value of a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R \rangle$ for a given initial belief $b \in \Delta(\mathcal{S})$ and a given horizon h is equal to the optimal value of the corresponding belief MDP for initial state b and horizon h.*

Solving the belief MDP. We now discuss how to solve the belief MDP. In practice, only finitely many beliefs in the belief MDP are reachable when starting from a fixed initial belief $b \in \Delta(\mathcal{S})$ and considering a fixed finite horizon h. Identifying these beliefs allows us to solve a finite-state MDP (e.g. using value iteration) to find the optimal value of the POMDP. We now explain how to compute the reachable beliefs. For this, we consider all possible *action-observation histories* (AOHs), which are sequences $a^0 o^1 a^1 \dots a^{t-1} o^t$ describing the actions taken and observations received. All relevant beliefs are now the beliefs that can be obtained by using the belief update t times on the initial belief $b \in \Delta(\mathcal{S})$ with the action-observation pair $a^u o^{u+1}$ for $0 \leq u \leq t-1$ for some $0 \leq t \leq h-1$. In this way, we can compute all reachable beliefs by mapping each AOH for $0 \leq t \leq h-1$ to a belief.

We note that we can also inductively extract an optimal policy π^*_{P} for the POMDP from an optimal policy π^*_{B} for the belief MDP, stage by stage. For stage 0, we let $\pi^*_{\mathsf{P}}(\emptyset) = \pi^*_{\mathsf{B}}(b, 0)$, where $\emptyset$ denotes the empty observation history and b is the initial belief. If we have extracted a policy up

to stage $t \geq 0$, then we can use this to compute the actions taken in the stages up to t for a given observation history $\tau^{t+1} = o^1 \dots o^{t+1}$, and hence transform it into an AOH $a^0 o^1 a^1 \dots a^t o^{t+1}$. Now we can compute the belief $b_{\tau^{t+1}}$ corresponding to this AOH and set $\pi^*_{\mathsf{P}}(\tau^{t+1}) = \pi^*_{\mathsf{B}}(b_{\tau^{t+1}}, t+1)$.

In addition, the computation of the value function of the belief MDP for the reachable beliefs also allows us to compute the optimal value corresponding to a POMDP whose initial belief is the belief corresponding to an AOH $a^0 o^1 a^1 \dots a^{t-1} o^t$ with t observations and whose horizon is $h - t$.

Size of the belief MDP. We now analyze how much time constructing and solving the belief MDP takes in the worst case. For this, we fist compute a bound on the number of reachable beliefs, which is the number of state of the (finite) MDP that we solve. The number of reachable beliefs is at most the number of AOHs, which is

$$\sum_{t=0}^{h-1} (|\mathcal{A}||\mathcal{O}|)^t = \frac{(|\mathcal{A}||\mathcal{O}|)^h - 1}{|\mathcal{A}||\mathcal{O}| - 1} \in O((|\mathcal{A}||\mathcal{O}|)^{h-1}).$$

One belief update can be done in time $O(|\mathcal{S}|^2)$. By applying belief updates recursively, we do one belief update per AOH of length $1 \leq t \leq h-1$. Computing the expected reward corresponding to a belief takes time $O(|\mathcal{S}|)$ per action. Hence, constructing the belief MDP can be done in time $O\left((|\mathcal{A}||\mathcal{O}|)^{h-1}(|\mathcal{S}|^2 + |\mathcal{A}||\mathcal{S}|)\right)$.

Time complexity of solving POMDPs using the belief MDP. To solve the belief MDP, we compute its value function $V^*\colon \Delta(\mathcal{S}) \times \{0, \dots, h\} \to \mathbb{R}$ on the reachable beliefs, such that $V^*(b,t)$ is the optimal value of the belief MDP for initial belief b over horizon t. To bound the time taken to solve the belief MDP using value iteration, note that from each belief b there are (at most) $|\mathcal{O}|$ reachable beliefs for a given action $a \in \mathcal{A}$, namely the beliefs of the form $b_{a,o}$ for some $o \in \mathcal{O}$. In particular, the recursion for computing the value function (Equation (2.1)) can be written as

$$V^*(b, t+1) = \max_{a \in \mathcal{A}} \Big[R(b,a) + \sum_{o \in \mathcal{O}} \Pr(o \mid b, a) V^*(b_{a,o}, t) \Big].$$

Moreover, each AOH is only relevant for one horizon. For the beliefs corresponding to the $O\left((|\mathcal{A}||\mathcal{O}|)^{h-1}\right)$ AOHs of length $h-1$, we only have to select the action yielding the highest reward, which can be done in time $O(|\mathcal{A}|)$ per AOH. For the $O\left((|\mathcal{A}||\mathcal{O}|)^{h-2}\right)$ beliefs corresponding to AOHs of length $t \leq h-2$, we have to consider all $O(|\mathcal{O}|)$ reachable beliefs from that belief for each action. Hence, this takes time $O(|\mathcal{A}||\mathcal{O}|)$ per AOH. Using this, it follows that the belief MDP can be solved in time $O\left((|\mathcal{A}||\mathcal{O}|)^{h-1}(|\mathcal{A}|+1)\right)$, which is dominated by time taken to construct the belief MDP. Hence, we can compute the optimal value for a POMDP in time

$$O\left((|\mathcal{A}||\mathcal{O}|)^{h-1}(|\mathcal{S}|^2 + |\mathcal{A}||\mathcal{S}|)\right).$$

In practice, the number of (distinct) beliefs is usually smaller, since not all observations have positive probability for each prior belief and each action, and since some AOHs may yield the same belief. If B_t denotes the set of beliefs up to horizon t, then the above analysis shows that we can bound the running time by $O\left(|B_{h-1}|(|\mathcal{S}|^2 + |\mathcal{A}||\mathcal{S}|) + |B_{h-2}||\mathcal{A}||\mathcal{O}|\right)$.

Exponential speedup over brute force search. Finally, we note that this algorithm shows that computing (the value of) an optimal policy for a POMDP can be done in time exponential in h, which is an exponential speedup over brute force search. For MDPs, the value iteration algorithm also gives an exponential speedup over brute force search. For MDPs, the reason for the exponential speedup is that in each stage t we can choose an optimal action $a \in \mathcal{A}$ for each state s independently from the choice for actions for other states $\tilde{s}$ in the same stage. The main reason for this exponential speedup for POMDPs is similar to the reason for MDPs. Namely, in each stage t we can choose an optimal action $a \in \mathcal{A}$ for an observation history $o^1 \dots o^t$ independently from the choice for actions for other observation histories $\tilde{o}^1 \dots \tilde{o}^t$ in the same stage.

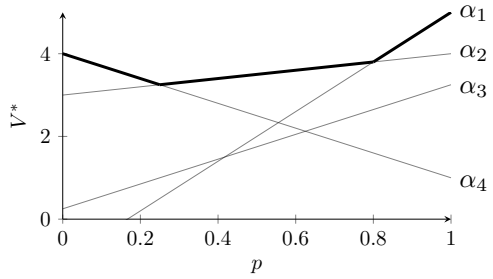


Figure 4: Example of a value function V^* over a belief simplex $\Delta(\mathcal{S})$ where $|\mathcal{S}| = 2$, which implies that each belief can be represented with a single probability p. We show a representation of V^* (thick lines) using four α-vectors, although the vector α_3 is dominated and can hence be pruned.

2.3.2 Exact Algorithms and α-vectors

We now discuss further exact algorithms for POMDPs. These algorithms exploit the structure of the value function V^* of the belief MDP, and in particular that there exists a representation of the value function using *α-vectors* [Smallwood and Sondik, 1973], as stated in the following theorem:

Theorem 2.2 ([Smallwood and Sondik, 1973])**.** *There exists a representation of the value function V^* using α-vectors: For each t there exist vectors $\alpha_1, \ldots, \alpha_z \in \mathbb{R}^{\mathcal{S}}$ such that*

$$V^*(b,t) = \max_{1 \le v \le z} \sum_{s \in \mathcal{S}} \alpha_v(s) b(s).$$

Hence, the value function is piecewise linear and convex (for a fixed stage t).

A graphical example of a representation of a value function using α-vectors is given in Figure 4. To prove Theorem 2.2, we can inductively construct the α-vectors. Let $\mathcal{V}_t$ denote the set of vectors corresponding to the value function for horizon t. For $t = 1$, we have the $|\mathcal{A}|$ vectors corresponding to the reward vectors of each action. If we have a set of vectors $\mathcal{V}_t$ for horizon t, then we can construct a set of vectors $\mathcal{V}_{t+1}$ for horizon $t+1$ of size $|\mathcal{A}||\mathcal{V}_t|^{|\mathcal{O}|}$ by considering each action that can be taken in the first stage and then for each observation a vector from $\mathcal{V}_t$ for the remaining t stages. We provide a detailed proof of Theorem 2.2 in Appendix A.1.

Algorithms. The optimal value of a POMDP for initial belief b over horizon t is equal to $V^*(b,t)$. Hence, finding a representation of the value function for horizon t using α-vectors is sufficient to compute the optimal value. However, when using the construction for the α-vectors sketched above, we have $|\mathcal{V}_1| = |\mathcal{A}|$ and $|\mathcal{V}_{t+1}| = |\mathcal{A}||\mathcal{V}_t|^{|\mathcal{O}|}$, which implies $|\mathcal{V}_t| = |\mathcal{A}|^{\sum_{i=0}^{t-1} |\mathcal{O}|^i} \in \Omega\big(|\mathcal{A}|^{|\mathcal{O}|^{t-1}}\big)$. Hence, the number of α-vectors is double exponential in t, which is clearly not tractable.

The number of α-vectors can be reduced by *pruning* dominated vectors from the set $\mathcal{V}_t$ for each t [Monahan, 1982], i.e. computing which vectors α_v never maximize $\sum_{s \in \mathcal{S}} \alpha_v(s) b(s)$, and removing those from $\mathcal{V}_t$. This is done using linear programming. As an example, the vector α_3 in Figure 4 can be pruned. Although pruning often results in a substantial reduction in the number of vectors, Monahan's algorithm still first generates all $|\mathcal{A}||\mathcal{V}_t|^{|\mathcal{O}|}$ vectors for stage $t+1$ before pruning them. *Incremental pruning* [Zhang and Liu, 1996] improves on Monahan's algorithm by considering the observations one by one and pruning each time before considering the next observation. Further improvements on incremental pruning have been made which reduce the total number of variables or constraints in the linear programs (LPs) [Cassandra *et al.*, 1997; Feng and Zilberstein, 2004; Hansen and Bowman, 2020], or exploit the structure of the LPs [Walraven and Spaan, 2017].

2.3.3 Approximate Algorithms

The algorithms considered so far can compute exact optimal values for POMDPs over a finite horizon. We proceed by discussing point-based methods, which generally provide lower bounds and upper bounds for the value function (either by explicitly computing both a lower and an

upper bound, or by computing one of the two bounds and giving theoretical guarantees about the tightness of the bound). We first discuss how upper bounds and lower bounds can be computed in general, and then proceed by discussing specific algorithms.

Upper bounds. Point-based methods compute upper bounds for the value function V^* on (a subset of) the belief simplex $\Delta(\mathcal{S})$ by computing upper bounds on a finite set of points $\mathcal{B} \subseteq \Delta(\mathcal{S})$. Since the value function is convex, these bounds for beliefs in $\mathcal{B}$ can be used to compute an upper bound on any belief b in the convex hull of $\mathcal{B}$. Formally, if we write $b = \sum_{i=1}^{n} \lambda_i b_i$ with $\sum_{i=1}^{n} \lambda_i = 1$ and $b_i \in \mathcal{B}$ for $1 \leq i \leq n$, then the convexity of the value function implies that

$$V^*(b,t) = V^*\left(\sum_{i=1}^{n} \lambda_i b_i, t\right) \leq \sum_{i=1}^{n} \lambda_i V^*(b_i,t).$$

Hence, if we have upper bounds on the $V^*(b_i,t)$, we can also compute an upper bound on $V^*(b,t)$. The coefficients λ_i in $b = \sum_{i=1}^{n} \lambda_i b_i$ giving the tightest upper bound can be computed using linear programming. Alternatively, if $\mathcal{B}$ contains all extreme points of the belief simplex (i.e., all $|\mathcal{S}|$ distributions that assign probability 1 to some state $s \in \mathcal{S}$), then a cheaper way to compute a reasonable upper bound on $V^*(b,t)$ is to consider writing b as a convex combination of an arbitrary point $b' \in \mathcal{B}$ and (at most) $|\mathcal{S}| - 1$ extreme points [Hauskrecht, 2000; Smith and Simmons, 2005].

Lower bounds. Lower bounds for the value function can be given using α-vectors, as introduced in Section 2.3.2. In particular, we can compute a set of α-vectors $\alpha_1, \ldots, \alpha_{z'} \in \mathbb{R}^{\mathcal{S}}$ such that

$$V^*(b,t) \geq \max_{1 \leq v \leq z'} \sum_{s \in \mathcal{S}} \alpha_v(s) b(s).$$

Another way to provide lower bounds is to compute a policy. When having an approximation of the value function, we can compute a policy by selecting actions which are best according to that approximate value function, similarly to how an optimal policy can be extracted from the (exact) value function for MDPs as in Equation (2.2) [Hauskrecht, 2000].

Algorithms. We now proceed by discussing algorithms that compute lower and/or upper bounds on the value function. Lovejoy [1991] provides lower and upper bounds on the value function on the belief simplex $\Delta(\mathcal{S})$ using a uniform grid consisting of all probability distributions whose probabilities are a multiple of $\frac{1}{M}$ for some M. However, the number of points in this discretization, which is $\binom{M+|\mathcal{S}|-1}{M} = \frac{(M+|\mathcal{S}|-1)!}{M!(|\mathcal{S}|-1)!}$, is prohibitive for large M and $|\mathcal{S}|$ (e.g. for $|\mathcal{S}| \geq 20$ and $M \geq 10$, this exceeds $2 \cdot 10^7$). Hauskrecht [2000] uses an adaptive non-uniform grid, by adding reachable beliefs as points to the grid. This is done by starting from the initial belief, and then repeatedly selecting the best action (according to the current approximation of the value function) and sampling an observation, until a new belief is found. Zhou and Hansen [2001] instead use a regular grid similar to the grid used by Lovejoy [1991], but with a variable resolution.

Point-Based Value Iteration (PBVI) [Pineau *et al.*, 2003] aims to approximate the value function V^* on all reachable beliefs from a given initial belief, i.e. all beliefs that can occur after updating the initial belief according to some action-observation history. Similarly to Hauskrecht [2000], the observations in this action-observation history are chosen by sampling. Heuristic Search Value Iteration (HSVI) [Smith and Simmons, 2004; Smith and Simmons, 2005] instead uses a *forward exploration heuristic* to select the observation leading to the next point, taking into account the likelihood of the observation and the difference between lower and upper bounds at the belief corresponding to this observation.[2] SARSOP [Kurniawati *et al.*, 2008] aims to further focus the region where V^* is approximated best to the beliefs that are reachable from the initial belief under an optimal policy. Another line of work focuses on finding approximations of the value function using few α-vectors [Dujardin *et al.*, 2015; Dujardin *et al.*, 2017]. Finally, Bork *et al.* [2020] use an abstraction-refinement framework to successively compute better abstractions of the belief MDP.

[2]Despite the similar name, HSVI is not related to A* heuristic search and the heuristic used by HSVI is quite different from the heuristics used in A* search: the heuristic used by HSVI selects an observation to explore further, while the heuristics used by A* are upper bounds on the value function.

2.3.4 Online Algorithms

The algorithms considered so far are offline algorithms, which provide a policy that specifies an action for all possible observation histories that can occur during policy execution. In contrast, online algorithms only provide actions for one sampled observation history while executing the provided actions. We discuss two online algorithms, which only use a simulator of the environment. As a result, they can scale up to much larger models. Partially Observable Monte-Carlo Planning (POMCP) [Silver and Veness, 2010] uses Monte Carlo Tree Search [Coulom, 2006] to estimate which action is the best. When deciding which action to select at a particular history, a number of simulations is done from that history onwards. To select actions in these simulation, the algorithm has to balance between exploring all actions and only selecting the actions that currently seem best. In comparison, DESPOT [Somani *et al.*, 2013] achieves better performance and better theoretical guarantees by using heuristic forward search for exploration, similarly to HSVI.

2.4 Complexity Results

Finally, we briefly discuss the complexity of solving POMDPs. The decision problem associated to POMDPs is the problem "Given an POMDP, an initial belief b^0, a horizon h, and a rational number V_{goal}, does there exist a policy with value at least V_{goal}?". This problem is PSPACE-complete [Papadimitriou and Tsitsiklis, 1987] when expressing h in unary. When expressing h in unary, the problem has size $\Omega(poly(|\mathcal{S}| + |\mathcal{A}| + |\mathcal{O}| + h))$.

The algorithm using the belief MDP from Section 2.3 shows that this decision problem is in EXPTIME. The belief MDP can also be used to show that the decision problem is in PSPACE, by calculating the optimal value $V^*(b^0, h)$. To do this in polynomial space, we first organize all beliefs in a belief tree, such that the root of the tree is the initial belief b^0 and the children of a belief b are all beliefs of the form $b_{a,o}$ (where we store the same belief multiple times if it occurs for multiple action-observation histories). We now make the following observation: once we have calculated $V^*(b, t+1)$ for some belief b, we no longer need to store the values $V^*(b_{a,o}, t)$. By exploring the belief tree in a depth-first manner, we then only need to store values $V^*(b_{a,o}, t)$ for one belief b per stage t simultaneously. Hence, we only need to store $O(|\mathcal{A}||\mathcal{O}|h)$ values of the value function simultaneously to calculate $V^*(b^0, h)$ for the initial belief b^0.

Moreover, we note that there also exist non-approximability results for POMDPs. For a POMDP $\mathcal{M}$, let $\text{opt}(\mathcal{M}, b^0, h)$ be its optimal value for initial belief b^0 and horizon h, and let $A(\mathcal{M}, b^0, h)$ be the value achieved by some algorithm A. Then we call A an ϵ-approximate algorithm if $(1-\epsilon)\text{opt}(\mathcal{M}, b^0, h) \leq A(\mathcal{M}, b^0, h)$ for all $\mathcal{M}, b^0, h$. Note that $A(\mathcal{M}, b^0, h) \leq \text{opt}(\mathcal{M}, b^0, h)$ always holds. Lusena *et al.* [2001] show that there is no polynomial time ϵ-approximate algorithm for solving POMDPs for any $0 < \epsilon < 1$, unless P = PSPACE. In particular, we expect any ϵ-approximate algorithm for POMDPs to take exponential time in the worst case.

3 Decentralized POMDPs (Dec-POMDPs)

In this section we introduce decentralized partially observable Markov decision processes (Dec-POMDPs), the main topic of this thesis. We start by defining Dec-POMDPs in Section 3.1. Section 3.2 explains policy evaluation for Dec-POMDPs. Section 3.3 introduces the underlying (PO)MDP of a Dec-POMDP, and uses this to introduce heuristics for Dec-POMDPs. Section 3.4 introduces some special cases of Dec-POMDPs. Section 3.5 states some complexity results for Dec-POMDPs, while Section 3.6 follows up with an informal discussion why Dec-POMDPs are harder than POMDPs. Finally, Section 3.7 formally states the problems that we solve in this thesis. For more background on Dec-POMDPs, we refer to the book by Oliehoek and Amato [2016].

3.1 Definition

Decentralized POMDPs generalize POMDPs to the multi-agent setting [Bernstein *et al.*, 2002; Oliehoek and Amato, 2016]. The agents each receive local observations and must locally select actions to execute. As shown in Figure 5, a Dec-POMDP models decision loops between each of the agents and the environment. In each stage, each of the agents individually selects an action, which together yields a joint action causing a state transition in the environment. Then the agents each get a local observation and earn, but do not observe, a reward.

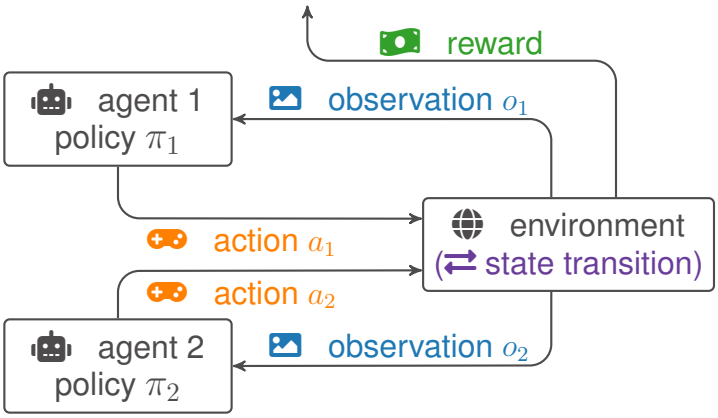


Figure 5: Schematic representation of a Dec-POMDP

We now turn to the formal definition of a Dec-POMDP.

Definition 3.1 (Dec-POMDP)**.** *A* decentralized partially observable Markov decision process *(*Dec-POMDP*) is a tuple* $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{O}}, T, O, R \rangle$ *where*

- $\mathcal{D} = \{1, \ldots, n\}$ *is a set of* n agents*,*
- $\mathcal{S}$ *is a finite set of* states*,*
- $\boldsymbol{\mathcal{A}} = \bigtimes_{i \in \mathcal{D}} \mathcal{A}_i$ *is a set of* joint actions $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$*, where each* $\mathcal{A}_i$ *is the finite set of* local actions *available to agent* i*,*
- $\boldsymbol{\mathcal{O}} = \bigtimes_{i \in \mathcal{D}} \mathcal{O}_i$ *is a set of* joint observations $\mathbf{o} = \langle o_1, \ldots, o_n \rangle$*, where each* $\mathcal{O}_i$ *is the finite set of* local observations *of agent* i*,*
- $T \colon \mathcal{S} \times \boldsymbol{\mathcal{A}} \to \Delta(\mathcal{S})$ *is the* transition function*,*
- $O \colon \boldsymbol{\mathcal{A}} \times \mathcal{S} \to \Delta(\boldsymbol{\mathcal{O}})$ *is the* observation function*, and*
- $R \colon \mathcal{S} \times \boldsymbol{\mathcal{A}} \to \mathbb{R}$ *is the* reward *function.*

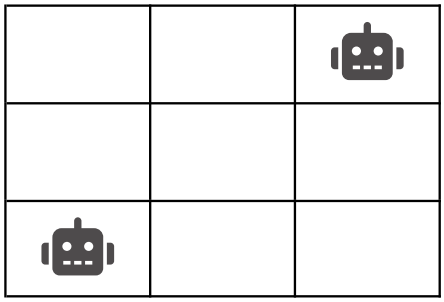


Figure 6: Schematic representation of the partially observable 3×3 Grid Dec-POMDP POGrid(3).

Figure 7: Schematic representation of DecTiger.

For $n = 1$ agent, a Dec-POMDP is a POMDP (Definition 2.2).

We write $T(s' \mid s, \mathbf{a})$ and $O(\mathbf{o} \mid \mathbf{a}, s')$ instead of $T(s, \mathbf{a})(s')$ and $O(\mathbf{a}, s')(\mathbf{o})$, respectively. A Dec-POMDP describes a system that starts in a state s^0 with probability $b(s^0)$ for some *initial belief* $b \in \Delta(\mathcal{S})$. At each stage t, each agent i individually selects a local action a_i^t. Together, this yields a joint action $\mathbf{a}^t = \langle a_1^t, \ldots, a_n^t \rangle$ with a corresponding reward $r^t = R(s^t, \mathbf{a}^t)$. The next state of the system is s^{t+1} with probability $T(s^{t+1} \mid s^t, \mathbf{a}^t)$. Finally, a joint observation $\mathbf{o}^{t+1}$ is drawn with probability $O(\mathbf{o}^{t+1} \mid \mathbf{a}^t, s^{t+1})$, and each agent i receives their local observation o_i^{t+1}.

Observation histories. We write $\tau_i = o_i^1 \ldots o_i^t$ for the *local observation history* (LOH) of agent i, and $\boldsymbol{\tau} = \mathbf{o}^1 \ldots \mathbf{o}^t$ for the *joint observation history*. We denote the set of all LOHs of agent i of length exactly ℓ and at most ℓ by $\mathcal{O}_i^\ell$ and $\mathcal{O}_i^{\leq \ell}$, respectively. We write $o_i^{[v,t]} = o_i^v \ldots o_i^t$ for the local observations of agent i between stage v and t. If the local observation in stage $t+1$ is $o \in \mathcal{O}_i$, then $o_i^{[v,t]} \cdot o$ denotes the local observations between stage v and $t+1$.

Policies. In a Dec-POMDP, each agent i has a *local policy* π_i, which together form the *joint policy* $\boldsymbol{\pi} = \langle \pi_1, \cdots, \pi_n \rangle$. A local policy for agent i maps LOHs for that agent to a local action $a_i \in \mathcal{A}_i$, formally: $\pi_i \colon \mathcal{O}_i^{\leq h-1} \to \mathcal{A}_i$. Let Π denote the set of joint policies. The value of executing a joint policy $\boldsymbol{\pi}$ for an initial belief b over a horizon h is then

$$V_{\boldsymbol{\pi}}(b, h) = \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{h-1} R(s^t, \mathbf{a}^t) \,\middle|\, s^0 \sim b \right],$$

where s^t and $\mathbf{a}^t$ are the state and joint action at stage t. The goal is to find a joint policy $\boldsymbol{\pi}$ that maximizes the expected reward $V_{\boldsymbol{\pi}}(b, h)$.

Examples. We now give two examples of Dec-POMDPs: A Dec-POMDP version of the 3×3 Grid problem (based on the 2×2 Grid problem with 2 observations by Amato *et al.* [2006]), as well as the classical Dec-POMDP benchmark problem DecTiger [Nair *et al.*, 2003].

Example 3.1 (3×3 Grid Dec-POMDP)**.** *We consider a* 3×3 *Grid Dec-POMDP, which in short we also denote by* POGrid(3) *(for* partially observable grid*). See Figure 6 for a schematic representation. It has two agents. There are* $9^2 = 81$ *states, giving the positions for both agents. The transitions, local actions and local observations are for each agent as in the* 3×3 *Grid POMDP (Example 2.1). However, the rewards are different: the reward is now 1 if the agents are on the same cell in the grid (according to the new state), while the reward is 0 otherwise. Initially, one of the agents is in the bottom left corner while the other agent is in the top right corner.*

Example 3.2 (Decentralized tiger problem)**.** *The* DecTiger *problem [Nair* et al.*, 2003] is a multi-agent variation of the Tiger problem [Kaelbling* et al.*, 1998]. See Figure 7 for a schematic representation. In this problem, there are two agents and two rooms, labelled left and right. One of the rooms contains a dangerous tiger, while the other room contains a valuable treasure. The state of the Dec-POMDP (*TL *or* TR*) indicates whether the tiger is in the left room or in the right room. Each agent has three actions: listen (*Li*), open the door to the left room (*OL*), or open the door to the right room (*OR*).*

*Listening gives a noisy observation regarding the location of the tiger (*HL *or* HR*), which is correct with probability 0.85, and incurs a small cost (*-1 *per agent). The state resets if any agent opens a door, in which case both agents get an uninformative observation (uniform distribution over the observations). If only one agent opens the door leading to the treasure, they receive a positive reward (*$+10$*). But, if both agents find the treasure, they receive a larger reward (*$+20$*). A large penalty is given for opening the door leading to the tiger (*-100 *if one agent opens a door with the tiger,* -50 *if both open the door with the tiger).*

In the DecTiger problem (Example 3.2), the agents need to find a policy to gather information regarding the location of the tiger and then decide whether to open a door. Hence, they need to coordinate on which stages they will open a door and on which stages they will listen, since opening a door will nullify the other agent's effort to gather information (as the state resets). An example of a local policy for an agent in DecTiger for horizon $h = 3$ is

$$\emptyset \mapsto \text{Li},\ (\text{HR}) \mapsto \text{Li},\ (\text{HL}) \mapsto \text{Li},\ (\text{HR}, \text{HR}) \mapsto \text{OL},$$
$$(\text{HR}, \text{HL}) \mapsto \text{Li},\ (\text{HL}, \text{HR}) \mapsto \text{Li}, (\text{HL}, \text{HL}) \mapsto \text{OR}.$$

Hence, in the first two stages the agent gathers information by listening, while in the last stage the agent opens a door if the information is unambiguous, and listens otherwise. It turns out that it is optimal for both agents to do this if $h = 3$.

3.2 Policy Evaluation

We now turn to beliefs and policy evaluation for Dec-POMDPs, which is very similar to beliefs and policy evaluation for POMDPs as explained in Section 2.3. In fact, each joint policy for a Dec-POMDP induces a policy for the underlying POMDP.

Beliefs. Computing joint probabilities $\Pr(\boldsymbol{\tau}^t, s^t)$ in a Dec-POMDP is similar to the corresponding computation in POMDPs. Let $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$ be a joint policy, where $\pi_i \colon \mathcal{O}_i^{\leq h-1} \to \mathcal{A}_i$. For $t = 0$, it holds that $\boldsymbol{\tau}^t$ is empty and that $\Pr(\boldsymbol{\tau}^0, s^0) = b(s^0)$. If $\boldsymbol{\tau}^{t+1} = \mathbf{o}^1 \ldots \mathbf{o}^t \mathbf{o}^{t+1}$, then we can recursively compute

$$\Pr(\mathbf{o}^1 \ldots \mathbf{o}^t \mathbf{o}^{t+1}, s^{t+1}) = \sum_{s^t \in \mathcal{S}} O(\mathbf{o}^{t+1} \mid \mathbf{a}^t, s^{t+1}) T(s^{t+1} \mid s^t, \mathbf{a}^t) \Pr(\mathbf{o}^1 \ldots \mathbf{o}^t, s^t),$$

where $\mathbf{a}^t = \langle a_1^t, \ldots, a_n^t \rangle$ and $a_i^t = \pi_i(o_i^1 \ldots o_i^t)$. For a given joint policy, there are $O(|\boldsymbol{\mathcal{O}}|^{h-1}|\mathcal{S}|)$ pairs $(\boldsymbol{\tau}, s)$, so we can compute these probabilities in time $O(|\boldsymbol{\mathcal{O}}|^{h-1}|\mathcal{S}|^2)$.

Policy evaluation. Using the joint probabilities $\Pr(\boldsymbol{\tau}^t, s^t)$, we can compute the value $V_{\boldsymbol{\pi}}(b, h)$ of a given joint policy $\boldsymbol{\pi}$ by iterating over all pairs $(\boldsymbol{\tau}^t, s^t)$ for each stage t:

$$V_{\boldsymbol{\pi}}(b, h) = \sum_{t=0}^{h-1} \sum_{\boldsymbol{\tau}^t \in \boldsymbol{\mathcal{O}}^t} \sum_{s^t \in \mathcal{S}} \Pr(\boldsymbol{\tau}^t, s^t) R(s^t, \boldsymbol{\pi}(\boldsymbol{\tau}^t)),$$

where we write $\boldsymbol{\pi}(\boldsymbol{\tau}^t) = \langle \pi_1(\tau_1^t), \ldots, \pi_n(\tau_n^t) \rangle$. This computation takes time $O(|\boldsymbol{\mathcal{O}}|^{h-1}|\mathcal{S}|)$. Hence, this allows us to do policy evaluation in time $O(|\boldsymbol{\mathcal{O}}|^{h-1}|\mathcal{S}|^2)$, including the time to compute the joint probabilities $\Pr(\boldsymbol{\tau}^t, s^t)$.

Brute force search. To analyze the complexity of brute force search, let us assume for simplicity that all local action and observation sets are equally large, and let $|\mathcal{A}_*|$ and $|\mathcal{O}_*|$ denote their sizes, respectively. The number of LOHs is then $n\sum_{t=0}^{h-1}|\mathcal{O}_*|^t = n\frac{|\mathcal{O}_*|^h-1}{|\mathcal{O}_*|-1}$, so there are $|\mathcal{A}_*|^{n\frac{|\mathcal{O}_*|^h-1}{|\mathcal{O}_*|-1}}$ joint policies (which we can upper bound by $|\mathcal{A}_*|^{n|\mathcal{O}_*|^h}$). Hence, the time complexity of brute force search is $O\Big(|\mathcal{A}_*|^{n\frac{|\mathcal{O}_*|^h-1}{|\mathcal{O}_*|-1}}|\boldsymbol{\mathcal{O}}|^{h-1}|\mathcal{S}|^2\Big)$, which is double exponential in h.

3.3 Upper Bounds for Dec-POMDPs

The decentralized nature of the Dec-POMDP makes it more challenging to solve than a POMDP. For this reason, we want to consider a relaxation of the decentralized problem. The *underlying POMDP* of a Dec-POMDP assumes that a centralized decision maker with access to the joint observations chooses the joint action in each stage. Equivalently, we can assume that each of the agents receives the joint observations and selects an action locally, since the agents can decide beforehand on which actions to take for a given joint observation history. We can go one step further by also removing partial observability, leading to the *underlying MDP*, where a centralized decision maker chooses the joint action in each stage based on the true state.

Definition 3.2 (Underlying (PO)MDP)**.** *Let $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{O}}, T, O, R\rangle$ be a Dec-POMDP. The underlying MDP is the MDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, T, R\rangle$. The underlying POMDP is the POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{O}}, T, O, R\rangle$.*

We note that the action (and observation) set still have a factorized structure in the underlying (PO)MDP, which makes the underlying MDP a *multi-agent MDP* [Boutilier, 1996] and the underlying POMDP a *multi-agent POMDP* [Pynadath and Tambe, 2002].

The optimal value $V_{\text{POMDP}}(b,h)$ of the underlying POMDP of a Dec-POMDP (for a given initial belief b and horizon h) is an upper bound for the optimal value $V_{\text{Dec-POMDP}}(b,h)$ of the Dec-POMDP [Szer *et al.*, 2005]. The reason for this is that each joint policy $\boldsymbol{\pi}$ for a Dec-POMDP induces a policy $\boldsymbol{\pi}_{\text{POMDP}}\colon \boldsymbol{\mathcal{O}}^{\leq h-1} \to \boldsymbol{\mathcal{A}}$ for the underlying POMDP, defined by $\boldsymbol{\pi}_{\text{POMDP}}(\boldsymbol{\tau}) = \langle \pi_1(\tau_1), \cdots, \pi_n(\tau_n)\rangle$. Hence, any value achievable by the Dec-POMDP is also achievable by the underlying POMDP, showing that $V_{\text{Dec-POMDP}}(b,h) \leq V_{\text{POMDP}}(b,h)$. In practice, $V_{\text{POMDP}}(b,h)$ is almost always strictly larger, since the additional information contained in the observations of the other agents can be used to make better choices regarding which action to take.

The optimal value $V_{\text{MDP}}(b,h)$ of the underlying MDP is in turn an upper bound for the POMDP value $V_{\text{POMDP}}(b,h)$, as shown by Hauskrecht [2000]. Intuitively, this holds because knowing the state is more informative than just knowing the observation history, since the rewards and transition probabilities are directly determined by the state while the observations only provide information about rewards and transition probabilities indirectly.

In summary, we have a hierarchy of bounds $V_{\text{Dec-POMDP}}(b,h) \leq V_{\text{POMDP}}(b,h) \leq V_{\text{MDP}}(b,h)$. When searching for policies for the Dec-POMDP, we use these upper bounds (for smaller horizons and other initial beliefs) to guide the search. In this context, we call V_{MDP} and V_{POMDP} the *MDP heuristic* and the *POMDP heuristic*, respectively. We further discuss this in Section 6.

3.4 Subclasses within Dec-POMDPs

We now introduce a few subclasses within the class of Dec-POMDPs [Becker *et al.*, 2004; Goldman and Zilberstein, 2004]. In particular, each of these subclasses consists of Dec-POMDPs with a factorized state space, which means that the state space can be written as a tuple of local states, where each local state describes one agent.

Definition 3.3 (Factorized state space)**.** *A Dec-POMDP has a factorized state space if we can write $\mathcal{S} = \bigtimes_{i\in\mathcal{D}} \mathcal{S}_i$, where $\mathcal{S}_i$ is a set of* local states *of agent i.*

Transition-independence means that the state or action of other agents do not influence the local state of a given agent, which implies that we can factorize the transition function.

Definition 3.4 (Transition-independence)**.** *A Dec-POMDP with a factorized state space $\bigtimes_{i\in\mathcal{D}} \mathcal{S}_i$ is* transition-independent *if we can write $T(s' \mid s, \mathbf{a}) = \prod_{i\in\mathcal{D}} T_i(s'_i \mid s_i, a_i)$ for local transition functions $T_i\colon \mathcal{S}_i \times \mathcal{A}_i \to \Delta(\mathcal{S}_i)$.*

Observation-independence means that the state or action of other agents do not influence the observations of a given agent. In particular, it implies that there are no actions in the Dec-POMDP that the agents can use to communicate.

Definition 3.5 (Observation-independence)**.** *A Dec-POMDP with a factorized state space* $\bigtimes_{i\in\mathcal{D}} \mathcal{S}_i$ *is* observation-independent *if we can write* $O(\mathbf{o} \mid \mathbf{a}, s') = \prod_{i\in\mathcal{D}} O_i(o_i \mid a_i, s'_i)$ *for local observation functions* $O_i \colon \mathcal{A}_i \times \mathcal{S}_i \to \Delta(\mathcal{O}_i)$.

Finally, we consider a class of Dec-POMDPs that is jointly fully observable, which means that the state is determined by the last joint observation. We formalize this using a map, which maps a joint observation $\mathbf{o} \in \boldsymbol{\mathcal{O}}$ to the only new state $s' \in \mathcal{S}$ in which it can occur.

Definition 3.6 (Dec-MDP)**.** *A* Dec-MDP *is a Dec-POMDP such that is jointly fully observable, i.e. such that there exists a map* $J \colon \boldsymbol{\mathcal{O}} \to \mathcal{S}$ *such that if* $O(\mathbf{o} \mid \mathbf{a}, s') > 0$*, then* $J(\mathbf{o}) = s'$.

Goldman and Zilberstein [2004] show that a transition- and observation-independent Dec-MDP (TOI-Dec-MDP) is also locally fully observable, which means that for each agent $i \in \mathcal{D}$ the last local observation o_i of that agent determines the local state s_i of that agent. Formally, there exists a map $J_i \colon \mathcal{O}_i \to \mathcal{S}_i$ such that if $O_i(o_i \mid a_i, s'_i) > 0$, then $J_i(o_i) = s'_i$.

The 3×3-Grid Dec-POMDP from Example 3.1, is transition- and observation-independent, but it is not a Dec-MDP since the joint observation does not determine the full state.

3.5 Complexity Results

We proceed by briefly discussing the complexity of solving Dec-POMDPs. The decision problem associated to Dec-POMDPs is the problem "Given an Dec-POMDP, an initial belief b, a horizon h, and a rational number V_{goal}, does there exist a policy with value at least V_{goal}?".

The decision problem associated to Dec-POMDPs is NEXP-complete when expressing h in unary [Bernstein *et al.*, 2002], thus we expect algorithms to take double exponential time to solve the decision problem. Non-deterministically guessing a policy and evaluating it (in exponential time) shows that this decision problem is indeed in NEXP. Rabinovich *et al.* [2003] show that also solving Dec-POMDPs ϵ-optimally is NEXP-hard. Nevertheless, TOI-Dec-MDPs form an easier subclass within the class of Dec-POMDPs: the associated decision problem is NP-complete [Goldman and Zilberstein, 2004]. The main reason for this drop in complexity is that it is sufficient to consider policies that depend only on the last local observation for TOI-Dec-MDPs.

Assuming that PSPACE $\neq$ NEXP, this shows that solving Dec-POMDPs is harder than solving POMDPs. In Section 3.6, we discuss the reasons for this on an informal level.

3.6 Challenges in Solving Dec-POMDPs

We now describe two challenges in solving Dec-POMDPs compared to solving POMDPs. These provide intuition for the complexity gap between POMDPs and Dec-POMDPs (see Section 3.5).

Dependence between different LOHs. In general, we cannot choose an optimal action for some local observation history (LOH) independently from the choice for other LOHs. Namely, the action agent i takes for LOH τ_i has implications for the optimal choice for all LOHs τ_j that another agent j could have observed when agent i observed τ_i. However, the choice for the actions on these LOHs τ_j in turn also influence the optimal action for other LOHs of agent i. Therefore, we can in general not split solving a Dec-POMDP into solving several independent problems with a lower horizon, as is possible for an MDP and a POMDP.

Insufficiency of state beliefs. In a POMDP, the state beliefs are a sufficient statistic for planning: if two observation histories induce the same state belief, then we can assume in an optimal policy that the same action is taken for these observation history. However, state beliefs are not sufficient statistics in Dec-POMDPs, because it is not just important what agent i believes about the state, but also what agent i believes about what agent j believes about the state, and

what agent i believes about what agent j believes about what agent k believes about the state, etc. The fact that state beliefs are not sufficient statistics limits the extent to which approaches similar to α-vectors (introduced in Section 2.3.2 for POMDPs) can be used for Dec-POMDPs.

A sufficient statistic for Dec-POMDPs is given by each agent's belief over the states and the observation histories of other agents. This sufficient statistic is discussed in more detail in Section 4.3. However, the size of this statistic typically grows exponentially with the horizon.

3.7 Problem Statement

The aim of this thesis is to provide algorithms for exact and approximate Dec-POMDP solving over a finite horizon. In exact solving, the algorithm has to guarantee that it outputs an optimal policy $\boldsymbol{\pi}^* \in \arg\max_{\boldsymbol{\pi}' \in \Pi} V_{\boldsymbol{\pi}'}(b,h)$. We formalize exact solving in the following problem statement.

Problem statement I (Exact Solving)**.** Given a Dec-POMDP, an initial belief b, and a horizon h, find a joint policy $\boldsymbol{\pi}$ that maximizes $V_{\boldsymbol{\pi}}(b,h)$.

The main performance metric of an exact algorithm is the computation time.

In approximate Dec-POMDP solving, the goal is to find good policies fast. In addition, we want to certify how good the provided policy is, which can be done by providing an upper bound for the value of an optimal policy. We formalize this in the following problem statement.

Problem statement II (Approximate Solving with Guarantees)**.** Given a Dec-POMDP, an initial belief b, and a horizon h, find a policy $\boldsymbol{\pi}$ and an upper bound $U(b,h)$ s.t.

$$V_{\boldsymbol{\pi}}(b,h) \quad \leq \quad \max_{\boldsymbol{\pi}' \in \Pi} V_{\boldsymbol{\pi}'}(b,h) \quad \leq \quad U(b,h),$$

where $V_{\boldsymbol{\pi}}(b,h)$ and $U(b,h)$ are close to each other.

For approximate solving, there is a trade-off between solution quality $V_{\boldsymbol{\pi}}(b,h)$ and computation time, and similarly between the tightness of the upper bound $U(b,h)$ and computation time.

4 Clustering

We now turn to clustering. When presenting a joint policy by listing for each local observation history (LOH) the action taken by the agent, the size of the policy is exponential in the horizon h. To migitate this, we cluster (group together) LOHs. Since the number of clustered policies is exponential in the number of clusters, clustering LOHs can lead to a substantial reduction in the size of the search space. Moreover, we study criteria for clustering LOHs losslessly, i.e. without loss in optimal policy value compared to another clustering.

We start by introducing clusterings in general in Section 4.1. In Section 4.2, we develop the multi-agent belief [Hansen *et al.*, 2004], an important tool for proving that a clustering is lossless. In Section 4.3 we discuss lossless clustering [Oliehoek *et al.*, 2009; Oliehoek *et al.*, 2013], which is an important ingredient in our exact algorithms and algorithms that find upper bounds. In Section 4.4 and Section 4.5 we introduce sliding window memory and lossless clustering for sliding window memory, respectively. Finally, in Section 4.6 we discuss probability-based clustering.

4.1 Clusterings

We start by defining a clustering.

Definition 4.1 (Clustering)**.** *A* clustering C *is a partition* $C_{i,t}$ *of the set* $\mathcal{O}_i^t$ *of length-*t *LOHs for each stage* $0 \leq t \leq h-1$ *and each agent* $i \in \mathcal{D}$*.*

We now introduce *cluster policies*, which map clusters to actions. Write $C_i = \bigcup_{t=0}^{h-1} C_{i,t}$ for the set of agent i's clusters. Let $f_{\mathcal{O}\to C} \colon \bigcup_{i\in\mathcal{D}} \bigcup_{t=0}^{h-1} \mathcal{O}_i^t \to \bigcup_{i\in\mathcal{D}} C_i$ be the map that assigns a local observation history to its cluster. A *local cluster policy* for agent i is a map $\pi_i^C \colon C_i \to \mathcal{A}_i$. The corresponding local policy π_i is defined by $\pi_i(\tau_i) = \pi_i^C(f_{\mathcal{O}\to C}(\tau_i))$ for each LOH τ_i. We denote the set of all cluster policies $\boldsymbol{\pi}^C = \langle \pi_1^C, \ldots, \pi_n^C \rangle$ corresponding to clustering C by Π^C. The value of a cluster policy $\boldsymbol{\pi}^C$ is the value of the corresponding policy $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$, where each π_i is the local policy corresponding to π_i^C.

We note that restricting to cluster policies is without loss of generality when the clustering is not fixed. In particular, we can consider the *trivial* clustering, i.e. the clustering in which each LOH gets its own cluster. Formally, the trivial clustering is the clustering defined by the partitions $C_{i,t} = \{\{o_i^{[1,t]}\} : o_i^{[1,t]} \in \mathcal{O}_i^t\}$. For this clustering, every policy $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$ can be written as a cluster policy $\boldsymbol{\pi}^C$, by defining $\pi_i^C(\{o_i^{[1,t]}\}) = \pi_i(o_i^{[1,t]})$.

4.1.1 Incremental Clustering

Intuitively, an *incremental* clustering is a clustering such that if two LOHs are clustered, then their extensions by the same local observation are also clustered together. Hence, if two observation histories $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ from stage t are clustered together in an incremental clustering, then the agent does not distinguish them in stage t nor in future stages. To formally define when a clustering is incremental, we use the equivalence relation $\equiv_{C_{i,t}}$ induced by the partition $C_{i,t}$, such that $o_i^{[1,t]} \equiv_{C_{i,t}} \tilde{o}_i^{[1,t]}$ if and only if the LOHs $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ are in the same cluster in $C_{i,t}$.

Definition 4.2 (Incremental clustering)**.** *Let* $\equiv_{C_{i,t}}$ *be the equivalence relation induced by the partition* $C_{i,t}$*. A clustering is* incremental *at stage* t *if*

$$\forall i \in \mathcal{D}.\ \forall o_i^{[1,t]}, \tilde{o}_i^{[1,t]} \in \mathcal{O}_i^t.\ \forall o \in \mathcal{O}_i : \left(o_i^{[1,t]} \equiv_{C_{i,t}} \tilde{o}_i^{[1,t]}\right) \implies \left(\left(o_i^{[1,t]} \cdot o\right) \equiv_{C_{i,t+1}} \left(\tilde{o}_i^{[1,t]} \cdot o\right)\right).$$

A clustering is incremental if it is incremental at each stage $0 \leq t \leq h-2$*.*

Example. We now provide an example of a clustering in the DecTiger problem (Example 3.2).

Example 4.1 (Clusterings for DecTiger)**.** *Consider the* DecTiger *problem over horizon* $h = 3$*. An example of a clustering* C *is given by*

$$C_{1,0} = C_{2,0} = \{\{\emptyset\}\}, \quad C_{1,1} = C_{2,1} = \{\{(\text{HL}),\ (\text{HR})\}\},$$
$$C_{1,2} = C_{2,2} = \{\{(\text{HL}, \text{HL})\},\ \{(\text{HL}, \text{HR}),\ (\text{HR}, \text{HL})\},\ \{(\text{HR}, \text{HR})\}\}.$$

Using this clustering, a local cluster policy $\pi_i^C : C_i \to \mathcal{A}_i$ *for agent* i *is given by*

$$\{\emptyset\} \mapsto \text{Li}, \quad \{(\text{HR}), (\text{HL})\} \mapsto \text{Li}, \quad \{(\text{HR}, \text{HR})\} \mapsto \text{OL},$$
$$\{(\text{HR}, \text{HL}), (\text{HL}, \text{HR})\} \mapsto \text{Li}, \quad \{(\text{HL}, \text{HL})\} \mapsto \text{OR}.$$

However, the clustering C *is not incremental. Namely, the LOHs* (HR) *and* (HL) *are clustered together, but their extensions with the same local observation* HL*, i.e.* (HR, HL) *and* (HL, HL)*, are not clustered together. Intuitively, we should not cluster together the observation histories* (HR) *and* (HL) *in an incremental clustering because the agent needs to distinguish them in a later stage, even though the agent does not need to distinguish them in stage 1. To create an incremental clustering* C'*, we can give* (HR) *and* (HL) *each their own cluster:*

$$C'_{1,0} = C'_{2,0} = \{\{\emptyset\}\}, \quad C'_{1,1} = C'_{2,1} = \{\{(\text{HL})\}, \ \{(\text{HR})\}\},$$
$$C'_{1,2} = C'_{2,2} = \{\{(\text{HL}, \text{HL})\}, \ \{(\text{HL}, \text{HR}), (\text{HR}, \text{HL})\}, \ \{(\text{HR}, \text{HR})\}\}.$$

Storing incremental clusterings. We now explain an algorithmic detail, namely how we can store incremental clusterings efficiently in practice. Storing a clustering by storing all partitions explicitly takes exponential ($\Omega\big(\sum_{i\in\mathcal{D}} |\mathcal{O}_i|^{h-1}\big)$) space, even if the number of clusters is small. An important reason for using incremental clusterings is that they can be stored more efficiently if the number of clusters is small.

Consider the clustering for a fixed agent i. The incrementality allows us to represent the clustering as a number of maps, one for each stage, mapping a cluster and the next observation to a new cluster in the next stage. Specifically, rather than storing the sets (clusters) in $C_{i,t}$ explicitly, we just store the number of clusters $|C_{i,t}|$ and refer to the clusters using an integer between 0 and $|C_{i,t}| - 1$. To store the clustering structure, we store maps $f_{i,t} : C_{i,t} \times \mathcal{O}_i \to C_{i,t+1}$, which map a cluster $c_i^t \in C_{i,t}$ and an observation $o_i^{t+1} \in \mathcal{O}_i$ to a cluster $c_i^{t+1} \in C_{i,t+1}$. Note that we always have $|C_{i,0}| = 1$, since in stage 0 we only have the empty observation history. For an LOH $\tau_i = o_i^{[1,v]}$, we can compute the corresponding cluster c_i^v recursively by taking the unique element $c_i^0 \in C_{i,0}$ and recursively defining $c_i^{t+1} = f_{i,t}(c_i^t, o_i^{t+1})$ for $0 \le t \le v-1$. Hence, we can indeed recover the clustering using these maps.

Finally, we note that the size of this representation is $O\big(\sum_{i\in\mathcal{D}} \sum_{t=0}^{h-2} |C_{i,t}||\mathcal{O}_i|\big)$, which is polynomial in $|\mathcal{O}_i|$ and h if the number of clusters $|C_{i,t}|$ is polynomial. Hence, using incremental clusterings allows us to store clusterings for large horizons.

4.1.2 Comparing Clusterings

We proceed by discussing some notions that can be used to compare two clusterings. A clustering C' is *coarser* than C if each partition $C'_{i,t}$ is coarser than $C_{i,t}$, i.e. if each cluster in $C'_{i,t}$ is a union of clusters in $C_{i,t}$. Put differently, a clustering C' is coarser than C if any two LOHs which are clustered in C are also clustered in C'. We call C' *finer* than C if C is coarser than C'.

Consider a clustering C. To limit the search space of cluster policies, we wish to further cluster (i.e., take unions of) the clusters in C, without further loss in optimal policy value. To express this, we define when a clustering C' is lossless with respect to C.

Definition 4.3 (Lossless clustering)**.** *We call a clustering* C' lossless *with respect to another clustering* C*, if* C' *is coarser than* C *and*

$$\max_{\pi\in\Pi^{C'}} V_\pi(b,h) = \max_{\pi\in\Pi^{C}} V_\pi(b,h),$$

where $\Pi^{C'}$ *and* Π^{C} *denote the set of all cluster policies corresponding to* C' *and* C*, respectively.*

The trivial clustering is the clustering which assigns every LOH to a different cluster. We call a clustering lossless, if it is lossless with respect to the trivial (i.e. no) clustering.

Example. We continue Example 4.1.

Example 4.2 (Clusterings for DecTiger)**.** *Consider the clusterings C and C' introduced in Example 4.1. The clustering C is coarser than C'. The clustering C' is in turn coarser than the trivial clustering. It can be shown that the clustering C is lossless (w.r.t. the trivial clustering), since the local policies corresponding to the local cluster policies given in Example 4.1 form an optimal joint policy for the Dec-POMDP. Hence, the clustering C' is also lossless.*

4.2 Multi-agent Belief

To select an optimal action, agents have to reason about the actions that other agents will take, both in the current stage and future stages. For this, we define the *multi-agent belief state* and the *multi-agent belief* [Hansen *et al.*, 2004]. To define the multi-agent belief, we first define the concept of a *subtree policy*, which describes how a particular agent acts from a specific LOH onwards.

Definition 4.4 (Subtree policy)**.** *Let $j \in \mathcal{D}$ be an agent and let π_j be its local policy. A* subtree policy $\gamma_j^{h-t} \colon \mathcal{O}_j^{\leq h-t-1} \to \mathcal{A}_i$ *describes how agent j acts given the observations they receive from stage t onwards. Specifically, the subtree policy corresponding to the LOH $o_j^{[1,t]}$ satisfies*

$$\gamma_j^{h-t}(o_j^{[t+1,v]}) = \pi_j(o_j^{[1,v]}).$$

Remark 4.1. *Note that different LOHs can yield the same subtree policy. In fact, if two LOHs yield the same subtree policy, then we could have clustered these LOHs in an incremental clustering. However, we only observe this after finding the subtree policy, while our goal is to determine that two LOHs can be clustered before defining a policy from these LOHs onwards.*

With the definition of a subtree policy, we now define the multi-agent belief.

Definition 4.5 (Multi-agent belief)**.** *A* multi-agent belief state *for agent $i \in \mathcal{D}$ and stage $0 \leq t \leq h-1$ is a pair $(s, \boldsymbol{\gamma}_{\neq i}^{h-t})$, where s is a state and*

$$\boldsymbol{\gamma}_{\neq i}^{h-t} = \langle \gamma_1^{h-t}, \ldots, \gamma_{i-1}^{h-t}, \gamma_{i+1}^{h-t}, \ldots, \gamma_n^{h-t} \rangle$$

represents a policy for all agents except agent i from stage t onwards. A multi-agent belief *for agent $i \in \mathcal{D}$ and stage t is a distribution over multi-agent belief states for agent i and stage t.*

Similarly to how the distribution over the states is a sufficient statistic for POMDPs, the multi-agent belief is a sufficient statistic for Dec-POMDPs.

Lemma 4.1 (Multi-agent belief is a sufficient statistic)**.** *The multi-agent belief for agent $i \in \mathcal{D}$ and stage t $(0 \leq t \leq h-1)$ is a sufficient statistic for computing the value of any subtree policy $\gamma_i^{h-t} \colon \mathcal{O}_i^{\leq h-t-1} \to \mathcal{A}_i$ that agent i can use from stage t onwards.*

We now explain the intuition behind this lemma. The subtree policy specifies the actions for agent i, while the multi-agent belief specifies a distribution over states and actions for the other agents, for each observation history. Moreover, since the multi-agent belief specifies a distribution over states, the Markov property of Dec-POMDPs allows us to calculate a distribution over these observation histories. Together, this allows us to compute the expected reward. We give a more detailed proof in Appendix A.2.1.

Since the multi-agent belief is a sufficient statistic for the value of any subtree policy of agent i, it follows that two LOHs that induce the same multi-agent belief have the same optimal subtree policies and can hence be clustered losslessly.

Corollary 4.1. *Suppose that two LOHs $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ of agent i induce the same multi-agent belief. Then they can be clustered losslessly.*

Proof. Let $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ be two LOHs that induce the same multi-agent belief. Then the value of any subtree policy γ_i^{h-t} is equal for $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$. Hence, the same subtree policies are optimal for $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$. In particular, we can without loss of policy value take the same optimal subtree policy for the LOHs $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$, which means that the LOHs can be clustered losslessly. □

4.3 Lossless Clustering

The goal of this section is to define a criterion that it allows us to cluster together LOHs *a priori*, i.e., before determining an optimal policy, and such that the resulting clustering is lossless, incremental, and not (always) the trivial clustering. Specifically, we discuss the *probabilistic equivalence* criterion [Oliehoek *et al.*, 2009; Oliehoek *et al.*, 2013], which is a sufficient criterion for when two local observation histories (LOHs) can be clustered together losslessly. We start by stating the criterion, and then prove that the resulting clustering is lossless (Section 4.3.1) and incremental (Section 4.3.2).

We first introduce some notation. Write $\mathbf{c}_{\neq i}^{t-1}$ for the tuple consisting of the cluster of each agent except agent i in stage $t-1$, and write $\mathbf{o}_{\neq i}^{t}$ for the tuple consisting of the local observations of each agent except agent i in stage t. Finally, in all probabilities that we write, we implicitly also condition on the past policy and the initial belief.

Definition 4.6 (Probabilistic equivalence)**.** *Consider a clustering and a policy up to stage $t-1$, and an initial belief. Two LOHs $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ of agent i are* probabilistically equivalent*, written $o_i^{[1,t]} \sim_p \tilde{o}_i^{[1,t]}$, if*

$$\Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^{t} \,\middle|\, o_i^{[1,t]}\right) = \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^{t} \,\middle|\, \tilde{o}_i^{[1,t]}\right)$$

for all $s^t \in \mathcal{S}$ and all tuples $\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^{t}$.

Note that $\sim_p$ is an equivalence relation. Write $\left[o_i^{[1,t]}\right]$ for the equivalence class of $o_i^{[1,t]}$. We now define the clustering corresponding to probabilistic equivalence:

Definition 4.7. *The clustering corresponding to probabilistic equivalence consists of the sets defined by $C_{i,t} = \left\{ [o_i^{[1,t]}] \;\middle|\; o_i^{[1,t]} \in \mathcal{O}_i^t \right\}$ for $i \in \mathcal{D}$ and $0 \leq t \leq h-1$.*

Since the definition of probabilistic equivalence requires a clustering up to stage $t-1$ to be given, Definition 4.7 should be interpreted as defining a clustering inductively. The clustering in stage 0 trivially consists of one equivalence class per agent (containing the empty observation history).

We now explain the intuition behind the criterion. If the clustering is incremental, then the pair $(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^{t})$ contains all information that the other agents will use to make decisions in future stages. Hence, probabilistic equivalence states that agent i's joint belief about the states and the information of the other agents is the same for both LOHs.

We note that Oliehoek *et al.* [2009] instead consider clustering local action-observation histories. However, when considering a fixed (past) policy (which Oliehoek *et al.* also require), we can compute the action-observation history from the observation history. Hence, we instead state the criterion in terms of local observation histories. In addition, rather than requiring that the beliefs over the LOHs $\mathbf{o}_{\neq i}^{[1,t]}$ of the agents except agent i are the same, we only require that the beliefs over the pairs $(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^{t})$ are the same, which is a weaker condition.

We make these changes for two reasons. Firstly, the condition stated here aligns better with the actual implementation. Namely, checking that $\Pr\left(s^t, \mathbf{o}_{\neq i}^{[1,t]} \,\middle|\, o_i^{[1,t]}\right) = \Pr\left(s^t, \mathbf{o}_{\neq i}^{[1,t]} \,\middle|\, \tilde{o}_i^{[1,t]}\right)$ requires checking exponentially many equalities (in t), which is hence not feasible for large t. Secondly, this condition is closer to the criterion used for lossless clustering for sliding window memory (which we discuss in Section 4.5).

Examples. We now give examples of probabilistic equivalence in DecTiger (Example 3.2) and in the partially observable grid problem POGrid(3) (Example 3.1).

Example 4.3 (Probabilistic equivalence in DecTiger)**.** *Suppose that the policy in stage 0 and 1 is such that no agent opens a door. Then the observation histories* (HR, HL) *and* (HL, HR) *are probabilistically equivalent. However,* (HR, HL) *and* (HL, HR) *are not probabilistically equivalent when an agent opens a door in stage 0 (but not in stage 1), since then the first observation is not relevant while the second observation is. This illustrates the need to take into account the policy of earlier stages when determining whether two observation histories can be clustered.*

Before turning to POGrid(3), we discuss how the probabilistic equivalence criterion simplifies for a problem which is transition- and observation-independent Dec-POMDP (Section 3.4).

Remark 4.2 (Probabilistic equivalence in a transition- and observation-independent problem)**.** *Consider a transition- and observation-independent Dec-POMDP. Hence, agents actually only need to reason about their own local state to check for probabilistic equivalence.*

We now formalize this. By definition, a transition- and observation-independent Dec-POMDP has a factorized state space. Write s_i for the local state of agent i and $s_{\neq i}$ for the tuple of local states of the other agents. Then we have

$$\Pr\left(s^t_{\neq i}, \mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i} \,\middle|\, o_i^{[1,t]}, s_i^t\right) = \Pr\left(s^t_{\neq i}, \mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i} \,\middle|\, \tilde{o}_i^{[1,t]}, s_i^t\right),$$

for any two LOHs $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$, since the local state and the local observations of agent i are not informative about the local state and local observations of the other agent. Hence, to show that $o_i^{[1,t]} \sim_p \tilde{o}_i^{[1,t]}$, it is sufficient to show that

$$\Pr\left(s_i^t \,\middle|\, o_i^{[1,t]}\right) = \Pr\left(s_i^t \,\middle|\, \tilde{o}_i^{[1,t]}\right).$$

Namely, multiplying this equation by $\Pr\left(s^t_{\neq i}, \mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i} \,\middle|\, o_i^{[1,t]}, s_i^t\right) = \Pr\left(s^t_{\neq i}, \mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i} \,\middle|\, \tilde{o}_i^{[1,t]}, s_i^t\right)$ and rewriting using the definition of conditional probability yields the probabilistic equivalence criterion $\Pr\left(s^t, \mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i} \,\middle|\, o_i^{[1,t]}\right) = \Pr\left(s^t, \mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i} \,\middle|\, \tilde{o}_i^{[1,t]}\right)$.

Example 4.4 (Probabilistic equivalence in POGrid(3))**.** *Recall that the partially observable grid problem* POGrid(3) *has a factorized state space, where agent i's local state is agent i's position in the grid. In addition, the problem is transition- and observation-independent, since actions of one agent do not affect the location of the other agent and observations of one agent do not provide any information about the location or observations of the other agent. Hence, we can use the simpler criterion for probabilistic equivalence as explained in Remark 4.2.*

Consider the agent that starts in the bottom left corner. Since agents observe in which column they are, we can call its observations col.1, col.2 *and* col.3*. Suppose that the agent always takes the action of moving right in the first three stages. Recall that this means that the agent likely moves to the right, but can also move in one of the other three directions or stay in-place. We now argue that the observation histories* (col.2, col.3, col.2) *and* (col.2, col.1, col.2) *are probabilistically equivalent. In both cases, the agent is in column 2 in stage 3. Moreover, because in both cases the column always changed, the row of the agent did not change. Hence, in both cases the agent is still in the bottom row. In particular, the belief about the state is the same for both observation histories, and for this problem this implies that the observation histories are probabilistically equivalent.*

4.3.1 Losslessness

We now prove that probabilistic equivalence clustering (Definition 4.7) is lossless, assuming that it is incremental. We show this by showing (in two steps) that two probabilistically equivalent LOHs induce the same multi-agent belief, so that we can use Corollary 4.1.

Lemma 4.2. *Let C be a clustering which is incremental at stage $t-1$. Let $\mathbf{c}^v_{\neq i}$ denote the tuple of clusters of the agents except agent i in stage v. If two LOHs of agent i induce the same distribution over pairs $(s, (\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i}))$, then they induce the same distribution over the pairs $(s, \mathbf{c}^t_{\neq i})$.*

Proof. Since the clustering is incremental at stage $t-1$, it holds that the clusters of agent j at stage t in C are unions of clusters of the form

$$\left\{o_j^{[1,t-1]} \cdot o_j^t \,\middle|\, o_j^{[1,t-1]} \in c_j^{t-1}\right\},$$

which is the cluster corresponding to the pair (c_j^{t-1}, o_j^t).

For a given cluster c_j^t, we can hence compute which pairs (c_j^{t-1}, o_j^t) cluster into c_j^t. Formally, we can define the set of these pairs as $\hat{C}_j(c_j^t) = \{(c_j^{t-1}, o_j^t) \colon \{o_j^{[1,t-1]} \cdot o_j^t \mid o_j^{[1,t-1]} \in c_j^{t-1}\} \subseteq c_j^t\}$. For a given tuple of clusters $\mathbf{c}_{\neq i}^t$, we can then define the set

$$\hat{C}_{\neq i}(\mathbf{c}_{\neq i}^t) = \{(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \colon \forall j \neq i . (c_j^{t-1}, o_j^t) \in \hat{C}_j(c_j^t)\}.$$

Then $\hat{C}_{\neq i}$ contains all pairs $(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t)$ that are clustered into $\mathbf{c}_{\neq i}^t$.

Hence, we can compute the distribution over the pairs $(s, \mathbf{c}_{\neq i}^t)$ from the distribution over the pairs $(s, (\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t))$ by aggregating over the pairs $(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \in \hat{C}_{\neq i}(\mathbf{c}_{\neq i}^t)$. To make this explicit, let $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ be two LOHs which induce the same distribution over pairs $(s, (\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t))$. Then we have $\Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, o_i^{[1,t]}\right) = \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, \tilde{o}_i^{[1,t]}\right)$. Hence, we can compute

$$\begin{aligned}
\Pr\left(s, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}\right) &= \sum_{(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \in \hat{C}_{\neq i}(\mathbf{c}_{\neq i}^t)} \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, o_i^{[1,t]}\right) \\
&= \sum_{(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \in \hat{C}_{\neq i}(\mathbf{c}_{\neq i}^t)} \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, \tilde{o}_i^{[1,t]}\right) \\
&= \Pr\left(s, \mathbf{c}_{\neq i}^t \mid \tilde{o}_i^{[1,t]}\right).
\end{aligned}$$

Hence, we conclude that the LOHs induce the same distribution over pairs $(s, \mathbf{c}_{\neq i}^t)$, as required. □

Note that Lemma 4.2 shows that probabilistically equivalent clusters induce the same distribution over pairs $(s, \mathbf{c}_{\neq i}^t)$. Next, we show that clusters that induce the same distribution over pairs $(s, \mathbf{c}_{\neq i}^t)$, also induce the same multi-agent belief.

Lemma 4.3. *Assume that the clustering is incremental. If two LOHs of agent i induce the same distribution over pairs $(s, \mathbf{c}_{\neq i}^t)$, then they induce the same multi-agent belief.*

Proof. Throughout the proof, fix a joint policy $\boldsymbol{\pi}$.[3] We first show that we can compute the multi-agent belief state corresponding to a single pair $(s, \mathbf{c}_{\neq i}^t)$. We do this per agent $j \neq i$. Let c_j^t be the current cluster of agent j and let $o_j^{[1,t]} \in c_j^t$ be any observation history in that cluster. Then the subtree policy $\gamma_j^{h-t} \colon \mathcal{O}_j^{\leq h-t-1} \to \mathcal{A}_j$ satisfies $\gamma_j^{h-t}(o_j^{[t+1,v]}) = \pi_j(o_j^{[1,v]})$ by Definition 4.4. Since the clustering is incremental, the result does not depend on which $o_j^{[1,t]} \in c_j^t$ we take. Hence, we can compute γ_j^{h-t} if we know c_j^t. This implies that we can compute the multi-agent belief state $(s, \boldsymbol{\gamma}_{\neq i}^{h-t})$ from $(s, \mathbf{c}_{\neq i}^t)$. Hence, if we have a distribution over pairs $(s, \mathbf{c}_{\neq i}^t)$, then we can compute the corresponding distribution over pairs $(s, \boldsymbol{\gamma}_{\neq i}^{h-t})$, i.e. the corresponding multi-agent belief. □

Using these two lemmas, it follows that probabilistic equivalence is lossless if it is incremental.

Lemma 4.4. *If probabilistic equivalence clustering is incremental, then it is lossless.*

Proof. Let $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ be two LOHs with $o_i^{[1,t]} \sim_p \tilde{o}_i^{[1,t]}$. By Definition 4.6, this means that the LOHs induce the same distribution over pairs $(s, (\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t))$. Lemma 4.2 and 4.3 then imply that they induce the same multi-agent belief. Finally, Corollary 4.1 then implies that the LOHs $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ can be clustered losslessly. □

[3] At the moment the clustering is computed, the algorithm has not yet specified actions beyond stage $t-1$. However, the argument holds for any (possible) joint policy $\boldsymbol{\pi}$, and the agents do have full knowledge of $\boldsymbol{\pi}$ when actually executing the policy.

4.3.2 Incrementality

Finally, we show that probabilistic equivalence clustering is incremental.

Lemma 4.5. *Probabilistic equivalence clustering is incremental.*

Proof. We use induction on the stage $t \geq 0$. Note that incrementality at stage 0 holds trivially. Now fix a stage $t \geq 1$ and assume that incrementality holds at stage $t-1$. Fix an agent i. Let $o_i^{[1,t]}$ and $\tilde{o}_i^{[1,t]}$ with $o_i^{[1,t]} \sim_p \tilde{o}_i^{[1,t]}$ be given. To show incrementality at stage t, we have to show that when extending both LOHs with the same observation $o_i^{t+1} = \tilde{o}_i^{t+1}$, we have $o_i^{[1,t+1]} \sim_p \tilde{o}_i^{[1,t+1]}$.

Let $\mathbf{a}_{\neq i}^{t+1}$ denote the joint action of all agents except agent i. We first compute

$$\begin{aligned}
&\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[1,t]}) = \sum_{s^t} \Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1}, s^t \mid o_i^{[1,t]}) \\
&\quad = \sum_{s^t} \Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[1,t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) \\
&\quad = \sum_{s^t, \mathbf{a}_{\neq i}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}_{\neq i}^{t+1}, \mathbf{c}_{\neq i}^t, o_i^{[1,t]}, s^t) \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[1,t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) \Big] \\
&\quad = \sum_{s^t, \mathbf{a}_{\neq i}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t) \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[1,t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) \Big] \\
&\quad = \sum_{s^t, \mathbf{a}_{\neq i}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t) \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) \Big].
\end{aligned}$$

In the fourth step, we replace the conditioning on $\mathbf{c}_{\neq i}^t$ and $o_i^{[1,t]}$ in $\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t)$ by a conditioning on a_i^{t+1} (together with the conditioning on $\mathbf{a}_{\neq i}^{t+1}$, this yields a conditioning on $\mathbf{a}^{t+1}$). We can add the conditioning on a_i^{t+1} since a_i^{t+1} is determined by $o_i^{[1,t]}$ (and the past policy, on which we implicitly also condition). We can then remove the conditioning on $\mathbf{c}_{\neq i}^t$ and $o_i^{[1,t]}$ due to the Markov property of Dec-POMDPs. In the final step, we remove the conditioning on $o_i^{[1,t]}$ and s^t in $\Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t)$ since $\mathbf{c}_{\neq i}^t$ already determines the actions the agents except agent i take.

Now we show that the final expression for the probability $\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[1,t]})$ given above is the same when we use $\tilde{o}_i^{[1,t]}$ instead of $o_i^{[1,t]}$, i.e. we show that

$$\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[1,t]}) = \Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid \tilde{o}_i^{[1,t]})$$

by showing that

$$\begin{aligned}
&\sum_{s^t, \mathbf{a}_{\neq i}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t) \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) \Big] \\
&\qquad = \sum_{s^t, \mathbf{a}_{\neq i}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t) \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid \tilde{o}_i^{[1,t]}) \Big].
\end{aligned}$$

For this, it suffices to prove that $\Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) = \Pr(s^t, \mathbf{c}_{\neq i}^t \mid \tilde{o}_i^{[1,t]})$. By the induction hypothesis, the clustering is incremental at stage $t-1$,which implies that we can apply Lemma 4.2. Note that $o_i^{[1,t]} \sim_p \tilde{o}_i^{[1,t]}$ implies that $\Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, o_i^{[1,t]}\right) = \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, \tilde{o}_i^{[1,t]}\right)$. Now Lemma 4.2 implies that $\Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[1,t]}) = \Pr(s^t, \mathbf{c}_{\neq i}^t \mid \tilde{o}_i^{[1,t]})$.

To finish the proof, note that

$$\Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i} \mid o_i^{[1,t+1]}) = \frac{\Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}, o_i^{t+1} \mid o_i^{[1,t]})}{\Pr(o_i^{t+1} \mid o_i^{[1,t]})}$$
$$= \frac{\Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}, o_i^{t+1} \mid o_i^{[1,t]})}{\sum_{s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}} \Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}, o_i^{t+1} \mid o_i^{[1,t]})}.$$

In this fraction, the numerator and denominator do not change when using $\tilde{o}_i^{[1,t]}$ instead of $o_i^{[1,t]}$, since $\Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}, o_i^{t+1} \mid o_i^{[1,t]}) = \Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}, o_i^{t+1} \mid \tilde{o}_i^{[1,t]})$ as shown above.

Hence, we conclude that

$$\Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i} \mid o_i^{[1,t+1]}) = \Pr(s^{t+1}, \mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i} \mid \tilde{o}_i^{[1,t+1]}).$$

for all $s^{t+1} \in \mathcal{S}$ and all tuples $\mathbf{c}^t_{\neq i}, \mathbf{o}^{t+1}_{\neq i}$.

This means that identical extensions of probabilistically equivalent clusters are probabilistically equivalent, which is what we need to show that the clustering is incremental. □

Using Lemma 4.4 and 4.5, we conclude:

Theorem 4.1. *Probabilistic equivalence clustering is lossless.*

Finally, we note that we also call probabilistic equivalence clustering *lossless clustering* (as in the literature), when it is clear from the context that this refers to this specific clustering.

4.4 Sliding Window Memory

Although lossless clustering (as introduced in Section 4.3) helps to reduce the number of (clustered) local observation histories (LOHs) on which an action needs to be specified, computing a policy still generally requires determining an action for exponentially many different observation histories. To find (good, but not necessarily optimal) policies for large horizons, we therefore restrict our search to the space of *sliding k-window memory* policies, which are policies that depend only on the last k observations, or on all observations if there are fewer than k of them. Finally, we emphasize that a best cluster policy corresponding to sliding k-window memory is not necessarily optimal.

We now introduce notation. Windows that end at stage t, start at stage $\ell_t = \max(t-k, 0)+1$. A local policy π_i has *sliding window memory*, if $o_i^{[\ell_t,t]} = \tilde{o}_i^{[\ell_t,t]}$ implies $\pi_i(o_i^{[1,t]}) = \pi_i(\tilde{o}_i^{[1,t]})$. We now define the clustering corresponding to sliding k-window memory.

Definition 4.8 (Sliding k-window memory)**.** *The clustering for sliding k-window memory consists of the partitions $C_{i,t}$ $(0 \leq t \leq h-1)$, where*

$$C_{i,t} = \left\{ \left\{ \tilde{o}_i^{[1,t]} \in \mathcal{O}_i^t \;\middle|\; \tilde{o}_i^{[\ell_t,t]} = o_i^{[\ell_t,t]} \right\} \;\middle|\; o_i^{[\ell_t,t]} \in \mathcal{O}_i^{\min\{t,k\}} \right\}.$$

We identify the equivalence class (cluster) of all LOHs that have suffix $o_i^{[\ell_t,t]}$ with exactly this suffix $o_i^{[\ell_t,t]}$. In particular, we refer to equivalence classes of sliding k-window memory as *suffixes*.

We have $|C_{i,t}| = |\mathcal{O}_i|^{\min\{t,k\}}$. Hence, the total number of clusters for a horizon h is bounded by $O(nh|\mathcal{O}_*|^k)$, where $|\mathcal{O}_*|$ is the size of the largest local observation set, which is polynomial in h and $|\mathcal{O}_*|$ for fixed k.

4.5 Lossless Clustering for Sliding Window Memory

Our next aim is to define a clustering that is lossless with respect to sliding k-window memory, but clusters more LOHs than sliding k-window memory itself. Hence, we want to define a criterion similar to the probabilistic equivalence criterion that allows us to (further) cluster suffixes without further loss in policy value. The final criterion is more complicated than the probabilistic equivalence criterion. However, we show that a few simpler and intuitive criteria do not yield a clustering that is lossless and incremental.

A first attempt would be to cluster two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ if they induce the same belief over the states and the pairs $(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t)$, i.e. if

$$\Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, o_i^{[\ell_t,t]}\right) = \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \,\middle|\, \tilde{o}_i^{[\ell_t,t]}\right) \tag{4.1}$$

for all $s^t \in \mathcal{S}$ and all tuples $\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t$. Using the DecTiger problem, we can show that this does not yield an incremental clustering.

Example 4.5. *Consider the* DecTiger *problem. We use windows of size $k = 2$. Suppose that the policy in stage 0, 1 and 2 is such that no agent opens a door. Then the LOHs* (HR, HL) *and* (HL, HR) *would be clustered according to Equation* (4.1)*, since the order of the observations does not matter when no agent opens a door inbetween. By incrementality, this means that we also cluster the LOHs* $(\text{HR}, \text{HL}, \text{HL})$ *and* $(\text{HL}, \text{HR}, \text{HL})$ *that we get by extending them with* HL*. Because we use windows of size $k = 2$, this implies that we cluster the suffixes* (HL, HL) *and* (HR, HL) *in stage 3. However, these suffixes are not equivalent according to Equation* (4.1)*.*

When using sliding window memory, the order of the observations now does matter for when the agent 'forgets' the observation, i.e. for when the observation leaves the window of k observations. This explains why the LOHs (HR, HL) and (HL, HR) from Example 4.5 should not be clustered when using sliding window memory. To account for this, we could require that the beliefs in Equation (4.1) are the same for the two suffixes $o_i^{[v,t]}$ and $\tilde{o}_i^{[v,t]}$ for any $v \in \{\ell_t, \dots, t\}$, i.e.

$$\Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \;\middle|\; o_i^{[v,t]}\right) = \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \;\middle|\; \tilde{o}_i^{[v,t]}\right) \tag{4.2}$$

for all $v \in \{\ell_t, \dots, t\}$. Intuitively, this means that the belief is also the same for the two suffixes when forgetting observations. Unfortunately, this still does not give rise to an incremental clustering. We show this by example in Appendix A.2.2.

The remainder of this section is organized as follows. First we define *weak equivalence*, which is essentially Equation (4.1), but with one change to make use of the fact that we cluster windows of size k. Then we introduce *suffix-equivalence*, which finally gives us an incremental clustering. Finally, we show that the clustering is indeed lossless and incremental.

4.5.1 Weak Equivalence

We now turn to *weak equivalence*, for which we first introduce the forget operator.

Forget operator. Recall that in probabilistic equivalence we consider distributions over pairs $(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t)$, because these pairs contains all information that the other agents will use to make decisions in future stages. In sliding k-window memory, this is also true, but when $t > k$, we in addition know that the agents 'forget' an observation, i.e., an observation leaves the window of k observations. To formalize this, we define the forget operator.

Definition 4.9 (Forget operator)**.** *Fix a clustering C up to stage $t-1$, and let $j \in \mathcal{D}$ be an agent. Let $\mathcal{C}$ be the collection of clusterings up to stage t that agree with C up to stage $t-1$, are incremental at stage $t-1$, and are coarser than sliding k-window memory. Let C^F be the finest clustering in $\mathcal{C}$. We define the* forget operator *F by defining $F(c_j^{t-1}, o_j^t)$ to be the cluster in C^F corresponding to any[4] LOH $o_j^{[1,t]}$ with $o_j^{[1,t-1]} \in c_j^{t-1}$. We write $F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t)$ for the tuple of clusters $F(c_j^{t-1}, o_j^t)$ for $j \neq i$.*

[4]Note that the incrementality guarantees that it does not matter which LOH we choose.

The following lemma (which we prove in Appendix A.2.3) ensures that the forget operator is well-defined, by showing that there is a *finest* clustering satisfying these conditions.

Lemma 4.6 (Well-definedness of forget operator)**.** *Fix a clustering C up to stage $t-1$. Then there exists a* finest *clustering C^F up to stage t agreeing with C up to stage $t-1$, which is incremental at stage $t-1$ and coarser than sliding k-window memory.*

Weak equivalence. With this in hand, we now define weak equivalence.

Definition 4.10 (Weak equivalence)**.** *Let a clustering up to stage $t-1$, a policy up to stage t and an initial belief be fixed. Two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ are* weakly equivalent*, written $o_i^{[\ell_t,t]} \sim_w \tilde{o}_i^{[\ell_t,t]}$, if*

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_t,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \tilde{o}_i^{[\ell_t,t]}\right)$$

for all $s^t \in \mathcal{S}$ and all tuples $\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t$.

Note that $\sim_w$ is an equivalence relation. The intuition behind this criterion is that this means that the suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ induce the same belief over states and the information that the other agents use to choose their actions, using that the other agents forget an observation if $t > k$. It is easier to satisfy than Equation (4.1).

4.5.2 Suffix-equivalence

As we have seen, weak equivalence does not yield an incremental clustering (the forget operator does not change this). However, we can prove incrementality if we restrict the shape of the clusters. In sliding k-window memory, all LOHs in a cluster have a common suffix $o_i^{[\ell_t,t]}$ of length k. To further cluster the LOHs, we allow clustering LOHs with a common suffix $o_i^{[m,t]}$ of any length. This length is at most k since we consider clusterings which are coarser than sliding k-window memory, but when the common suffix is shorter, we cluster more LOHs into one cluster than in sliding k-window memory. We define these clusters using the following equivalence relation, which we call *suffix-equivalence*.

Definition 4.11 (Suffix-equivalence)**.** *Consider two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ with largest common suffix $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$. They are* suffix-equivalent*, written $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$, if for all $\hat{o}_i^{[\ell_t,t]}$ with $\hat{o}_i^{[m,t]} = o_i^{[m,t]}$, we have $o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$.*

In this definition, we allow $m > t$, in which case the suffix is empty, and then we cluster all LOHs of agent i together. By definition, suffix-equivalence directly implies weak equivalence. In Appendix A.2.4, we show that suffix-equivalence is an equivalence relation.

Lemma 4.7. *Suffix-equivalence $\sim$ is an equivalence relation.*

We write $[o_i^{[\ell_t,t]}]$ for the set of LOHs that have a suffix in the equivalence class of $o_i^{[\ell_t,t]}$. Using these sets, we define the clustering corresponding to clustered sliding window memory.

Definition 4.12 (Clustered sliding window memory)**.** *The clustering corresponding to clustered sliding k-window memory consists of the sets defined by $C_{i,t} = \left\{[o_i^{[\ell_t,t]}] \;\middle|\; o_i^{[\ell_t,t]} \in \mathcal{O}_i^{\min\{t,k\}}\right\}$ for $i \in \mathcal{D}$ and $0 \leq t \leq h-1$.*

We note that clusters (as defined in Definition 4.1) contain LOHs, but we will say that a cluster contains a suffix $o_i^{[\ell_t,t]}$ if it contains all LOHs with the suffix $o_i^{[\ell_t,t]}$. The following lemma (which we prove Appendix A.2.5) shows that clusters in clustered sliding window memory correspond to a suffix (possibly of length smaller than the window size k).

Lemma 4.8. *Let c_i^t be a cluster in clustered sliding window memory. Then there exists a suffix $o_i^{[m,t]}$ such that c_i^t contains precisely the suffixes with suffix $o_i^{[m,t]}$.*

We give an example of a clustering in clustered sliding window memory.

Example 4.6 (Clustered sliding window memory)**.** *Consider a Dec-POMDP with three local observations X, Y and Z, and suppose that we use window size $k = 3$. Then a possible clustering would be to cluster all 9 suffixes whose most recent local observation is X, all 3 suffixes that end in XZ, all 3 suffixes that end in XY and keep the other 12 suffixes separate in their own clusters.*

An example of a clustering that is not possible is one where precisely the 6 suffixes that in XY or XZ are clustered, because this does not correspond to a single shorter common suffix. In fact, the suffixes ending in XY and XZ can only be in the same cluster if all 27 suffixes are clustered in one cluster (with the empty suffix as shorter common suffix).

A new insight presented in this thesis (compared to Koops *et al.* [2024]) is that Equation (4.2) holds for any two suffixes that are clustered according to the definition of suffix-equivalence.

Lemma 4.9. *Consider two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ which are suffix-equivalent, i.e. $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$. Then Equation* (4.2) *holds for these two suffixes, i.e.*

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \;\middle|\; o_i^{[v,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \;\middle|\; \tilde{o}_i^{[v,t]}\right)$$

for all $v \in \{\ell_t, \ldots, t\}$, all $s^t \in \mathcal{S}$ and all tuples $\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t$.

Proof. Let $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ be the largest common suffix of $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$. If $v \geq m$, then $o_i^{[v,t]} = \tilde{o}_i^{[v,t]}$, and hence the claim holds trivially. Now suppose that $v < m$. Since $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$, we have $o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$ for all $\hat{o}_i^{[\ell_t,t]}$ with $\hat{o}_i^{[m,t]} = o_i^{[m,t]}$, so in particular for all $\hat{o}_i^{[\ell_t,t]}$ with $\hat{o}_i^{[v,t]} = o_i^{[v,t]}$. This implies that

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \;\middle|\; o_i^{[\ell_t,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \;\middle|\; \hat{o}_i^{[\ell_t,t]}\right) \qquad (\star)$$

for all $\hat{o}_i^{[\ell_t,t]}$ with $\hat{o}_i^{[v,t]} = o_i^{[v,t]}$. Write $\hat{S} = \left\{\hat{o}_i^{[\ell_t,t]} \;\middle|\; \hat{o}_i^{[v,t]} = o_i^{[v,t]}\right\}$. Hence, we have

$$\begin{aligned}
\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t), o_i^{[v,t]}\right) &= \sum\nolimits_{\hat{o}_i^{[\ell_t,t]} \in \hat{S}} \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t), \hat{o}_i^{[\ell_t,t]}\right) \\
&= \sum\nolimits_{\hat{o}_i^{[\ell_t,t]} \in \hat{S}} \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \hat{o}_i^{[\ell_t,t]}\right) \Pr\left(\hat{o}_i^{[\ell_t,t]}\right) \\
&= \sum\nolimits_{\hat{o}_i^{[\ell_t,t]} \in \hat{S}} \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_t,t]}\right) \Pr\left(\hat{o}_i^{[\ell_t,t]}\right) \\
&= \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_t,t]}\right) \sum\nolimits_{\hat{o}_i^{[\ell_t,t]} \in \hat{S}} \Pr\left(\hat{o}_i^{[\ell_t,t]}\right) \\
&= \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_t,t]}\right) \Pr\left(o_i^{[v,t]}\right)
\end{aligned}$$

where we use in the first (and last) step that the observed suffix is $o_i^{[v,t]}$ if and only if the observed suffix is a (longer) suffix from $\hat{S}$, and in the third step we use $(\star)$.
By conditioning on $o_i^{[v,t]}$, we conclude that

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[v,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_t,t]}\right).$$

Since $\sim$ is symmetric, we also have $\tilde{o}_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$, so it follows in the same way that

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \tilde{o}_i^{[v,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \tilde{o}_i^{[\ell_t,t]}\right).$$

Finally, we also have $o_i^{[\ell_t,t]} \sim_w \tilde{o}_i^{[\ell_t,t]}$, which implies

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_t,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \tilde{o}_i^{[\ell_t,t]}\right).$$

Combining these three equalities yields

$$\Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[v,t]}\right) = \Pr\left(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \tilde{o}_i^{[v,t]}\right),$$

which is the equality that we had to prove. □

This result allows us to use a simpler definition of clustered sliding window memory compared to the definition we used in Koops *et al.* [2024], where we instead used Equation (4.2) (with the forget operator) as definition of weak equivalence (called belief-equivalence in Koops *et al.* [2024]). This simpler definition does not yield a reduction in the number of clusters (since both definitions are equivalent), but it does provide a faster way to check whether two suffixes should be clustered.

4.5.3 Losslessness

We now prove that clustering sliding window memory (Definition 4.12) is lossless, assuming that it is incremental. Similarly to Section 4.3.1, we do this in two steps. However, we can reuse the second step (Lemma 4.3) directly. We first show a statement similar to Lemma 4.2.

Lemma 4.10. *Let C be a clustering which is incremental at stage $t-1$ and coarser than sliding k-window memory clustering. Let $\mathbf{c}^v_{\neq i}$ denote the tuple of clusters of the agents except agent i in stage v. If two suffixes of agent i induce the same distribution over pairs $(s, F(\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i}))$, then they induce the same distribution over the pairs $(s, \mathbf{c}^t_{\neq i})$.*

Proof. By definition, the clusters in $F(\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i})$ are the clusters in the finest clustering C^F which is incremental at stage $t-1$ and coarser than sliding k-window memory clustering. Since C is incremental at stage $t-1$ and coarser than sliding k-window memory clustering, it follows that C^F is finer than C, so clusters in C are unions of clusters in C^F.

Hence, we can compute the distribution over the pairs $(s, \mathbf{c}^t_{\neq i})$ from the distribution over the pairs $(s, F(\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i}))$ by aggregating over all tuples of clusters in C^F that are clustered to the same cluster in C, which implies the claim. □

Using this, we can show that clustered sliding k-window memory is lossless if it is incremental.

Lemma 4.11. *If a clustering is incremental, coarser than sliding k-window memory and finer than weak equivalence, then it is lossless with respect to sliding k-window memory.*

Proof. Let C be a clustering satisfying the conditions of the lemma. Fix an agent i, a stage t, and a cluster $c^t_i \in C_{i,t}$. Let $o^{[\ell_t,t]}_i, \tilde{o}^{[\ell_t,t]}_i \in c^t_i$ be given. Since $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$ are in the same cluster and C is finer than weak equivalence, we have $o^{[\ell_t,t]}_i \sim_w \tilde{o}^{[\ell_t,t]}_i$. By the definition of weak equivalence, this implies that $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$ induce the same distribution over pairs $(s, F(\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i}))$. By Lemma 4.10 and 4.3, it follows that $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$ induce the same multi-agent beliefs. Hence, Corollary 4.1 implies that $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$ can be clustered together losslessly. □

4.5.4 Incrementality

Finally, we show that clustered sliding window memory is incremental. The overall strategy of the proof is similar to Lemma 4.5, but we need several extra steps.

Lemma 4.12. *Clustered sliding window memory is incremental.*

Proof. We use induction on the stage $t \geq 0$. Note that incrementality at stage 0 holds trivially. Now fix $t \geq 1$ and assume that incrementality holds at stage $t-1$. Fix an agent i. Let suffixes $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$ with $o^{[\ell_t,t]}_i \sim \tilde{o}^{[\ell_t,t]}_i$ be given. To show incrementality at stage t, we have to show that $o^{[\ell_{t+1},t+1]}_i \sim \tilde{o}^{[\ell_{t+1},t+1]}_i$ when extending both suffixes with the same observation $o^{t+1}_i = \tilde{o}^{t+1}_i$.

Let $o^{[m,t+1]}_i = \tilde{o}^{[m,t+1]}_i$ be the largest common suffix of $o^{[\ell_{t+1},t+1]}_i$ and $\tilde{o}^{[\ell_{t+1},t+1]}_i$. Then $m \leq t+1$ since $o^{t+1}_i = \tilde{o}^{t+1}_i$. Note that $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$ have common suffix $o^{[m,t]}_i = \tilde{o}^{[m,t]}_i$. If $m = \ell_{t+1}$, then we have $o^{[\ell_{t+1},t+1]}_i = \tilde{o}^{[\ell_{t+1},t+1]}_i$ and hence trivially $o^{[\ell_{t+1},t+1]}_i \sim \tilde{o}^{[\ell_{t+1},t+1]}_i$. From now on assume that $m > \ell_{t+1}$. Then $o^{[m,t]}_i = \tilde{o}^{[m,t]}_i$ is the largest common suffix of $o^{[\ell_t,t]}_i$ and $\tilde{o}^{[\ell_t,t]}_i$. By Lemma 4.8 this implies that all clusters $\hat{o}^{[\ell_t,t]}_i$ with $o^{[m,t]}_i = \hat{o}^{[m,t]}_i$ are clustered, i.e. $o^{[\ell_t,t]}_i \sim \hat{o}^{[\ell_t,t]}_i$ for all $\hat{o}^{[\ell_t,t]}_i$ with $o^{[m,t]}_i = \hat{o}^{[m,t]}_i$. Since $\ell_t \leq \ell_{t+1}$, Lemma 4.9 then implies that

$$\Pr\left(s^t, F(\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i}) \mid o^{[\ell_{t+1},t]}_i\right) = \Pr\left(s^t, F(\mathbf{c}^{t-1}_{\neq i}, \mathbf{o}^t_{\neq i}) \mid \hat{o}^{[\ell_{t+1},t]}_i\right) \qquad (\star)$$

for all $\hat{o}_i^{[\ell_t,t]}$ with $o_i^{[m,t]} = \hat{o}_i^{[m,t]}$.

Let $\mathbf{a}_{\neq i}^{t+1}$ denote the joint action of all agents except agent i. We first compute

$$\begin{aligned}
&\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[\ell_{t+1},t]}) = \sum_{s^t} \Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1}, s^t \mid o_i^{[\ell_{t+1},t]}) \\
&= \sum_{s^t} \Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[\ell_{t+1},t]}) \\
&= \sum_{s^t, \mathbf{a}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) \Pr(\mathbf{a}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[\ell_{t+1},t]}) \Big] \\
&= \sum_{s^t, \mathbf{a}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t) \Pr(\mathbf{a}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[\ell_{t+1},t]}) \Big]. \\
&= \sum_{s^t, \mathbf{a}^{t+1}} \Big[\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t) \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t) \Pr(a_i^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) \Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[\ell_{t+1},t]}) \Big].
\end{aligned}$$

In the fourth step, we remove the conditioning on $\mathbf{c}_{\neq i}^t$ and $o_i^{[\ell_{t+1},t]}$ in $\Pr(s^{t+1}, \mathbf{o}^{t+1} \mid \mathbf{a}^{t+1}, s^t)$ due to the Markov property of Dec-POMDPs. In the final step, we use that

$$\begin{aligned}
\Pr(\mathbf{a}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) &= \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t, a_i^{t+1}) \Pr(a_i^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) \\
&= \Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t) \Pr(a_i^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t)
\end{aligned}$$

where we remove the conditioning on $o_i^{[\ell_{t+1},t]}$, s^t, and a_i^{t+1} in $\Pr(\mathbf{a}_{\neq i}^{t+1} \mid \mathbf{c}_{\neq i}^t)$ since $\mathbf{c}_{\neq i}^t$ already completely determines the actions the agents except agent i take.

To show that $o_i^{[\ell_{t+1},t+1]} \sim \tilde{o}_i^{[\ell_{t+1},t+1]}$, we have to show that $o_i^{[\ell_{t+1},t+1]} \sim_w \hat{o}_i^{[\ell_{t+1},t+1]}$ for all $\hat{o}_i^{[\ell_{t+1},t+1]}$ such that $o_i^{[m,t+1]} = \hat{o}_i^{[m,t+1]}$. Let $\hat{o}_i^{[\ell_{t+1},t+1]}$ such that $o_i^{[m,t+1]} = \hat{o}_i^{[m,t+1]}$ be given.

Now we show that the final expression for the probability $\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[\ell_{t+1},t]})$ given above is the same when we use $\hat{o}_i^{[\ell_{t+1},t]}$ instead of $o_i^{[\ell_{t+1},t]}$. We show this for each of the four factors individually.

- The first two factors do not depend on $o_i^{[\ell_{t+1},t]}$.
- For the third factor we have to show that
$$\Pr(a_i^{t+1} \mid \mathbf{c}_{\neq i}^t, o_i^{[\ell_{t+1},t]}, s^t) = \Pr(a_i^{t+1} \mid \mathbf{c}_{\neq i}^t, \hat{o}_i^{[\ell_{t+1},t]}, s^t).$$
Recall that $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ is the largest common suffix of $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$, and that $m > \ell_{t+1}$. By Lemma 4.8, this implies that all suffixes with suffix $o_i^{[m,t]}$ are clustered together. This in turn implies that $o_i^{[m,t]}$ determines the action a_i^{t+1} that agent i takes. Since $o_i^{[m,t]} = \hat{o}_i^{[m,t]}$ (we assume that $o_i^{[m,t+1]} = \hat{o}_i^{[m,t+1]}$), this shows that the factors are equal.
- For the fourth factor, we use ($\star$), which states that
$$\Pr(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid o_i^{[\ell_{t+1},t]}) = \Pr(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t) \mid \hat{o}_i^{[\ell_{t+1},t]}).$$
By the induction hypothesis, the clustering is incremental at stage $t-1$. Hence, we can use Lemma 4.10 to go from a distribution over pairs $(s^t, F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t))$ to distribution over pairs $(s^t, \mathbf{c}_{\neq i}^t)$, which yields the required equality
$$\Pr(s^t, \mathbf{c}_{\neq i}^t \mid o_i^{[\ell_{t+1},t]}) = \Pr(s^t, \mathbf{c}_{\neq i}^t \mid \hat{o}_i^{[\ell_{t+1},t]}).$$

Finally, note that

$$\begin{aligned}
\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1} \mid o_i^{[\ell_{t+1},t+1]}) &= \frac{\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[\ell_{t+1},t]})}{\Pr(o_i^{t+1} \mid o_i^{[\ell_{t+1},t]})} \\
&= \frac{\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[\ell_{t+1},t]})}{\sum_{s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}} \Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[\ell_{t+1},t]})}.
\end{aligned}$$

In this fraction, the numerator and denominator do not change when using $\hat{o}_i^{[\ell_{t+1},t]}$ instead of $o_i^{[\ell_{t+1},t]}$, since $\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid o_i^{[\ell_{t+1},t]}) = \Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}, o_i^{t+1} \mid \hat{o}_i^{[\ell_{t+1},t]})$ as shown above. Hence, it follows that

$$\Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1} \mid o_i^{[\ell_{t+1},t+1]}) = \Pr(s^{t+1}, \mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1} \mid \hat{o}_i^{[\ell_{t+1},t+1]}).$$

We can now sum over all $\mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}$ that end up in the same cluster $F(\mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1})$. Hence, we conclude that

$$\Pr(s^{t+1}, F(\mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}) \mid o_i^{[\ell_{t+1},t+1]}) = \Pr(s^{t+1}, F(\mathbf{c}_{\neq i}^t, \mathbf{o}_{\neq i}^{t+1}) \mid \hat{o}_i^{[\ell_{t+1},t+1]}),$$

This implies that $o_i^{[\ell_{t+1},t+1]} \sim_w \hat{o}_i^{[\ell_{t+1},t+1]}$ for all $\hat{o}_i^{[\ell_{t+1},t+1]}$ such that $o_i^{[m,t+1]} = \hat{o}_i^{[m,t+1]}$, which finally implies that $o_i^{[\ell_{t+1},t+1]} \sim \tilde{o}_i^{[\ell_{t+1},t+1]}$. This means that identical extensions of suffix-equivalent clusters are suffix-equivalent, which is what we need to show that the clustering is incremental. □

Using Lemma 4.11 and 4.12, we conclude:

Theorem 4.2. *Clustered sliding k-window memory is lossless w.r.t. sliding k-window memory.*

Proof. We show that the conditions of Lemma 4.11 are satisfied. Lemma 4.12 implies that clustered sliding k-window memory is incremental. Since we cluster suffixes of length k further together, it follows that clustered sliding k-window memory is coarser than sliding k-window memory. Finally, since suffix-equivalence implies weak equivalence, clustered sliding k-window memory is finer than the clustering corresponding to weak equivalence. Hence, Lemma 4.11 implies that clustered sliding k-window memory is lossless with respect to sliding k-window memory. □

4.6 Probability-Based Clustering

Using (clustered) sliding k-window memory avoids the exponential blowup in the number of clusters that occurs in the general case for lossless clustering. Nevertheless, even for $k = 2$ or $k = 3$ the number of clusters can in practice already be too large for our search algorithm to find near-optimal policies within the space of sliding k-window memory policies, especially if little additional (lossless) clustering of k-windows is possible.

To further limit the number of clusters, and focus the search on clusters that have a higher probability, we can also cluster suffixes together to a smaller common suffix if the probability corresponding to the smaller common suffix is still smaller than some threshold $p_{\max}$.

To define this, we first introduce the probability of a suffix $\Pr(o_i^{[m,t]})$, where we implicitly condition on the initial belief and on the joint policy up to stage t. This is the probability that agent i observes the observations in $o_i^{[m,t]}$ in stage m up to stage t.

Definition 4.13 (Approximate equivalence)**.** *Let $0 < p_{\max} < 1$, and let k be a window size. Consider two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ with largest common suffix $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$. They are* approximately equivalent*, written $o_i^{[\ell_t,t]} \approx \tilde{o}_i^{[\ell_t,t]}$, if $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ or $\Pr(o_i^{[m,t]}) \leq p_{\max}$. Here $\sim$ denotes suffix-equivalence corresponding to window size k.*

Lemma 4.13. *Approximate equivalence $\approx$ is an equivalence relation.*

We prove Lemma 4.13 in Appendix A.2.6. Since approximate equivalence is an equivalence relation, its equivalence classes can be used to define a clustering.

Definition 4.14 (Probability-based clustering)**.** *The clustering corresponding to probability-based clustering with window size k and threshold $p_{\max}$ consists of the sets defined by*

$$C_{i,t} = \left\{ [o_i^{[\ell_t,t]}] \;\middle|\; o_i^{[\ell_t,t]} \in \mathcal{O}_i^{\min\{t,k\}} \right\}$$

for $i \in \mathcal{D}$ and $0 \leq t \leq h-1$, where $[o_i^{[\ell_t,t]}]$ denotes the set of LOHs that have a suffix in the equivalence class of $o_i^{[\ell_t,t]}$ of approximate equivalence with window size k and threshold $p_{\max}$.

In Appendix A.2.7, we show that clusters in probability-based clustering also correspond to a suffix, similarly to clusters in clustered sliding window memory.

Lemma 4.14. *Let c_i^t be a cluster in probability-based clustering. Then there exists a suffix $o_i^{[m,t]}$ such that c_i^t contains precisely the suffixes with suffix $o_i^{[m,t]}$.*

Next, we show that probability-based clustering is incremental.

Lemma 4.15. *Probability-based clustering is incremental.*

Proof. Let two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ with $o_i^{[\ell_t,t]} \approx \tilde{o}_i^{[\ell_t,t]}$ be given. Let $o_i^{t+1} = \tilde{o}_i^{t+1}$ be an additional observation. To show incrementality, we have to show that $o_i^{[\ell_{t+1},t+1]} \approx \tilde{o}_i^{[\ell_{t+1},t+1]}$. Let $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ be the largest common suffix of $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$. If $m \leq \ell_{t+1}$, then we have $o_i^{[\ell_{t+1},t+1]} = \tilde{o}_i^{[\ell_{t+1},t+1]}$ and hence $o_i^{[\ell_{t+1},t+1]} \approx \tilde{o}_i^{[\ell_{t+1},t+1]}$. Assume from now on that $m > \ell_{t+1}$.

Since $o_i^{[\ell_t,t]} \approx \tilde{o}_i^{[\ell_t,t]}$, we have $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ or $\Pr(o_i^{[m,t]}) \leq p_{\max}$. We distinguish these two cases. First suppose that $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$. Then Lemma 4.12 implies that $o_i^{[\ell_{t+1},t+1]} \sim \tilde{o}_i^{[\ell_{t+1},t+1]}$, which in turn implies that $o_i^{[\ell_{t+1},t+1]} \approx \tilde{o}_i^{[\ell_{t+1},t+1]}$.

Now suppose that $\Pr(o_i^{[m,t]}) \leq p_{\max}$. Since $m > \ell_{t+1}$ and $o_i^{t+1} = \tilde{o}_i^{t+1}$, it follows that $o_i^{[m,t+1]} = \tilde{o}_i^{[m,t+1]}$ is the largest common suffix of $o_i^{[\ell_{t+1},t+1]}$ and $\tilde{o}_i^{[\ell_{t+1},t+1]}$. Since $\Pr(o_i^{[m,t+1]}) \leq \Pr(o_i^{[m,t]}) \leq p_{\max}$, this implies that $o_i^{[\ell_{t+1},t+1]} \approx \tilde{o}_i^{[\ell_{t+1},t+1]}$. □

Finally, we show an upper bound on the maximum number of clusters in probability-based clustering. It should be noted that this is a worst-case bound. The number of clusters is typically much smaller.

Lemma 4.16 (Maximum number of clusters in probability-based clustering)**.** *Suppose that we use probability-based clustering with window size k and threshold $0 < p_{\max} < 1$. Then the number of clusters of agent i is upper bounded by $\frac{k}{p_{\max}}|\mathcal{O}_i|$.*

Proof. If the clustering contains only a single cluster (with the empty suffix as common suffix), then the claim holds. Otherwise, every cluster corresponds to a suffix $o_i^{[m,t]}$ with $m \leq t$. For a cluster corresponding to suffix $o_i^{[m,t]}$, call $o_i^{[m+1,t]}$ its *parent suffix*. This is a suffix of length between 0 and $k-1$. Since the suffixes with common suffix $o_i^{[m+1,t]}$ are not all clustered together, we must have $\Pr(o_i^{[m+1,t]}) > p_{\max}$. This implies that there are less than $\frac{1}{p_{\max}}$ parent suffixes of a given length, so at most $\frac{k}{p_{\max}}$ parent suffixes in total. For each parent suffix, there are at most $|\mathcal{O}_i|$ clusters. Hence, the total number of clusters is at most $\frac{k}{p_{\max}}|\mathcal{O}_i|$. □

5 Multi-Agent A*

In this section, we discuss *multi-agent A** (MAA*), which is the search algorithm that we use to search through the space of policies. In Section 5.1, we give a general introduction to A* algorithms. In Section 5.2, we discuss the MAA* algorithm for Dec-POMDPs [Szer *et al.*, 2005]. Sections 5.3 and 5.4 proceed by discussing *generalized MAA** (GMAA*) [Oliehoek *et al.*, 2008b] and its state-of-the-art extension *GMAA*-ICE* [Spaan *et al.*, 2011; Oliehoek *et al.*, 2013], respectively.

Next, we turn to our contributions. Section 5.5 discusses *small-step MAA**, our novel variant of MAA*. Finally, in Section 5.6 we discuss two optimizations to more efficiently find an optimal policy for the last stage given the policy for the earlier stages.

5.1 A* Algorithms

We start by giving an introduction to A* algorithms [Hart *et al.*, 1968; Russell and Norvig, 2020]. A* algorithms were originally developed for solving shortest-path problems, but can be applied to a variety of combinatorial optimization problems.

We introduce A* search over a rooted search tree $T = (\mathcal{V}, \mathcal{E})$ for a particular problem. The leaves of T represent a fully specified solution to the problem, while the other nodes of the tree represent partially specified solutions to the problem. Let $\mathcal{F} \subseteq \mathcal{V}$ denote the set of leaves. There is a value function $V\colon \mathcal{F} \to \mathbb{R}$ that gives the value of a solution, and the goal is to find a solution $f^* \in \arg\max_{f \in \mathcal{F}} V(f)$ maximizing the value.

To guide the search, we need a heuristic $Q\colon \mathcal{V} \to \mathbb{R}$ that assigns a heuristic value to each node in the tree. The algorithm uses a priority queue L storing nodes $v \in \mathcal{V}$ sorted in descending order by heuristic value $Q(v)$. Initially, L only contains the root of the search tree T. In each step, a node v with the highest heuristic value is removed from L. If $v \in \mathcal{F}$, i.e. if v is a leaf, then that node is returned as the final solution. Otherwise, all children of v in the search tree are added to the priority queue L. Algorithm 1 summarizes this algorithm.

Algorithm 1 A*

Input: Rooted search tree $T = (\mathcal{V}, \mathcal{E})$, heuristic $Q\colon \mathcal{V} \to \mathbb{R}$
Output: An $f \in \mathcal{F}$, where $\mathcal{F} \subseteq \mathcal{V}$ is the set of leaves of T

```
L ← PriorityQueue()                          ▷ sorted in descending order by Q-value
L.push(root(T))
while true do
  v ← L.pop()
  if v ∈ F then
    return v
  for all children v' of v in T do
    L.push(v')
```

We note that A* algorithms are presented differently in the context of finding shortest paths.

Remark 5.1. *In the context of shortest path-finding, A* is applied to a general graph $G = (\mathcal{V}, \mathcal{E})$. A single node $v_{\text{Goal}} \in \mathcal{V}$ is the final destination, rather than each leaf representing a complete solution. Also, rather than having a root, we specify a starting node.*

In addition, heuristics need to be defined differently if G is not a tree. Namely, a node $v \in \mathcal{V}$ may then be reachable in multiple ways from the starting node. As a result, a path-dependent cost function from the start to v and a heuristic providing an estimate of the remaining *distance from v to the goal are used. Since we use a (search) tree, we use a heuristic $Q\colon \mathcal{V} \to \mathbb{R}$, where $Q(v)$ is an estimate of the* total *distance from the starting node via the node v to the goal.*

Notation. Next, we introduce some notation. The set of *extensions* $E(v)$ of a node $v \in \mathcal{V}$ is the set of fully specified solutions $f \in \mathcal{F}$ that are a descendant of v. Also, let $\text{root}(T)$ denote the root of the search tree T and let v^- denote the parent of a node $v \in \mathcal{V}$.

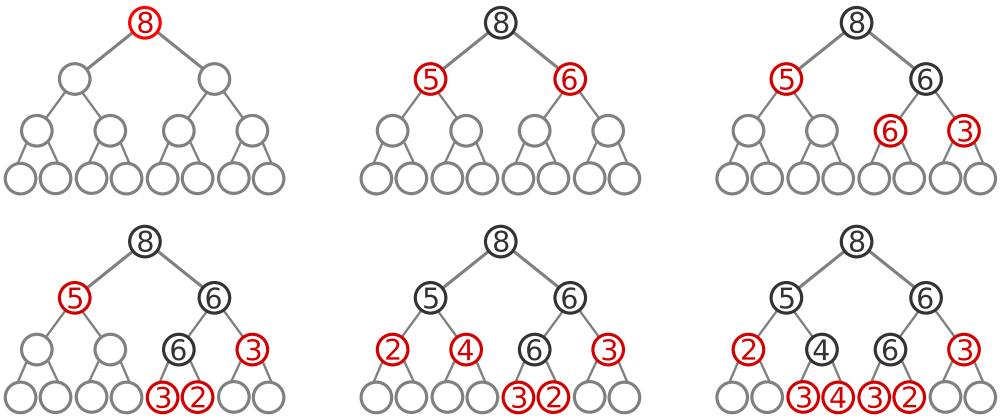


Figure 8: Example run of an A* algorithm. The numbers in the nodes denote heuristic values, the red nodes are the nodes in the priority queue, and dark gray nodes are nodes that have been expanded already.

We now turn to a key invariant of the A* algorithm.

Lemma 5.1 (Invariant)**.** *The following is an invariant of A* search (Algorithm 1): until the algorithm pops and returns a leaf, it holds that for each $f \in \mathcal{F}$, the priority queue L contains a node $v \in \mathcal{V}$ such that $f \in E(v)$, i.e. such that f is an descendant of v.*

Proof. We prove the invariant by induction on the while loop in Algorithm 1. The invariant holds initially because each leaf $f \in \mathcal{F}$ is a descendant of the root of T. Now suppose that the invariant holds at the start of the while loop. Let $v \in \mathcal{V}$ be the popped node and suppose that $v \notin \mathcal{F}$. If $f \in \mathcal{F}$ is not a descendant of the popped node v, then nothing changes. Otherwise, if $f \in \mathcal{F}$ is a descendant of v, then f is also a descendant of one of the children of v (which are added to L), since $v \neq f$. Hence, the invariant still holds after popping v and adding all children of v to L. This completes the induction and hence the proof. □

Example 5.1 (A* search)**.** *Figure 8 gives a concrete example of an A* search over a search tree T with 8 nodes. The six trees in Figure 8 show the heuristics that have been computed after each iteration of the while loop in Algorithm 1, the nodes that are in the priority queue (in red) and the nodes that have been expanded already (in dark gray). Each time, a node in the priority queue with the highest heuristic value is expanded. In the final tree, the node with the highest heuristic value 4 is a leaf, which is hence returned as final solution. This example also illustrates the invariant proven in Lemma 5.1: in each tree, it holds that each leaf is a descendant of a red node.*

5.1.1 Admissible Heuristics

The A* algorithm returns an optimal solution f^* if the heuristic Q is *admissible*, and equals the value $V(f)$ for fully specified solutions. We first define admissible heuristics:

Definition 5.1 (Admissible heuristic)**.** *A heuristic $Q\colon \mathcal{V} \to \mathbb{R}$ is called* admissible *if it is an overapproximation, i.e. $Q(v) \geq \max_{f \in E(v)} V(f)$ for all $v \in \mathcal{V}$.*

When using A* with an admissible heuristic, the resulting algorithm finds an optimal solution.

Lemma 5.2 (A* with an admissible heuristic is exact)**.** *Let $T = (\mathcal{V}, \mathcal{E})$ be a rooted search tree, let $\mathcal{F} \subseteq \mathcal{V}$ denote the set of leaves of T, and let $V\colon \mathcal{F} \to \mathbb{R}$ be a value function. If $Q\colon \mathcal{V} \to \mathbb{R}$ is an admissible heuristic and $Q(f) = V(f)$ for all $f \in \mathcal{F}$, then A* search (Algorithm 1) outputs an optimal solution $f^* \in \arg\max_{f \in \mathcal{F}} V(f)$.*

Proof. We use the invariant shown in Lemma 5.1. Let $f^* \in \mathcal{F}$ be the solution that Algorithm 1 returns and let $f \in \mathcal{F}$ be any fully specified solution. Let L be the priority queue just before f^* is popped. By the invariant, there exists a $v \in \mathcal{V}$ in L such that $f \in E(v)$. Then

$$V(f^*) = Q(f^*) \geq Q(v) \geq \max_{f' \in E(v)} V(f') \geq V(f).$$

where the first inequality holds because L is a priority queue and f^* was popped before v, and the second inequality holds because Q is admissible. Since $f \in \mathcal{F}$ was any fully specified solution, this shows that $V(f^*) \geq V(f)$ for all $f \in \mathcal{F}$, i.e. $f^* \in \arg\max_{f \in \mathcal{F}} V(f)$. □

Using a similar argument, we can also show that the highest heuristic value in the priority queue at any point is an upper bound for the optimal value.

Lemma 5.3. *Let $T = (\mathcal{V}, \mathcal{E})$ be a rooted search tree, let $\mathcal{F} \subseteq \mathcal{V}$ denote the set of leaves of T, and let $V \colon \mathcal{F} \to \mathbb{R}$ be a value function. Let $Q \colon \mathcal{V} \to \mathbb{R}$ be an admissible heuristic. Assume that the A* search (Algorithm 1) has not terminated after expanding M nodes. Let $v_M \in \mathcal{V}$ be a node with the highest heuristic value $Q(v_M)$ in the priority queue after expanding M nodes. Let $V(f^*)$ denote the value of an optimal solution. Then $Q(v_M) \geq V(f^*)$.*

Proof. We use the invariant proven in Lemma 5.1. Let $f \in \mathcal{F}$ be any fully specified solution. Let L be the priority queue after the A* has expanded M nodes. By the invariant, there exists a $v \in \mathcal{V}$ in the priority queue L such that $f \in E(v)$. Then

$$Q(v_M) \geq Q(v) \geq \max_{f' \in E(v)} V(f') \geq V(f).$$

where the first inequality holds because v_M is a node with the highest heuristic value in the priority queue, and the second inequality holds because Q is admissible. Since $f \in \mathcal{F}$ was any fully specified solution, this shows that $Q(v_M) \geq V(f)$ for all $f \in \mathcal{F}$ and hence also $Q(v_M) \geq V(f^*)$. □

Lemma 5.3 shows that A* algorithms can also be used as an anytime algorithm to compute upper bounds on the value of an optimal solution, by terminating at any point and returning the highest heuristic value in the priority queue.

5.1.2 Consistent Heuristics

We now discuss consistent heuristics. Since the value of an admissible heuristic at a node $v \in \mathcal{V}$ is an upper bound on the value of all fully specified descendants of v, and deeper nodes have fewer descendants, it is natural for the heuristic values to decrease as we go deeper in the search tree. In particular, the heuristic value of a node should be at most the heuristic value of its parent. If this holds for all nodes and also $Q(f) = V(f)$ for all $f \in \mathcal{F}$, we call the heuristic *consistent*.

Definition 5.2 (Consistent heuristic)**.** *Let $T = (\mathcal{V}, \mathcal{E})$ be a rooted search tree, let $\mathcal{F} \subseteq \mathcal{V}$ denote the set of leaves of T, and let $V \colon \mathcal{F} \to \mathbb{R}$ be a value function. A heuristic $Q \colon \mathcal{V} \to \mathbb{R}$ is called* consistent *if 1) for all nodes $v \in \mathcal{V} \setminus \{\text{root}(T)\}$, it holds that $Q(v^-) \geq Q(v)$ and 2) for all $f \in \mathcal{F}$, it holds that $Q(f) = V(f)$.*

When using a consistent heuristic, the heuristic values of the nodes expanded by the A* algorithm become smaller as the algorithm progresses. We show this in the following lemma.

Lemma 5.4. *Let $Q \colon \mathcal{V} \to \mathbb{R}$ be a consistent heuristic. Let $v_j \in \mathcal{V}$ denote the jth node expanded by A* search (Algorithm 1). Then $Q(v_j) \geq Q(v_{j+1})$. Hence, the sequence $\{Q(v_j)\}$ is descending.*

Proof. Let L_j and L_{j+1} denote the set of nodes in the queue before and after expanding v_j, respectively. If v_j is expanded, then it holds that $Q(v_j) \geq Q(v)$ for all $v \in L_j$, since A* expands a node with the highest heuristic value in the priority queue. When expanding v_j, we add all children v' of v_j to the priority queue. Since Q is consistent, we have $Q(v_j) \geq Q(v')$ for all children v'. Hence, we also have $Q(v_j) \geq Q(v)$ for all $v \in L_{j+1}$. Since the next node expanded v_{j+1} is a node from L_{j+1}, we conclude that $Q(v_j) \geq Q(v_{j+1})$. □

If a heuristic is consistent and coincides with the value function on fully specified solutions, then it is admissible. Intuitively, this holds because the heuristic value of a node is an upper bound for the heuristic value of all descendants for a consistent heuristic.

Lemma 5.5. *Let $T = (\mathcal{V}, \mathcal{E})$ be a rooted search tree. Let $Q \colon \mathcal{V} \to \mathbb{R}$ be a consistent heuristic. Then Q is admissible.*

Proof. We show that Q is admissible using induction on the search tree (starting at the leaves). For leaves, which contain a fully specified solution f, we have $Q(f) = V(f) = \max_{f' \in E(f)} V(f')$, since $E(f) = \{f\}$. Now let v be a node which is not a leaf and let $v'_1, \ldots, v'_m$ be its children. Note that $E(v) = E(v'_1) \cup \ldots \cup E(v'_m)$. Since Q is consistent, we have $Q(v) \geq Q(v'_j)$ for $1 \leq j \leq m$. Hence,

$$Q(v) \geq \max_{1 \leq j \leq m} Q(v'_j) \geq \max_{1 \leq j \leq m} \max_{f' \in E(v'_j)} V(f') = \max_{f' \in E(v'_1) \cup \ldots \cup E(v'_m)} V(f') = \max_{f' \in E(v)} V(f').$$

This completes the induction. Hence, we conclude that Q is admissible. □

Using Lemma 5.4, we can say which nodes A* expands, except nodes for which the heuristic value equals the value of the optimal solution. For these nodes it depends on how nodes are ordered in the priority queue which have the same heuristic value.

Lemma 5.6. *Let $T = (\mathcal{V}, \mathcal{E})$ be a rooted search tree. Let $Q \colon \mathcal{V} \to \mathbb{R}$ be a consistent heuristic. Let $V(f^*)$ denote the value of an optimal solution. Then A* expands all nodes $v \in \mathcal{V}$ with $Q(v) > V(f^*)$ and no nodes with $Q(v) < V(f^*)$.*

Proof. By Lemma 5.5, Q is admissible. The final node that A* pops is therefore an optimal, fully specified solution f^*, with heuristic value $Q(f^*) = V(f^*)$. Let $v \in \mathcal{V}$ be a node that is expanded. By Lemma 5.4, it then follows that $Q(v) \geq Q(f^*) = V(f^*)$. In particular, no nodes with $Q(v) < V(f^*)$ are expanded.

Conversely, consider a node with $Q(v) > V(f^*)$ and suppose that it is not expanded. Let v' be the most shallow ancestor of v' that is not expanded. Then the parent of v' is expanded (since v' is not the root), so v' is in the priority queue from that point onwards and in particular when the final node is popped. However, since Q is consistent we have $Q(v') \geq Q(v) > V(f^*)$, which means that v' would be popped rather than f^* at the end. This is a contradiction, so we conclude that all nodes v with $Q(v) > V(f^*)$ are expanded. □

If an admissible heuristic is not consistent, then we can make it consistent without losing admissibility [Mérõ, 1984]. We show this in the following lemma. We note that the lemma is not true in the context of path-finding where A* is applied to a general graph [Holte, 2010].

Lemma 5.7. *Let $Q \colon \mathcal{V} \to \mathbb{R}$ be an admissible heuristic such that $Q(f) = V(f)$ for $f \in \mathcal{F}$. Define the heuristic $Q_c \colon \mathcal{V} \to \mathbb{R}$ inductively by $Q_c(\text{root}(T)) = Q(\text{root}(T))$ and $Q_c(v) = \min\{Q_c(v^-), Q(v)\}$ if $v \neq \text{root}(T)$. Then Q_c is admissible and consistent.*

Proof. We show admissibility by induction on the tree (starting at the root). For $v = \text{root}(T)$, we have the inequality

$$Q_c(\text{root}(T)) = Q(\text{root}(T)) \geq \max_{f \in E(\text{root}(T))} V(f),$$

since Q is admissible. Now let $v \neq \text{root}(T)$ be a node and assume that admissibility holds for the parent v^- of v, i.e. $Q_c(v^-) \geq \max_{f \in E(v^-)} V(f)$. This implies that $Q_c(v^-) \geq \max_{f \in E(v)} V(f)$ since $E(v) \subseteq E(v^-)$. We also have $Q(v) \geq \max_{f \in E(v)} V(f)$ since Q is admissible. Together, this yields

$$Q_c(v) = \min\{Q_c(v^-), Q(v)\} \geq \max_{f \in E(v)} V(f),$$

which is what we need to show that Q_c is admissible.

For consistency, first note that $Q_c(v) = \min\{Q_c(v^-), Q(v)\} \leq Q_c(v^-)$ if $v \neq \text{root}(T)$. Also, we have $Q_c(f) = \min\{Q_c(f^-), Q(f)\} \leq Q(f) = V(f)$. On the other hand, $Q_c(f) \geq V(f)$ since Q_c is admissible. Hence, we conclude that $Q_c(f) = V(f)$. Hence, Q_c is consistent. □

5.2 Multi-Agent A*

We now explain multi-agent A* (MAA*) [Szer *et al.*, 2005], an exact A* algorithm for finding an optimal policy for Dec-POMDPs over a finite horizon h. In short, MAA* searches through the space of policies by incrementally fixing the actions that agents take in the individual stages. Hence, expanding a node in the MAA* search tree amounts to choosing actions for all agents and for all LOHs in a particular stage. In the context of Dec-POMDP solving, a fully specified solution is a joint policy $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$, where $\pi_i \colon \mathcal{O}_i^{\leq h-1} \to \mathcal{A}_i$ is a local policy for agent i. To define the search tree used by MAA*, we need to define partially specified policies.

Partial policies. A *local partial policy* for agent i is a partial function $\varphi_i \colon \mathcal{O}_i^{\leq h-1} \to \mathcal{A}_i$. We write $\varphi_i(\tau_i) = \bot$ if φ_i is not defined on $\tau_i \in \mathcal{O}_i^{\leq h-1}$. A *partial policy* is a tuple of local partial policies $\boldsymbol{\varphi} = \langle \varphi_1, \ldots, \varphi_n \rangle$.[5] In MAA*, we only consider partial policies for which there exists a t such that each local partial policy is defined on precisely the LOHs consisting of at most t observations, i.e. $\varphi_i(\tau_i) = \bot$ if and only if $\tau_i \notin \mathcal{O}_i^{\leq t}$. In this case, we say that the *stage* $\sigma(\boldsymbol{\varphi})$ of a partial policy $\boldsymbol{\varphi}$ is $t+1$. A partial policy $\boldsymbol{\varphi}'$ *extends* a partial policy $\boldsymbol{\varphi}$, denoted by $\boldsymbol{\varphi} <_E \boldsymbol{\varphi}'$, if $\boldsymbol{\varphi}'$ agrees with $\boldsymbol{\varphi}$ on all LOHs for which $\boldsymbol{\varphi}$ specifies the action. Formally,

$$\boldsymbol{\varphi} <_E \boldsymbol{\varphi}' \iff \forall i \in \mathcal{D}. \forall \tau_i \in \mathcal{O}_i^{\leq h-1}. \varphi_i(\tau_i) \in \{\bot, \varphi'_i(\tau_i)\}.$$

The *extensions* of a partial policy $\boldsymbol{\varphi}$ (as defined in Section 5.1) are the fully specified policies extending $\boldsymbol{\varphi}$, i.e. $E(\boldsymbol{\varphi}) = \{\boldsymbol{\pi} \in \Pi \mid \boldsymbol{\varphi} <_E \boldsymbol{\pi}\}$.

The empty policy is the partial policy which is not defined at any observation history, and we define the stage of the empty policy to be 0. The stage of fully specified policies is h.

Example 5.2. *Examples of partial policies for* DecTiger *(Example 3.2) include the empty policy,* $\langle \{\emptyset \mapsto \text{Li}\}, \{\emptyset \mapsto \text{Li}\} \rangle$ *and* $\langle \{\emptyset \mapsto \text{Li}, (\text{HL}) \mapsto \text{Li}, (\text{HR}) \mapsto \text{OL}\}, \{\emptyset \mapsto \text{Li}, (\text{HL}) \mapsto \text{Li}, (\text{HR}) \mapsto \text{OL}\} \rangle$.

MAA* search tree. We now define the MAA* search tree.

Definition 5.3 (MAA* search tree)**.** *In a MAA* search tree for a Dec-POMDP, the nodes are the partial policies for which there exists a t such that each local partial policy is defined on precisely the LOHs consisting of at most t observations, i.e. $\varphi_i(\tau_i) = \bot$ if and only if $\tau_i \notin \mathcal{O}_i^{\leq t}$. The root node is the empty policy, and the children of a partial policy $\boldsymbol{\varphi}$ of stage $\sigma(\boldsymbol{\varphi}) < h$ are exactly the partial policies $\boldsymbol{\varphi}'$ such that $\boldsymbol{\varphi} <_E \boldsymbol{\varphi}'$ and the stage of $\boldsymbol{\varphi}'$ is $\sigma(\boldsymbol{\varphi}) + 1$. The leaves of the search tree are the fully specified policies, which have stage $\sigma(\boldsymbol{\varphi}) = h$.*

The MAA* algorithm (which we also call *classical MAA** to disambiguate it from our novel variant) applies A* to the MAA* search tree. To do an A* search, we also have to specify a heuristic. To ensure that MAA* finds an optimal policy, we have to use an admissible heuristic.

Let Φ_M denote the set of partial policies in the MAA* search tree. A heuristic $Q \colon \Phi_M \to \mathbb{R}$ is admissible if $Q(\boldsymbol{\varphi}) \geq \max_{\boldsymbol{\pi} \in E(\boldsymbol{\varphi})} V_{\boldsymbol{\pi}}(b, h)$. We discuss specific choices for the heuristic in Section 6.

Analysing the MAA* search tree. The height of the MAA* search tree is $h+1$. The nodes at depth t in the tree are precisely the partial policies $\boldsymbol{\varphi}$ such that $\sigma(\boldsymbol{\varphi}) = t$. To analyse the outdegree of the search tree, assume for simplicity that all local action and observation sets are equally large, and let $|\mathcal{A}_*|$ and $|\mathcal{O}_*|$ denote their sizes, respectively. MAA* constructs a stage-$(t+1)$ policy in one step from a stage-t policy, specifying actions for all $n|\mathcal{O}_*|^t$ length-t LOHs for all n agents. As a result, the outdegree of a node at depth t is $|\mathcal{A}_*|^{n|\mathcal{O}_*|^t}$, which is double exponential in t. Since MAA* expands at least one node at depth $h-1$, it follows that the running time of MAA* is at least $\Omega\big(|\mathcal{A}_*|^{n|\mathcal{O}_*|^{h-1}}\big)$. The total number of joint policies is upper bounded by $O\big(|\mathcal{A}_*|^{n|\mathcal{O}_*|^h}\big)$ if $|\mathcal{O}_*| \geq 2$. As Seuken and Zilberstein [2008] note, this implies that classical MAA* can at best solve Dec-POMDPs of one horizon larger than a brute force search which iterates over all policies.

[5] In the Section 5.5, we introduce a restriction on the sets of LOHs on which these partial policies are defined.

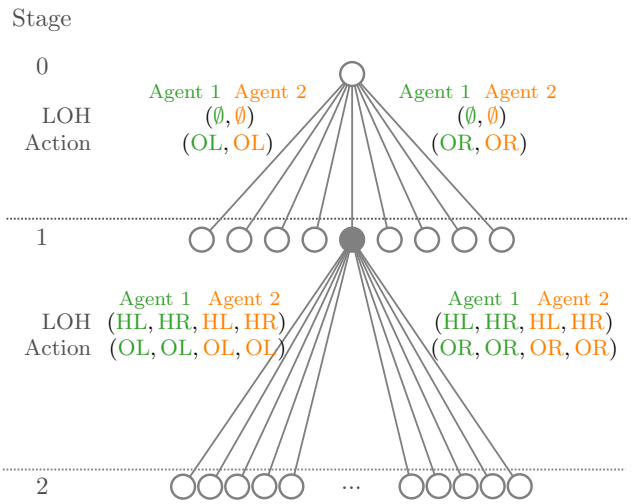


Figure 9: A partial MAA* search tree for the DecTiger problem (Example 3.2), where we expanded the root in stage 0 but only the filled gray node in stage 1. The edges are labelled with the actions specified for each agent. For example, the leftmost edge in stage 0 corresponds to assigning the action OL to the empty LOH $\emptyset$ for both agent 1 and 2.

Example. We now give an example of an MAA* search tree, using DecTiger (Example 3.2).

Example 5.3. *Figure 9 gives a partial MAA* search tree for the* DecTiger *problem. The outdegree of the root is* $|\mathcal{A}_*|^n = 3^2 = 9$*: there are three choices (*OL, Li *or* OR*) for the action that the first agent takes in stage 0 of the Dec-POMDP (corresponding to the empty observation history* $\emptyset$*) and similarly three choices for the action of the second agent.*

*In stage 1 of the Dec-POMDP, the agents each have one local observation (*HL *or* HR*). In Figure 9, we expand one node (the filled gray node) in stage 1, which corresponds to the partial policy* $\langle\{\emptyset \mapsto \text{Li}\}, \{\emptyset \mapsto \text{Li}\}\rangle$*. The outdegree of this node is* $|\mathcal{A}_*|^{n|\mathcal{O}_*|} = 3^4 = 81$*. Namely, we have to specify the action of agent 1 after observing* HL*, the action of agent 1 after observing* HR*, the action of agent 2 after observing* HL *and the action of agent 2 after observing* HR*. For each of these, there are three choices.*
The bottom left node in Figure 9 corresponds to the partial policy

$$\langle\{\emptyset \mapsto \text{Li}, (\text{HL}) \mapsto \text{OL}, (\text{HR}) \mapsto \text{OL}\}, \{\emptyset \mapsto \text{Li}, (\text{HL}) \mapsto \text{OL}, (\text{HR}) \mapsto \text{OL}\}\rangle.$$

If $h \geq 3$*, then the outdegree of this node is* $|\mathcal{A}_*|^{n|\mathcal{O}_*|^2} = 3^8 = 6\,561$*. Namely, we have to specify an action for both agents for each of the* $|\mathcal{O}_*|^2 = 4$ *LOHs of length 2. Similarly, if* $h \geq 4$*, then the outdegree of this node is* $|\mathcal{A}_*|^{n|\mathcal{O}_*|^3} = 3^{16} = 43\,046\,721$*. Finally, for* $h \geq 5$*, we would have to expand at least one node with outdegree* $|\mathcal{A}_*|^{n|\mathcal{O}_*|^4} = 3^{32} > 10^{15}$*. Hence, we see that it is infeasible to solve* DecTiger *using (classical) MAA* for* $h \geq 5$*.*

5.3 Generalized MAA*

We proceed by discussing generalized MAA* (GMAA*) [Oliehoek *et al.*, 2008b]. GMAA* generalizes MAA* by pinpointing and generalizing two operators that are used implicitly in MAA*, namely the `Select` operator and the `Next` operator. GMAA* uses a *policy pool* P to store policies, which is simply a set of policies. Algorithm 2 gives pseudocode for the GMAA* algorithm. We now discuss the `Select` and `Next` operators, and possible choices for these operators.

The `Select` operator chooses which partial policy from the policy pool to expand. In MAA*, `Select` chooses a partial policy with the highest heuristic value from the policy pool (and implements the policy pool as a priority queue to do this efficiently). Alternatives would be, for example, depth-first search (expanding a deepest node in the search tree) or breadth-first search (expanding a shallowest node in the search tree). However, to the best knowledge of the author, in practice all MAA*-variants use the same `Select` operator as MAA*.

Algorithm 2 GMAA*

Input: Dec-POMDP $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{O}}, T, O, R \rangle$, initial belief b, horizon h
Output: A joint policy $\boldsymbol{\pi}^*$ for the Dec-POMDP for initial belief b and horizon h and its value v^*

```
P ← {empty policy}                          ▷ policy pool, initialized with empty policy
v* ← −∞, π* ← None                          ▷ keep track of (value of) best policy found
while P ≠ ∅ do
  φ ← Select(P)
  Φ_Next ← Next(φ)
  if Φ_Next ∩ Π ≠ ∅ then                    ▷ If Φ_Next contains fully specified policies,
    π' ← arg max_{Φ_Next ∩ Π} V_π'(b, h)    ▷ find the best fully specified policy in Φ_Next
    if V_π'(b, h) > v* then                 ▷ new best policy found
      v* ← V_π'(b, h), π* ← π'
      P ← {φ' ∈ P | Q(φ') > v*}             ▷ prune policy pool
  P ← (P \ {φ}) ∪ (Φ_Next \ Π)              ▷ add partial policies to the pool, and remove φ
return π*, v*
```

The `Next` operator specifies which partial policies are added to the policy pool after expanding a partial policy $\boldsymbol{\varphi}$. In MAA*, when expanding a partial policy such that $\sigma(\boldsymbol{\varphi}) = t$, $\texttt{Next}(\boldsymbol{\varphi})$ is the set of all $|\mathcal{A}_*|^{n|\mathcal{O}_*|^t}$ children of the policy. Different operators for `Next` have been considered, giving up optimality. Emery-Montemerlo *et al.* [2004] propose *forward-sweep policy computation*, for which $\texttt{Next}(\boldsymbol{\varphi})$ only contains a child with the highest heuristic value. Oliehoek *et al.* [2008b] propose k-GMAA*, for which $\texttt{Next}(\boldsymbol{\varphi})$ contains k children with the highest heuristic values.

5.4 GMAA*-ICE

We now introduce the state-of-the-art exact solver for Dec-POMDPs, GMAA* with incremental clustering and expansion (GMAA*-ICE) [Spaan *et al.*, 2011; Oliehoek *et al.*, 2013]. GMAA*-ICE builds on GMAA*-IC, which applies lossless clustering [Oliehoek *et al.*, 2009] (see Section 4.3) to MAA*. GMAA*-IC can be seen as an instance of GMAA*, where the $\texttt{Next}(\boldsymbol{\varphi})$ is the set of all (flat) partial policies which correspond to a cluster policy and which are a child of $\boldsymbol{\varphi}$.

GMAA*-ICE adds incremental expansion to this, which we explain now. The main idea of incremental expansion is to add children of a node $\boldsymbol{\varphi}$ incrementally, i.e., one by one, in order of heuristic value. Within the A* search performed by GMAA*-ICE, this is done by storing *placeholder* nodes in the policy pool. The value of a placeholder node is equal to the heuristic value of its last child that was added to the pool. Each time the placeholder node is selected for expansion, a node with the next highest heuristic value is added to the policy pool and the heuristic value of the placeholder is updated accordingly.

It remains to explain how the children of a node $\boldsymbol{\varphi}$ can be generated incrementally in order of heuristic value. This is done using a *collaborative Bayesian game* (CBG), which we define in the context of Dec-POMDP solving.

Definition 5.4 (Collaborative Bayesian game)**.** *A* collaborative Bayesian game *(*CBG*) corresponding to a Dec-POMDP $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{O}}, T, O, R \rangle$, an initial belief b, a horizon h, and a partial policy $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) = t$, is a tuple $\langle \mathcal{D}, \boldsymbol{\mathcal{A}}, \boldsymbol{\Theta}, \Pr, Q \rangle$, where*

- $\boldsymbol{\Theta} = \bigtimes_{i \in \mathcal{D}} \Theta_i$ *is a set of* joint types $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_n \rangle$ *specifying a type for each agent, where each $\Theta_i = C_{i,t}$ is a set of local types θ_i corresponding to the clusters of agent i in stage t,*
- $\Pr \in \Delta(\boldsymbol{\Theta})$ *is a probability distribution over joint types, where $\Pr(\boldsymbol{\theta})$ is the probability of the joint cluster corresponding to $\boldsymbol{\theta}$, conditioned on the initial belief b and the policy $\boldsymbol{\varphi}$ up to stage $t-1$,*
- $Q\colon \boldsymbol{\Theta} \times \boldsymbol{\mathcal{A}} \to \mathbb{R}$ *is a heuristic function, which provides a heuristic for the value of executing joint action $\mathbf{a} \in \boldsymbol{\mathcal{A}}$ for a joint type $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ over horizon $h - t$.*

The CBG models the choice of actions taken in stage t of the Dec-POMDP.

The CBG can now be solved using the Bayesian game branch and bound (BaGaBaB) algorithm [Oliehoek *et al.*, 2010]. The BaGaBaB algorithm itself also uses A* search. Hence, when used inside GMAA*-ICE, BaGaBaB uses a second, nested A* search. In each step of the A* search corresponding BaGaBaB, a joint action for one joint type $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is chosen, in such a way that the chosen action is consistent with other joint actions chosen already. This means that if $\theta_i = \widetilde{\theta}_i$, then agent i needs to take the same local action for the joint types $\boldsymbol{\theta}$ and $\widetilde{\boldsymbol{\theta}}$. The outdegree of nodes in the search tree used by BaGaBaB is therefore $|\boldsymbol{\mathcal{A}}|$ if all local types in $\boldsymbol{\theta}$ have not been seen before, but smaller otherwise.

When using A* with an admissible heuristic, it is guaranteed that the first ℓ fully specified solutions found are indeed ℓ solutions with the highest heuristic value. Hence, the BaGaBaB algorithm can indeed be used to generate the children of a node $\boldsymbol{\varphi}$ incrementally in order of heuristic value, as required by GMAA*-ICE.

5.5 Small-Step MAA*

We now turn to small-step multi-agent A*, our variant of MAA* that uses a novel search tree with a constant outdegree rather than a double exponential outdegree. This is achieved by only fixing one additional action of the policy each time. We introduce small-step MAA* explicitly using clusters and cluster policies, as introduced in Section 4.1.

Expansion order. Small-step MAA* explores clusters in a fixed order: first by stage, then by agent, and then according to a given order on the clusters. We formalize this as follows.

Definition 5.5 (Expansion order)**.** *Let a total order $\preceq_{i,t}$ on $C_{i,t}$ for $i \in \mathcal{D}$ and $0 \leq t \leq h-1$ be given. We define an order $\preceq$, called the* expansion order*, on the set of all clusters $\bigcup_{i \in \mathcal{D}} C_i$ by $c_i^t \preceq \tilde{c}_j^{t'}$ if and only if*

- *$t < t'$, or*
- *$t = t'$ and $i < j$, or*
- *$t = t'$ and $i = j$ and $c_i^t \preceq_{i,t} \tilde{c}_j^{t'}$.*

Partial cluster policies. We now define partial cluster policies, which we from now on simply call partial policies. A *local partial policy* is a partial function $\varphi_i \colon C_i \to \mathcal{A}_i$. We write $\varphi_i(c_i) = \bot$ if $\varphi_i(c_i)$ is not defined at c_i. A *partial policy* is a tuple $\boldsymbol{\varphi} = \langle \varphi_1, \ldots, \varphi_n \rangle$ such that the local partial policies φ_i are defined on precisely agent i's clusters among the first d clusters in the expansion order for some d. In contrast to MAA*, it is therefore possible that a local partial policy is defined on some, but not all clusters of a given length.

The *stage* $\sigma(\boldsymbol{\varphi})$ *of* $\boldsymbol{\varphi}$ is u if $\boldsymbol{\varphi}$ is defined on all clusters of length $u-1$, but not on all clusters of length u. A partial policy $\boldsymbol{\varphi}'$ *extends* a partial policy $\boldsymbol{\varphi}$, written $\boldsymbol{\varphi} <_E \boldsymbol{\varphi}'$, if $\boldsymbol{\varphi}'$ agrees with $\boldsymbol{\varphi}$ on all clusters on which $\boldsymbol{\varphi}$ is defined, formally:

$$\boldsymbol{\varphi} <_E \boldsymbol{\varphi}' \iff \forall i \in \mathcal{D}. \forall c_i \in C_i. \varphi_i(c_i) \in \{\bot, \varphi_i'(c_i)\}.$$

Small-step search tree. Using the partial cluster policies, we define the small-step search tree:

Definition 5.6 (Small-step search tree)**.** *The* small-step search tree *for a Dec-POMDP is a tree whose nodes are the partial policies, the root node is the empty policy, and the children of a partial policy $\boldsymbol{\varphi}$ are exactly the partial policies $\boldsymbol{\varphi}'$ such that (1) $\boldsymbol{\varphi} <_E \boldsymbol{\varphi}'$ and (2) $\boldsymbol{\varphi}'$ is defined on one additional LOH compared to $\boldsymbol{\varphi}$.*

Small-step MAA* applies A* to the small-step search tree. Let Φ denote the set of partial policies. A heuristic $Q \colon \Phi \to \mathbb{R}$ is admissible if $Q(\boldsymbol{\varphi}) \geq \max_{\boldsymbol{\pi} \in E(\boldsymbol{\varphi})} V_{\boldsymbol{\pi}}(b, h)$.

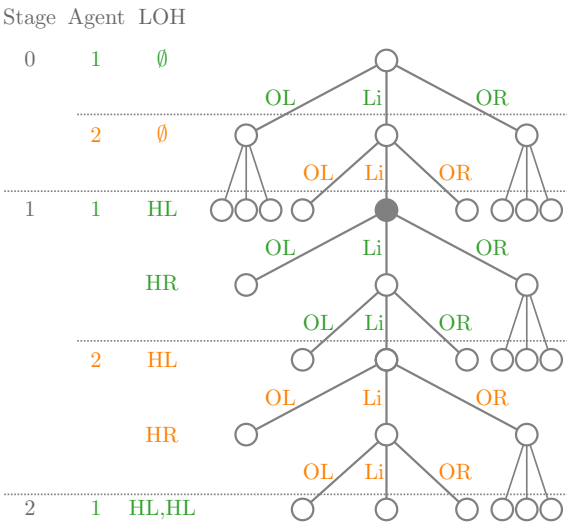


Figure 10: A partial tree of the DecTiger problem at stage 1, where we expanded all nodes in stage 0 but only a subset of the nodes in stage 1 after expanding the gray node (filled).

Example. To give an example and to compare small-step MAA* to classical MAA* as explained in Section 5.2, we apply small-step MAA* to the trivial clustering C, i.e. the clustering that assigns every LOH to its own cluster. Hence, we can identify $C_{i,t}$ with $|\mathcal{O}_i|^t$. To fix an expansion order, we need to define an order on $C_{i,t}$. We do this by ordering the LOHs of agent i and stage t lexicographically according to a given order on the observations. We now apply small-step MAA* with the trivial clustering to DecTiger (Example 3.2).

Example 5.4. *Figure 10 gives a partial small-step search tree for the* DecTiger *problem, where we use the order* HR < HL *on the observations. The outdegree of each node is* $|\mathcal{A}_*| = 3$*, corresponding to the choice of an action by specific agent for a specific LOH. The filled gray node in stage 1 corresponds to the partial policy* $\langle\{\emptyset \mapsto \text{Li}\}, \{\emptyset \mapsto \text{Li}\}\rangle$*. Expanding this node corresponds to specifying an action for the LOH* (HL) *of agent 1.*

In this tree, we can extract the policy of a node by backtracking. For example, the policy in the bottom-left node is $\langle\{\emptyset \mapsto \text{Li}, \text{HL} \mapsto \text{Li}, \text{HR} \mapsto \text{Li}\}, \{\emptyset \mapsto \text{Li}, \text{HL} \mapsto \text{Li}, \text{HR} \mapsto \text{OL}\}\rangle$.

Comparison to classical MAA*. Classical MAA*, as introduced in Section 5.2, constructs a stage-$(t+1)$ policy in one step from a stage-t policy, specifying actions for all length-t LOHs for all n agents. As a result, each stage-t policy has $O\big(|\mathcal{A}_*|^{n|\mathcal{O}_*|^t}\big)$ children in the A* search tree, where $|\mathcal{A}_*|$ and $|\mathcal{O}_*|$ denote the sizes of the largest local action and observation sets, respectively. In small-step MAA*, this is split up in $n|\mathcal{O}_*|^t$ levels, where each node has up to $|\mathcal{A}_*|$ children and just adds an action for one agent for one particular LOH.

Example 5.5. *In the tree from Figure 10, small-step MAA* reaches only 18 nodes below the gray node (12 nodes in stage 1 and 6 nodes in stage 2) by expanding only 6 nodes below the gray node in stage 1. By contrast, MAA* reaches 81 nodes at stage 2 after expanding the gray node.*

Compared to MAA*, our search tree limits the outdegree of the nodes from double exponential (w.r.t. the stage t) to constant (w.r.t. the stage t), at the cost of increasing the tree height from linear to exponential. In the best case, the heuristic is such that every node that does not have an optimal descendant, has a heuristic value that is smaller than the optimal value. In that case, small-step A* can find an optimal solution in time which is exponential in the horizon (or polynomial in the number of LOHs/clusters).

On the other hand, suppose that we use classical MAA* and small-step MAA* with the same consistent heuristic Q (which also satisfies $Q(\boldsymbol{\pi}) = V(\boldsymbol{\pi})$ for fully specified policies $\boldsymbol{\pi}$). Then Lemma 5.6 implies that any partial policy $\boldsymbol{\varphi}$ with $Q(\boldsymbol{\varphi}) > V(\boldsymbol{\pi}^*)$ which is in the MAA* tree,

is expanded by both MAA* and small-step MAA*. In particular, small-step MAA* expands at least as many policies as classical MAA*, but small-step MAA* may avoid reaching (doubly exponentially) many nodes with a heuristic value smaller than the optimal value.

Comparison to GMAA*-ICE. We now compare small-step MAA* to GMAA*-ICE. Both small-step MAA* to GMAA*-ICE use clustering, which reduces the number of clusters from $n|\mathcal{O}_*|^t$ to $\sum_{i\in\mathcal{D}}|C_{i,t}|$. However, for most problems the number of clusters $|C_{i,t}|$ still grows exponentially in t. Hence, the outdegree of a MAA*-like search tree for cluster policies, which is $|\mathcal{A}_*|^{\sum_{i\in\mathcal{D}}|C_{i,t}|}$, is in general still double exponential in t. Both small-step MAA* and GMAA*-ICE aim to avoid this double exponential blowup, but do this in a different way.

As a first approximation, one can argue that small-step MAA* flattens the nested A* searches that BaGaBaB does inside GMAA*-ICE into a single A* search. However, there are also important differences. Firstly, the A* search that BaGaBaB does assigns actions for each agent for a single joint type/cluster. In contrast, small-step MAA* assigns actions for the different agents successively. This also implies that the search tree used by BaGaBaB has an outdegree up to $|\boldsymbol{\mathcal{A}}|$, while the small-step search tree always has an outdegree of $|\mathcal{A}_i|$ for some $i \in \mathcal{D}$. Secondly, BaGaBaB already generates a child for which all actions for an additional stage are assigned the first time a node is expanded, while small-step MAA* just assigns one action in that stage.

5.6 Last Stage Optimization

In the last stage of a Dec-POMDP, choosing an optimal action is a bit easier, since agents only have to reason about the immediate reward they receive, rather than also reasoning about the effect of the actions on the next state of the system. Nevertheless, agents still have to take into account that each local action has an effect on multiple joint clusters.

To simplify notation, write $h' = h - 1$. We write $\delta_i \colon C_{i,h'} \to \mathcal{A}_i$ for the cluster policy that agent i uses in the last stage. Such a policy for a specific stage is called a *decision rule* [Oliehoek *et al.*, 2008b]. The maximization problem corresponding to the last stage can then be written as

$$\max_{\substack{\delta_i \colon C_{i,h'}\to\mathcal{A}_i \\ 1\le i\le n}} \sum_{c_1^{h'}\in C_{1,h-1}} \cdots \sum_{c_n^{h'}\in C_{n,h'}} \left[\sum_{s^{h'}\in\mathcal{S}} \Pr(\mathbf{c}^{h'}, s^{h'}) R(s^{h'}, \boldsymbol{\delta}(\mathbf{c}^{h'})) \right],$$

where $\boldsymbol{\delta}(\mathbf{c}^{h'}) = \langle \delta_1(c_1^{h'}), \ldots, \delta_n(c_n^{h'}) \rangle$. We now explain how we can solve this maximization problem more efficiently than searching over all δ_i using small-step MAA*.

5.6.1 Last Stage of the Last Agent

When using small-step MAA*, we assign actions for the first $n-1$ agents before assigning any actions for the last agent. Hence, at some point a partial policy $\boldsymbol{\varphi}$ has specified all actions for the first $n-1$ agents, but no actions for the last agent. Let $\delta_i \colon C_{i,h'} \to \mathcal{A}_i$ be the decision rules that $\boldsymbol{\varphi}$ specifies for $1 \le i \le n-1$. Hence, it remains to maximize over the decision rule $\delta_n \colon C_{n,h'} \to \mathcal{A}_n$. We now show that for the last agent, we can maximize over the points $\delta_n(c_n^{h'})$ independently. For this, we rewrite the maximization problem as follows:

$$\begin{aligned}
&\max_{\delta_n \colon C_{n,h'}\to\mathcal{A}_n} \sum_{c_1^{h'}\in C_{1,h'}} \cdots \sum_{c_{n-1}^{h'}\in C_{n-1,h'}} \sum_{c_n^{h'}\in C_{n,h'}} \left[\sum_{s^{h'}\in\mathcal{S}} \Pr(\mathbf{c}^{h'}, s^{h'}) R(s^{h'}, \boldsymbol{\delta}(\mathbf{c}^{h'})) \right] \\
&\quad = \max_{\delta_n \colon C_{n,h'}\to\mathcal{A}_n} \sum_{c_n^{h'}\in C_{n,h'}} \left[\sum_{c_1^{h'}\in C_{1,h'}} \cdots \sum_{c_{n-1}^{h'}\in C_{n-1,h'}} \left[\sum_{s^{h'}\in\mathcal{S}} \Pr(\mathbf{c}^{h'}, s^{h'}) R(s^{h'}, \boldsymbol{\delta}(\mathbf{c}^{h'})) \right] \right] \\
&\quad = \sum_{c_n^{h'}\in C_{n,h'}} \max_{\delta_n(c_n^{h'})\in\mathcal{A}_n} \left[\sum_{c_1^{h'}\in C_{1,h'}} \cdots \sum_{c_{n-1}^{h'}\in C_{n-1,h'}} \left[\sum_{s^{h'}\in\mathcal{S}} \Pr(\mathbf{c}^{h'}, s^{h'}) R(s^{h'}, \boldsymbol{\delta}(\mathbf{c}^{h'})) \right] \right].
\end{aligned}$$

In the first step, we only interchange the order of the summations. In the second step, we then note that the objective is a sum of terms, each of which only depends on a single action $\delta_n(c_n^{h'}) \in \mathcal{A}_n$ of agent n. Hence, we can maximize over each $\delta_n(c_n^{h'})$ independently inside the outer summation. This optimization is by default turned on in our algorithm.

5.6.2 Last Stage of Other Agents

For the other agents, we cannot decouple the objective as in Section 5.6.1. However, in some cases, it is still possible to show that a particular action cannot be optimal, by showing that it is suboptimal whatever the other agents do. This is in particular useful for clusters that have a small probability, since in this case the difference in overall heuristic value for the optimal action and a clearly suboptimal action can still be very small. To show that an action is suboptimal, we define the regret of choosing $\widetilde{a}_j \in \mathcal{A}_j$ rather than action $a_j \in \mathcal{A}_j$.

Definition 5.7 (Regret)**.** *Consider an agent $j \in \mathcal{D}$ and assume that actions for the first $j-1$ agents have been specified. Write $\mathbf{c}_{<j}^{h'}$ and $\mathbf{c}_{>j}^{h'}$ for the tuples $\langle c_1^{h'}, \ldots, c_{j-1}^{h'} \rangle$ and $\langle c_{j+1}^{h'}, \ldots, c_n^{h'} \rangle$, respectively, and $\mathbf{a}_{>j}$ for a tuple $\langle a_{j+1}, \ldots, a_n \rangle$ of actions of the last $n-j$ agents. The* regret *of choosing action $\widetilde{a}_j \in \mathcal{A}_j$ rather than action $a_j \in \mathcal{A}_j$ on the cluster $c_j^{h'} \in C_{j,h'}$ is then given by*

$$\begin{aligned} &\mathit{regret}(c_j^{h'}, a_j, \widetilde{a}_j) \\ &\quad = \sum_{\mathbf{c}_{>j}^{h'}} \max_{\mathbf{a}_{>j}} \left[\sum_{\mathbf{c}_{<j}^{h'}} \sum_{s^{h'} \in \mathcal{S}} \Pr(\mathbf{c}^{h'}, s^{h'}) \Big[R\big(s^{h'}, \boldsymbol{\delta}(\mathbf{c}_{<j}^{h'}, a_j, \mathbf{a}_{>j})\big) - R\big(s^{h'}, \boldsymbol{\delta}(\mathbf{c}_{<j}^{h'}, \widetilde{a}_j, \mathbf{a}_{>j})\big) \Big] \right], \end{aligned}$$

where $\boldsymbol{\delta}(\mathbf{c}_{<j}^{h'}, a_j, \mathbf{a}_{>j}) = \langle \delta_1(c_1^{h'}), \ldots, \delta_{j-1}(c_{j-1}^{h'}), a_j, a_{j+1}, \ldots, a_n \rangle$.

The regret is (an upper bound for) the largest difference in reward that can occur when switching from action $\widetilde{a}_j \in \mathcal{A}_j$ to action $a_j \in \mathcal{A}_j$. Namely, we consider all actions of the last $n-j$ agents, and take those actions that yield the largest difference. If $\mathit{regret}(c_j^{h'}, a_j, \widetilde{a}_j) < 0$, then the action $\widetilde{a}_j$ *strictly dominates* the action $a_j \in \mathcal{A}_j$, i.e., $\widetilde{a}_j$ always gives a better reward. In that case, we do not have to consider taking the action a_j. If $\mathit{regret}(c_j^{h'}, a_j, \widetilde{a}_j) = 0$, then the action $\widetilde{a}_j$ *weakly dominates* the action $a_j \in \mathcal{A}_j$, which means that we do not need to consider a_j provided that we do consider at least one action $\hat{a}_j$ such that $\mathit{regret}(c_j^{h'}, a_j, \hat{a}_j) \geq 0$.

In practice, the computations required for computing the regret can be relatively expensive and if the best action to take strongly depends on the actions of other agents, then typically no actions are dominated. Hence, computing the regrets is switched off by default in our algorithms.

6 Heuristics

Having set up the general framework of A* search for Dec-POMDPs in Section 5, we now turn to the (admissible) heuristics used to guide the search. In Section 6.1, we summarize three heuristics from the literature. In Section 6.2, we explain how these heuristics can be used in the small-step search tree. Section 6.3 provides a general framework for defining admissible heuristics. Finally, in Section 6.4 we show that the BG heuristic is admissible.

6.1 Precomputed Heuristics for MAA*

We start by defining three admissible heuristics $Q\colon \Phi_M \to \mathbb{R}$ from the literature that can be used for classical MAA*. These heuristics have in common that they require a precomputation to be used. We defer the proofs that these heuristics are admissible to Section 6.3, to illustrate our general framework for admissible heuristics.

6.1.1 MDP Heuristic

We first consider the MDP heuristic Q_{MDP} [Emery-Montemerlo *et al.*, 2004]. This heuristic was first proposed in the context of POMDPs [Littman *et al.*, 1995]. Intuitively, $Q_{\mathsf{MDP}}(\boldsymbol{\varphi})$ represents the value that can be achieved by an extension of $\boldsymbol{\varphi}$ if the agents could observe the full state for the remaining stages on which $\boldsymbol{\varphi}$ is not defined yet. Equivalently, we can say that the agents can observe the full state in all stages, since the policy in the first stages is already fixed and can hence not retroactively make use of the state information.

To compute the MDP heuristic, we precompute the optimal value $V_{\mathsf{MDP}}(s,t)$ of the underlying MDP for each initial state $s \in \mathcal{S}$ and each horizon $1 \leq t \leq h$, as explained in Section 2.1. Consider a partial policy $\boldsymbol{\varphi}$ of stage $\sigma(\boldsymbol{\varphi}) = t$. In classical MAA*, this means that $\boldsymbol{\varphi}$ is defined on all LOHs of length at most $t-1$, but not on any LOH of length t or larger. Since this specifies the policy up to stage t, we can compute the realized reward $V_{\boldsymbol{\varphi}}(b,t)$ up to stage t and probabilities $\Pr(\boldsymbol{\tau}, s^t)$ for $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t$ as explained in Section 3.2. Then we define $Q_{\mathsf{MDP}}(\boldsymbol{\varphi})$ as

$$Q_{\mathsf{MDP}}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b^0,t) + \sum_{\boldsymbol{\tau}\in\boldsymbol{\mathcal{O}}^t} \sum_{s^t\in\mathcal{S}} \Pr(\boldsymbol{\tau}, s^t) V_{\mathsf{MDP}}(s^t, h-t),$$

where $b^0 \in \Delta(\mathcal{S})$ is the initial belief. In Corollary 6.1, we show that Q_{MDP} is admissible. To use the MDP heuristic, we need to precompute $V_{\mathsf{MDP}}(s,t)$, which can be done in time $O(|\boldsymbol{\mathcal{A}}||\mathcal{S}|^2 h)$. Using the MDP heuristic then takes time $O(|\mathcal{S}|)$ per observation history per heuristic computation.

6.1.2 POMDP Heuristic

Next, we consider the POMDP heuristic Q_{POMDP} [Szer *et al.*, 2005; Roth *et al.*, 2005]. Intuitively, $Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$ represents the value that can be achieved by an extension of $\boldsymbol{\varphi}$ if the agents could observe the joint observations, rather than just their own local observations.

To compute the POMDP heuristic, we precompute the value function $V_{\mathsf{POMDP}}(b,t)$ of the belief MDP corresponding to the underlying POMDP, as explained in Section 2.3.1. Consider a partial policy $\boldsymbol{\varphi}$ of stage $\sigma(\boldsymbol{\varphi}) = t$. We define $Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$ similarly to the MDP heuristic:

$$Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b^0,t) + \sum_{\boldsymbol{\tau}\in\boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) V_{\mathsf{POMDP}}(b_{\boldsymbol{\tau}}, h-t),$$

where $b_{\boldsymbol{\tau}} \in \Delta(\mathcal{S})$ is the state belief corresponding to the joint observation history $\boldsymbol{\tau}$, i.e. the belief $b_{\boldsymbol{\tau}}(s^t) = \Pr(s^t \mid \boldsymbol{\tau})$. In Corollary 6.2, we show that Q_{POMDP} is admissible. Let B_t denote the set of reachable beliefs (from the given initial belief) up to horizon t. As shown in Section 2.3.1, the precomputation of $V_{\mathsf{POMDP}}(b,t)$ takes time $O\left(|B_{h-1}|(|\mathcal{S}|^2 + |\boldsymbol{\mathcal{A}}||\mathcal{S}|) + |B_{h-2}||\boldsymbol{\mathcal{A}}||\boldsymbol{\mathcal{O}}|\right)$. Using the POMDP heuristic then takes time $O(1)$ per observation history per heuristic computation, excluding the time for computing the beliefs. Here $|B_{h-1}| \in O((|\boldsymbol{\mathcal{A}}||\boldsymbol{\mathcal{O}}|)^{h-1})$ is the dominating factor in this time complexity.

6.1.3 BG Heuristic

Next, we turn to the Bayesian Game (BG) heuristic Q_{BG} [Oliehoek and Vlassis, 2007; Oliehoek *et al.*, 2008b]. Intuitively, $Q_{\mathsf{BG}}(\boldsymbol{\varphi})$ represents the value that can be achieved by an extension of $\boldsymbol{\varphi}$ if the agents receive the joint observations with a delay of one time step, in addition to their own local observation. Hence, in stage t, an agent can use the joint observations up to stage $t-1$ and their own local observation corresponding to stage t. A possible way to compute this heuristic is to use Bayesian games, which explains the name of the heuristic.

Like the POMDP heuristic, the BG heuristic can be computed using the belief MDP. However, the required precomputations are more extensive. Specifically, we precompute $Q'_{\mathsf{BG}}(b,t,\mathbf{a})$ for all $1 \leq t \leq h$, all beliefs b that are reachable from the given initial belief in exactly $h-t$ steps, and all joint actions $\mathbf{a} \in \boldsymbol{\mathcal{A}}$. As in Section 2.3.1, let $b_{\mathbf{a},\mathbf{o}}$ denote the posterior belief when the prior belief is b, action $\mathbf{a}$ is taken, and observation $\mathbf{o}$ is received. Then we define $Q'_{\mathsf{BG}}(b,1,\mathbf{a}) = R(b,\mathbf{a})$ and

$$Q'_{\mathsf{BG}}(b,t+1,\mathbf{a}) = R(b,\mathbf{a}) + \max_{\substack{\delta_i:\,\mathcal{O}_i \to \mathcal{A}_i \\ 1\leq i\leq n}} \sum_{o_1\in\mathcal{O}_1} \cdots \sum_{o_n\in\mathcal{O}_n} \left[\Pr(\mathbf{o} \mid b,\mathbf{a}) Q'_{\mathsf{BG}}(b_{\mathbf{a},\mathbf{o}},t,\mathbf{a}')\right], \tag{6.1}$$

where $\mathbf{a}' = \langle \delta_1(o_1), \ldots, \delta_n(o_n)\rangle$. By maximizing over functions $\delta_i \colon \mathcal{O}_i \to \mathcal{A}_i$, we model that the action that each agent chooses can only depend on their own local observation, rather than the full joint observation.

Using the Q'_{BG}-values, we can compute $V_{\mathsf{BG}}(b,t) = \max_{\mathbf{a}\in\boldsymbol{\mathcal{A}}} Q'_{\mathsf{BG}}(b,t,\mathbf{a})$. To use the BG heuristic to compute a heuristic value $Q_{\mathsf{BG}}(\boldsymbol{\varphi})$ for a partial policy $\boldsymbol{\varphi}$ with stage $\sigma(\boldsymbol{\varphi}) = t$, we then define

$$Q_{\mathsf{BG}}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b^0,t) + \sum_{\boldsymbol{\tau}\in\boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) V_{\mathsf{BG}}(b_{\boldsymbol{\tau}}, h-t). \tag{6.2}$$

In Corollary 6.3, we show that Q_{BG} is admissible.

We now discuss the cost of precomputing Q'_{BG}. The cost for computing all reachable beliefs and the corresponding rewards $R(b,\mathbf{a})$ is $O\left(|B_{h-1}|(|\mathcal{S}|^2 + |\boldsymbol{\mathcal{A}}||\mathcal{S}|)\right)$, as for the POMDP heuristic. We use Equation (6.1) in total $O(|B_{h-2}||\boldsymbol{\mathcal{A}}|)$ times. When solving the maximization problem in Equation (6.1) using brute force, the number of possibilities for the δ_i is $O\left(\prod_{i\in\mathcal{D}} |\mathcal{A}_i|^{|\mathcal{O}_i|}\right)$, or $O\left(|\mathcal{A}_*|^{n|\mathcal{O}_*|}\right)$ where $|\mathcal{A}_*|$ and $|\mathcal{O}_*|$ are the sizes of the (largest) local action and observation sets. For given δ_i, calculating the corresponding value can be done in time $O(|\boldsymbol{\mathcal{O}}|)$. Hence, the total time complexity in this case is $O\left(|B_{h-1}|(|\mathcal{S}|^2 + |\boldsymbol{\mathcal{A}}||\mathcal{S}|) + |B_{h-2}||\boldsymbol{\mathcal{A}}||\boldsymbol{\mathcal{O}}||\mathcal{A}_*|^{n|\mathcal{O}_*|}\right)$.

In practice, Bayesian game solvers are used to solve the maximization problem in Equation (6.1), which however have the same worst-case time complexity. Compared to the POMDP heuristic, we therefore have an extra factor $|\mathcal{A}_*|^{n|\mathcal{O}_*|}$ in the second term. For problems with a larger number of actions and observations, this quickly becomes infeasible.

Faster precomputation. We now discuss a novel way to precompute the Q'_{BG} values. The idea is similar to the idea used in the last stage optimization for the last agent (Section 5.6.1). Namely, we again show that we can maximize over the points $\delta_n(o_n)$ of the last agent independently. For this, we rewrite the maximization problem in Equation (6.1) as

$$\begin{aligned} Q'_{\mathsf{BG}}(b,t+1,\mathbf{a}) &= R(b,\mathbf{a}) + \max_{\substack{\delta_i:\,\mathcal{O}_i \to \mathcal{A}_i \\ 1\leq i\leq n}} \sum_{o_n\in\mathcal{O}_n} \sum_{o_1\in\mathcal{O}_1} \cdots \sum_{o_{n-1}\in\mathcal{O}_n} \left[\Pr(\mathbf{o} \mid b,\mathbf{a}) Q'_{\mathsf{BG}}(b_{\mathbf{a},\mathbf{o}},t,\mathbf{a}')\right], \\ &= R(b,\mathbf{a}) + \max_{\substack{\delta_i:\,\mathcal{O}_i \to \mathcal{A}_i \\ 1\leq i\leq n-1}} \sum_{o_n\in\mathcal{O}_n} \max_{a'_n\in\mathcal{A}_n} \sum_{o_1\in\mathcal{O}_1} \cdots \sum_{o_{n-1}\in\mathcal{O}_n} \left[\Pr(\mathbf{o} \mid b,\mathbf{a}) Q'_{\mathsf{BG}}(b_{\mathbf{a},\mathbf{o}},t,\mathbf{a}')\right], \end{aligned}$$

where now $\mathbf{a}' = \langle \delta_1(o_1), \ldots, \delta_{n-1}(o_{n-1}), a'_n\rangle$. We now only have to brute force over the policies of $n-1$ agents, while maximizing over the actions of the last agent only contributes a factor $|\mathcal{A}_*|$ to the time complexity. This reduces the time complexity per Q'_{BG} to $O\left(|\mathcal{A}_*|^{(n-1)|\mathcal{O}_*|+1}|\boldsymbol{\mathcal{O}}|\right)$, yielding a total time complexity of $O\left(|B_{h-1}|(|\mathcal{S}|^2 + |\boldsymbol{\mathcal{A}}||\mathcal{S}|) + |B_{h-2}||\boldsymbol{\mathcal{A}}||\boldsymbol{\mathcal{O}}||\mathcal{A}_*|^{(n-1)|\mathcal{O}_*|+1}\right)$.

Moreover, we can speed up the computation in practice for some beliefs b by noting that not all (local) observations are possible when taking action $\mathbf{a}$ in a state s in the support of b. This can substantially limit the number of size of the domain of the policies we brute force over.

6.2 Precomputed Heuristics for Small-step MAA*

We proceed by discussing how to extend the precomputed heuristics Q_{MDP}, Q_{POMDP} and Q_{BG} to all policies in the small-step search tree. To do this, we have to be able to reflect in the heuristic value that a partial policy $\boldsymbol{\varphi}$ assigns actions for some, but not all agents for some observation history.

We introduce the notion of a *partial (joint) action*, which is a tuple of actions $\langle a_1, \ldots, a_i \rangle$ of the first i agents for some $0 \leq i \leq n$, where $i = 0$ corresponds to the empty tuple. Let $\boldsymbol{\mathcal{A}}_\Phi = \bigcup_{i=0}^{n} \bigtimes_{j=0}^{i} \mathcal{A}_j$ be the set of partial actions. If $|\mathcal{A}_j| \geq 2$ for all $j \in \mathcal{D}$, then $|\boldsymbol{\mathcal{A}}_\Phi| \in O(|\boldsymbol{\mathcal{A}}|)$ since

$$|\boldsymbol{\mathcal{A}}_\Phi| = \sum_{i=0}^{n} \prod_{j=0}^{i} |\mathcal{A}_j| = |\boldsymbol{\mathcal{A}}| \sum_{i=0}^{n} \prod_{j=i+1}^{n} |\mathcal{A}_j|^{-1} \leq |\boldsymbol{\mathcal{A}}| \sum_{i=0}^{n} \prod_{j=i+1}^{n} \frac{1}{2} = |\boldsymbol{\mathcal{A}}| \sum_{i=0}^{n} \frac{1}{2^{n-i}} < 2|\boldsymbol{\mathcal{A}}|.$$

We now explain how to define partial actions based on a partial policy. Consider a partial policy $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) = t$, and let $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t$ be an observation history. Let $\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}} = \langle \varphi_1(\tau_1), \ldots, \varphi_i(\tau_i) \rangle$, where i is the last agent for which $\boldsymbol{\varphi}$ specifies an action for $\boldsymbol{\tau}$, i.e. such that $\varphi_j(\tau_j) \neq \bot$ for $1 \leq j \leq i$ and $\varphi_j(\tau_j) = \bot$ for $i+1 \leq j \leq n$. These partial actions are the partial actions to which we apply our precomputed heuristics. Next, we extend the precomputation to partial policies. In each case, it can be shown that the time complexity of the precomputation remains the same as for MAA* (up to lower order terms), although more values need to be stored.

MDP heuristic. We precompute a heuristic $Q'_{\mathsf{MDP}}(s, t, \mathbf{a})$ for each initial state $s \in \mathcal{S}$, each horizon $1 \leq t \leq h$ and each partial action $\mathbf{a} \in \boldsymbol{\mathcal{A}}_\Phi$, using a small modification of value iteration. First, we let $Q'_{\mathsf{MDP}}(s, 1, \mathbf{a}) = R(s, \mathbf{a})$ for $\mathbf{a} \in \boldsymbol{\mathcal{A}}$. For partial actions $\langle a_1, \ldots, a_i \rangle \in \boldsymbol{\mathcal{A}}_\Phi \setminus \boldsymbol{\mathcal{A}}$, we then recursively define $Q'_{\mathsf{MDP}}(s, t, \langle a_1, \ldots, a_i \rangle) = \max_{a_{i+1} \in \mathcal{A}_{i+1}} Q'_{\mathsf{MDP}}(s, t, \langle a_1, \ldots, a_{i+1} \rangle)$. Finally, we have

$$Q'_{\mathsf{MDP}}(s, t+1, \mathbf{a}) = R(s, \mathbf{a}) + \sum_{s' \in \mathcal{S}} T(s' \mid s, \mathbf{a}) Q'_{\mathsf{MDP}}(s', t, \langle\rangle),$$

where $\langle\rangle$ is the empty tuple, which by definition (implicitly) picks out the optimal action for Q'_{MDP}. For a partial policy $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) = t$, we define $Q_{\mathsf{MDP}}(\boldsymbol{\varphi})$ as

$$Q_{\mathsf{MDP}}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b^0, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \sum_{s^t \in \mathcal{S}} \Pr(\boldsymbol{\tau}, s^t) Q'_{\mathsf{MDP}}(s^t, h - t, \mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}}).$$

POMDP heuristic. For the POMDP heuristic, we precompute $Q'_{\mathsf{POMDP}}(b, t, \mathbf{a})$ for $1 \leq t \leq h$, all beliefs b that are reachable from the given initial belief in exactly $h - t$ steps, and all partial actions $\mathbf{a} \in \boldsymbol{\mathcal{A}}_\Phi$. Similarly to the MDP heuristic, we let $Q'_{\mathsf{POMDP}}(b, 1, \mathbf{a}) = R(b, \mathbf{a})$ for $\mathbf{a} \in \boldsymbol{\mathcal{A}}$ and recursively define $Q'_{\mathsf{POMDP}}(b, t, \langle a_1, \ldots, a_i \rangle) = \max_{a_{i+1} \in \mathcal{A}_{i+1}} Q'_{\mathsf{POMDP}}(b, t, \langle a_1, \ldots, a_{i+1} \rangle)$. For $\mathbf{a} \in \boldsymbol{\mathcal{A}}$, the value iteration equation becomes

$$Q'_{\mathsf{POMDP}}(b, t+1, \mathbf{a}) = R(b, \mathbf{a}) + \sum_{\mathbf{o} \in \boldsymbol{\mathcal{O}}} \left[\Pr(\mathbf{o} \mid b, \mathbf{a}) Q'_{\mathsf{POMDP}}(b_{\mathbf{a},\mathbf{o}}, t, \langle\rangle) \right].$$

Finally, for a partial policy $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) = t$, we define $Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$ as

$$Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b^0, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) Q'_{\mathsf{POMDP}}(b_{\boldsymbol{\tau}}, h - t, \mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}}).$$

BG heuristic. For the BG heuristic, the required modifications are even smaller. We keep the computation for the heuristic values $Q'_{\mathsf{BG}}(b, t, \mathbf{a})$ (with $\mathbf{a} \in \boldsymbol{\mathcal{A}}$) as introduced in Section 6.1.3, but extend this to partial actions using $Q'_{\mathsf{BG}}(s, t, \langle a_1, \ldots, a_i \rangle) = \max_{a_{i+1} \in \mathcal{A}_{i+1}} Q'_{\mathsf{BG}}(s, t, \langle a_1, \ldots, a_{i+1} \rangle)$. Heuristics for a partial policy $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) = t$ are defined as for the POMDP heuristic:

$$Q_{\mathsf{BG}}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b^0, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) Q'_{\mathsf{BG}}(b_{\boldsymbol{\tau}}, h - t, \mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}}).$$

6.3 General Framework for Admissible Heuristics

In this section, we present a general framework that can be used to define admissible heuristics. Besides defining heuristics and giving a short proof that they are admissible, this framework can also be used to short proofs for inequalities between different heuristics.

6.3.1 Generalized Policies

To be able to define the heuristics, we first introduce a more general type of policy. Intuitively, these generalized policies allow the agents to use additional information to decide which actions to take. Formally, a *generalized local policy* $\pi_i \colon \bigcup_{v=0}^{h-1}(\boldsymbol{\mathcal{O}}^v \times \mathcal{S}^{v+1}) \to \mathcal{A}_i$ maps histories of states and joint observations to a local action.[6] Note that there is one more state than observation, since in stage v the system has gone through $v+1$ states (s^0 up to s^v), but the agents have received only v observations ($\mathbf{o}^1$ up to $\mathbf{o}^v$).

A *generalized (joint) policy* is a tuple $\langle \pi_1, \ldots, \pi_n \rangle$ of generalized local policies. Let Π_{gen} denote the set of all generalized policies. A generalized policy $\boldsymbol{\pi}$ *extends* a partial policy $\boldsymbol{\varphi}$, denoted by $\boldsymbol{\varphi} <_E \boldsymbol{\pi}$, if π_i agrees with φ_i and ignores the state information on all OHs corresponding to an LOH for which φ_i specifies the action. Formally,

$$\boldsymbol{\varphi} <_E \boldsymbol{\pi} \iff \forall i \in \mathcal{D}. \forall (\boldsymbol{\tau}, \boldsymbol{s}) \in \textstyle\bigcup_{v=0}^{h-1}(\boldsymbol{\mathcal{O}}^v \times \mathcal{S}^{v+1}).\ \varphi_i(\tau_i) \in \{\bot, \pi_i(\boldsymbol{\tau}, \boldsymbol{s})\}.$$

If $\varphi_i(\tau_i) \neq \bot$, then this forces that $\pi_i(\boldsymbol{\tau}, \boldsymbol{s}) = \varphi_i(\tau_i)$ for all $\boldsymbol{s}$ and all joint OHs whose LOH for agent i is τ_i. This means that agent i does not use any information that is not actually available in a Dec-POMDP setting to decide which action to take on this specific LOH τ_i. On the other hand, if $\varphi_i(\tau_i) = \bot$, then this puts no constraints on the $\pi_i(\boldsymbol{\tau}, \boldsymbol{s})$. Finally, let

$$E_{\mathsf{gen}}(\boldsymbol{\varphi}) = \left\{ \boldsymbol{\pi} \in \Pi_{\mathsf{gen}} \mid \boldsymbol{\varphi} <_E \boldsymbol{\pi} \right\}$$

be the set of generalized extensions of a partial policy $\boldsymbol{\varphi}$ agreeing with $\boldsymbol{\varphi}$.

6.3.2 Framework

To explain our general framework for admissible heuristics, we first write the Dec-POMDP optimization problem as an optimization problem over generalized policies, as follows:

$$\begin{aligned} &\max_{\boldsymbol{\pi} \in \Pi_{\mathsf{gen}}} V_{\boldsymbol{\pi}}(b, h) \\ &\quad \text{subject to} \quad \pi_i(\boldsymbol{\tau}, \boldsymbol{s}) = \pi_i(\widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}}) \quad \text{for all } (i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \in \mathcal{I}, \text{ with} \\ &\mathcal{I} = \bigcup_{v=0}^{h-1} \left\{ (i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \mid i \in \mathcal{D},\ \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^v,\ \boldsymbol{s}, \widetilde{\boldsymbol{s}} \in \mathcal{S}^{v+1},\ \tau_i = \widetilde{\tau}_i \right\}. \end{aligned} \tag{6.3}$$

Using $\mathcal{I}$, we quantify over all agents and pairs of traces. The condition $\tau_i = \widetilde{\tau}_i$ in the definition of $\mathcal{I}$ selects the pairs $(\boldsymbol{\tau}, \boldsymbol{s})$ that agent i cannot distinguish, since they induce the same LOH for agent i. On these pairs, the agent therefore has to take the same action. Using this framework, we can also express the requirement for an admissible heuristic Q:

$$\begin{aligned} Q(\boldsymbol{\varphi}) \geq \max_{\boldsymbol{\pi} \in E_{\mathsf{gen}}(\boldsymbol{\varphi})} & V_{\boldsymbol{\pi}}(b, h) \\ & \text{subject to} \quad \pi_i(\boldsymbol{\tau}, \boldsymbol{s}) = \pi_i(\widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}}) \quad \text{for all } (i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \in \mathcal{I}. \end{aligned} \tag{6.4}$$

Hence, instead of maximizing over Π_{gen} in Equation (6.3), we maximize over $E_{\mathsf{gen}}(\boldsymbol{\varphi})$.

We can now define a general class of admissible heuristics by relaxing some of these constraints, i.e. by considering a subset $\mathcal{I}' \subseteq \mathcal{I}$ of the constraints in Equation (6.4). Denote the corresponding heuristic by $Q_{\mathcal{I}'}(\boldsymbol{\varphi})$. Formally, we define:

[6] Including (past) observations states seems unnecessary for policies that have access to the current state, however, we will add constraints that the policy cannot depend on the most recent state.

Definition 6.1. *Let $\mathcal{I}' \subseteq \mathcal{I}$ be given. We define the heuristic $Q_{\mathcal{I}'} \colon \Phi \to \mathbb{R}$ by*

$$Q_{\mathcal{I}'}(\boldsymbol{\varphi}) = \max_{\boldsymbol{\pi} \in E_{\mathsf{gen}}(\boldsymbol{\varphi})} V_{\boldsymbol{\pi}}(b,h)$$
$$\text{subject to} \quad \pi_i(\boldsymbol{\tau}, \boldsymbol{s}) = \pi_i(\widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}}) \text{ for all } (i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \in \mathcal{I}'.$$

Next, we show that $Q_{\mathcal{I}'}$ is admissible and consistent.

Theorem 6.1 (Admissibility)**.** *$Q_{\mathcal{I}'}$ is admissible for all $\mathcal{I}' \subseteq \mathcal{I}$.*

Proof. To show admissibility, it suffices to check Equation (6.4). Compared to the right hand side in Equation (6.4), $Q_{\mathcal{I}'}$ has fewer constraints since $\mathcal{I}' \subseteq \mathcal{I}$. Since relaxing constraints in a maximization problem can only increase the optimal value, the result follows. □

Theorem 6.2 (Consistency)**.** *$Q_{\mathcal{I}'}$ is consistent for all $\mathcal{I}' \subseteq \mathcal{I}$.*

Proof. We first show that $Q_{\mathcal{I}'}(\boldsymbol{\varphi}^-) \geq Q_{\mathcal{I}'}(\boldsymbol{\varphi})$ for all partial policies $\boldsymbol{\varphi}$ which are not the empty policy, where $\boldsymbol{\varphi}^-$ denotes the parent of $\boldsymbol{\varphi}$. When comparing the definitions of $Q_{\mathcal{I}'}(\boldsymbol{\varphi}^-)$ and $Q_{\mathcal{I}'}(\boldsymbol{\varphi})$, the only difference is that we maximize over $E_{\mathsf{gen}}(\boldsymbol{\varphi}^-)$ rather than $E_{\mathsf{gen}}(\boldsymbol{\varphi})$. Since we have $E_{\mathsf{gen}}(\boldsymbol{\varphi}) \subseteq E_{\mathsf{gen}}(\boldsymbol{\varphi}^-)$ and maximizing over a larger domain can only increase the optimal value, the inequality $Q_{\mathcal{I}'}(\boldsymbol{\varphi}^-) \geq Q_{\mathcal{I}'}(\boldsymbol{\varphi})$ follows.

To show that $Q_{\mathcal{I}'}(\boldsymbol{\pi}) = V_{\boldsymbol{\pi}}(b,h)$, we note that $E_{\mathsf{gen}}(\boldsymbol{\pi}) = \{\boldsymbol{\pi}\}$. Since $\boldsymbol{\pi}$ is a Dec-POMDP policy, it satisfies the Dec-POMDP constraints which correspond to the set $\mathcal{I}$. Since $\mathcal{I}' \subseteq \mathcal{I}$, it follows that $\boldsymbol{\pi}$ in particular satisfies all constraints imposed by $Q_{\mathcal{I}'}$. Hence, it follows that $\boldsymbol{\pi}$ is feasible for the maximization problem defining $Q_{\mathcal{I}'}$, and hence also maximal (since it is the only candidate solution). This yields that $Q_{\mathcal{I}'}(\boldsymbol{\pi}) = V_{\boldsymbol{\pi}}(b,h)$. We conclude that $Q_{\mathcal{I}'}$ is consistent. □

We note that Theorem 6.2 and Lemma 5.5 also imply Theorem 6.1.

We now explain how to define the set $\mathcal{I}'$ for the heuristic $Q_{\mathcal{I}'}$ using an *information function* F with domain $\mathcal{D} \times \bigcup_{v=0}^{h-1} (\boldsymbol{\mathcal{O}}^v \times \mathcal{S}^{v+1})$. Namely, we let

$$\mathcal{I}'_{\mathsf{F}} = \bigcup_{v=0}^{h-1} \Big\{ (i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \mid i \in \mathcal{D},\ \ \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^v,\ \ \boldsymbol{s}, \widetilde{\boldsymbol{s}} \in \mathcal{S}^{v+1},\ \ \mathsf{F}(i, \boldsymbol{\tau}, \boldsymbol{s}) = \mathsf{F}(i, \widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}}) \Big\}.$$

$\mathsf{F}(i, \boldsymbol{\tau}, \boldsymbol{s})$ describes which information from $(\boldsymbol{\tau}, \boldsymbol{s})$ agent i can use to choose which action to take, since the constraints in $\mathcal{I}'_{\mathsf{F}}$ state that agent i has to take the same action on pairs $(\boldsymbol{\tau}, \boldsymbol{s})$ which cannot be distinguished using the information $\mathsf{F}(i, \boldsymbol{\tau}, \boldsymbol{s})$. If $\mathsf{F}(i, \boldsymbol{\tau}, \boldsymbol{s}) = \mathsf{F}(i, \widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}})$ implies $\tau_i = \widetilde{\tau}_i$, then $\mathcal{I}'_{\mathsf{F}} \subseteq \mathcal{I}$, and hence $Q_{\mathcal{I}'_{\mathsf{F}}}$ is admissible and consistent. Finally, we write Q_{F} instead of $Q_{\mathcal{I}'_{\mathsf{F}}}$.

6.3.3 Precomputed Heuristics

Next, we use this framework to show some properties of Q_{MDP}, Q_{POMDP} and Q_{BG}.

Corollary 6.1. *Q_{MDP} is admissible and consistent.*

Proof. The MDP heuristic corresponds to the set $\mathcal{I}'_{\mathsf{MDP}} = \emptyset$. The agents can then use all state information and joint observations to decide on an action, but in practice only use the most recent state. Since $\mathcal{I}'_{\mathsf{MDP}} \subseteq \mathcal{I}$, Theorem 6.1 and 6.2 imply that Q_{MDP} is admissible and consistent. □

Corollary 6.2. *Q_{POMDP} is admissible and consistent.*

Proof. The POMDP heuristic corresponds to the information function $\mathsf{POMDP}(i, \boldsymbol{\tau}, \boldsymbol{s}) = \boldsymbol{\tau}$. In the definition of $Q_{\mathcal{I}'_{\mathsf{POMDP}}}$, the constraints then mean that the agents can use all joint observations, but still have to take the same action on pairs $(\boldsymbol{\tau}, \boldsymbol{s})$ that can only be distinguished using state information. Hence, this corresponds to the POMDP heuristic.

Since the condition $\boldsymbol{\tau} = \widetilde{\boldsymbol{\tau}}$ implies $\tau_i = \widetilde{\tau}_i$, we indeed have $\mathcal{I}'_{\mathsf{POMDP}} \subseteq \mathcal{I}$. Hence, Theorem 6.1 and 6.2 imply that Q_{POMDP} is admissible and consistent. □

6.4 Admissibility of the BG Heuristic

For Q_{BG}, we cannot directly use the framework introduced in Section 6.3. In fact, the Q_{BG} heuristic is in general not consistent. Intuitively, the reason for this is that Equation (6.2) effectively allows agents to use the joint observation of stage t in stage t if $\sigma(\boldsymbol{\varphi}) = t$, while the Q'_{BG} values used if $\sigma(\boldsymbol{\varphi}) = t - 1$ only allow the agents to use local observation of stage t. Hence, if $\sigma(\boldsymbol{\varphi}^-) < \sigma(\boldsymbol{\varphi})$, then agents can use more information for $\boldsymbol{\varphi}$ than for $\boldsymbol{\varphi}^-$. Put differently, for $\boldsymbol{\varphi}$ more constraints are relaxed than for $\boldsymbol{\varphi}^-$. This explains why we can have $Q_{\mathsf{BG}}(\boldsymbol{\varphi}^-) < Q_{\mathsf{BG}}(\boldsymbol{\varphi})$ (if $\sigma(\boldsymbol{\varphi}^-) < \sigma(\boldsymbol{\varphi})$).

To still be able to use the framework to study the Q_{BG} heuristic, we define a heuristic $Q_{\mathsf{BG},t}$ for each $t \geq 0$ which can be written using the framework, such that $Q_{\mathsf{BG}}(\boldsymbol{\varphi}) = Q_{\mathsf{BG},t}(\boldsymbol{\varphi})$ for $\sigma(\boldsymbol{\varphi}) = t$. The $Q_{\mathsf{BG},t}$-heuristic corresponds to revealing the joint observation history in stage t and the joint observation history of stage $v - 1$ in stages $v > t$.

Corollary 6.3. *Q_{BG} is admissible.*

Proof. Let $\boldsymbol{\tau} = \mathbf{o}^{[1,v]} \in \boldsymbol{\mathcal{O}}^v$ be a joint observation history and let $i \in \mathcal{D}$ be an agent. Then the $Q_{\mathsf{BG},t}$ heuristic defined above corresponds to the information function

$$\mathsf{BG}_t(i, \mathbf{o}^{[1,v]}, \boldsymbol{s}) = \begin{cases} o_i^{[1,v]} & \text{if } v < t \\ \mathbf{o}^{[1,v]} & \text{if } v = t \\ (\mathbf{o}_{\neq i}^{[1,v-1]}, o_i^{[1,v]}) & \text{if } v > t. \end{cases}$$

Namely, $\mathsf{BG}_t(i, \mathbf{o}^{[1,v]}, \boldsymbol{s})$ represents the information that agent i can use in stage v: the joint observation history in stage t, and the joint observation history of stage $v - 1$ and the own local observation in stage v in stages $v > t$.

Since the condition $\mathsf{BG}_t(i, \boldsymbol{\tau}, \boldsymbol{s}) = \mathsf{BG}_t(i, \widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}})$ implies $\tau_i = \widetilde{\tau}_i$, we indeed have $\mathcal{I}'_{\mathsf{BG}_t} \subseteq \mathcal{I}$. Hence, Theorem 6.1 implies that $Q_{\mathsf{BG},t}$ is admissible for all $t \geq 0$. Since $Q_{\mathsf{BG}}(\boldsymbol{\varphi}) = Q_{\mathsf{BG},t}(\boldsymbol{\varphi})$ for partial policies $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) = t$, it follows from this that Q_{BG} is admissible. □

Finally, we can also use the framework to show the inequality between the heuristics Q_{MDP}, Q_{POMDP} and Q_{BG} (which was first shown by Oliehoek and Vlassis [2007]):

Lemma 6.1. *For any partial policy $\boldsymbol{\varphi}$, we have $Q_{\mathsf{BG}}(\boldsymbol{\varphi}) \leq Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}) \leq Q_{\mathsf{MDP}}(\boldsymbol{\varphi})$.*

Proof. We have $\mathcal{I}'_{\mathsf{MDP}} \subseteq \mathcal{I}'_{\mathsf{POMDP}} \subseteq \mathcal{I}'_{\mathsf{BG},t}$. Since relaxing constraints in a maximization problem can only increase the optimal value, it follows that $Q_{\mathsf{BG},t}(\boldsymbol{\varphi}) \leq Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}) \leq Q_{\mathsf{MDP}}(\boldsymbol{\varphi})$ for all $t \geq 0$. This in turn implies that $Q_{\mathsf{BG}}(\boldsymbol{\varphi}) \leq Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}) \leq Q_{\mathsf{MDP}}(\boldsymbol{\varphi})$. □

7 Recursive Small-Step Multi-Agent A* (RS-MAA*)

In this section, we study our exact algorithm *recursive small-step multi-agent A* (RS-MAA*)*. It applies small-step MAA* with lossless clustering [Oliehoek *et al.*, 2009] and a recursive heuristic with early termination $Q_{M,d}$, which we introduce in Section 7.1 and 7.2. In Section 7.3, we discuss how we store and reuse heuristics in our implementation.

7.1 Recursive Heuristics

We now turn to the recursive heuristics that we use in our exact algorithm recursive small-step multi-agent A* (RS-MAA*). For an exact solver, using a tight heuristic is for most problems essential to avoid having to explore a double exponential number of nodes. The reason for this is that there are many clusters with a small probability, and taking suboptimal actions on these clusters only leads to a slightly lower policy value compared to the optimal policy value. When an admissible heuristic is not sufficiently tight, the heuristic values corresponding to partial policies leading to these suboptimal policies are larger than the optimal policy value, which means that the A* search must explore them to guarantee exactness. To avoid this, we propose tight recursive heuristics, which are tighter than the heuristics currently available in the literature.

We first introduce *fixed-depth heuristics* with some depth d. However, since we can only compute these heuristics efficiently if $\sigma(\boldsymbol{\varphi}) \geq d$, we also introduce *variable-depth heuristics* whose depth is $\sigma(\boldsymbol{\varphi})$ if $\sigma(\boldsymbol{\varphi}) < d$ and d if $\sigma(\boldsymbol{\varphi}) \geq d$.

7.1.1 Fixed-Depth Heuristics

We start by defining *fixed-depth heuristics*. For the fixed-depth heuristic $Q_{\mathsf{Dec},d}$ with depth d, agents are allowed to use the first d joint observations in addition to their own local observations. We formalize this as follows, using the framework from Section 6.3.

Definition 7.1 (Fixed-depth heuristic)**.** *Let $\boldsymbol{\tau} = \mathbf{o}^{[1,v]} \in \boldsymbol{\mathcal{O}}^v$ be a joint observation history and let $i \in \mathcal{D}$ be an agent. Write $\mathsf{FD}_d\big(i, \mathbf{o}^{[1,v]}, \boldsymbol{s}\big) = \big(\mathbf{o}_{\neq i}^{[1,\min\{v,d\}]}, o_i^{[1,v]}\big)$ for the information that agent i can use: the first d joint observations (or fewer if there are less than d), and its own local observations. Then the* fixed-depth heuristic *$Q_{\mathsf{Dec},d}$ is the heuristic Q_{FD_d}.*

From our general framework, we directly see that $Q_{\mathsf{Dec},d}$ is admissible and consistent:

Lemma 7.1. *$Q_{\mathsf{Dec},d}$ is admissible and consistent.*

Proof. The condition $\mathsf{FG}_d(i, \boldsymbol{\tau}, \boldsymbol{s}) = \mathsf{FG}_d(i, \widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}})$ implies $\tau_i = \widetilde{\tau}_i$, which implies that $\mathcal{I}'_{\mathsf{FD}_d} \subseteq \mathcal{I}$. Hence, Theorem 6.1 and 6.2 imply that $Q_{\mathsf{Dec},d}$ is admissible and consistent. □

For $d = 0$, the fixed-depth heuristic corresponds to the original optimization problem (6.3). From now on, we therefore assume that $d \geq 1$. For stage-t partial policies with $d \leq t$, the fixed-depth heuristic is easier to compute than the Dec-POMDP value itself. Next, we explain this, first informally and then formally. Since $d \leq t$, the policy for the first d stages is specified by $\boldsymbol{\varphi}$, and we can compute the realized reward $V_{\boldsymbol{\varphi}}(b, d)$ up to stage d.

Intuition for the fixed-depth heuristic. We now first provide some intuition why the fixed-depth heuristic with depth d is easier to compute than the Dec-POMDP value itself. Recall from Section 3.6 that one of the challenges in solving Dec-POMDPs is that we cannot decompose solving a Dec-POMDP into solving several independent, smaller problems, as is possible for an MDP and a POMDP. The fixed-depth heuristic makes such a decomposition into smaller problems possible. Namely, the fixed-depth heuristic effectively removes all constraints from the Dec-POMDP optimization problem that involve both $\pi_i(\boldsymbol{\tau}, \boldsymbol{s})$ and $\pi_i(\widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}})$, where the first d observations in $\boldsymbol{\tau}$ and $\widetilde{\boldsymbol{\tau}}$ are different. Hence, given that the policy is fixed for stage $t \leq d-1$, the problem effectively falls apart in $|\boldsymbol{\mathcal{O}}|^d$ independent subproblems, where each subproblem corresponds to the variables $\pi_i(\boldsymbol{\tau}, \boldsymbol{s})$ where $\boldsymbol{\tau}$ has some specific prefix of length d. Each of these subproblems corresponds to a Dec-POMDP with horizon $h - d$, which leads to the recursive nature of the heuristic.

Shortened policy. Consider a partial policy $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) \geq d$. To compute the fixed-depth heuristic of depth d, we solve Dec-POMDPs with horizon $h-d$. If $\boldsymbol{\varphi}$ has already specified actions for LOHs of length larger than d, then this should also be reflected in the problems with horizon $h-d$. For this, we define the *shortened partial policy* $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$, which specifies for which observation histories $\boldsymbol{\varphi}$ has already specified an action: $(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})_i(\widetilde{\tau}) = \varphi_i(\tau_i\widetilde{\tau})$.

Computing the fixed-depth heuristic. We now explain how to compute the fixed-depth heuristic with depth d in practice for partial policies $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) \geq d$. Revealing the first d joint observations $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^d$ provides the agents with a new joint belief $b_{\boldsymbol{\tau}}$ after d stages. Hence, for each length-d joint observation history we solve a Dec-POMDP with horizon $h-d$, the new belief $b_{\boldsymbol{\tau}}$ as initial belief, and the shortened partial policy $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ as initial policy of which all considered policies must be an extension. This leads to the following equation:

$$Q_{\mathsf{Dec},d}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b,d) + \sum_{\boldsymbol{\tau}\in\boldsymbol{\mathcal{O}}^d} \Pr(\boldsymbol{\tau}) \cdot \max_{\boldsymbol{\pi}\in E(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})} V_{\boldsymbol{\pi}}(b_{\boldsymbol{\tau}}, h-d)\,. \tag{7.1}$$

Comparing different depths. When computing $Q_{\mathsf{Dec},d}$, we allow each agent to also base their decision on these first d joint observations. The smaller d, the less (additional) information agents have, so the tighter the heuristic is. This provides the intuition for the following lemma. We again prove the lemma by using our general framework.

Lemma 7.2. *Let $\boldsymbol{\varphi}$ be a partial policy. If $d' \leq d$, then $Q_{\mathsf{Dec},d'}(\boldsymbol{\varphi}) \leq Q_{\mathsf{Dec},d}(\boldsymbol{\varphi})$.*

Proof. If $d' \leq d$, then $\mathsf{FD}_d\big(i,\boldsymbol{\tau},\boldsymbol{s}\big) = \mathsf{FD}_d\big(i,\widetilde{\boldsymbol{\tau}},\widetilde{\boldsymbol{s}}\big)$ implies $\mathsf{FD}_{d'}\big(i,\boldsymbol{\tau},\boldsymbol{s}\big) = \mathsf{FD}_{d'}\big(i,\widetilde{\boldsymbol{\tau}},\widetilde{\boldsymbol{s}}\big)$. Hence, we have $\mathcal{I}'_{\mathsf{FD}_d} \subseteq \mathcal{I}'_{\mathsf{FD}_{d'}}$. So the maximization problem corresponding to depth d relaxes constraints compared to the maximization problem corresponding to depth d'. Since relaxing constraints in a maximization problem can only increase the optimal value, the result follows. □

On the other hand, the heuristic $Q_{\mathsf{Dec},d}$ is harder to compute for smaller d, since by Equation (7.1) we have to solve Dec-POMDPs with a larger horizon for smaller d.

7.1.2 Variable-Depth Heuristics

We now turn to *variable-depth heuristics*. These heuristics take into account that we do not want to compute $Q_{\mathsf{Dec},d}(\boldsymbol{\varphi})$ if $\sigma(\boldsymbol{\varphi}) < d$, because then Eq. (7.1) cannot be used. To solve this, we use $Q_{\mathsf{Dec},d}$ unless $\sigma(\boldsymbol{\varphi}) < d$, in which case we use $Q_{\mathsf{Dec},\sigma(\boldsymbol{\varphi})}(\boldsymbol{\varphi})$. We do not compute heuristics for policies with $\sigma(\boldsymbol{\varphi}) = 0$, which we model by setting these heuristic values to ∞. Hence, the actual depth d' used starts at 1 for policies with $\sigma(\boldsymbol{\varphi}) = 1$, and then gradually increases $\sigma(\boldsymbol{\varphi})$ until an upper limit d is reached. We call the corresponding variable-depth heuristic Q_d.

However, the heuristic $Q_{\mathsf{Dec},\sigma(\boldsymbol{\varphi})}(\boldsymbol{\varphi})$ is not consistent. Namely, whenever we increase the stage $\sigma(\boldsymbol{\varphi})$, we increase the depth of the heuristic and hence switch to a looser heuristic (by Lemma 7.2), which means that the heuristic value can increase rather than decrease. To solve this, we use the construction from Lemma 5.7 to make the heuristic consistent, i.e. when defining the heuristic, we take the minimum with the heuristic value of the parent.

Definition 7.2 (Variable-depth heuristic)**.** *Let $\boldsymbol{\varphi}$ be a partial policy. The* variable-depth heuristic *Q_d is recursively defined by $Q_d(\boldsymbol{\varphi}) = \infty$ if $\sigma(\boldsymbol{\varphi}) = 0$, and if $\sigma(\boldsymbol{\varphi}) \geq 1$ we define*

$$Q_d(\boldsymbol{\varphi}) = \min\{Q_{\mathsf{Dec},\min\{\sigma(\boldsymbol{\varphi}),d\}}(\boldsymbol{\varphi}), Q_d(\boldsymbol{\varphi}^-)\}.$$

We can also set $d = \infty$; then $\min\{\sigma(\boldsymbol{\varphi}), d\} = \sigma(\boldsymbol{\varphi})$. Next, we show that Q_d is admissible and consistent, using the properties of $Q_{\mathsf{Dec},d'}$.

Theorem 7.1. *Q_d is admissible and consistent for $d \geq 1$.*

Proof. We first show that $Q_d(\boldsymbol{\varphi}) \geq Q_{\mathsf{Dec},1}(\boldsymbol{\varphi})$ using induction. For $\sigma(\boldsymbol{\varphi}) = 0$ this holds trivially since then $Q_d(\boldsymbol{\varphi}) = \infty$. If $\sigma(\boldsymbol{\varphi}) \geq 1$, then we have $Q_{\mathsf{Dec},\min\{\sigma(\boldsymbol{\varphi}),d\}}(\boldsymbol{\varphi}) \geq Q_{\mathsf{Dec},1}(\boldsymbol{\varphi})$ by Lemma 7.2 and $Q_d(\boldsymbol{\varphi}^-) \geq Q_{\mathsf{Dec},1}(\boldsymbol{\varphi}^-) \geq Q_{\mathsf{Dec},1}(\boldsymbol{\varphi})$ by the induction hypothesis and since $Q_{\mathsf{Dec},1}$ is consistent. Together, this implies that $Q_d(\boldsymbol{\varphi}) \geq Q_{\mathsf{Dec},1}(\boldsymbol{\varphi})$. Since $Q_{\mathsf{Dec},1}$ is admissible (by Lemma 7.1), it follows that Q_d is also admissible.

To show that Q_d is consistent, we note that $Q_d(\boldsymbol{\varphi}) \leq Q_d(\boldsymbol{\varphi}^-)$ holds by construction. Also, we have $Q_d(\boldsymbol{\pi}) \geq V_{\boldsymbol{\pi}}(b,h)$ since Q_d is admissible, while $Q_d(\boldsymbol{\pi}) \leq Q_{\mathsf{Dec},\min\{h,d\}}(\boldsymbol{\pi}) = V_{\boldsymbol{\pi}}(b,h)$ since the heuristic $Q_{\mathsf{Dec},\min\{h,d\}}$ is consistent. Hence, $Q_d(\boldsymbol{\pi}) = V_{\boldsymbol{\pi}}(b,h)$ for fully specified Dec-POMDP policies $\boldsymbol{\pi}$. We conclude that Q_d is consistent. □

Comparison to recursive MAA*. We now compare the recursive heuristics proposed in this section with the recursive heuristic proposed by Szer *et al.* [2005], which is also called recursive MAA*. In essence, they propose the heuristic $Q_{\mathsf{Dec},\sigma(\boldsymbol{\varphi})}$, but with one difference. Namely, rather than revealing the first d joint observations, Szer *et al.* [2005] reveals the dth state. As a result, fewer different heuristics need to be computed, but the resulting heuristic is also less tight. We revisit the idea of revealing the state in Section 9.1.

7.2 Terminating Heuristic Computations Early

Although the recursive heuristics Q_d are usually very tight, they are also rather expensive to compute. We compute the recursive heuristics using Equation (7.1), for which we need to solve Dec-POMDPs with a smaller horizon. In turn, we solve these Dec-POMDPs with a smaller horizon again using an A* algorithm with a consistent heuristic. Empirically, we see that the highest heuristic value in the priority queue of the A* search quickly decreases to a value close to the optimal value, but exploring all possibilities with a heuristic value just above optimal value takes a lot of time. However, when computing heuristic values, we do not necessarily need the precise optimal value of the Dec-POMDPs with a smaller horizon. Using an upper bound for this optimal value still gives rise to an admissible heuristic. Due to the anytime nature of A* shown in Lemma 5.3, the highest heuristic value of an open node is an upper bound for the expected reward of an optimal policy. Hence, we can terminate the A* search corresponding to a heuristic computation early. The resulting heuristic is (slightly) less tight than without early termination, but typically still much tighter than the tightest heuristic from the literature, Q_{BG}. Moreover, this early termination makes the recursive heuristic substantially faster to compute.

7.2.1 Static Termination

We first discuss early termination based on a fixed limit M on the number of nodes expanded. We call this *static termination*, since the number of node expansions is fixed.

More precisely, we terminate heuristic computations after exploring M nodes and return the highest heuristic value of an open node at that point. We call this value $L_M(\boldsymbol{\varphi}, Q)$. If we use a consistent heuristic, then the A* algorithm explores nodes in order of decreasing values, as shown in Lemma 5.4. Then $L_M(\boldsymbol{\varphi}, Q)$ is the Mth largest value in the tree, or, if A* terminates within M steps, the value of an optimal fully specified policy. Next, we formalize $L_M(\boldsymbol{\varphi}, Q)$.

Definition 7.3. *Consider a search tree with a heuristic $Q\colon \Phi \to \mathbb{R}$ and some fixed partial policy $\boldsymbol{\varphi}$. We define the set of candidate policies $\Phi_c(\boldsymbol{\varphi}) = \{\boldsymbol{\varphi} <_E \boldsymbol{\varphi}' \wedge \sigma(\boldsymbol{\varphi}') \geq 1\}$. Let w be an ordered list over $\Phi_c(\boldsymbol{\varphi})$ such that $Q(w_i) \geq Q(w_j)$ for $i < j$, and let $k^* = \min\{k \mid w_k \in \Pi\}$ be the smallest index where w contains a fully specified policy. Then we define $L_M(\boldsymbol{\varphi}, Q) = Q(w_{\min\{k^*,M\}})$.*

Note that we only count partial policies with stage larger than 0, as we do not compute heuristic values for partial policies with stage 0; formally, we assign these nodes an infinite heuristic value. If $K = M$, then $L_M(\boldsymbol{\varphi}, Q) = Q(w_M)$ is the Mth largest heuristic value in the search tree. On the other hand, if $K = k^*$, then $L_M(\boldsymbol{\varphi}, Q) = Q(w^*_k) = \max_{\boldsymbol{\pi} \in \Phi_c(\boldsymbol{\varphi}) \cap \Pi} Q(\boldsymbol{\pi})$, i.e., the value of an optimal fully specified policy in Φ_c.

Recall that the heuristic Q_d can be written in terms of optimal values of Dec-POMDPs with a lower horizon. For the heuristic $Q_{M,d}$, we (recursively) substitute the precise values with the values obtained after early termination.

Definition 7.4 (Recursive heuristic with early termination)**.** *We define $Q_{M,d}(\boldsymbol{\varphi}) = \infty$ for $\sigma(\boldsymbol{\varphi}) = 0$. For $\sigma(\boldsymbol{\varphi}) \geq 1$, we define $Q_{M,d}(\boldsymbol{\varphi})$ inductively over the horizon. For $h = 1$, partial policies with $\sigma(\boldsymbol{\varphi}) \geq 1$ are fully specified and we define $Q_{M,d}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b, 1)$. For $h = \ell + 1$, $Q_{M,d}(\boldsymbol{\varphi})$ is*

$$\min\left\{V_{\boldsymbol{\varphi}}(b,t) + \sum_{\boldsymbol{\tau}\in\boldsymbol{\mathcal{O}}^t}\left[\Pr(\boldsymbol{\tau})L_M(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{M,d})\right], Q_{M,d}(\boldsymbol{\varphi}^-)\right\}, \quad \textit{where } t = \min\{d, \sigma(\boldsymbol{\varphi})\}.$$

Note that the $Q_{M,d}$ values used for computing $L_M(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$ are values with horizon $\ell + 1 - t \leq \ell$, so these are defined already. Hence, the recursion is well-founded.

Next, we show that terminating heuristics earlier leads to heuristics that are less tight.

Lemma 7.3 (Terminating earlier leads to larger heuristic values)**.** *Let $\boldsymbol{\varphi}$ be a partial policy. Then $Q_d(\boldsymbol{\varphi}) \leq Q_{M',d}(\boldsymbol{\varphi}) \leq Q_{M,d}(\boldsymbol{\varphi})$ for all $d \geq 1$ and for all $M' \geq M$.*

Proof. We use notation as in Definition 7.3. For any N, we can write

$$L_N(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{N,d}) = \max\left\{\max_{\boldsymbol{\pi}\in\Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})\cap\Pi} Q_{N,d}(\boldsymbol{\pi}), Q_{N,d}(w_N)\right\}.$$

We show the second inequality by induction on the horizon. For $h = 1$, $Q_{M',d}(\boldsymbol{\varphi})$ and $Q_{M,d}(\boldsymbol{\varphi})$ are defined in the same way, so they are equal. Now assume that $Q_{M',d}(\boldsymbol{\varphi}) \leq Q_{M,d}(\boldsymbol{\varphi})$ for all $h \leq \ell$. To prove that $Q_{M',d}(\boldsymbol{\varphi}) \leq Q_{M,d}(\boldsymbol{\varphi})$ for all partial policies $\boldsymbol{\varphi}$ with horizon $\ell + 1$, we apply induction on $\boldsymbol{\varphi}$. As base case for this induction, note that $Q_{M',d}(\boldsymbol{\varphi}) = Q_{M,d}(\boldsymbol{\varphi}) = \infty$ if $\sigma(\boldsymbol{\varphi}) = 0$. Now let $\boldsymbol{\varphi}$ be a partial policy for horizon $\ell + 1$ with $\sigma(\boldsymbol{\varphi}) \geq 1$, and write $t = \min\{d, \sigma(\boldsymbol{\varphi})\}$. Let $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t$ be given. We first show that $L_{M'}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{M',d}) \leq L_M(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{M,d})$. Let v and w denote the ordered list over $\Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$ corresponding to $Q_{M',d}$ and $Q_{M,d}$, respectively. By the induction hypothesis, we have $Q_{M,d}(\boldsymbol{\varphi}') \leq Q_{M,d}(\boldsymbol{\varphi}')$ for all $\boldsymbol{\varphi}' \in \Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$, since $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ and hence the partial policies in $\Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$ have horizon at most ℓ. Hence,

$$Q_{M',d}(v_{M'}) \leq Q_{M',d}(v_M) \leq Q_{M,d}(w_M),$$

where the second inequality follows from the fact that the $Q_{M',d}$-values are at most the $Q_{M,d}$-values, so the Mth largest $Q_{M',d}$-value is at most the Mth largest $Q_{M,d}$-value.

Since $Q_{M',d}(\boldsymbol{\pi}) \leq Q_{M,d}(\boldsymbol{\pi})$ for all $\boldsymbol{\pi} \in \Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) \cap \Pi$ by the induction hypothesis, it follows that

$$\begin{aligned} L_{M'}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{M',d}) &= \max\left\{\max_{\boldsymbol{\pi}\in\Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})\cap\Pi} Q_{M',d}(\boldsymbol{\pi}), Q_{M',d}(v_{M'})\right\} \\ &\leq \max\left\{\max_{\boldsymbol{\pi}\in\Phi_c(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})\cap\Pi} Q_{M,d}(\boldsymbol{\pi}), Q_{M,d}(w_M)\right\} = L_M(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{M,d}) \end{aligned}$$

for all $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t$. Finally, we have $Q_{M',d}(\boldsymbol{\varphi}^-) \leq Q_{M,d}(\boldsymbol{\varphi}^-)$ by the induction hypothesis corresponding to the inner induction on $\boldsymbol{\varphi}$. Together, this shows that $Q_{M',d}(\boldsymbol{\varphi}) \leq Q_{M,d}(\boldsymbol{\varphi})$, which completes the induction and hence the proof of the second inequality.

The first inequality follows from the second inequality, since we have $Q_d(\boldsymbol{\varphi}) = Q_{M',d}(\boldsymbol{\varphi})$ for M' sufficiently large (e.g. at least if M' is larger than the number of partial policies of the considered horizon h, and hence also larger than the number of partial policies of any smaller horizon). □

We now use this lemma to show that $Q_{M,d}$ is admissible and consistent.

Theorem 7.2. *$Q_{M,d}$ is admissible and consistent for $d \geq 1$ and $M \geq 1$.*

Proof. Since $Q_{M,d}(\boldsymbol{\varphi}) \geq Q_d(\boldsymbol{\varphi})$ and Q_d is admissible by Theorem 7.1, it follows that $Q_{M,d}$ is also admissible. To show that $Q_{M,d}$ is consistent, we note that $Q_{M,d}(\boldsymbol{\varphi}) \leq Q_{M,d}(\boldsymbol{\varphi}^-)$ holds by construction. Also, we have $Q_{M,d}(\boldsymbol{\pi}) \geq V_{\boldsymbol{\pi}}(b, h)$ since $Q_{M,d}$ is admissible.

Hence, it remains to show that $Q_{M,d}(\boldsymbol{\pi}) \leq V_{\boldsymbol{\pi}}(b,h)$. We show this by induction on the horizon h. For $h=1$, we have $Q_{M,d}(\boldsymbol{\pi}) = V_{\boldsymbol{\pi}}(b,h)$ by definition, since $\sigma(\boldsymbol{\pi}) = 1$. Now assume that $Q_{M,d}(\boldsymbol{\varphi}) \leq V_{\boldsymbol{\pi}}(b,h)$ for all $h \leq \ell$. Let $\boldsymbol{\pi}$ be a fully specified policy for horizon $\ell+1$. If $d \geq \ell+1$, then $Q_{M,d}(\boldsymbol{\pi}) = V_{\boldsymbol{\pi}}(b,\ell+1)$, as required. Otherwise, we have $d < \ell+1$, and hence

$$Q_{M,d}(\boldsymbol{\pi}) \leq V_{\boldsymbol{\pi}}(b,d) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^d} \left[\Pr(\boldsymbol{\tau}) L_M(\boldsymbol{\pi}|_{\boldsymbol{\tau}}, Q_{M,d})\right].$$

Note that $\boldsymbol{\pi}|_{\boldsymbol{\tau}}$ is also a fully specified policy, so $\Phi_c(\boldsymbol{\pi}|_{\boldsymbol{\tau}}) = \{\boldsymbol{\pi}|_{\boldsymbol{\tau}}\}$ and hence the induction hypothesis implies that $L_M(\boldsymbol{\pi}|_{\boldsymbol{\tau}}, Q_{M,d}) = Q_{M,d}(\boldsymbol{\pi}|_{\boldsymbol{\tau}}) \leq V_{\boldsymbol{\pi}|_{\boldsymbol{\tau}}}(b_{\boldsymbol{\tau}}, \ell+1-d)$. Hence, we conclude that $Q_{M,d}(\boldsymbol{\pi}) \leq V_{\boldsymbol{\pi}}(b,\ell+1)$, which completes the induction and hence the proof. □

7.2.2 Relation to the POMDP Heuristic

We now compare the heuristic $Q_{1,d}$ to the precomputed heuristics Q_{POMDP}.

Theorem 7.3. *$Q_{1,d}(\boldsymbol{\varphi}) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$ for all $d \geq 1$ and all partial policies $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) \geq 1$.*

Proof. We show the statement using induction on the horizon h of $\boldsymbol{\varphi}$. Let b denote the initial belief. If $h=1$, then partial policies $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) \geq 1$ are fully specified and then we have $Q_{1,d}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b,1) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$. Now assume that $Q_{1,d}(\boldsymbol{\varphi}) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$ for all $h \leq \ell$ and all partial policies $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) \geq 1$.

We now prove using induction on $\boldsymbol{\varphi}$ that $Q_{1,d}(\boldsymbol{\varphi}) \geq Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$ for all partial policies $\boldsymbol{\varphi}$ with horizon $\ell+1$, with equality if $\sigma(\boldsymbol{\varphi}) \geq 1$. For the base case, note that the inequality is trivial if $\sigma(\boldsymbol{\varphi}) = 0$, since then $Q_{1,d}(\boldsymbol{\varphi}) = \infty$ by definition. Now suppose that $\sigma(\boldsymbol{\varphi}) \geq 1$. Let $t = \min\{d, \sigma(\boldsymbol{\varphi})\}$. Then $t \geq 1$. We first show that

$$L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$$

for all $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t$. If $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) \geq 1$, then $L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) = Q_{1,d}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$, where the second equality follows from the induction hypothesis since $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ has horizon $\ell+1-t \leq \ell$.

If $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) = 0$, then $L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d})$ is the highest heuristic value $Q_{1,d}(\boldsymbol{\varphi}')$ among all $\boldsymbol{\varphi}'$ with $\boldsymbol{\varphi}|_{\boldsymbol{\tau}} <_E \boldsymbol{\varphi}'$ and $\sigma(\boldsymbol{\varphi}') \geq 1$. Since $Q_{1,d}$ is consistent, this highest heuristic value is achieved by a $\boldsymbol{\varphi}'$ that only specifies the joint action for stage 0. Hence, the computation corresponding to $L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d})$ corresponds to maximizing over all joint actions in stage 0 that agree with the partial action that $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ that specifies for stage 0. To formalize this, let $\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}} = \langle \varphi_1(\tau_1), \ldots, \varphi_i(\tau_i) \rangle$ be the partial action that $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ specifies for stage 0, and let $E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}})$ denote the set of all joint actions agreeing with $\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}}$:

$$E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}}) = \left\{ \langle \varphi_1(\tau_1), \ldots, \varphi_i(\tau_i), a_{i+1}, \ldots, a_n \rangle \;\middle|\; a_{i+1} \in \mathcal{A}_{i+1}, \ldots, a_n \in \mathcal{A}_n \right\}.$$

Finally, let $\boldsymbol{\varphi}'_{\mathbf{a}}$ be the partial policy corresponding to initial belief $\boldsymbol{\tau}$, with horizon $\ell+1-t$ that assigns joint action $\mathbf{a}$ in stage 0. Then we can write

$$\begin{aligned}
L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) &= \max_{\mathbf{a} \in E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}})} Q_{1,d}(\boldsymbol{\varphi}'_{\mathbf{a}}) \\
&= \max_{\mathbf{a} \in E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}})} Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}'_{\mathbf{a}}) \\
&= \max_{\mathbf{a} \in E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}})} \left[V_{\boldsymbol{\varphi}'_{\mathbf{a}}}(b_{\boldsymbol{\tau}}, 1) + \sum_{\mathbf{o} \in \boldsymbol{\mathcal{O}}} \Pr(\mathbf{o} \mid b_{\boldsymbol{\tau}}, \mathbf{a}) Q'_{\mathsf{POMDP}}((b_{\boldsymbol{\tau}})_{\mathbf{a},\mathbf{o}}, \ell - t, \langle\rangle) \right] \\
&= \max_{\mathbf{a} \in E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}})} \left[R(b_{\boldsymbol{\tau}}, \mathbf{a}) + \sum_{\mathbf{o} \in \boldsymbol{\mathcal{O}}} \Pr(\mathbf{o} \mid b_{\boldsymbol{\tau}}, \mathbf{a}) Q'_{\mathsf{POMDP}}((b_{\boldsymbol{\tau}})_{\mathbf{a},\mathbf{o}}, \ell - t, \langle\rangle) \right] \\
&= \max_{\mathbf{a} \in E_{\boldsymbol{\mathcal{A}}}(\mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}})} Q'_{\mathsf{POMDP}}\big(b_{\boldsymbol{\tau}},\ \ell{+}1{-}t,\ \mathbf{a}\big) \\
&= Q'_{\mathsf{POMDP}}\big(b_{\boldsymbol{\tau}},\ \ell{+}1{-}t,\ \mathbf{a}_{\boldsymbol{\varphi},\boldsymbol{\tau}}\big) \\
&= Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}).
\end{aligned}$$

In the first equality, we use the property of $L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d})$ shown earlier. The second equality follows from the induction hypothesis, since $\boldsymbol{\varphi}'_{\mathbf{a}}$ has horizon $\ell + 1 - t \leq \ell$. The third equality follows from the definition of Q_{POMDP} from Section 6.2, since $\sigma(\boldsymbol{\varphi}'_{\mathbf{a}}) = 1$, rewritten slightly using the correspondence between joint observations and joint observation histories of length 1. In the fourth equality we just rewrite $V_{\boldsymbol{\varphi}'_{\mathbf{a}}}(b_{\boldsymbol{\tau}}, 1) = R(b_{\boldsymbol{\tau}}, \mathbf{a})$. The fifth equality is the value iteration equation of Q'_{POMDP} from Section 6.2. The sixth equality follows from (repeatedly) applying the definition of Q'_{POMDP} for partial actions, and the final equality is the definition of Q_{POMDP} from Section 6.2, using that $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) = 0$. Hence, we conclude that $L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$ also holds if $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) = 0$, and hence it holds for all $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t$.

Write $u = \sigma(\boldsymbol{\varphi}) - t$. Note that $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) \in \{u, u+1\}$. The case $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) = u + 1$ occurs if $\boldsymbol{\varphi}$ specified actions for all LOHs of length $\sigma(\boldsymbol{\varphi})$ with prefix $\boldsymbol{\tau}$ (but, by definition, not yet on all LOHs of length $\sigma(\boldsymbol{\varphi})$). In that case, $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ does not specify actions for any LOH of length $\sigma(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) = u+1$, so $\mathbf{a}_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, \widetilde{\boldsymbol{\tau}} \cdot \widetilde{\mathbf{o}}} = \langle\rangle$ for any $\widetilde{\boldsymbol{\tau}} \cdot \widetilde{\mathbf{o}} \in \boldsymbol{\mathcal{O}}^{u+1}$ (with $\widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^u$). Hence, we can write

$$
\begin{aligned}
Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) &= V_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}}(b_{\boldsymbol{\tau}}, u{+}1) \; + \sum_{\widetilde{\boldsymbol{\tau}} \cdot \widetilde{\mathbf{o}} \in \boldsymbol{\mathcal{O}}^{u+1}} \Pr(\widetilde{\boldsymbol{\tau}}, \widetilde{\mathbf{o}}) Q'_{\mathsf{POMDP}}\big((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}} \cdot \widetilde{\mathbf{o}}}, \ell - t - u, \langle\rangle\big) \\
&= V_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}}(b_{\boldsymbol{\tau}}, u) + \sum_{\widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^u} [\Pr(\widetilde{\boldsymbol{\tau}}) R((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}}}, \widetilde{\mathbf{a}})] + \sum_{\widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^u} \Pr(\widetilde{\boldsymbol{\tau}}) \sum_{\widetilde{\mathbf{o}} \in \boldsymbol{\mathcal{O}}} \Pr(\widetilde{\mathbf{o}}) Q'_{\mathsf{POMDP}}\big((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}} \cdot \widetilde{\mathbf{o}}}, \ell{-}t{-}u, \langle\rangle\big) \\
&= V_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}}(b_{\boldsymbol{\tau}}, u) + \sum_{\widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^u} \Pr(\widetilde{\boldsymbol{\tau}}) \left[R((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}}}, \widetilde{\mathbf{a}}) + \sum_{\widetilde{\mathbf{o}} \in \boldsymbol{\mathcal{O}}} \Pr(\widetilde{\mathbf{o}}) Q'_{\mathsf{POMDP}}\big((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}}})_{\widetilde{\mathbf{a}}, \widetilde{\mathbf{o}}}, \ell{-}t{-}u, \langle\rangle\big) \right] \\
&= V_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}}(b_{\boldsymbol{\tau}}, u) + \sum_{\widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^u} \Pr(\widetilde{\boldsymbol{\tau}}) Q'_{\mathsf{POMDP}}\big((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}}}, \, \ell{-}t{-}u{+}1, \, \mathbf{a}_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, \widetilde{\boldsymbol{\tau}}}\big),
\end{aligned}
$$

where in the second and third line we shorten $\widetilde{\mathbf{a}} = \mathbf{a}_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, \widetilde{\boldsymbol{\tau}}}$. This shows that we can use the equation for $Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$ corresponding to policies of stage u, even if the stage $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ actually $u+1$ (but no actions for LOHs of length $u+1$ are specified).

Using the equality $L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}})$, we can now write

$$
\begin{aligned}
&V_{\boldsymbol{\varphi}}(b, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) \\
&\quad = V_{\boldsymbol{\varphi}}(b, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}) \\
&\quad = V_{\boldsymbol{\varphi}}(b, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \Pr(\boldsymbol{\tau}) \left[V_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}}(b_{\boldsymbol{\tau}}, u) + \sum_{\widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^u} \Pr(\widetilde{\boldsymbol{\tau}}) Q'_{\mathsf{POMDP}}\big((b_{\boldsymbol{\tau}})_{\widetilde{\boldsymbol{\tau}}}, \, \ell{-}t{-}u{+}1, \, \mathbf{a}_{\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, \widetilde{\boldsymbol{\tau}}}\big) \right] \\
&\quad = V_{\boldsymbol{\varphi}}(b, t+u) + \sum_{\boldsymbol{\tau} \cdot \widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^{t+u}} \Pr(\boldsymbol{\tau} \cdot \widetilde{\boldsymbol{\tau}}) Q'_{\mathsf{POMDP}}\big(b_{\boldsymbol{\tau} \cdot \widetilde{\boldsymbol{\tau}}}, \, \ell{-}t{-}u{+}1, \, \mathbf{a}_{\boldsymbol{\varphi}, \boldsymbol{\tau} \cdot \widetilde{\boldsymbol{\tau}}}\big) \\
&\quad = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}).
\end{aligned}
$$

Finally, the induction hypothesis implies that $Q_{1,d}(\boldsymbol{\varphi}^-) \geq Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}^-) \geq Q_{\mathsf{POMDP}}(\boldsymbol{\varphi})$, where the inequality holds since Q_{POMDP} is consistent. Hence, we have

$$
\begin{aligned}
Q_{1,d}(\boldsymbol{\varphi}) &= \min \left\{ V_{\boldsymbol{\varphi}}(b, t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \left[\Pr(\boldsymbol{\tau}) L_1(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{1,d}) \right], Q_{1,d}(\boldsymbol{\varphi}^-) \right\} \\
&= \min \left\{ Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}), Q_{1,d}(\boldsymbol{\varphi}^-) \right\} = Q_{\mathsf{POMDP}}(\boldsymbol{\varphi}).
\end{aligned}
$$

This completes the induction and hence the proof. □

7.2.3 Dynamic Termination

In Section 7.2.1, we have shown how heuristic computations can be terminated early statically, based on a fixed maximum number of iterations M. We can go further and also terminate the heuristic computation of $Q(\boldsymbol{\varphi})$ when the highest heuristic value of an open node is much smaller than the heuristic value of the parent $Q(\boldsymbol{\varphi}^-)$, which is an upper bound for $Q(\boldsymbol{\varphi})$ for consistent heuristics. We call this *dynamic termination*.

Specifically, let $u = Q(\boldsymbol{\varphi}^-)$. Then we terminate the heuristic computation after M iterations or if the new heuristic value v satisfies $v \leq u - \alpha \max\{|u|, 1\}$, where α is a parameter, called the *threshold*. The intuition for this is that a large drop in computing the heuristic value is sufficient to avoid expanding that node again (at least for a long time), so spending more time computing heuristics for this node is likely a waste of time.

Next, we formalize this heuristic, similarly to the formalization of static termination. We first define the generalization of $L_M(\boldsymbol{\varphi}, Q)$, which we denote by $L_{M,\alpha}(\boldsymbol{\varphi}, Q)$.

Definition 7.5. *Consider a search tree with a heuristic $Q\colon \Phi \to \mathbb{R}$ and some fixed partial policy $\boldsymbol{\varphi}$. We define the set of candidate policies $\Phi_c(\boldsymbol{\varphi}) = \{\boldsymbol{\varphi} <_E \boldsymbol{\varphi}' \wedge \sigma(\boldsymbol{\varphi}') \geq 1\}$. Let w be an ordered list over $\Phi_c(\boldsymbol{\varphi})$ such that $Q(w_i) \geq Q(w_j)$ for $i < j$, and let $k^* = \min\{k \mid w_k \in \Pi\}$ be the smallest index where w contains a fully specified policy.*

Write $u = Q(\boldsymbol{\varphi}^-)$ and $\underline{u} = u - \alpha \max\{|u|, 1\}$. Let $k' = \min\{k \mid Q(w_k) \leq \underline{u}\}$ be the smallest index for which the heuristic value $Q(w_k)$ is smaller than $\underline{u}$. If such index does not exist, then we set $k' = \infty$. Finally, we define $L_{M,\alpha}(\boldsymbol{\varphi}, Q) = Q(w_{\min\{k^*, k', M\}})$.

Using the value $L_{M,\alpha}(\boldsymbol{\varphi}, Q)$, we now define the recursive heuristic with dynamic termination, which we denote by $Q_{M,d,\alpha}$:

Definition 7.6 (Recursive heuristic with dynamic early termination)**.**
First, we define $Q_{M,d,\alpha}(\boldsymbol{\varphi}) = \infty$ for $\sigma(\boldsymbol{\varphi}) = 0$. For $\sigma(\boldsymbol{\varphi}) \geq 1$, we define $Q_{M,d,\alpha}(\boldsymbol{\varphi})$ inductively over the horizon. For $h = 1$, partial policies with $\sigma(\boldsymbol{\varphi}) \geq 1$ are fully specified and we define $Q_{M,d,\alpha}(\boldsymbol{\varphi}) = V_{\boldsymbol{\varphi}}(b, 1)$. For $h = \ell + 1$, $Q_{M,d,\alpha}(\boldsymbol{\varphi})$ is

$$\min\left\{V_{\boldsymbol{\varphi}}(b,t) + \sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^t} \left[\Pr(\boldsymbol{\tau}) L_{M,\alpha}(\boldsymbol{\varphi}|_{\boldsymbol{\tau}}, Q_{M,d,\alpha})\right], Q_{M,d,\alpha}(\boldsymbol{\varphi}^-)\right\}, \quad \textit{where } t = \min\{d, \sigma(\boldsymbol{\varphi})\}.$$

We note that $Q_{M,d,\alpha}(\boldsymbol{\varphi}) = Q_{M,d}(\boldsymbol{\varphi})$ if α is sufficiently large. In general, we always have $Q_{M,d,\alpha}(\boldsymbol{\varphi}) \geq Q_{M,d}(\boldsymbol{\varphi})$. Also, for $M = 1$ we always terminate heuristic computations after 1 step. Hence, dynamic termination has no effect for $M = 1$, so $Q_{1,d,\alpha}(\boldsymbol{\varphi}) = Q_{1,d}(\boldsymbol{\varphi})$ for any α.

7.2.4 Dynamically Increasing Depth

We now turn to another technical detail how we use these heuristics in practice. While Lemma 7.2 tells us that $Q_d(\boldsymbol{\varphi}) \leq Q_{d'}(\boldsymbol{\varphi})$ if $d \leq d'$, the inequality $Q_{M,d}(\boldsymbol{\varphi}) \leq Q_{M,d'}(\boldsymbol{\varphi})$ does not necessarily hold, i.e., taking the lowest depth does not always lead to the tightest heuristic when terminating heuristic computations early. We first give some intuition why $Q_{M,d}(\boldsymbol{\varphi}) \leq Q_{M,d'}(\boldsymbol{\varphi})$ might not hold, in particular if $\sigma(\boldsymbol{\varphi})$ is large. For larger d, a shortened policy $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ with $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^d$ specifies fewer actions. Hence, the observation histories (or more generally, clusters) on which $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$ does not yet specify an action are shorter, and hence in general have a higher probability. Hence, the M iterations are spent in a better way for larger d, namely on clusters with a higher probability, which have a larger impact on the (heuristic) value.

To overcome this weakness, we actually compute $Q_{d'}(\boldsymbol{\varphi})$ for all $d' \geq d$. Computing the heuristic $Q_d(\boldsymbol{\varphi})$ is much more expensive than computing $Q_{d'}(\boldsymbol{\varphi})$ for $d' > d$, so the additional runtime for computing heuristics for higher depths is negligible. Each time we expand a node $\boldsymbol{\varphi}$, we update d to $\arg\min_{d'} Q_{d'}(\boldsymbol{\varphi})$ (taking the largest value of d' in case of a tie) for all descendants of that node.

7.3 Storing and Reusing Heuristics

Next, we turn to an important aspect of our implementation, namely how we reuse heuristics.

A key feature of our algorithms is that heuristics are computed on-the-fly, in contrast to precomputing them. Recall that we solve the Dec-POMDP for smaller horizons and different initial beliefs. The combination of horizon and initial belief (and initial part of the policy) can appear in multiple computations. To avoid recomputing those heuristic values, we memoize all values. To enable finding values that we already computed, we characterize each heuristic as a tuple containing 1) the horizon, 2) the initial belief, 3) an index describing the partial policy, and 4) a list of indices describing the clustering structure at each stage.

We additionally reuse computations for which we completed the computation of Q_d, i.e., for which we did not terminate early. Those values can be reused when evaluating extended policies. In particular, whenever we do not terminate early, we not only find a heuristic value, but also a fully specified policy $\boldsymbol{\pi}$. Then, for all $\boldsymbol{\varphi}'$ satisfying $\boldsymbol{\varphi} <_E \boldsymbol{\varphi}' <_E \boldsymbol{\pi}$ we have $Q(\boldsymbol{\varphi}) = Q(\boldsymbol{\varphi}') = Q(\boldsymbol{\pi})$. Hence, we can also store the heuristic value for all such $\boldsymbol{\varphi}'$ without explicitly computing them. In practice, we only store it for policies $\boldsymbol{\varphi}'$ of the same stage as clustering in later stages may depend on the observations that are revealed.

8 Policy-Finding Multi-Agent A* (PF-MAA*)

Next, we turn to *policy-finding multi-agent A* (PF-MAA*)*, our policy-finding algorithm. In Section 8.1, we introduce *queue pruning*, which is a method to ensure that A* finds a (possibly suboptimal) solution faster. In Section 8.2, we introduce heuristics tailored to policy-finding. Finally, in Section 8.3, we explain how we compute the heuristics for PF-MAA* in practice using the concept of *horizon reduction*.

PF-MAA* applies small-step MAA* with clustered sliding window memory (Section 4.5), queue pruning (Section 8.1) and the heuristic $Q_{\mathsf{maxr},r}$ or $Q_{\mathsf{MDP},r}$ (Section 8.2).

8.1 Queue Pruning

For our policy-finding algorithm PF-MAA*, we prune the priority queue to find a good policy faster. In particular, we limit the number of policies expanded to $h \cdot L$ for some L, while guaranteeing that we find a fully specified policy. We prune the A* priority queue using a progress measure *prog*. Partial policies in the priority queue are still processed in descending order of heuristic value. However, a partial policy $\boldsymbol{\varphi}$ is only expanded if $\mathit{prog}(\boldsymbol{\varphi}) \geq N$, where N is the number of policies expanded already. Otherwise, the partial policy $\boldsymbol{\varphi}$ is pruned, i.e. removed from the queue without further processing.

Algorithm 3 shows in pseudocode how queue pruning is done for A* algorithms in general. Alternatively, we can see queue pruning as a modification of the `Next` operator in GMAA*, where $\texttt{Next}(\boldsymbol{\varphi})$ remains unchanged if $\mathit{prog}(\boldsymbol{\varphi}) \geq N$, but we set $\texttt{Next}(\boldsymbol{\varphi}) = \emptyset$ if $\mathit{prog}(\boldsymbol{\varphi}) < N$.

Algorithm 3 A* with queue pruning

Input: Rooted search tree $T = (\mathcal{V}, \mathcal{E})$, heuristic $Q\colon \mathcal{V} \to \mathbb{R}$, progress measure $\mathit{prog}\colon \mathcal{V} \to \mathbb{R}$
Output: An $f \in \mathcal{F}$, where $\mathcal{F} \subseteq \mathcal{V}$ is the set of leaves of T

```
L ← PriorityQueue()                          ▷ sorted in descending order by Q-value
N ← 0                                        ▷ number of nodes expanded
L.push(root(T))
while true do
  v ← L.pop()
  if v ∈ F then
    return v
  if prog(v) < N then                        ▷ prune nodes with insufficient progress
    continue
  for all children v' of v in T do
    L.push(v')
  N ← N + 1
```

Necessity of queue pruning. We now explain why queue pruning is necessary for PF-MAA*. When running (small-step) MAA* to find an optimal policy, it is not possible to prune the priority queue. It can be observed empirically that too many nodes are expanded even before the first fully specified policy is added to the priority queue, to be able to find a policy in reasonable time. Hence, we have to limit the number of policies expanded in the shallow levels of the search tree. In fact, we really have to limit the number of policies expanded up to each level of the tree, to ensure that the algorithm will explore deeper nodes in the search tree sufficiently fast. To be precise, since we want to make sure that the algorithm finds a fully specified policy after $h \cdot L$ policy expansions, we make sure that the algorithm finds a policy with stage $\sigma(\varphi) \geq t$ after $t \cdot L$ policy expansions for each $1 \leq t \leq h$.

8.1.1 Definition of the Progress Measure

Having explained the general concept of queue pruning, we turn to the specific progress measure *prog* that is used to determine which policies are pruned.

There are two design goals for the progress measure *prog*:

- It has to limit the number of policies expanded up to each level of the search tree, to ensure that a fully specified policy is found within $h \cdot L$ policy expansions. We call this *correctness*.
- It should prune the least promising policies.

The first design goal is met by using a progress measure that increases by at least 1 each time we go down a level in the tree. We prove this for our progress measure in Section 8.1.2.

We now focus on the intuition for designing a progress measure that meets the second design goal. When using a consistent heuristic for policy finding, the heuristic values by definition go down as we go deeper in the search tree. Intuitively, this holds because a larger part of the heuristic value becomes the actual realized reward in the stages for which the policy is already specified, rather than the overestimation provided by the heuristic for remaining stages (for which the policy is not specified yet). So consistent heuristics are more optimistic for partial policies for which fewer actions (or actions for clusters with lower probability) have been specified. Hence, a deep policy with some heuristic value is more likely to have a good policy as descendant than a shallow policy with the same heuristic value. This explains why it is preferable to prune the shallow node when aiming to find a good policy.

First attempt: progress measure *prog′*. Due to differences in the number of clusters per stage between different policies, we should however not directly use the depth of the node in the search tree (which is just the total number of LOHs to which an action is assigned) as progress measure, but rather take into account how many stages this represents, and within each stage, for how many agents the policy has been fully specified. A first idea for a progress measure could therefore be as follows: Let $\boldsymbol{\varphi}$ be a partial policy such that $\boldsymbol{\varphi}$ has specified an action for all clusters of length $\sigma(\boldsymbol{\varphi}) - 1$, for all clusters of length $\sigma(\boldsymbol{\varphi})$ for i out of n agents, and for c out of $|C_{i+1,\sigma(\boldsymbol{\varphi})}|$ clusters of length $\sigma(\boldsymbol{\varphi})$ of agent $i+1$. Then we could consider

$$prog'(\boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + c \cdot \frac{L}{n \left|C_{i+1,\sigma(\boldsymbol{\varphi})}\right|}.$$

The progress measure *prog′* aims to divide the available iterations evenly over the stages, within each stage evenly over the agents, and per agent evenly over the clusters of the agent in that stage.

Progress measure *prog*. We now turn to the actual progress measure *prog* that we use. With *prog*, we also take into account the probability of each of the clusters for the agent $i+1$ for which we currently specify actions. Specifically, *prog* is defined as

$$prog(\boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + c + p \cdot \left(\frac{L}{n} - \left|C_{i+1,\sigma(\boldsymbol{\varphi})}\right|\right).$$

where p is the probability that the LOH of agent $i+1$ is in one of the first c clusters, i.e. the probability of the clusters of agent $i+1$ to which $\boldsymbol{\varphi}$ already assigns an action. The progress measure *prog′* gives comparable, but slightly worse results than the progress measure *prog*.

To provide intuition for the progress measure *prog*, we note the following. The contribution of the overestimation of the heuristic on a given cluster to the overall heuristic value, is proportional to the probability of that cluster. Hence, if so far we mostly assigned actions to clusters with small probability (for the current agent), the heuristic value will be more optimistic, because the overestimations that we reduced by assigning actions have a smaller weight. Hence, a smaller p should correspond to a smaller progress measure, so that it is pruned rather than a partial policy with the same heuristic value progressed to the same stage and agent, but which has assigned actions to clusters with a larger probability.

However, when completely removing c from the progress measure and replacing it by $p \cdot \frac{L}{n}$, we no longer ensure that the queue always contains a policy which is not pruned. Therefore, we also include c in the progress measure (which is at most $\left|C_{i+1,\sigma(\boldsymbol{\varphi})}\right|$), and include the probability p with the remaining weight $\left(\frac{L}{n} - \left|C_{i+1,\sigma(\boldsymbol{\varphi})}\right|\right)$. We assume that $L \geq n|C_{i,t}|$, which implies that

this weight is nonnegative. Then the progress measure always increases by at least 1 when going down a level in the tree, and hence the queue always contains a policy which is not pruned. We show this next.

8.1.2 Correctness of the Progress Measure

We now show that our progress measure is correct, i.e. that a fully specified policy is eventually found. The definition of the progress measure has been chosen to ensure that $\mathit{prog}(\boldsymbol{\varphi}') \geq \mathit{prog}(\boldsymbol{\varphi})+1$ if $\boldsymbol{\varphi}'$ is a child of $\boldsymbol{\varphi}$ in the search tree. Hence, if $\boldsymbol{\varphi}$ is expanded, then its children are possible candidates for expansion in the next iteration. We formalize this in two lemmas and a corollary.

Lemma 8.1. *Assume that $L \geq n|C_{i,t}|$ for all $0 \leq t \leq h-1$ and all $i \in \mathcal{D}$. Let $\boldsymbol{\varphi}$ be a partial policy and let $\boldsymbol{\varphi}'$ be a child of $\boldsymbol{\varphi}$ in the small-step search tree. Then $\mathit{prog}(\boldsymbol{\varphi}') \geq \mathit{prog}(\boldsymbol{\varphi}) + 1$.*

The proof of this lemma, which we give in Appendix A.3, consists of a case analysis, depending on whether we move to the next cluster for the same agent, the next agent for the same stage or the next stage. This lemma implies that it is an invariant of the algorithm that the queue always contains a policy which is not pruned.

Lemma 8.2. *Assume that $L \geq n|C_{i,t}|$ for all $0 \leq t \leq h-1$ and all $i \in \mathcal{D}$. Until a fully specified policy is found, the queue always contains a policy $\boldsymbol{\varphi}$ such that $\mathit{prog}(\boldsymbol{\varphi}) \geq N$, where N is the number of policies expanded already.*

Proof. Initially, we have $N = 0$ and the queue starts with a policy $\boldsymbol{\varphi}$ with $\mathit{prog}(\boldsymbol{\varphi}) \geq 0$.[7] Now suppose that we expand some policy $\boldsymbol{\varphi}$ when N policies are expanded already. Then $\mathit{prog}(\boldsymbol{\varphi}) \geq N$. If $\boldsymbol{\varphi}$ is not fully specified, then we add its children $\boldsymbol{\varphi}'$ to the queue which satisfy

$$\mathit{prog}(\boldsymbol{\varphi}') \geq \mathit{prog}(\boldsymbol{\varphi}) + 1 \geq N + 1$$

by Lemma 8.1. Since $N+1$ policies are expanded after expanding $\boldsymbol{\varphi}$, this completes the proof. □

Corollary 8.1. *Assume that $L \geq n|C_{i,t}|$ for all $0 \leq t \leq h-1$ and all $i \in \mathcal{D}$. Then PF-MAA* finds a fully specified policy within $h \cdot L$ policy expansions.*

Proof. For any partial policy $\boldsymbol{\varphi}$, we have $\mathit{prog}(\boldsymbol{\varphi}) \leq h \cdot L$, with equality if and only if $\boldsymbol{\varphi}$ is fully specified. Now suppose that we have already expanded $h \cdot L$ policies, but have not yet found a fully specified policy. Then Lemma 8.2 implies that the queue contains a policy with $\mathit{prog}(\boldsymbol{\varphi}) \leq h \cdot L$, which is hence a fully specified policy. Moreover, we prune all policies which are not fully specified policy, so after $h \cdot L$ policy expansions PF-MAA* returns the fully specified policy with the highest value in the queue if it has not returned a fully specified policy earlier already. □

8.2 Heuristics for Policy-Finding

Next, we discuss the heuristics used in policy-finding multi-agent A* (PF-MAA*). Recall from Section 8.1 that in PF-MAA*, we prune the priority queue used by the A* search. Due to this pruning, heuristics that are useful for exact solving (or finding upper bounds) are not necessarily useful for policy-finding. Before discussing the actual heuristics, we therefore discuss some properties which make a heuristic useful for policy-finding.

Highest heuristic values for good policies. The most important property of a heuristic for policy-finding is that 'good' policies (i.e., partial policies leading to (near-)optimal policies) have the highest heuristic values, to ensure that the search is steered towards these policies, while 'bad' policies are pruned. In particular, it is sufficient if the heuristic overestimates all policies by the same amount, or overestimates good policies more than bad policies. On the other hand, it is problematic if bad policies are initially (i.e., for low stages) overestimated more compared to good policies of the same stage, even if this is corrected in the next stage. Namely, the good policies are then already pruned when reaching the next stage. This is in stark contrast with heuristics used for exact solving, where it is acceptable if bad policies are initially overestimated more, as long as this is corrected in the next stage.

[7] For heuristic computations, the initial policy $\boldsymbol{\varphi}$ in the queue is not necessarily the empty policy.

Computationally cheap. Another useful property for a heuristic for policy-finding is that it is cheap to compute, as this allows for more exploration of the search space within given time and memory limits. As a consequence of desiring a computational cheap heuristic, the heuristics are often rather loose. However, since we prune policies, is not necessary to use a tight heuristic to limit the number of nodes expanded. By contrast, for exact solving the fastest option can be to use a very expensive heuristic, if the number of nodes expanded is prohibitive otherwise.

Value information gathering and communication actions. To be able to steer the search towards good policies, a heuristic for policy-finding should sufficiently value information gathering and communication actions. Information gathering actions are actions that provide relatively much information about the state through the (local) observations, while communication actions allow agents to transmit (a summary of) their local observations to other agents. Typically, these actions are costly. This makes the MDP heuristic typically a poor choice for policy-finding, since the effect of gathering information is not reflected in the estimate of future rewards provided by the MDP heuristic, while the costs of the information gathering action are taken into account. Similarly, in problems where communication actions are important, also the POMDP heuristic is likely to be perform poorly. In practice, the policy-finding heuristics that we use take into account the effects of these actions on the near future, by modelling the near future as a Dec-POMDP.

8.2.1 Maximum Reward Heuristic

We use the following simple *maximum reward heuristic* $Q_{\mathsf{maxr},r}(\boldsymbol{\varphi})$, where $\sigma(\boldsymbol{\varphi}) < r$. This heuristic computes a Dec-POMDP heuristic for $\boldsymbol{\varphi}$ with horizon r, and upper bounds the reward over the remaining $h - r$ stages by the maximum reward (over all state-action pairs) per stage. We now introduce some notation. We say that the *horizon* of a partial policy is t if the domain of each local partial policy φ_i is $\mathcal{O}_i^{\leq t-1}$. Also, we let $\boldsymbol{\varphi}|^{\leq r}$ be the policy where the horizon is changed to r, i.e. the domain of each local partial policy φ_i is changed to $\mathcal{O}_i^{\leq r-1}$. Let $R_{\max} = \max_{(s,\mathbf{a})\in\mathcal{S}\times\boldsymbol{\mathcal{A}}} R(s,\mathbf{a})$.

Definition 8.1 (Maximum reward heuristic)**.** *Let $\boldsymbol{\varphi}$ be a policy with horizon h and $\sigma(\boldsymbol{\varphi}) < r$. Let $M \geq 1$ and $d \geq 1$ be given. The* maximum reward heuristic *is defined by*

$$Q_{\mathsf{maxr},r}(\boldsymbol{\varphi}) = Q_{M,d}\left(\boldsymbol{\varphi}|^{\leq r}\right) + (h-r)R_{\max}.$$

Although this heuristic is clearly not tight in general, it is still useful when finding policies: search guided by this heuristic is essentially a local search, choosing policies yielding good reward over the next $r - \sigma(\boldsymbol{\varphi})$ stages. In particular, this heuristic overestimates the future beyond stage r equally for all partial policies.

8.2.2 Terminal Reward MDP Heuristic

To take into account the effect of actions on later stages to some extent, we can also use the *terminal reward MDP heuristic* $Q_{\mathsf{MDP},r}(\boldsymbol{\varphi})$, where $r > \sigma(\boldsymbol{\varphi})$. This is a Dec-POMDP heuristic for $\boldsymbol{\varphi}$ with horizon r for a Dec-POMDP with terminal rewards, where these terminal rewards represent the QMDP value for the remaining $h - r$ stages. The QMDP value is the MDP value where the agents take the same joint action in stage 0 [Littman *et al.*, 1995]. To define this heuristic formally, let $Q'_{\mathsf{MDP}}(s,t,\mathbf{a})$ be the optimal value of the underlying MDP with initial state s and horizon t, given that action $\mathbf{a}$ is taken in stage 0.

Definition 8.2 (Terminal reward MDP heuristic)**.** *Let $\boldsymbol{\varphi}$ be a policy with horizon h and $\sigma(\boldsymbol{\varphi}) < r$. If $h > r$, we consider the given Dec-POMDP with horizon r and a terminal reward $R_T \colon \Delta(\mathcal{S}) \to \mathbb{R}$, awarded based on the final joint belief after r stages. The terminal reward is defined by*

$$R_T(b) = \max_{\mathbf{a}\in\boldsymbol{\mathcal{A}}} \sum_{s\in\mathcal{S}} b(s) \cdot Q'_{\mathsf{MDP}}(s, h-r, \mathbf{a}). \tag{8.1}$$

The awarded terminal reward is then $\sum_{\boldsymbol{\tau}\in\boldsymbol{\mathcal{O}}^r} \Pr(\boldsymbol{\tau}) R_T(b_{\boldsymbol{\tau}})$. Finally, let $M \geq 1$ and $d \geq 1$ be given. Then we define the terminal reward MDP heuristic *$Q_{\mathsf{MDP},r}(\boldsymbol{\varphi})$ as the heuristic $Q_{M,d}\left(\boldsymbol{\varphi}|^{\leq r}\right)$ corresponding to this Dec-POMDP with horizon r and terminal rewards. If $h \leq r$, then the terminal reward MDP heuristic $Q_{\mathsf{MDP},r}(\boldsymbol{\varphi})$ is simply $Q_{M,d}(\boldsymbol{\varphi})$.*

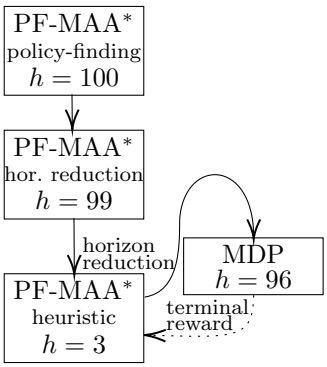


Figure 11: Schematic call graph for PF-MAA*, for $r = 3$. A dotted line indicates that a result is provided.

8.3 Horizon Reduction

In PF-MAA*, heuristic values are computed recursively, similarly to RS-MAA*. However, the heuristics introduced in Section 8.2 only apply if $\sigma(\boldsymbol{\varphi}) < r$. To avoid having to compute heuristics for large horizons, PF-MAA* first applies a *horizon reduction*, reducing the computation of a heuristic for a horizon h' Dec-POMDP to a computation for horizon r Dec-POMDP with terminal rewards representing the remaining $h' - r$ stages. For PF-MAA*, this terminal reward is an MDP value or just $h' - r$ times the maximal reward, as explained in Section 8.2.

To further explain how heuristics are computed, we now turn to the two different modes for A* and the concept of *horizon reduction*. Specifically, we have a *policy-finding mode*, a *horizon reduction step* and a *heuristic (computation) mode*. In the initial call, PF-MAA* is in the policy-finding mode. In all further (recursive) calls, PF-MAA* makes a horizon reduction step when called with a horizon $h' > r$ and are in heuristic mode when called with a horizon $h' \leq r$. In policy-finding mode, no horizon reduction is applied, since this precludes finding a policy for the full horizon h. In a horizon reduction step, we reduce computing a heuristic for a horizon h' Dec-POMDP to computing a heuristic for a horizon r Dec-POMDP with terminal rewards representing the remaining $h' - r$ stages. In heuristic mode, we do not have to find a policy, but only return an upper bound for the value of an optimal policy.

In policy-finding and heuristic mode, PF-MAA* behaves similarly to RS-MAA*. In particular, we compute a heuristic corresponding to revealing the first d joint observations for each agent, which in practice means that we consider all possible joint observation histories $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^d$ at stage d. For each observation history $\boldsymbol{\tau}$, we consider the shortened partial policy $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$, which gives the actions the agents take when receiving particular observations from stage d onwards. Then we compute heuristics for horizon $h' - d$ with initial policy $\boldsymbol{\varphi}|_{\boldsymbol{\tau}}$, and weigh these with the probabilities of the observation history $\boldsymbol{\tau}$ to compute a heuristic. In both modes, d starts at 1. In heuristic mode, d increases to d' when using the $Q_{d'}$ heuristic. In policy-finding mode for PF-MAA*, d increases further as the stage of the partial policies $\boldsymbol{\varphi}$ in the queue increase.

Call graph for PF-MAA*. Figure 11 shows a schematic call graph for PF-MAA* when using the $Q_{\mathsf{MDP},r}$ heuristic with $r = 3$. Initially, we have $d = 1$ and call PF-MAA* in horizon reduction mode with horizon 99. Then the horizon is reduced to 3, and the corresponding call to PF-MAA* in heuristic mode uses the terminal rewards, which are values of an MDP with horizon 96, when needed. In the figure, we only show these initial calls. Later in the execution of the algorithm, d increases; then the initial call of PF-MAA* in policy-finding mode makes calls to PF-MAA* with horizon $100 - d$, which will in turn make a horizon reduction if $100 - d \geq r$, and use values of an MDP with horizon $100 - d - r$ as terminal rewards.

9 Terminal Reward Multi-Agent A* (TR-MAA*)

Terminal reward multi-agent A (TR-MAA*)* computes upper bounds. It applies small-step MAA* with lossless clustering and a terminal reward heuristic, which we introduce in Section 9.1. In Section 9.2, we explain the horizon reduction in the context of TR-MAA*.

9.1 Heuristics for Tight Upper Bounds

We continue by discussing the heuristics used in our upper bound algorithm, terminal reward multi-agent A* (TR-MAA*). Among the heuristics used so far in the literature, only the MDP heuristic [Littman *et al.*, 1995] can be computed effectively for large horizons. However, the MDP heuristic is generally a very loose overestimation. Tighter heuristics, such as the POMDP heuristic and the BG heuristic, cannot be precomputed and stored for large horizons within reasonable time and memory limits [Oliehoek *et al.*, 2013]. Similarly, even when using the cheapest recursive heuristic $Q_{1,\infty}$, RS-MAA* reaches time or memory limits on some benchmarks before even computing the first heuristic value.

The main reason for the infeasiblity of computing the POMDP heuristic and the BG heuristic is that for large horizons h, the number of reachable beliefs from the initial belief in h stages is too large. Although the number of reachable beliefs is typically much smaller than the theoretical maximum $O\left((|\boldsymbol{\mathcal{A}}||\boldsymbol{\mathcal{O}}|)^{h-1}\right)$, it is often still exponential in h, and hence infeasible for large h. Due to its on-the-fly heuristic computations, RS-MAA* can avoid considering all reachable beliefs. Nevertheless, RS-MAA* still needs to consider at least all beliefs reachable by all optimal policies, of which there are $O\left(|\boldsymbol{\mathcal{O}}|^{h-1}\right)$ per policy. While this upper bound is again not always reached, even the number of reachable beliefs by an optimal policy grows exponentially with the horizon for several of the standard Dec-POMDP benchmarks.

To design a heuristic that scales better with the horizon, we must therefore avoid exploring all these beliefs. We do this by periodically revealing the state. If the state is revealed every r stages, then all beliefs that we encounter in heuristic computations are reachable in r stages from a belief that assigns probability 1 to some state. Hence, there are at most $O\left(|\mathcal{S}|(|\boldsymbol{\mathcal{A}}||\boldsymbol{\mathcal{O}}|)^{r-1}\right)$ such beliefs. Importantly, this upper bound is independent of the horizon. In practice, we model our heuristic using the general framework introduced in Section 6.3 and compute the heuristic using terminal rewards. For this reason, we call the heuristics *terminal reward heuristics*. Empirically, these heuristics are usually tighter than the POMDP heuristic.

While the overestimation due to revealing a stage can be rather large, it only happens once per r stages. Moreover, we can ensure that the state information is actually only used from stage $r+1$ onwards, to make the heuristic tighter. Finally, as a new insight in this thesis (compared to Koops *et al.* [2024]), we present *aperiodic terminal reward heuristics*, where there is a choice between revealing state $r-1$ or state r. This allows the heuristic to reveal the less valuable state information by taking the minimum over the two options.

9.1.1 Terminal Reward Heuristics

In our terminal reward heuristics, we allow policies to depend on the rth state from some stage r onwards. We can then split the original problem with horizon h as a Dec-POMDP with horizon $r < h$ and a terminal reward representing the remaining $h-r$ stages. This is illustrated in Figure 12 (on the next page) on the left. The terminal reward is a Dec-POMDP value, with horizon $h-r$ and the revealed state as initial belief. This is similar to Eq. (8.1), but using Dec-POMDP values instead of MDP values. Since we reveal the state, the agents can use more information to decide which actions they take. This therefore gives an upper bound.

To formalize revealing the rth state using our framework from Section 6.3, define the information function TR'_r by $\mathsf{TR}'_r(i, \boldsymbol{\tau}, \boldsymbol{s}) = \tau_i$ if $(\boldsymbol{\tau}, \boldsymbol{s}) \in \boldsymbol{\mathcal{O}}^v \times \mathcal{S}^{v+1}$ with $v < r$, while $\mathsf{TR}'_r(i, \boldsymbol{\tau}, \boldsymbol{s}) = (\tau_i, s^r)$ if $(\boldsymbol{\tau}, \boldsymbol{s}) \in \boldsymbol{\mathcal{O}}^v \times \mathcal{S}^{v+1}$ with $v \geq r$. Then we can write this heuristic as $Q_{\mathsf{TR}'_r}$. In particular, the heuristic is admissible and consistent. We use the prime in TR'_r as we refine this heuristic later.

Note that rather than revealing only the rth state, we can equivalently reveal both the rth state and the first r joint observations, since these joint observations give no further information when

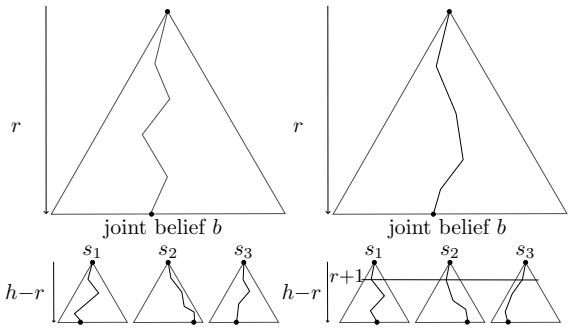


Figure 12: Revealing the state for a joint belief b at stage r over three states s_1, s_2, s_3. On the left the state is revealed at stage r, on the right it is revealed at stage $r+1$. In the latter case, the three policies are forced to take same action in stage 0 from belief b.

already having state information. Similarly to the recursive heuristics, the problem therefore decomposes into independent subproblems after stage r, although the optimal values of these subproblems are terminal rewards for this heuristic.

In practice, however, we do not use the optimal values of these subproblems, but values computed using an early terminated A* search for the subproblem, as in Section 7.2. We formalize this computation using terminal rewards for the tighter version of the heuristic.

Revealing the state at stage $r+1$. We now explain how to compute a tighter variant of this heuristic, where we allow policies to depend on the rth joint belief from stage r onwards and on the rth state from stage $r+1$ onwards. This is illustrated in Figure 12 on the right.

We now discuss how to compute this heuristic. We consider this problem as a Dec-POMDP with horizon r and a terminal reward depending on the joint belief. Since we reveal the joint belief b in stage r, we can compute each of these terminal rewards separately. We compute for each state s and each joint action $\mathbf{a}$, a heuristic $Q(s, h{-}r, \mathbf{a})$ for the Dec-POMDP with horizon $h-r$ where the initial state is s and the agents take action $\mathbf{a}$ in stage 0. To define this heuristic, let $\boldsymbol{\varphi}'_{s,h-r,\mathbf{a}}$ denote the partial policy of horizon $h-r$ corresponding to the initial belief assigning probability 1 to state s, where it is already fixed that the agents take action $\mathbf{a}$ in stage 0.

Definition 9.1 (Terminal reward heuristic)**.** *Let $r \geq 1$ be given and let $\boldsymbol{\varphi}$ be a policy with horizon h and $\sigma(\boldsymbol{\varphi}) < r$. Let $M \geq 1$ and $d \geq 1$ be given. We define the* terminal reward heuristic $Q_{\mathsf{TR},r}$ *recursively over the horizon h of $\boldsymbol{\varphi}$. If $h \leq r$, then we define $Q_{\mathsf{TR},r}(\boldsymbol{\varphi}) = Q_{M,d}(\boldsymbol{\varphi})$. If $h > r$, we consider the given Dec-POMDP with horizon r and a terminal reward $R_T \colon \Delta(\mathcal{S}) \to \mathbb{R}$, awarded based on the final joint belief after r stages. The terminal reward is defined by*

$$R_T(b) = \max_{\mathbf{a} \in \boldsymbol{\mathcal{A}}} \sum_{s \in \mathcal{S}} b(s) \cdot Q(s, h{-}r, \mathbf{a}), \tag{9.1}$$

where $Q(s, h{-}r, \mathbf{a}) = Q_{\mathsf{TR},r}(\boldsymbol{\varphi}'_{s,h-r,\mathbf{a}})$. The awarded terminal reward is then $\sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^r} \Pr(\boldsymbol{\tau}) R_T(b_{\boldsymbol{\tau}})$. Finally, we define the terminal reward heuristic $Q_{\mathsf{TR},r}(\boldsymbol{\varphi})$ as the heuristic $Q_{M,d}\big(\boldsymbol{\varphi}|^{\leq r}\big)$ corresponding to this Dec-POMDP with horizon r and terminal rewards.

Note that taking the maximum over the actions outside of the weighted sum in Equation (9.1) ensures that the same action is taken in stage 0, i.e. that the state information is not used in stage 0 (which is stage r of the original Dec-POMDP). Next, we show that $Q_{\mathsf{TR},r}$ is admissible.

Theorem 9.1. *$Q_{\mathsf{TR},r}$ is admissible.*

Proof. We use induction on the horizon h of $\boldsymbol{\varphi}$. If $h \leq r$, then $Q_{\mathsf{TR},r}(\boldsymbol{\varphi}) = Q_{M,d}(\boldsymbol{\varphi})$, so the admissibility of $Q_{\mathsf{TR},r}$ follows from the admissibility of $Q_{M,d}$ (Theorem 7.2).

We now first argue that the tighter heuristic where we would use the precise optimal values for the heuristic $Q(s, h{-}r, \mathbf{a})$ in Equation (9.1) and compute the optimal value of Dec-POMDP

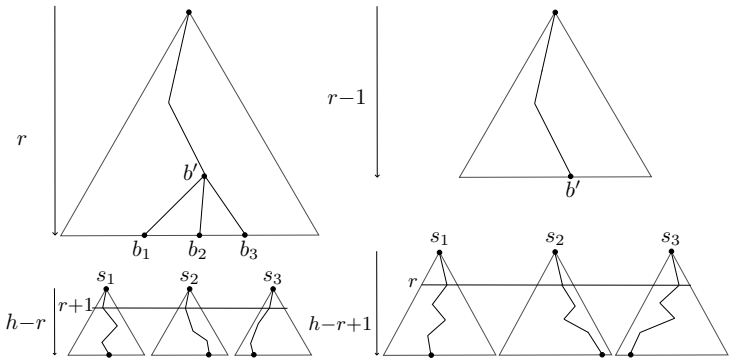


Figure 13: In the aperiodic terminal reward heuristic, we consider two options for the belief b': either we reveal the rth joint belief (b_1, b_2 or b_3) from stage r onwards and the rth state from stage $r+1$ onwards, or we reveal the $(r-1)$th state from stage r onwards.

with horizon r and terminal rewards, is admissible. Namely, we can formalize this heuristic using our framework from Section 6.3. Define the information function TR_r by

$$\mathsf{TR}_r(i, \mathbf{o}^{[1,v]}, \boldsymbol{s}) = \begin{cases} o_i^{[1,v]} & \text{if } v < r, \\ \mathbf{o}^{[1,v]} & \text{if } v = r, \\ (o_i^{[1,v]}, \mathbf{o}_{\neq i}^{[1,r]}, s^r) & \text{if } v > r. \end{cases}$$

Then we can write this heuristic as Q_{TR_r}, which is hence admissible. Note that we include $\mathbf{o}_{\neq i}^{[1,r]}$ for $v > r$ in the information function to ensure that the agent in particular knows $\mathbf{o}^{[1,r]}$, which was the information used by agent i to decide on the action in stage r.

In $Q_{\mathsf{TR},r}$, we instead use heuristics for the optimal values for $Q(s, h-r, \mathbf{a})$. These heuristics are upper bounds since $Q_{\mathsf{TR},r}$ is admissible for horizon $h-r$ by the induction hypothesis. In addition, use a heuristic for the optimal value of Dec-POMDP with horizon r and terminal rewards. This is also an upper bound since $Q_{M,d}$ is admissible. Hence, $Q_{\mathsf{TR},r}$ is an upper bound for Q_{TR_r}. Since Q_{TR_r} is admissible, we conclude that $Q_{\mathsf{TR},r}$ is also admissible. □

In practice, since computing $Q(s, h-r, \mathbf{a}) = Q_{\mathsf{TR},r}(\boldsymbol{\varphi}'_{s,h-r,\mathbf{a}})$ is relatively expensive, we first compute the MDP heuristic $Q'_{\mathsf{MDP}}(s, h-r, \mathbf{a})$ for the Dec-POMDP with horizon $h-r$, and only compute $Q(s, h-r, \mathbf{a})$ for the actions $\mathbf{a} \in \boldsymbol{\mathcal{A}}$ which based on the MDP values can possibly attain the maximum in Equation (9.1). This does not affect the value of the heuristic.

Finally, since we compute the terminal reward heuristic $Q_{\mathsf{TR},r}$ only for policies $\boldsymbol{\varphi}$ with $\sigma(\boldsymbol{\varphi}) < r$, the number of policies for which a heuristic is computed only depends on the full horizon h through the horizon of $\boldsymbol{\varphi}$. In particular, the number of policies for which a heuristic is computed is bounded by $O\big(|\mathcal{A}_*|^{n|\mathcal{O}_*|^r} h\big)$, which is linear in h. In practice, however, not all $O\big(|\mathcal{A}_*|^{n|\mathcal{O}_*|^r}\big)$ heuristics per horizon are computed. Since for larger horizons a larger fraction of these policies is computed, running times scale in practice (slightly) worse than linearly in h.

9.1.2 Aperiodic Terminal Reward Heuristics

We now turn to *aperiodic terminal reward heuristics*, which are a further tightening of the terminal reward heuristics. Rather than having a fixed period r with which states are revealed, the aperiodic terminal reward heuristic reveals the state after $r-1$ or r stages, depending on which yields the tightest heuristic. Moreover, this choice can be made independently for each joint observation history at stage $r-1$. We illustrate this in Figure 13. The aperiodic terminal reward heuristic can in particular be significantly tighter than the terminal reward heuristic if state information is not equally valuable at every stage of the problem.

We compute the aperiodic terminal reward heuristic using a Dec-POMDP with horizon r and a terminal reward. However, we only use the first $r-1$ stages in the usual way, while we integrate the last stage with the terminal reward, which we compute at stage $r-1$. This terminal reward

depends on both the joint belief $b_{\boldsymbol{\tau}} \in \Delta(\mathcal{S})$ and the action $\mathbf{a} \in \boldsymbol{\mathcal{A}}$ that a fully specified policy $\boldsymbol{\pi}$ (with horizon r) specifies for $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^{r-1}$. For each joint observation history $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^{r-1}$, we compute two values. Firstly, we compute a value $R_T^0(b, \mathbf{a})$ corresponding to revealing the rth state: a reward $R(b, \mathbf{a})$ for stage r plus a terminal reward for the remaining $h - r$ stages, where we reveal the rth joint belief from stage r onwards and the rth state from stage $r + 1$ onwards. Secondly, we compute a value $R_T^{-1}(b, \mathbf{a})$ corresponding to revealing the $(r-1)$th state: a terminal reward for the remaining $h - r + 1$ stages, where we reveal the $(r-1)$th state from stage r onwards. Since we use the action $\mathbf{a}$ specified by a decentralized, fully specified policy $\boldsymbol{\pi}$ (with horizon r), we do not reveal extra information in stage $r-1$ in this case.

We now provide the definition of the aperiodic terminal reward heuristic.

Definition 9.2 (Aperiodic terminal reward heuristic)**.** *Let $r \geq 2$ be given and let $\boldsymbol{\varphi}$ be a policy with horizon h and $\sigma(\boldsymbol{\varphi}) < r$. Let $M \geq 1$ and $d \geq 1$ be given. We define the* aperiodic terminal reward heuristic *$Q_{\mathsf{ATR},r}$ recursively over the horizon h. If $h \leq r$, then $Q_{\mathsf{ATR},r}(\boldsymbol{\varphi}) = Q_{M,d}(\boldsymbol{\varphi})$. If $h > r$, we consider the given Dec-POMDP with horizon r and a terminal reward $R_T \colon \Delta(\mathcal{S}) \times \boldsymbol{\mathcal{A}} \to \mathbb{R}$, awarded based on the joint belief after $r - 1$ stages and an action specified by a joint policy $\boldsymbol{\pi}$ (which has horizon r). This terminal reward is defined by*

$$\begin{aligned}
R_T^0(b, \mathbf{a}) &= R(b, \mathbf{a}) + \sum_{\mathbf{o} \in \boldsymbol{\mathcal{O}}} \left[O(\mathbf{o} \mid b, \mathbf{a}) \max_{\mathbf{a}' \in \boldsymbol{\mathcal{A}}} \sum_{s \in \mathcal{S}} b_{\mathbf{a},\mathbf{o}}(s) \cdot Q(s, h{-}r, \mathbf{a}') \right] \\
R_T^{-1}(b, \mathbf{a}) &= \sum_{s \in \mathcal{S}} b(s) \cdot Q(s, h{-}r{+}1, \mathbf{a}) \\
R_T(b, \mathbf{a}) &= \min \left\{ R_T^{-1}(b, \mathbf{a}), R_T^0(b, \mathbf{a}) \right\},
\end{aligned}$$

where $Q(s, t, \mathbf{a}) = Q_{\mathsf{ATR},r}(\boldsymbol{\varphi}'_{s,t,\mathbf{a}})$. The awarded terminal reward is then

$$\sum_{\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^{r-1}} \Pr(\boldsymbol{\tau}) R_T(b_{\boldsymbol{\tau}}, \boldsymbol{\pi}(\boldsymbol{\tau})),$$

where $\boldsymbol{\pi}(\boldsymbol{\tau}) = \langle \pi_1(\tau_1), \ldots, \pi_n(\tau_n) \rangle$. Finally, we define $Q_{\mathsf{ATR},r}(\boldsymbol{\varphi})$ as the heuristic $Q_{M,d}\left(\boldsymbol{\varphi}|^{\leq r}\right)$ corresponding to this Dec-POMDP with horizon r and terminal rewards.

Next, we argue that $Q_{\mathsf{ATR},r}$ is admissible. The intuition is again that for any Dec-POMDP policy $\boldsymbol{\pi}$, we can define corresponding policies for recursively defined heuristics that yield the same heuristic value as the policy value of $\boldsymbol{\pi}$. This holds since at any point in the heuristic computations, agents at least have access to their own local observation history.

Theorem 9.2. *$Q_{\mathsf{ATR},r}$ is admissible.*

Proof (sketch). We use induction on the horizon h of $\boldsymbol{\varphi}$. If $h \leq r$, then $Q_{\mathsf{ATR},r}(\boldsymbol{\varphi}) = Q_{M,d}(\boldsymbol{\varphi})$, so the admissibility of $Q_{\mathsf{ATR},r}$ follows from the admissibility of $Q_{M,d}$ (Theorem 7.2).

Similarly to the proof of Theorem 9.1, we first consider a tighter heuristic where we calculate the optimal values of the horizon r Dec-POMDP with terminal rewards and any Dec-POMDP values used in the terminal rewards. Then we can again formalize this heuristic using a generalization of our framework from Section 6.3, where we allow the set of constraints to depend on the policy. For a given policy $\boldsymbol{\pi}$ up to stage r, we first define a selector $\mathsf{S} \colon \boldsymbol{\mathcal{O}}^{\geq r-1} \to \{0, -1\}$. If $\boldsymbol{\tau} \in \boldsymbol{\mathcal{O}}^{r-1}$, then we define $\mathsf{S}(\boldsymbol{\tau}) = 0$ if $R_T^0(b_{\boldsymbol{\tau}}, \boldsymbol{\pi}(\boldsymbol{\tau})) < R_T^{-1}(b_{\boldsymbol{\tau}}, \boldsymbol{\pi}(\boldsymbol{\tau}))$, and $\mathsf{S}(\boldsymbol{\tau}) = -1$ otherwise. In the selector, we use the actual heuristic values for R_T^0 and R_T^{-1}, computed recursively using early termination. For $v \geq r$, we ignore the extra observations, i.e. we define $\mathsf{S}(\mathbf{o}^{[1,v]}) = \mathsf{S}(\mathbf{o}^{[1,r-1]})$. Then we define the information function ATR_r by

$$\mathsf{ATR}_r(i, \mathbf{o}^{[1,v]}, \boldsymbol{s}) = \begin{cases} o_i^{[1,v]} & \text{if } v < r, \\ \mathbf{o}^{[1,v]} & \text{if } v = r \text{ and } \mathsf{S}(\boldsymbol{\tau}) = 0, \\ (o_i^{[1,v]}, \mathbf{o}_{\neq i}^{[1,r]}, s^r) & \text{if } v > r \text{ and } \mathsf{S}(\boldsymbol{\tau}) = 0, \\ (o_i^{[1,v]}, s^{r-1}) & \text{if } v \geq r \text{ and } \mathsf{S}(\boldsymbol{\tau}) = -1. \end{cases}$$

Then the corresponding heuristic Q_{ATR_r} is admissible.

Similarly to the proof of Theorem 9.1, we can now argue that the actual heuristic $Q_{\mathsf{ATR},r}$ is an upper bound for Q_{ATR_r}, and hence also admissible. □

We have $Q_{\mathsf{ATR},r}(\boldsymbol{\varphi}) \leq Q_{\mathsf{TR},r}(\boldsymbol{\varphi})$, since we recover the terminal reward heuristic if we set $R_T(b, \mathbf{a}) = R^0_T(b, \mathbf{a})$ instead of taking the minimum with $R^{-1}_T(b, \mathbf{a})$.

9.2 Horizon Reduction

TR-MAA* also applies a horizon reduction as explained in Section 8.3. For TR-MAA*, the terminal reward is a heuristic computed as explained in Section 9.1. If $h > r$, we also apply the horizon reduction to compute a heuristic for the root node. We note that for TR-MAA*, it would be possible to start in horizon reduction mode. Then TR-MAA* terminates with a policy for a horizon r Dec-POMDP with terminal rewards, and the corresponding policy value is an upper bound. Initial experiments indicate that this is typically faster, but gives worse bounds.

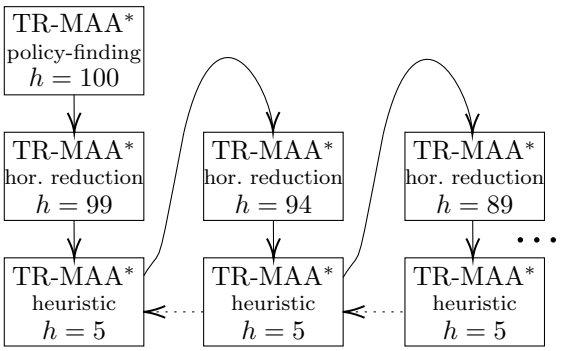


Figure 14: Schematic call graph for TR-MAA*, for $r = 5$. A dotted line indicates that a result is provided. The call graph continues further to the right with $h = 84$, $h = 79$, etc.

Call graph for TR-MAA*. Figure 14 shows a schematic call graph for TR-MAA* (for $r = 5$). Initially, we have $d = 1$ and call TR-MAA* in horizon reduction mode with horizon 99. Then the horizon is reduced to 5, and the corresponding call to TR-MAA* in heuristic mode uses the terminal rewards, for which it makes calls to TR-MAA* with horizon 94. This again makes a horizon reduction, and the corresponding call to TR-MAA* in heuristic mode with horizon 5 uses terminal rewards with horizon 89, which in turn uses terminal rewards with horizon 84, etc., until we reach horizon 4 for which we directly compute a heuristic. As for PF-MAA*, the figure only shows the case where $d = 1$, while d can increase during the execution of the algorithm.

10 Empirical Evaluation

We now turn to the empirical evaluation of the algorithms. In Section 10.1, we present the setup and the benchmarks used. Section 10.2 shows the performance of small-step MAA* with a precomputed heuristic in comparison to the state-of-the-art exact solver GMAA*-ICE [Oliehoek *et al.*, 2013] with the same precomputed heuristics. Section 10.3 provides an evaluation for RS-MAA*, showing the efficacy of recursive heuristics. Finally, in Section 10.4 and 10.5 we compare PF-MAA* and TR-MAA* to the state-of-the-art ϵ-optimal solver FB-HSVI [Dibangoye *et al.*, 2014a; Dibangoye *et al.*, 2016] (for $\epsilon = 0.01$) and the state-of-the-art approximate solver, which is the genetic algorithm GA-FSC [Eker and Akin, 2013].

10.1 Setup and Benchmarks

Setup. We implemented our algorithms RS-MAA*, PF-MAA* and TR-MAA* in a common code base in Python 3 that does not have any dependencies.[8] We execute it using PyPy. All experiments were run on a 3.7GHz Intel Core i9 CPU running Ubuntu 22.04.1 LTS. We limited each process to 16GB of memory and 1800 seconds of CPU time (unless mentioned otherwise).

Benchmarks. We used the standard benchmarks from the literature: DecTiger [Nair *et al.*, 2003], FireFight [Oliehoek *et al.*, 2008b] (3 fire levels, 3 or 4 houses), Grid with two observations [Amato *et al.*, 2006], BoxPushing [Seuken and Zilberstein, 2007a], Grid3x3 [Amato *et al.*, 2009], Mars [Amato and Zilberstein, 2009], Hotel [Spaan and Melo, 2008], Recycling [Amato *et al.*, 2007], and Broadcast [Hansen *et al.*, 2004]. We provide some statistics about the sizes and properties of these benchmarks in Table 1.

benchmark	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}_*\rvert$	$\lvert\mathcal{O}_*\rvert$	TI	OI	Dec-MDP
DecTiger	2	3	2	✗	✗	✗
FireFight ($n_h = 3$)	432	3	2	✗	✗	✗
FireFight ($n_h = 4$)	2025	4	2	✗	✗	✗
Grid	16	5	2	✓	✓	✗
BoxPushing	100	4	5	✗	✗	✗
Grid3x3	81	5	9	✓	✓	✓
Mars	256	6	8	✗	✗	✗
Hotel	16	3	4	✓	✓	✓
Recycling	4	3	2	✓	✓	✓
Broadcast	4	2	2	✓	✗	✗

Table 1: Properties of the used benchmarks: the number of states $\lvert\mathcal{S}\rvert$, the number of local actions $\lvert\mathcal{A}_*\rvert$ of each agent, the number of local observations $\lvert\mathcal{O}_*\rvert$ of each agent, and whether the benchmark is transition-independent (TI), observation-independent (OI), or a Dec-MDP. All problems have 2 agents.

We now provide a description of the benchmarks. For DecTiger, we refer to Example 3.2.

The FireFight problems [Oliehoek *et al.*, 2008b] considers two firefighters that aim to extinguish fires in a row of $n_h \in \{3, 4\}$ houses. Each house has a fire level (0, 1 or 2), where the lowest fire level 0 corresponds to no fire. In each time step, the fire level of a house may increase if it is already on fire or if one of its neighbours is on fire. If both firefighters extinguish fires in the same house, they completely extinguish the fire in that house. On the other hand, if only one agent extinguishes fire, then the fire level drops by only one level, or remains the same with some probability if a neighbouring house is burning. The agents receive a noisy observation of the (new) fire level at the house which they extinguished. In each stage, the cost (negative reward) is the sum of the fire levels of the houses. When the fires are extinguished completely, they remain extinguished and the reward is 0 for the remaining stages.

[8] Source code and scripts to reproduce the results are available at https://zenodo.org/records/13374100.

The GRID problem [Amato *et al.*, 2006] is a 2×2 version of the partially observable grid problem introduced in Example 3.1. In particular, agents only observe in which of the columns in the grid they are, and the agents receive a reward if they are in the same cell in the grid. This problem is TOI (transition- and observation-independent), since each agent only has an influence on its own position, and only observes its own position.

The BOXPUSHING problem [Seuken and Zilberstein, 2007a] considers two agents that have to push boxes one cell upwards. There are small boxes (yielding a reward of 10) and a large box (yielding a reward of 100), that the agents have to push together. The possible actions of the agents are: turn left, turn right, move forward, and do nothing. As observation, the agents see the contents of the cell in front of them: empty cell, wall, other agent, small box, large box.

The GRID3X3 problem [Amato *et al.*, 2009], considers agents in a 3×3 grid that want to meet in the bottom left corner or the top right corner. In contrast to the problem introduced in Example 3.1, however, the agents fully observe their own position. Like GRID, this problem is TOI. However, GRID3X3 is also a Dec-MDP, because the agents fully observe their own locations, and hence jointly observe the full state, which describes both locations.

The MARS problem [Amato and Zilberstein, 2009] considers two mars rovers moving in a 2×2 grid. There are two cells in the grid that need to be sampled by a single agent and two cells that need to be drilled by both agents simultaneously. Agents observe their own position and whether their current cell was already sampled or drilled. MARS is not a Dec-MDP, since the last joint observation does not determine the full state. Nevertheless, the MARS problem does have the special property that the joint action-observation history determines the full underlying state.

In the HOTEL problem [Spaan and Melo, 2008], we consider two travel agents that each can put customers in their own hotel (which has only one place) or in a common resort. This problem is TOI (transition- and observation-independent) and a Dec-MDP.

In the RECYCLING problem [Amato *et al.*, 2007], the agents aim to empty recycling bins without running out of battery. For each agent, there are two local states, describing whether the battery level is high or low. The agents can earn reward by emptying small bins or by collaborating to empty a large bin, but receive a negative reward for running out of battery. This problem is also TOI (transition- and observation-independent) and a Dec-MDP.

Finally, the BROADCAST problem [Hansen *et al.*, 2004] considers two agents that need to pass messages to each other over a shared channel. Agents have a buffer of messages which can contain 0 or 1 message; if it does not contain a message, then it will contain a message in the next time step with a 90% probability for agent 1 and a 10% probability for agent 2. If both agents attempt to send a message, then there is a collision on the channel and neither message is received. The reward is 1 when successfully transmitting a message and 0 otherwise. The agents receive a noisy observation regarding whether a collision happened. This problem is transition-independent: for each agent, there are two local states, describing whether their buffer has 0 or 1 message. However, the problem is not observation-independent (since the observation depends on the *joint* action).

10.2 Small-Step MAA*

To evaluate the efficacy of the small-step search tree in comparison to incremental expansion, we compare small-step MAA* to generalized MAA* with incremental clustering and expansion (GMAA*-ICE). We use the same precomputed heuristic (Q_{MDP}, Q_{POMDP} or Q_{BG}) for both algorithms and where both algorithms use lossless clustering [Oliehoek *et al.*, 2009]. Both algorithms use sparse data structures. Hence, besides implementation details, the only difference between GMAA*-ICE and small-step MAA* is that GMAA*-ICE uses incremental expansion while small-step MAA* uses the small-step search tree.

For GMAA*-ICE, we use the C++ implementation in the MADP toolbox [Oliehoek *et al.*, 2017]. For the BG heuristic, the MADP toolbox offers several representations: a tree-based representation, a hybrid representation and a vector-based representation with tree-based incremental pruning (corresponding to the options `QBG`, `QBGhybrid` and `QBGTreeIncPrune`, respectively). We tested each option and provide per benchmark the best result.

Since in this section we primarily want to understand the value of the small-step search tree, we separate timings for precomputing heuristics and the planning itself. In addition to this, the timings for the heuristic computations can be used to study the effect of the faster precomputation of the Q_{BG} heuristic introduced in Section 6.1.3.

10.2.1 Results

Table 2 presents the results. In short, small-step MAA* is able to compute an optimal solution for all benchmarks and all horizons where GMAA*-ICE is able to compute an optimal solution, using the same heuristic, with a single exception: FireFight with three houses and horizon $h = 5$, where small-step MAA* goes out of memory when using Q_{MDP} while GMAA*-ICE is able to compute a solution in 216 seconds. On the other hand, when using Q_{MDP}, small-step MAA* scales to higher horizons than GMAA*-ICE for several benchmarks: Grid3x3 (6 rather than 5), Mars (7 rather than 4), and Hotel (6 rather than 4). On the benchmarks where GMAA*-ICE finishes the heuristic computation for Q_{POMDP} and Q_{BG}, both GMAA*-ICE and small-step MAA* are able to complete the planning part within the time limit.

However, GMAA*-ICE is in some cases much faster. For example, for Grid with horizon $h = 4$ and the Q_{MDP} heuristic, GMAA*-ICE takes only 5 seconds while our algorithm takes 214 seconds (expanding over 10 million nodes). Similarly, GMAA*-ICE is almost a factor 5 faster on DecTiger, Recycling and Broadcast. A possible reason for this is that the Bayesian game solver used internally in GMAA*-ICE expands clusters in descending order of probability, while small-step MAA* orders the clusters lexicographically (by the smallest LOH in the cluster). The reason that small-step MAA* uses a fixed, lexicographic ordering of the clusters is that this helps with heuristic reuse, but this is only relevant when using heuristics that are computed on-the-fly, rather than heuristics which are precomputed as in this experiment.

Overall, we see that the small-step search tree is competitive with incremental expansion, but that both methods have strengths and weaknesses.

Heuristic timings. For Q_{MDP}, both small-step MAA* and GMAA*-ICE finish the heuristic computation within 1 second, with the exception of FireFight ($n_h = 3$) for $h = 1000$, where our implementation takes 2 seconds. We attribute this difference mainly to the difference in speed of C++ compared to PyPy.

For Q_{POMDP}, the implementation used by the MADP toolbox explicitly computes and stores a heuristic for all action-observation histories (other implementations of Q_{POMDP} in the MADP toolbox gave errors). In contrast, we compute Q_{POMDP} using the belief MDP, and therefore compute a heuristic value for each belief induced by an action-observation history. Since in practice many different action-observation histories give rise to the same joint belief, we compute and store much fewer heuristic values.

For Q_{BG}, we really see the effect of the faster precomputation of the Q_{BG} heuristic introduced in Section 6.1.3. In particular, GMAA*-ICE is not at able to compute Q_{BG} for any horizon for the largest benchmark problems, Grid3x3 and Mars. Indeed, for these benchmarks, the factor $|\mathcal{A}_*|^{n|\mathcal{O}_*|}$ in the complexity of the precomputation done by Q_{BG} is larger than 10^{12}. In addition, we see that the speedup generally increases with the size of the local action and observation sets benchmark: for Broadcast, our computation is a factor 10 faster, for Hotel, our computation is a factor 80 faster and for BoxPushing our computation is more than 500 times faster.

		Q_{MDP}				Q_{POMDP}				Q_{BG}			
		ICE		S-MAA*		ICE		S-MAA*		ICE		S-MAA*	
h	value	t_Q	t_P	t_Q	t_P	t_Q	t_P	t_Q	t_P	t_Q	t_P	t_Q	t_P
DecTiger													
3	5.190812	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1
4	4.802755	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1
5	7.026451	<1	38	<1	35	2	2	<1	1	<1	<1	<1	<1
6	10.381625	<1	MO	<1	MO	80	164	<1	744	<1	15	<1	68
FireFight ($n_h = 3$)													
4	−6.578834	<1	3	<1	9	8	<1	<1	<1	15	<1	1	<1
5	−7.069874	<1	216	<1	MO	335	<1	4	26	46	<1	4	25
6	−7.175591	<1	7	<1	34	TO		47	<1	762	<1	59	<1
7	−7.175591	<1	4	<1	8	MO		MO		MO		MO	
100	−7.175591	<1	4	<1	4	MO		MO		MO		MO	
1000	−7.175591	<1	7	2	4	MO		MO		MO		MO	
FireFight ($n_h = 4$)													
3	−11.135643	<1	9	<1	2	7	<1	<1	<1	14	<1	<1	<1
4	−14.106819	<1	144	<1	214	606	<1	11	<1	1191	<1	12	<1
Grid													
3	1.550444	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1
4	2.241577	<1	5	<1	214	2	<1	<1	<1	19	<1	1	<1
5	2.970496	<1	MO	<1	MO	MO		2	1	MO		9	<1
6	3.717168	<1	MO	<1	MO	MO		30	26	MO		124	1
BoxPushing													
3	66.081000	<1	<1	<1	<1	<1	<1	<1	<1	279	<1	<1	<1
4	98.593613	<1	127	<1	17	2	130	<1	17	TO		1	10
Grid3x3													
3	0.133200	<1	<1	<1	<1	1	<1	<1	<1	MO		<1	<1
4	0.432900	<1	<1	<1	<1	MO		<1	<1	MO		<1	<1
5	0.895656	<1	2	<1	<1	MO		<1	<1	MO		2	<1
6	1.492987	<1	MO	<1	11	MO		<1	11	MO		7	7
Mars													
3	9.380000	<1	<1	<1	<1	1	<1	<1	<1	MO		<1	<1
4	10.180800	<1	<1	<1	<1	MO		<1	<1	MO		1	<1
5	13.266538	<1	MO	<1	41	MO		<1	40	MO		2	4
6	18.623165	<1	MO	<1	13	MO		<1	12	MO		4	1
7	20.900724	<1	MO	<1	62	MO		1	60	MO		6	4
Hotel													
4	22.187500	<1	15	<1	3	<1	15	<1	3	<1	<1	<1	<1
5	27.187500	<1	MO	<1	109	MO		<1	111	<1	<1	<1	<1
6	32.187500	<1	MO	<1	1055	MO		<1	1057	5	<1	<1	<1
7	37.187500	<1	MO	<1	MO	MO		<1	MO	6	<1	<1	<1
100	502.187500	<1	MO	<1	MO	MO		<1	MO	72	<1	1	<1
1000	5002.187500	<1	MO	<1	MO	MO		<1	MO	403	2	5	1
Recycling													
10	31.863889	<1	<1	<1	1	MO		<1	1	<1	<1	<1	<1
20	62.633136	<1	41	<1	46	MO		<1	46	<1	<1	<1	<1
70	216.479290	<1	MO	<1	MO	MO		<1	MO	<1	11	<1	36
80	247.248521	<1	MO	<1	MO	MO		<1	MO	<1	63	<1	287
Broadcast													
10	9.290000	<1	<1	<1	<1	MO		<1	<1	<1	<1	<1	<1
50	45.501604	<1	8	<1	7	MO		<1	<1	8	<1	1	<1
60	54.570564	<1	79	<1	69	MO		<1	<1	11	<1	1	<1
70	63.611284	<1	MO	<1	MO	MO		<1	<1	17	<1	1	<1
500	452.738119	<1	MO	<1	MO	MO		4	19	426	1	43	17
900	814.709393	<1	MO	<1	MO	MO		8	15	881	6	88	15

Table 2: Results and running times (seconds) for GMAA*-ICE (ICE) in comparison to small-step MAA* (S-MAA*). TO and MO denote timeout (>1800s) and memout (>16GB). t_Q and t_P denote the times for heuristic precomputations and planning, respectively.

10.3 Recursive Small-Step Multi-Agent A* (RS-MAA*)

Next, we evaluate our exact algorithm RS-MAA*. We compare the performance different recursive heuristics to the performance of GMAA*-ICE and the performance of small-step MAA* with precomputed heuristics, as shown in the previous section.

10.3.1 Hyperparameters

We evaluate RS-MAA* on five different recursive heuristics $Q_{M,d}$, to show the behaviour of the different recursive heuristics. We now discuss the selection of the hyperparameters M, d and α.

Hyperparameters. The most important parameter is the depth d of the recursive heuristic. Lower values of d give tighter heuristics but are more expensive to compute. Setting the number of iterations M too low leads to an overly loose heuristic. However, setting M high leads to generally higher computation times. Due to the recursive nature of the algorithm, however, the time can scale as badly as $O(M^h)$. Typically, the computational effort is much less, in particular due to heuristic reuse and the fact that recursive heuristics at later stages are easier to compute. Furthermore, some heuristics already find a complete policy before M iterations and most heuristics do not nest h layers deep (except if $d = 1$). The effect of the threshold α used in dynamic termination of heuristic computations is similar to the effect of M, but harder to quantify. In particular, higher values of α yield tighter heuristics and are harder to compute.

Hyperparameter selection. For most problems, Q_1 and Q_2 are too expensive to compute, but Q_3 is a good compromise. For easier problems, Q_∞ also works well.

In a preliminary evaluation for $d = 3$ using $M \in \{100, 200, 400\}$ and $\alpha \in \{0.1, 0.2\}$, we observed that $M = 100$ sometimes expands much more nodes than $M = 200$, but increasing M to 400 did not reduce the number of nodes (significantly) compared to $M = 200$, and hence typically increased running times. Hence, we set $M = 200$ for $d = 3$ (and $d = 2$, shown for comparison). The threshold α had a small effect, so we conservatively set it to 0.2.

For easier problems where Q_∞ works well, we observed that even smaller values of M such as $M = 25$, which empirically often gives tighter bounds than Q_{BG}, works well. For comparison, we also show the cheapest recursive heuristic $Q_{1,\infty}$, which is equal to the precomputed heuristic Q_{POMDP} by Theorem 7.3.

10.3.2 Results

Table 3 and Table 4 shows the results for RS-MAA*. We first compare the results to the results from GMAA*-ICE and small-step MAA* (from Table 2). Next, we compare results for different recursive heuristics and discuss the characteristics of the benchmarks leading to the results.

RS-MAA* vs. other exact solvers. With the exception of FireFight with three houses, recursive heuristics are able to scale at least as far as precomputed heuristics. Moreover, on DecTiger, FireFight with four houses, BoxPushing, Mars, Recycling and Broadcast, we are able to scale further than both GMAA*-ICE and small-step MAA* with a precomputed heuristic. FireFight's structure makes $h = 5$ expensive (no potential for clustering). The recursive nature of RS-MAA* suffers from this when computing values for $h \geq 6$.

On DecTiger, we scale to horizon 12 rather than horizon 6. For this, the tightness of the heuristic is essential. The Q_{BG} heuristic reveals too much information which leads to unrealistically high heuristic values. On Q_3, however, only information from the first two stages is shared, which under an optimal policy becomes irrelevant in later stages. This leads to a very tight heuristic in the later stages.

On Recycling, we scale to horizon 1500 rather than 80. For Recycling, heuristics need to be tighter for larger horizons. For example, we see that the $Q_{25,\infty}$ heuristic scales to horizon 350, while the tighter $Q_{200,\infty}$ heuristic scales to horizon 1500. Nevertheless, the $Q_{25,\infty}$ heuristic is already tighter than the Q_{BG} heuristic.

Q_{POMDP} **vs.** $Q_{1,\infty}$**.** Recall from Theorem 7.3 that the POMDP heuristic Q_{POMDP} and the heuristic $Q_{1,\infty}$ are equal to each other. When comparing results for Q_{POMDP} (from Table 2) and $Q_{1,\infty}$, we see that using the recursive heuristic $Q_{1,\infty}$ is typically between 10% and 50% slower than using the precomputed heuristics. We attribute this increase to the overhead incurred for working with the recursive heuristics. On the other hand, using $Q_{1,\infty}$ makes it possible to compute an exact result for FireFight with four houses with $h = 5$, whereas a memory limit was reached during the precomputation of Q_{POMDP}.

Q_3 **vs.** Q_2**.** The increased tightness of the tighter recursive heuristics Q_d leads to better scalability on DecTiger, Mars and BoxPushing. On DecTiger and Mars, Q_3 is sufficiently tight and using the Q_2 heuristic leads to (much) larger running times since it takes longer to compute deeper heuristics. On the other hand, for BoxPushing the tighter heuristic Q_2 performs much better.

High horizons. For Recycling and Hotel, we scale up to horizon 1500 and horizon 2000, respectively. The main reason for this is that Recycling and Hotel are TOI-Dec-MDPs. As a result, the number of clusters (per agent and per stage) is limited to 2 for Recycling and to 4 for Hotel. On the other hand, Grid3x3 is also a TOI-Dec-MDP, but we do not scale well on Grid3x3. The reason for this is that the search space is still large with 9 possible local observations for each agent.

For Broadcast, we scale up to horizon 1500. In this case, the main reason for this is that for the optimal policy found by the solver, the agents never both send a message and hence there never is a collision, so the observations are not informative. Moreover, the Q_{POMDP} heuristic is tight for the optimal policy, since the Dec-POMDP policy is in fact also optimal for the underlying POMDP.

Regret computations. Finally, we discuss some additional experiments where we enable regret computations as discusses in Section 5.6.2. For most benchmarks, the regret computations do not have an effect on the number of nodes expanded, but do increase the running time by up to 40%. However, for BoxPushing, the regret computations do have a substantial effect. When using the $Q_{200,3}$ heuristic, the running time for $h = 4$ decreases from 11 seconds to 1 seconds when enabling regret computations. Moreover, the number of nodes expanded by the main A* search decreases from 682 683 to 184. Moreover, $h = 5$ becomes solvable with Q_3 in 38 seconds (expanding around 2 million nodes). The reason for the difference for BoxPushing is that agents can earn a reward without cooperating with the other agent by pushing a small box. If it is possible to push a small box to the goal in the last stage, then this action clearly dominates the other actions. In contrast, for other problems the best action often strongly depends on what the other agent does, and hence no actions are shown to be dominated using the regret computations.

h	value	$Q_{1,\infty}$	$Q_{25,\infty}$	$Q_{200,\infty}$	$Q_{200,3}$	$Q_{200,2}$
DecTiger						
5	7.026451	1	<1	<1	<1	<1
6	10.381625	853	26	20	<1	1
7	9.993568	MO	MO	MO	2	5
8	12.217263	MO	MO	MO	2	12
9	15.572437	MO	MO	MO	8	34
10	15.184380	MO	MO	MO	516	TO
11	17.408076	MO	MO	MO	202	TO
12	20.763250	MO	MO	MO	659	TO
FireFight ($n_h = 3$)						
3	−5.736969	<1	1	<1	<1	1
4	−6.578834	1	2	2	2	2
5	−7.069874	36	38	34	9	11
6	−7.175591	72	96	84	85	98
7	−7.175591	MO	MO	MO	MO	MO
FireFight ($n_h = 4$)						
3	−11.135643	1	1	1	1	1
4	−14.106819	16	18	15	15	18
5	−16.835395	650	602	613	MO	MO
Grid						
3	1.550444	1	1	1	1	1
4	2.241577	1	2	2	2	2
5	2.970496	5	10	9	9	10
6	3.717168	83	109	103	106	111
BoxPushing						
3	66.081000	<1	<1	<1	<1	<1
4	98.593613	19	12	11	11	3
5	107.729851	MO	MO	TO	TO	6
Grid3x3						
3	0.133200	<1	<1	<1	<1	<1
4	0.432900	<1	1	1	1	1
5	0.895656	1	2	2	2	2
6	1.492987	13	13	10	10	25
Mars						
3	9.380000	<1	<1	<1	<1	<1
4	10.180800	1	1	1	1	1
5	13.266538	44	2	2	2	2
6	18.623165	15	4	4	4	24
7	20.900724	70	8	9	18	14
8	22.478798	MO	MO	MO	33	245
9	24.320398	MO	MO	MO	122	1778

Table 3: Results and running times for RS-MAA* for five different heuristics $Q_{M,d}$, on a range of standard benchmarks. TO and MO denote timeout (>1800s) and memout (>16GB).

h	value	$Q_{1,\infty}$	$Q_{25,\infty}$	$Q_{200,\infty}$	$Q_{200,3}$	$Q_{200,2}$
			HOTEL			
3	16.875000	<1	<1	<1	<1	<1
4	22.187500	3	<1	<1	<1	<1
5	27.187500	119	<1	<1	<1	<1
6	32.187500	1128	1	1	1	1
7	37.187500	MO	1	1	1	2
10	52.187500	MO	1	2	3	3
50	252.187500	MO	2	9	49	83
60	302.187500	MO	2	11	61	478
70	352.187500	MO	2	14	76	TO
100	502.187500	MO	3	21	118	TO
500	2502.187500	MO	13	121	706	TO
1000	5002.187500	MO	27	259	1528	TO
1500	7502.187500	MO	41	409	TO	TO
2000	10002.187500	MO	56	552	TO	TO
			RECYCLING			
10	31.863889	1	<1	<1	1	2
15	47.248521	2	1	1	2	3
20	62.633136	53	1	1	3	7
25	78.017751	MO	1	1	6	12
50	154.940828	MO	1	2	43	85
100	308.786982	MO	1	4	242	456
250	770.325444	MO	4	11	945	1735
350	1078.017751	MO	33	16	1450	TO
400	1231.863905	MO	MO	18	1694	TO
500	1539.556213	MO	MO	23	TO	TO
1000	3078.017751	MO	MO	48	TO	TO
1500	4616.479290	MO	MO	112	TO	TO
			BROADCAST			
10	9.290000	<1	<1	<1	<1	<1
50	45.501604	1	4	5	5	6
100	90.760423	2	14	42	42	48
250	226.500545	9	103	715	730	821
300	271.751958	12	146	1142	1160	1341
350	316.999551	16	191	1662	1673	TO
500	452.738119	40	330	TO	TO	TO
800	724.214207	MO	674	TO	TO	TO
900	814.709393	101	821	TO	TO	TO
1000	905.200368	MO	958	TO	TO	TO
1500	1357.663165	MO	1649	TO	TO	TO

Table 4: Results and running times for RS-MAA* for five different heuristics $Q_{M,d}$, on a range of standard benchmarks. TO and MO denote timeout (>1800s) and memout (>16GB).

10.4 Policy-Finding Multi-Agent A* (PF-MAA*)

Next, we turn to empirical results for our policy-finding algorithm PF-MAA*.

10.4.1 Setup and Hyperparameters

Setup. We ran PF-MAA* for six configurations: three configurations aimed at finding policies fast and limited to 60 seconds of CPU time each, and three configurations aimed at finding high-quality policies limited to 600 seconds.

comparison			fast results			high-quality results		
heur.	k	L	heur.	k	L	heur.	k	L
Q_{MDP}	2	1000	$Q_{\mathsf{MDP},1}$	1	20	$Q_{\mathsf{MDP},2}$	2	1000
Q_{MDP}	3	10 000	$Q_{\mathsf{MDP},1}$	2	100	$Q_{\mathsf{MDP},2}$	3	10 000
			$Q_{\mathsf{MDP},1}$	3	100	$Q_{\mathsf{maxr},3}$	3	10 000

Table 5: Hyperparameters for PF-MAA*.

Hyperparameters. Next, we discuss the hyperparameter selection. We report the configurations that we used in Table 5. For PF-MAA*, the main hyperparameters are the window size k for the sliding-window policies (Section 4.4 and 4.5), whether to use the maximum reward heuristic $Q_{\mathsf{maxr},r}$ or the terminal reward MDP heuristic $Q_{\mathsf{MDP},r}$ (Section 8.2), the depth r of the heuristic, and the iteration limit L per stage (Section 8.1). Initial experiments indicated that $k = 1$ is too small for most benchmarks, whereas for $k = 4$, the search space becomes too large, which generally led to worse results than using $k = 3$. We also found that $r = 1$ clearly guides the search worse than larger r, whereas $r \geq 4$ is too expensive (since it needs to be computed for all $h \cdot L$ policies). This yielded eight different configurations: Using the heuristics $Q_{\mathsf{maxr},r}$ and $Q_{\mathsf{MDP},r}$ with $k \in \{2, 3\}$ and $r \in \{2, 3\}$. Based on our tests, we selected three configurations for the quality runs with a 600 second time limit: ($k = 2$, $Q_{\mathsf{MDP},2}$), ($k = 3$, $Q_{\mathsf{MDP},2}$), and ($k = 3$, $Q_{\mathsf{maxr},3}$). For $k = 2$ we use $L = 1000$, while for $k = 3$ we use $L = 10\,000$. Increasing L yields higher running times, and typically also better results. However, for the smaller search space for $k = 2$, using $L = 1000$ is usually sufficient. For the fast runs with a time limit of 60 seconds, we selected the cheapest heuristic $Q_{\mathsf{MDP},1}$ and chose L small, but still large enough to amply satisfy the constraint $L \geq n|C_{i,t}|$ from Corollary 8.1 (with the exception of Mars for $k = 3$). In each case, the $Q_{200,3}$ heuristic is used to solve the Dec-POMDPs.

10.4.2 Results

In Table 6, we first give the results for the baselines FB-HSVI and GA-FSC. The next three columns give results for PF-MAA*, using either the Q_{MDP} heuristic (as comparison), all fast configurations, or all high-quality configurations. For reference, we also give the value for the random policy (i.e. the policy that uniformly randomizes over all actions) and the best-known *upper bound* (from the literature, RS-MAA* or TR-MAA*). We now discuss the results. In Appendix B, we provide further results and results split out per configuration.

State-of-the-art policies. Using PF-MAA*, we compute policies comparable with or better than the state-of-the-art for most benchmarks. In particular, for BoxPushing, we provide the best known policies for $h \geq 10$. Although the improvement over GA-FSC in absolute sense is not large, the upper bound shows that our improvement is a significant step towards optimality. On DecTiger, we find better solutions (with $k = 3$) than FB-HSVI and GA-FSC. It is known that with $k = 4$, better policies exist for e.g. $h = 50$. However, the policy space with $k = 4$ is so large that PF-MAA* does not find the (almost) optimal policies in that space.

Fast versus high-quality configurations. Even the fast configuration typically yields policies that are reasonably close to the best-known policy. Nevertheless, the high-quality configuration is always able to find (slightly) better policies.

Novel heuristics outperform Q_{MDP}. Our algorithm PF-MAA* consistently finds better policies using the novel heuristics $Q_{\mathsf{maxr},r}$ and $Q_{\mathsf{MDP},r}$ from Section 8.2 than using Q_{MDP} (for the same number of iterations L; see Table 5). We note that Q_{MDP} is the only heuristic in the literature that scales up to high horizons on all benchmarks.

Challenges for PF-MAA*. On GRID, the best policies for large horizons require actions that are sub-optimal in the short run, and are hence hard to find using PF-MAA*. On MARS, FB-HSVI yields better results than PF-MAA*. We conjecture that this is due to the larger search space for MARS. To reduce the search space, we used probability-based clustering with $p_{\max} = 0.2$; this improves the values to 125.84 and 243.02 for $h = 50$ and $h = 100$ respectively.

$Q_{\mathsf{MDP},r}$ vs. $Q_{\mathsf{maxr},r}$. In Appendix B, we provide results split out per configuration. Here we summarize an important finding. The terminal reward MDP heuristic $Q_{\mathsf{MDP},2}$ works well on all benchmarks except DECTIGER for $k = 3$. On the other hand, $Q_{\mathsf{maxr},3}$ works very well for DECTIGER and BOXPUSHING, but poorly on other benchmarks. This is primarily due to the increase to $r = 3$: $Q_{\mathsf{maxr},2}$ performs poorly on these benchmarks. However, for DECTIGER and BOXPUSHING $Q_{\mathsf{maxr},3}$ also yields (slightly) better policies than $Q_{\mathsf{MDP},3}$, besides being more scalable.

h	FB-HSVI	GA-FSC	PF-MAA*			random	upper
			Q_{MDP}	fast	quality		
DECTIGER							
20	28.75		28.01	27.14	29.12	−924.4	38.28
50	80.66		79.92	79.05	81.03	−2311.1	101.32
100	170.90	169.30	169.30	169.30	170.91	−4622.2	206.44
FIREFIGHT ($n_h = 4$)							
5			−17.05	−17.08	−16.85	−20.6	−16.84
6			−19.55	−19.50	−19.09	−24.7	−19.04
10			−23.93	−22.93	−22.99	−41.4	−22.81
20			−26.97	−24.82	MO	−82.7	−23.64
50			−28.35	−25.09	MO	−202.2	−20.65
GRID							
7	4.47		4.44	4.41	4.47	1.4	4.47
10			6.69	6.68	6.82	2.2	7.17
20			14.23	14.36	14.81	4.7	17.03
50		40.49	36.86	37.46	MO	12.2	47.26
BOXPUSHING							
10	223.74	216.98	220.35	221.12	224.28	−8.3	224.51
20	458.10	468.15	402.46	466.34	474.97	−20.5	476.21
50	1134.70	1201.05	949.75	1207.67	1209.80	−57.9	1211.96
100		2420.26	1864.49	2431.40	2433.51	−120.5	2437.45
GRID3X3							
7	2.19		2.192	2.174	2.192	0.1	2.192
10	4.68		4.675	4.671	4.685	0.1	4.695
20	14.35		14.350	14.337	14.366	0.4	14.393
50	44.32		44.335	44.322	44.352	1.1	44.379
MARS							
10	26.31		24.63	24.44	26.31	−13.6	26.32
20	52.13		46.09	49.08	49.37	−25.6	52.71
50	128.95		116.73	117.03	122.56	−62.0	132.41
100	249.92		221.70	221.75	234.07	−122.7	264.85

Table 6: Lower bounds, i.e. the value of the policies obtained with different policy-finding algorithms. MO denotes memout (>16GB), blanks denote results that are not available in the literature.

10.5 Terminal Reward Multi-Agent A* (TR-MAA*)

Finally, we turn to our upper bound algorithm TR-MAA*. We ran TR-MAA* with a limit of 120 seconds and a limit of 1800 seconds, respectively.

10.5.1 Hyperparameters

For TR-MAA*, the main parameter is the depth r and the heuristic $Q_{d'}$ chosen for solving the horizon-r Dec-POMDPs. Higher values of r and lower values of d' lead to tighter, more expensive heuristics. For small horizons, it can be better to choose a slightly cheaper heuristic that allows for more exploration in the search tree and hence better bounds for the reward in the initial stages. For larger horizons (e.g. 50), it appears in general to be best to choose r as large as possible, while still ensuring that the heuristic for the root of the tree can be computed within the time limit.

We selected $r = 4$ with the Q_3 heuristic. Like in RS-MAA*, Q_1 and Q_2 are too expensive and therefore require lower values of r. For the fast results, we selected $r = 2$.

10.5.2 Results

In Table 7, we give FB-HSVI's result (if ϵ-optimal) and results for RS-MAA* and TR-MAA* for time limits of 120s and 1800s, respectively. Finally, we give the best-known *lower bound* (from a known policy) and the MDP value, which is a trivial upper bound. We now discuss the results.

A scalable upper bound. On all benchmarks, we provide an upper bound for horizons up to 50 which is better than Q_{MDP}. This is an improvement over RS-MAA*, which only reaches horizon 10 on BoxPushing, horizon 6 on Grid and horizon 5 on FireFight with four houses. For larger horizons, RS-MAA* cannot even compute the cheapest recursive heuristic, $Q_{1,\infty}$, within the memory limit. On BoxPushing, the bound is also tight. It is at least 40 times closer to the optimum than the MDP value.

For Grid, heuristic reuse is very hard. For Grid, relatively few action-observation histories lead to the same belief. As a result, RS-MAA* reaches the memory limit for $h \geq 7$, and also TR-MAA* reaches the memory limit for the more expensive $r = 4$ and higher horizons.

Revealing the state is too optimistic. When RS-MAA* can compute an upper bound, TR-MAA* generally gives worse upper bounds than RS-MAA* with a cheap heuristic. An exception is Mars, where revealing the joint observation history allows for computing the state and is hence just as 'bad' as revealing the state. For $h = 50$ and $h = 100$, TR-MAA* is hence able to compute tighter upper bounds (using $r = 4$) than RS-MAA*.

Exact solving. In some cases, the heuristic used by TR-MAA* is sufficiently tight that the algorithm is able to find an optimal policy within the time limit. In particular, TR-MAA* is the first solver to compute the exact value for Grid $h = 7$, namely 4.4739533, and for FireFight with four houses $h = 6$, namely -19.0385409. In addition, TR-MAA* can also compute the optimal value for Grid3x3 $h = 7$, namely 2.1923737. However, this result can also be achieved by RS-MAA* when providing a near-optimal solution.

h	FB-HSVI	RS-MAA*		TR-MAA*		lower	MDP
		fast	quality	fast	quality		
DecTiger							
20		38.28	38.28	104.90	57.02	29.12	400.00
50		101.32	101.32	316.66	161.55	81.03	1000.00
100		206.44	206.44	669.55	337.00	170.91	2000.00
FireFight ($n_h = 4$)							
5		TO	−16.84	−16.84	MO	−16.84	−11.07
6		TO	MO	−19.04	MO	−19.04	−11.76
10		TO	MO	−22.81	MO	−22.93	−12.42
20		TO	MO	−23.64	MO	−24.82	−12.46
50		TO	MO	−20.65	MO	−25.09	−12.46
Grid							
7	4.48	TO	MO	4.55	4.47	4.47	5.81
10		TO	MO	7.35	7.17	6.82	8.81
20		TO	MO	17.26	17.03	14.81	18.81
50		TO	MO	47.26	47.27	40.49	48.81
BoxPushing							
10	223.75	TO	224.51	226.79	224.62	224.28	244.85
20		TO	MO	479.12	476.21	474.97	511.13
50		TO	MO	1217.23	1211.96	1209.80	1306.24
100		TO	MO	2446.86	2437.45	2433.51	2628.14
Grid3x3							
7	2.20	2.193	2.192	2.193	2.192	2.192	2.312
10	4.69	4.700	4.696	4.697	4.695	4.685	4.882
20	14.36	14.400	14.393	14.397	14.393	14.366	14.629
50		44.392	44.379	TO	44.379	44.352	44.618
Mars							
10	26.32	26.32	26.32	26.36	26.32	26.31	28.61
20		52.86	52.71	52.94	52.77	52.13	57.52
50		132.82	132.82	133.04	132.41	128.95	144.97
100		287.46	265.73	266.28	264.85	249.92	288.97

Table 7: Upper bounds on the value of optimal policies, obtained with algorithms that find such bounds. TO and MO denote timeout (>120s for the fast configuration) and memout (>16GB). Results for FB-HSVI were not available for FireFight. For other benchmarks, a blank denotes that no upper bound was reported for that horizon.

11 Related Work

11.1 A* Algorithms

This work directly builds on a series of works on A* algorithms for Dec-POMDPs, including multi-agent A* (MAA*) [Szer *et al.*, 2005], generalized MAA* (GMAA*) [Oliehoek *et al.*, 2008b] and GMAA*-ICE [Oliehoek *et al.*, 2013]. We refer to Section 5.2, 5.3 and 5.4 for an explanation of these algorithms, and to the discussion at the end of Section 5.5 for a comparison between these algorithms and small-step MAA*.

Using A* algorithms for approximate solving of Dec-POMDPs has also been proposed previously. In particular, Oliehoek *et al.* [2008b] propose k-GMAA*, which in each step only adds the k best children of each node as computed using a Bayesian game. For better scalability, Emery-Montemerlo *et al.* [2004] combine this (for $k = 1$) with only approximately solving the Bayesian games and lossy clustering based on a probability threshold. Instead of only adding the k best children, our algorithm adds all children and prunes them when necessary. Therefore, our algorithm is able to use its resources to search at points where it is less clear what is the best action to choose. Finally, Szer *et al.* [2005] mention a heuristic where the overestimate for future reward is weighted with some factor $w < 1$. This is not an admissible heuristic, but it results in MAA* finding a (possibly suboptimal) policy faster. The disadvantage of this approach, however, is that it is hard to predict how many nodes will be expanded for a particular value of w.

General A* literature. Finally, also similar ideas have appeared in the general A* literature. The idea of splitting up steps in the A* search tree into smaller steps to limit the outdegree of the tree was proposed before by Cazenave [2010], and applied to multi-agent path finding and multiple sequence alignment. Russell [1992] studies Simplified Memory-Bounded A* (SMA*), which bounds the memory required by A* until the first solution is found by pruning policies with a low heuristic from the priority queue. However, applying these ideas to Dec-POMDPs is novel.

11.2 Dynamic Programming

Another line of work in both exact and approximate Dec-POMDP solving is on algorithms that use dynamic programming (DP) [Hansen *et al.*, 2004]. The core workings of these DP algorithms is quite different from A* algorithms, as they are *bottom-up* rather than *top-down*. This means that the DP algorithms start by considering 1-step policies for the final stage of the Dec-POMDPs and iteratively build larger (t-step) policy trees covering the last t stages of the Dec-POMDP for some t. Hansen *et al.* [2004] propose an exact dynamic programming algorithm that incrementally prunes policy trees for each t, thereby eliminating policy trees that are *dominated*, i.e. that cannot be optimal regardless of the policy used by other agents. The idea of eliminating dominated policy trees is similar to the last stage optimizations that we propose in Section 5.6.1, however.

Another exact dynamic programming algorithm was proposed by Szer and Charpillet [2006]. Their point-based dynamic programming algorithm takes into account which beliefs are reachable and only computes optimal policy trees for those beliefs.

Several adaptations have been proposed to create an approximate algorithm. Memory Bounded Dynamic Programming (MBDP) [Seuken and Zilberstein, 2007b] limits the number of policies considered of a particular length. This is somewhat similar to queue pruning. Improved MBDP [Seuken and Zilberstein, 2007a] improves scalability w.r.t. the number of observations by only considering the most likely observations and choosing a fixed policy for the remaining observations. This technique is reminiscent of probability-based clustering. MBDP-OC (MDBP with Observation Compression) [Carlin and Zilberstein, 2008] instead groups observations to reduce the effective number of observations. MBDP-AS (MDBP Anyspace) [Kumar and Zilberstein, 2009] further reduces space requirements by pruning policies based on similarity.

11.3 Other Algorithms for Dec-POMDP Solving

Early work on approximately solving Dec-POMDPs includes JESP [Nair *et al.*, 2003], which computes a Nash equilibrium, and DICEPS [Oliehoek *et al.*, 2008a] which uses the cross-entropy method. The genetic algorithm GA-FSC [Eker and Akin, 2013] searches through finite state controllers and finds the best known policies on several benchmarks. The state-of-the-art ϵ-optimal algorithm FB-HSVI transforms the Dec-POMDP into a continuous-state MDP [Dibangoye *et al.*, 2016] and solves it using an adaption of heuristic search value iteration. Wu *et al.* [2017] propose a baseline regret minimization algorithm to improve existing policies. In the area of exact solving, there has also been work on mixed-integer linear programming [Aras and Dutech, 2010]. Finally, Dibangoye *et al.* [2012] present a dedicated fast ϵ-optimal algorithm for Dec-MDPs.

11.4 Reinforcement Learning

There is also significant work on model-free reinforcement learning (RL) algorithms. Kraemer and Banerjee [2016] propose RLaR, in which agents rehearse using information not available during execution. Bono *et al.* [2018] improve policy gradient techniques in the multi-agent setting. Dibangoye and Buffet [2018] propose oSARSA, which uses a continuous-state MDP. Mao *et al.* [2020] use machine learning techniques to create an information state embedding. Finally, the model-based RL algorithm Team-Imitate-Synchronize [Abdoo *et al.*, 2022] learns a centralized team policy, which is imitated by a decentralized policy and improved using synchronization.

11.5 Other Related Work

Finally, we discuss some related work that solve slightly different problems. Amato *et al.* [2019] consider Dec-POMDPs with macro-actions. Lauri *et al.* [2020] and Lauri and Oliehoek [2020] consider information-gathering Dec-POMDPs, with a terminal reward depending on the information that agents have jointly gathered about state of the Dec-POMDP. Xie *et al.* [2020] and Peralez *et al.* [2024] consider problems with one-sided information sharing.

Finally, there is also related work in the area of infinite-horizon Dec-POMDP solving [Amato *et al.*, 2010; Pajarinen and Peltonen, 2011; MacDermed and Isbell, 2013; Dibangoye *et al.*, 2014b; You *et al.*, 2021]. However, in the context of infinite-horizon Dec-POMDPs, policies are usually represented as finite state controllers (FSCs).

12 Conclusion

In this section, we summarize the thesis and provide some directions for future research. First, we give a summary of our theoretical and empirical contributions.

Clustering. In Section 4, we discussed clustering. We first provided an overview of lossless incremental clustering as introduced by Oliehoek *et al.* [2009], which remains an important ingredient in our exact algorithms and algorithms that find upper bounds on the optimal value. Next, we introduced sliding k-window memory. By avoiding the exponential blowup typically observed for lossless clustering, sliding k-window memory allows us to specify policies for larger horizons. Furthermore, we developed a lossless clustering for sliding k-window memory, which allows us to further reduce the number of clusters without additional loss in policy value compared to just using sliding k-window memory. This is an important ingredient for our policy-finding algorithm PF-MAA*. Finally, we discussed probability-based clustering, which is a first attempt to reduce the number of clusters even further, with only limited cost in policy value.

Search. In Section 5, we studied a series of search algorithms for Dec-POMDPs building on MAA*. We first provided a number of general properties of A* algorithms. Next, we introduced existing algorithms from the literature: MAA*, GMAA* and GMAA*-ICE. Thereafter we introduced our novel small-step search tree. Finally, we introduced optimizations to find optimal policies for the last stage of the Dec-POMDP faster.

Heuristics. In Section 6, 7, 8 and 9, we provided an arsenal of heuristics for our A* search. We started by discussing heuristics from the literature (Q_{MDP}, Q_{POMDP} and Q_{BG}) and extending them to the small-step search tree. We also provided a faster precomputation for the Q_{BG} heuristic. Next, we introduced a general framework for admissible heuristics based on revealing additional information to the agents. In particular, we showed how revealing the first d joint observations gives rise to a heuristic that can be computed recursively, using Dec-POMDPs with a smaller horizon. By terminating A* searches computing these heuristics early, we make this recursive heuristic more scalable while preserving tightness. Next, we studied what properties make a heuristic a good heuristic for policy-finding, and provided two such heuristics. Finally, we used our general framework again to define heuristics that occasionally reveal the state. These heuristics are more scalable than the recursive heuristics, but still very tight. These heuristics can be computed using Dec-POMDPs with terminal rewards.

Algorithms. We synthesize these contributions into three algorithms, as outlined in the introduction. All three algorithms build on small-step MAA* (Section 5.5). Recursive small-step multi-agent A* (RS-MAA*) is our exact algorithm. It applies small-step MAA* with lossless clustering [Oliehoek *et al.*, 2009] and a recursive heuristic with early termination $Q_{M,d}$ (as introduced in Section 7.1 and 7.2). Policy-finding multi-agent A* (PF-MAA*) is our policy-finding algorithm. It applies small-step MAA* with clustered sliding window memory (Section 4.5), queue pruning (Section 8.1) and the heuristic $Q_{\mathsf{maxr},r}$ or $Q_{\mathsf{MDP},r}$ (Section 8.2). Terminal reward multi-agent A* (TR-MAA*) computes upper bounds. It applies small-step MAA* with lossless clustering and a terminal reward heuristic (Section 9.1). Together, the algorithms PF-MAA* and TR-MAA* form an approximate solver with guarantees.

Experiments. Finally, we implemented the algorithms in a common code base and evaluated the algorithms on a rich set of standard benchmarks. Our exact algorithm RS-MAA* is able to reach higher horizons on all 'hard' benchmarks, i.e. on all benchmarks where existing algorithms could not already scale up to horizon 500 or further. Our policy-finding algorithm PF-MAA* is able to find good policies fast for horizons up to at least 50 for every benchmark. Finally, our upper bound algorithm TR-MAA* is able to compute upper bounds tighter than the MDP heuristic on all benchmarks up to horizon at least 50. On BoxPushing, the provided upper bound is only 0.5% larger than the value of our best policy, for horizons up to 100.

Future Research

We now identify some directions for future research in Dec-POMDP solving.

- **Lossless clustering w.r.t. an arbitrary clustering.** It would be interesting to develop methods to find a lossless clustering with respect to any given clustering. This is in particular useful when already having a clustering which is known to be lossless based on domain knowledge, but where the probabilistic equivalence criterion does not identify this.
- **Lossy clusterings.** Secondly, more refined variations of probability-based clustering could be considered, which (for example) also take into account the distance between the beliefs of the suffixes that are clustered. In addition, it would be interesting to show bounds on the resulting loss in policy value, as is possible for POMDPs (see, e.g. Pineau *et al.* [2003]).
- **Clustering methods from machine learning.** Another possible direction is using clustering methods from machine learning, such as k-means clustering. When doing this, a method needs to be found to deal efficiently with the belief over the information of other agents, to avoid having to cluster in a space which is too high-dimensional.
- **Cluster order in search.** The Bayesian game solver used internally in GMAA*-ICE orders (joint) clusters within each stage by probability. In contrast, small-step MAA* orders (local) clusters lexicographically. It remains to be investigated whether ordering local clusters by probability (within each stage) yields an improvement for small-step MAA*. In addition, it may be possible to choose an ordering which takes into account which heuristics have been computed already, to limit the time spent on computing new heuristics.
- **Improved upper bounds by focusing on important clusters.** To find upper bounds, we currently use lossless clustering. To improve the upper bounds, one possibility would be to discard clusters with a low probability, using an appropriate upper bound the reward on discarded clusters. This would allow us to tighten the bound on high-probability clusters in later stages. Besides implementation, a research question here is to come up with a criterion regarding which clusters to discard.
- **Automated heuristic selection.** The best heuristic varies from benchmark to benchmark. For better ease of use on new benchmarks, it would be useful to develop tools that select an appropriate heuristic automatically.
- **Using a portfolio of heuristics for policy-finding.** A related, but slightly different direction is the use of a portfolio of heuristics for policy-finding. Rather than using one heuristic, that has to appropriately value both information gathering actions and communication actions, it might be possible to use one heuristic that values information gathering actions and one heuristic that values communication actions. In addition to coming up with suitable heuristics, this requires designing a way to combine the heuristics.
- **Factored models.** In several cases, a main limiting factor for scaling further in our solver is working with distributions over states and observation histories, due to the size of the state space. In order to be able to scale up to models with a larger state space that are hence able to model real-world cases more realistically, we propose to consider factored models, where the state space consists of several independent components.

References

[Abdoo *et al.*, 2022] Eliran Abdoo, Ronen I. Brafman, Guy Shani, and Nitsan Soffair. Team-Imitate-Synchronize for solving Dec-POMDPs. In *ECML/PKDD*, pages 216–232, 2022.

[Amato and Zilberstein, 2009] Christopher Amato and Shlomo Zilberstein. Achieving goals in decentralized POMDPs. In *AAMAS*, pages 593–600, 2009.

[Amato *et al.*, 2006] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimal fixed-size controllers for decentralized POMDPs. In *AAMAS MSDM workshop*, 2006.

[Amato *et al.*, 2007] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *UAI*, pages 1–8, 2007.

[Amato *et al.*, 2009] Christopher Amato, Jilles Steeve Dibangoye, and Shlomo Zilberstein. Incremental policy generation for finite-horizon DEC-POMDPs. In *ICAPS*, 2009.

[Amato *et al.*, 2010] Christopher Amato, Blai Bonet, and Shlomo Zilberstein. Finite-state controllers based on mealy machines for centralized and decentralized POMDPs. In *AAAI*, pages 1052–1058, 2010.

[Amato *et al.*, 2019] Christopher Amato, George D. Konidaris, Leslie P. Kaelbling, and Jonathan P. How. Modeling and planning with macro-actions in decentralized POMDPs. *JAIR*, 64:817–859, 2019.

[Aras and Dutech, 2010] Raghav Aras and Alain Dutech. An investigation into mathematical programming for finite horizon decentralized POMDPs. *JAIR*, 37:329–396, 2010.

[Åström, 1965] Karl Johan Åström. Optimal control of Markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1):174–205, 1965.

[Becker *et al.*, 2004] Raphen Becker, Shlomo Zilberstein, Victor R. Lesser, and Claudia V. Goldman. Solving transition independent decentralized Markov decision processes. *JAIR*, 22:423–455, 2004.

[Bellman, 1957] Richard Bellman. A Markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[Bernstein *et al.*, 2002] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[Bhustali and Andriotis, 2023] Prateek Bhustali and Charalampos P. Andriotis. Assessing the optimality of decentralized inspection and maintenance policies for stochastically degrading engineering systems. In *BNAIC/BeNeLearn 2023: Joint International Scientific Conferences on AI and Machine Learning*, 2023.

[Bono *et al.*, 2018] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron, and Olivier Simonin. Cooperative multi-agent policy gradient. In *ECML/PKDD*, volume 11051 of *Lecture Notes in Computer Science*, pages 459–476. Springer, 2018.

[Bork *et al.*, 2020] Alexander Bork, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. Verification of indefinite-horizon POMDPs. In *ATVA*, volume 12302 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2020.

[Boutilier, 1996] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *TARK*, pages 195–210, 1996.

[Carlin and Zilberstein, 2008] Alan Carlin and Shlomo Zilberstein. Value-based observation compression for DEC-POMDPs. In *AAMAS*, pages 501–508, 2008.

[Cassandra *et al.*, 1997] Anthony R. Cassandra, Michael L. Littman, and Nevin Lianwen Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *UAI*, pages 54–61, 1997.

[Cazenave, 2010] Tristan Cazenave. Partial move A*. In *22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 25–31. IEEE, 2010.

[Chanel *et al.*, 2013] Caroline Ponzoni Carvalho Chanel, Florent Teichteil-Königsbuch, and Charles Lesire. Multi-target detection and recognition by UAVs using online POMDPs. In *AAAI*, pages 1381–1387, 2013.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2006.

[Dibangoye and Buffet, 2018] Jilles Steeve Dibangoye and Olivier Buffet. Learning to act in continuous Dec-POMDPs. In *JFPDA*. HAL, 2018.

[Dibangoye *et al.*, 2012] Jilles Steeve Dibangoye, Christopher Amato, and Arnaud Doniec. Scaling up decentralized MDPs through heuristic search. In *UAI*, pages 217–226, 2012.

[Dibangoye *et al.*, 2014a] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. Optimally solving Dec-POMDPs as continuous-state MDPs: Theory and algorithms. Technical report, Tech. Rep. RR-8517, Inria, 2014.

[Dibangoye *et al.*, 2014b] Jilles Steeve Dibangoye, Olivier Buffet, and François Charpillet. Error-bounded approximations for infinite-horizon discounted decentralized pomdps. In *ECML/PKDD (1)*, pages 338–353, 2014.

[Dibangoye *et al.*, 2016] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. Optimally solving Dec-POMDPs as continuous-state MDPs. *JAIR*, 55:443–497, 2016.

[Dujardin *et al.*, 2015] Yann Dujardin, Tom Dietterich, and Iadine Chades. α-min: A compact approximate solver for finite-horizon POMDPs. In *IJCAI*, pages 2582–2588, 2015.

[Dujardin *et al.*, 2017] Yann Dujardin, Tom Dietterich, and Iadine Chades. Three new algorithms to solve N-POMDPs. In *AAAI*, pages 4495–4501, 2017.

[Eaton *et al.*, 2017] Christopher M. Eaton, Edwin K. P. Chong, and Anthony A. Maciejewski. Robust UAV path planning using POMDP with limited FOV sensor. In *CCTA*, pages 1530–1535. IEEE, 2017.

[Eker and Akin, 2013] Baris Eker and H. Levent Akin. Solving decentralized POMDP problems using genetic algorithms. *AAMAS*, 27(1):161–196, 2013.

[Emery-Montemerlo *et al.*, 2004] Rosemary Emery-Montemerlo, Geoffrey J. Gordon, Jeff G. Schneider, and Sebastian Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS*, pages 136–143, 2004.

[Feng and Zilberstein, 2004] Zhengzhu Feng and Shlomo Zilberstein. Region-based incremental pruning for POMDPs. In *UAI*, pages 146–153, 2004.

[Goldman and Zilberstein, 2004] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR*, 22:143–174, 2004.

[Hansen and Bowman, 2020] Eric A. Hansen and Thomas Bowman. Improved vector pruning in exact algorithms for solving POMDPs. In *UAI*, pages 1258–1267, 2020.

[Hansen *et al.*, 2004] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, pages 709–715, 2004.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968.

[Hauskrecht, 2000] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *JAIR*, 13:33–94, 2000.

[Hemmati *et al.*, 2015] Mahdi Hemmati, Abdulsalam Yassine, and Shervin Shirmohammadi. A Dec-POMDP model for congestion avoidance and fair allocation of network bandwidth in rate-adaptive video streaming. In *SSCI*, pages 1182–1189. IEEE, 2015.

[Holte, 2010] Robert C. Holte. Common misconceptions concerning heuristic search. In *SOCS*, pages 46–51. AAAI Press, 2010.

[Kaelbling *et al.*, 1998] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[Koops *et al.*, 2023] Wietze Koops, Nils Jansen, Sebastian Junges, and Thiago D. Simão. Recursive small-step multi-agent A* for Dec-POMDPs. In *IJCAI*, pages 5402–5410, 2023.

[Koops *et al.*, 2024] Wietze Koops, Sebastian Junges, and Nils Jansen. Approximate Dec-POMDP solving using multi-agent A*. In *IJCAI*, pages 6743–6751, 2024.

[Kraemer and Banerjee, 2016] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[Kumar and Zilberstein, 2009] Akshat Kumar and Shlomo Zilberstein. Dynamic programming approximations for partially observable stochastic games. In *FLAIRS*. AAAI Press, 2009.

[Kurniawati *et al.*, 2008] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*. The MIT Press, 2008.

[Lauri and Oliehoek, 2020] Mikko Lauri and Frans A. Oliehoek. Multi-agent active perception with prediction rewards. In *NeurIPS*, 2020.

[Lauri *et al.*, 2020] Mikko Lauri, Joni Pajarinen, and Jan Peters. Multi-agent active information gathering in discrete and continuous-state decentralized POMDPs by policy graph improvement. *AAMAS*, 34(2):42, 2020.

[Littman *et al.*, 1995] Michael L. Littman, Anthony R. Cassandra, and Leslie P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, pages 362–370, 1995.

[Lovejoy, 1991] William S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Oper. Res.*, 39(1):162–175, 1991.

[Lusena *et al.*, 2001] Christopher Lusena, Judy Goldsmith, and Martin Mundhenk. Nonapproximability results for partially observable Markov decision processes. *JAIR*, 14:83–103, 2001.

[Lyu *et al.*, 2023] Mingyang Lyu, Yibo Zhao, Chao Huang, and Hailong Huang. Unmanned aerial vehicles for search and rescue: A survey. *Remote. Sens.*, 15(13):3266, 2023.

[MacDermed and Isbell, 2013] Liam MacDermed and Charles L. Isbell. Point based value iteration with optimal belief compression for Dec-POMDPs. In *NeurIPS*, pages 100–108, 2013.

[Mao *et al.*, 2020] Weichao Mao, Kaiqing Zhang, Erik Miehling, and Tamer Basar. Information state embedding in partially observable cooperative multi-agent reinforcement learning. In *CDC*, pages 6124–6131. IEEE, 2020.

[Mérõ, 1984] László Mérõ. A heuristic search algorithm with modifiable estimate. *Artif. Intell.*, 23(1):13–27, 1984.

[Monahan, 1982] George E. Monahan. State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.

[Nair *et al.*, 2003] Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, pages 705–711, 2003.

[Oliehoek and Amato, 2016] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Briefs in Intelligent Systems. Springer, 2016.

[Oliehoek and Vlassis, 2007] Frans A. Oliehoek and Nikos Vlassis. Q-value heuristics for approximate solutions of Dec-POMDPs. In *AAAI Spring Symposium: Game Theoretic and Decision Theoretic Agents*, pages 31–37, 2007.

[Oliehoek *et al.*, 2008a] Frans A. Oliehoek, Julian F. P. Kooij, and Nikos Vlassis. The cross-entropy method for policy search in decentralized POMDPs. *Informatica (Slovenia)*, 32(4):341–357, 2008.

[Oliehoek *et al.*, 2008b] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *JAIR*, 32:289–353, 2008.

[Oliehoek *et al.*, 2009] Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *AAMAS*, pages 577–584, 2009.

[Oliehoek *et al.*, 2010] Frans A. Oliehoek, Matthijs T. J. Spaan, Jilles Steeve Dibangoye, and Christopher Amato. Heuristic search for identical payoff bayesian games. In *AAMAS*, pages 1115–1122, 2010.

[Oliehoek *et al.*, 2013] Frans A. Oliehoek, Matthijs T. J. Spaan, Christopher Amato, and Shimon Whiteson. Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *JAIR*, 46:449–509, 2013.

[Oliehoek *et al.*, 2017] Frans A. Oliehoek, Matthijs T. J. Spaan, Bas Terwijn, Philipp Robbel, and João V. Messias. The MADP toolbox: An open source library for planning and learning in (multi-)agent systems. *JMLR*, 18:1–5, 2017.

[Oliehoek, 2010] Frans A. Oliehoek. *Value-based planning for teams of agents in stochastic partially observable environments*. 2010. PhD thesis, University of Amsterdam.

[Pajarinen and Peltonen, 2011] Joni Pajarinen and Jaakko Peltonen. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *NeurIPS*, pages 2636–2644, 2011.

[Papadimitriou and Tsitsiklis, 1987] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[Peralez *et al.*, 2024] Johan Peralez, Aurélien Delage, Olivier Buffet, and Jilles Steeve Dibangoye. Solving hierarchical information-sharing dec-POMDPs: An extensive-form game approach. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pages 40414–40438, 2024.

[Pineau *et al.*, 2003] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

[Pynadath and Tambe, 2002] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.

[Rabinovich *et al.*, 2003] Zinovi Rabinovich, Claudia V. Goldman, and Jeffrey S. Rosenschein. The complexity of multiagent systems: the price of silence. In *AAMAS*, pages 1102–1103, 2003.

[Roth *et al.*, 2005] Maayan Roth, Reid G. Simmons, and Manuela M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *AAMAS*, pages 786–793, 2005.

[Russell and Norvig, 2020] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[Russell, 1992] Stuart Russell. Efficient memory-bounded search methods. In *ECAI*, pages 1–5, 1992.

[Seuken and Zilberstein, 2007a] Sven Seuken and Shlomo Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *UAI*, pages 344–351. AUAI Press, 2007.

[Seuken and Zilberstein, 2007b] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *IJCAI*, pages 2009–2015, 2007.

[Seuken and Zilberstein, 2008] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *AAMAS*, 17(2):190–250, 2008.

[Silver and Veness, 2010] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *NeurIPS*, pages 2164–2172, 2010.

[Smallwood and Sondik, 1973] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Oper. Res.*, 21(5):1071–1088, 1973.

[Smith and Simmons, 2004] Trey Smith and Reid G. Simmons. Heuristic search value iteration for POMDPs. In *UAI*, pages 520–527, 2004.

[Smith and Simmons, 2005] Trey Smith and Reid G. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, pages 542–547, 2005.

[Somani *et al.*, 2013] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. DESPOT: online POMDP planning with regularization. In *NeurIPS*, pages 1772–1780, 2013.

[Spaan and Melo, 2008] Matthijs T. J. Spaan and Francisco S. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *AAMAS*, pages 525–532, 2008.

[Spaan *et al.*, 2011] Matthijs T. J. Spaan, Frans A. Oliehoek, and Christopher Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *IJCAI*, pages 2027–2032, 2011.

[Spaan, 2012] Matthijs T. J. Spaan. Partially observable Markov decision processes. In Marco A. Wiering and Martijn van Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 387–414. Springer, 2012.

[Szer and Charpillet, 2006] Daniel Szer and François Charpillet. Point-based dynamic programming for dec-pomdps. In *AAAI*, pages 1233–1238, 2006.

[Szer *et al.*, 2005] Daniel Szer, François Charpillet, and Shlomo Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *UAI*, pages 576–590, 2005.

[Tang *et al.*, 2020] Qinqin Tang, Renchao Xie, Fei Richard Yu, Tao Huang, and Yunjie Liu. Decentralized computation offloading in IoT fog computing system with energy harvesting: A Dec-POMDP approach. *IEEE Internet Things J.*, 7(6):4898–4911, 2020.

[Waharte and Trigoni, 2010] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *EST*, pages 142–147. IEEE Computer Society, 2010.

[Walraven and Spaan, 2017] Erwin Walraven and Matthijs T. J. Spaan. Accelerated vector pruning for optimal POMDP solvers. In *AAAI*, pages 3672–3678, 2017.

[Wu *et al.*, 2017] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Multi-agent planning with baseline regret minimization. In *IJCAI*, pages 444–450, 2017.

[Xie *et al.*, 2020] Yuxuan Xie, Jilles Dibangoye, and Olivier Buffet. Optimally solving two-agent decentralized POMDPs under one-sided information sharing. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 10473–10482, 2020.

[You *et al.*, 2021] Yang You, Vincent Thomas, Francis Colas, and Olivier Buffet. Solving infinite-horizon Dec-POMDPs using finite state controllers within JESP. In *ICTAI*, pages 427–434. IEEE, 2021.

[Zhang and Liu, 1996] Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical report, 1996.

[Zhang *et al.*, 2015] Jing Zhang, Li Xu, Shuming Zhou, and Xiucai Ye. A novel sleep scheduling scheme in green wireless sensor networks. *J. Supercomput.*, 71(3):1067–1094, 2015.

[Zhou and Hansen, 2001] Rong Zhou and Eric A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, pages 707–716, 2001.

[Zhu *et al.*, 2021] Xiaolong Zhu, Fernando Vanegas, Felipe Gonzalez, and Conrad Sanderson. A multi-UAV system for exploration and target finding in cluttered and GPS-denied environments. In *International Conference on Unmanned Aircraft Systems*, pages 721–729. IEEE, 2021.

A Omitted proofs

A.1 Existence of α-vectors (Theorem 2.2)

In this appendix we prove the existence of a representation of the value function V^* of the belief MDP corresponding to a POMDP using α-vectors. As a result, we show that the value function is piecewise linear and convex.

Theorem 2.2 ([Smallwood and Sondik, 1973]). *There exists a representation of the value function V^* using α-vectors: For each t there exist vectors $\alpha_1, \ldots, \alpha_z \in \mathbb{R}^{\mathcal{S}}$ such that*

$$V^*(b,t) = \max_{1 \le v \le z} \sum_{s \in \mathcal{S}} \alpha_v(s) b(s).$$

Hence, the value function is piecewise linear and convex (for a fixed stage t).

Proof. We construct the α-vectors inductively. For $t = 1$, we have the $|\mathcal{A}|$ vectors corresponding to the reward vectors of each action. Formally, if $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$, then we can define the vectors $\alpha_v \in \mathbb{R}^{\mathcal{S}}$ by $\alpha_v(s) = R(s, a_v)$ for $1 \le v \le |\mathcal{A}|$. Now suppose that a representation of the value function using α-vectors exists for stage t. Then we can write $V^*(b,t) = \max_{v \in [z]} \sum_{s \in \mathcal{S}} \alpha_v(s) b(s)$, where we write $[z] = \{1, \ldots, z\}$. We now show that there also exists a representation of the value function using α-vectors for stage $t+1$. We have

$$\begin{aligned}
V^*(b,t+1) &= \max_{a \in \mathcal{A}} \left[R(b,a) + \sum_{o \in \mathcal{O}} \Pr(o \mid b,a) V^*(b_{a,o}, t) \right] \\
&= \max_{a \in \mathcal{A}} \left[R(b,a) + \sum_{o \in \mathcal{O}} \left[\Pr(o \mid b,a) \max_{v_o \in [z]} \sum_{s' \in \mathcal{S}} \alpha_{v_o}(s') b_{a,o}(s') \right] \right] \\
&= \max_{a \in \mathcal{A}} \max_{v \in [z]^{\mathcal{O}}} \left[R(b,a) + \sum_{o \in \mathcal{O}} \left[\Pr(o \mid b,a) \sum_{s' \in \mathcal{S}} \alpha_{v_o}(s') b_{a,o}(s') \right] \right] \\
&= \max_{(a,v) \in \mathcal{A} \times [z]^{\mathcal{O}}} \left[R(b,a) + \sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} \left[\alpha_{v_o}(s') \Pr(o \mid b,a) b_{a,o}(s') \right] \right] \\
&= \max_{(a,v) \in \mathcal{A} \times [z]^{\mathcal{O}}} \left[R(b,a) + \sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} \left[\alpha_{v_o}(s') O(o \mid a, s') \sum_{s \in \mathcal{S}} T(s' \mid s,a) b(s) \right] \right] \\
&= \max_{(a,v) \in \mathcal{A} \times [z]^{\mathcal{O}}} \left[R(b,a) + \sum_{s \in \mathcal{S}} \left[\sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} \alpha_{v_o}(s') O(o \mid a, s') T(s' \mid s,a) \right] b(s) \right] \\
&= \max_{(a,v) \in \mathcal{A} \times [z]^{\mathcal{O}}} \sum_{s \in \mathcal{S}} \left[R(s,a) + \sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} \alpha_{v_o}(s') O(o \mid a, s') T(s' \mid s,a) \right] b(s).
\end{aligned}$$

In the first line we use the value iteration equation for POMDPs. In the second line we substitute the representation of $V^*(b_{a,o}, t)$ using α-vectors. In the third line we interchange the summation over $o \in \mathcal{O}$ and the maximization over $v_o \in [z]$. As a result, we have to maximize over v_o for each $o \in \mathcal{O}$ outside of the summation. The fourth line is rewriting. In the fifth line we substitute in the expression for $b_{a,o}(s')$ derived in Section 2.3.1, and the last two lines are rewriting steps. This shows that we have a representation of the value function using α-vectors for stage $t+1$, which completes the induction, and hence the proof that there exists a representation of the value function using α-vectors.

It remains to show that the value function is piecewise linear and convex. For this, note that the value function is linear on each set $M_v = \{b \in \Delta(\mathcal{S}) \colon \alpha_v(s) b(s) \ge \alpha_{v'}(s) b(s) \text{ for all } 1 \le v' \le z\}$. Since the sets M_v are polyhedral sets and together cover the full belief simplex $\Delta(\mathcal{S})$, it follows that the value function is piecewise linear. Finally, since the value function is the pointwise maximum of linear functions, it is the pointwise maximum of convex functions. Since the pointwise maximum of convex functions is convex, we conclude that the value function is convex. □

A.2 Clustering (Section 4)

We proceed by giving the omitted proofs from the section on clustering (Section 4). Specifically, we show that the multi-agent belief is a sufficient statistic, that Equation (4.2) does not yield an incremental clustering, that the forget operator is well-defined, that suffix-equivalence is an equivalence relation, that equivalence classes of suffix-equivalence correspond to a suffix, and that approximate equivalence is an equivalence relation.

A.2.1 Multi-agent belief is a sufficient statistic (Lemma 4.1)

We now prove Lemma 4.1, which states that the multi-agent belief of an agent is a sufficient statistic for computing the value (i.e., the expected reward) of a subtree policy of that agent.

Lemma 4.1 (Multi-agent belief is a sufficient statistic)**.** *The multi-agent belief for agent $i \in \mathcal{D}$ and stage t ($0 \leq t \leq h-1$) is a sufficient statistic for computing the value of any subtree policy $\gamma_i^{h-t} \colon \mathcal{O}_i^{\leq h-t-1} \to \mathcal{A}_i$ that agent i can use from stage t onwards.*

Proof. We first show that we can calculate the value $V\Big((s^t, \boldsymbol{\gamma}_{\neq i}^{h-t}), \gamma_i^{h-t}\Big)$ of a subtree policy γ_i^{h-t} of agent i corresponding to a multi-agent belief state $(s^t, \boldsymbol{\gamma}_{\neq i}^{h-t})$. We prove this by induction on t, starting at $t = h-1$ downwards to $t = 0$. For $t = h-1$, the subtree policy γ_i^{h-t} specifies one action a_i^{h-1} for agent i while $\boldsymbol{\gamma}_{\neq i}^{h-t}$ specifies actions $\mathbf{a}_{\neq i}^{h-1}$ for the agents other than agent i. Together, this gives a joint action $\mathbf{a}^{h-1}$, and $V\Big((s^t, \boldsymbol{\gamma}_{\neq i}^{h-t}), \gamma_i^{h-t}\Big) = R(s^{h-1}, \mathbf{a}^{h-1})$ for $t = h-1$.

To go from stage t to stage $t-1$, we first define the *policy consumption operator*, denoted by $\Downarrow$ [Oliehoek, 2010]. For an agent j, let $\big(\gamma_j^{h-u}\Downarrow_{o_j^{u+1}}\big) \colon \mathcal{O}_j^{\leq h-u-2} \to \mathcal{A}_j$ be the subtree policy defined by $\big(\gamma_j^{h-u}\Downarrow_{o_j^{u+1}}\big)\big(o_j^{[u+2,v]}\big) = \gamma_j^{h-u}\big(o_j^{[u+1,v]}\big)$. Moreover, define

$$\boldsymbol{\gamma}_{\neq i}^{h-u}\Downarrow_{\mathbf{o}^{u+1}} = \Big\langle \gamma_1^{h-u}\Downarrow_{o_1^{u+1}}, \ldots, \gamma_{i-1}^{h-u}\Downarrow_{o_{i-1}^{u+1}}, \gamma_{i+1}^{h-u}\Downarrow_{o_{i+1}^{u+1}}, \ldots, \gamma_n^{h-u}\Downarrow_{o_n^{u+1}} \Big\rangle.$$

The subtree policy γ_i^{h-t+1} specifies one action a_i^{t-1} for agent i while $\boldsymbol{\gamma}_{\neq i}^{h-t+1}$ specifies actions $\mathbf{a}_{\neq i}^{t-1}$ for the agents other than agent i. Together, this gives a joint action $\mathbf{a}^{t-1}$. By conditioning on the next state s^t and the next observation $\mathbf{o}^t$, we can write

$$\begin{aligned} &V\Big((s^{t-1}, \boldsymbol{\gamma}_{\neq i}^{h-t+1}), \gamma_i^{h-t+1}\Big) \\ &\quad = R(s, \mathbf{a}^{t-1}) + \sum_{s^t \in \mathcal{S}} \sum_{\mathbf{o}^t \in \boldsymbol{\mathcal{O}}} T(s^t \mid s^{t-1}, \mathbf{a}^{t-1}) O(\mathbf{o}^t \mid \mathbf{a}^{t-1}, s^t) V\Big((s^t, \boldsymbol{\gamma}_{\neq i}^{h-t+1}\Downarrow_{\mathbf{o}^t}), \gamma_i^{h-t+1}\Downarrow_{o_i^t}\Big), \end{aligned}$$

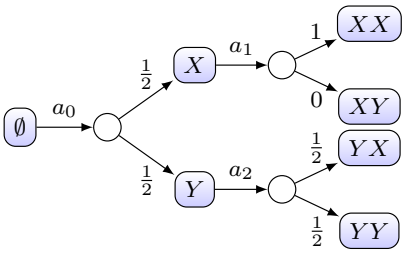
which shows that we can calculate the value of a subtree policy γ_i^{h-t+1} for a multi-agent belief state $(s^{t-1}, \boldsymbol{\gamma}_{\neq i}^{h-t+1})$ in stage $t-1$, which completes the induction.

Now suppose that we have a multi-agent belief b_i which assigns probabilities to multi-agent belief states $(s^t, \boldsymbol{\gamma}_{\neq i}^{h-t})$. Then we can compute the corresponding value of a subtree policy γ_i^{h-t} of agent i by taking the average over the values corresponding to the multi-agent belief states according to the probabilities specified by b_i:

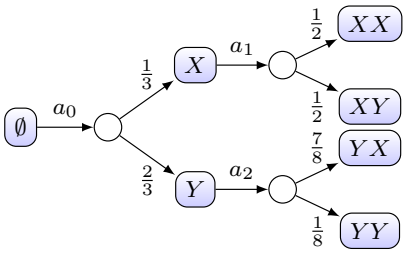
$$V\Big(b_i, \gamma_i^{h-t}\Big) = \sum_{s^t} \sum_{\boldsymbol{\gamma}_{\neq i}^{h-t}} b_i(s^t, \boldsymbol{\gamma}_{\neq i}^{h-t}) V\Big((s^t, \boldsymbol{\gamma}_{\neq i}^{h-t}), \gamma_i^{h-t}\Big).$$

This shows that we can compute the value of any subtree policy γ_i^{h-t} based on the multi-agent belief. Hence, the multi-agent belief is a sufficient statistic. □

Finally, we note that the value defined in this way matches the value of a policy as defined in Section 3, since both values compute the expected reward. Specifically, suppose that b_i is the multi-agent belief of agent i after observing LOH $o_i^{[1,t]}$ and γ_i^{h-t} is the subtree policy that agent i uses from LOH $o_i^{[1,t]}$ onwards. Then $V\big(b_i, \gamma_i^{h-t}\big)$ is the expected reward over the last $h-t$ stages conditional on agent i's LOH being $o_i^{[1,t]}$.



(a) Probabilities corresponding to state s_0.



(b) Probabilities corresponding to state s_1.

Figure 15: Probabilities for each of the observations given the true state, corresponding to Example A.1.

A.2.2 Equation (4.2) does not yield an incremental clustering

In this Appendix, we give an example showing that clustering two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ if

$$\Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \;\middle|\; o_i^{[v,t]}\right) = \Pr\left(s^t, \mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t \;\middle|\; \tilde{o}_i^{[v,t]}\right).$$

for all $v \in \{\ell_t, \ldots, t\}$, does not yield an incremental clustering.

Example A.1. *Consider a Dec-POMDP with one agent, two states (which we call s_0 and s_1), three actions (a_0, a_1 and a_2) and two observations (X and Y). The state does not change: $T(s' \mid a_i, s) = 1$ if $s = s'$, and 0 otherwise. For the observations, we have $O(X \mid a_0, s_0) = \frac{1}{2}$, $O(X \mid a_1, s_0) = 1$, $O(X \mid a_2, s_0) = \frac{1}{2}$, $O(X \mid a_0, s_1) = \frac{1}{3}$, $O(X \mid a_1, s_1) = \frac{1}{2}$ and $O(X \mid a_2, s_1) = \frac{7}{8}$. The probabilities for the observation Y can be computed as the complements of these probabilities. The initial belief assigns probability $\frac{1}{2}$ to both s_0 and s_1. Rewards are not relevant for this example.*

Now consider the following policy in stage 0 and 1: $\emptyset \mapsto a_0$, $(X) \mapsto a_1$, and $(Y) \mapsto a_2$. Figure 15 depicts the resulting observation probabilities for both states. We use windows of size $k = 2$.

We now show that (X, X) and (Y, Y) are equivalent according to Equation (4.2). We instead write these observation histories instead as XX and YY. Using Figure 15, we see that $\Pr(XX \mid s_0) = \frac{1}{2} \cdot 1 = \frac{1}{2}$ and $\Pr(XX \mid s_1) = \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$ and $\Pr(YY \mid s_0) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$ and $\Pr(YY \mid s_1) = \frac{2}{3} \cdot \frac{1}{8} = \frac{1}{12}$. Using Bayes' rule, it follows that

$$\Pr(s_0 \mid XX) = \frac{\Pr(XX \mid s_0)\Pr(s_0)}{\Pr(XX)} = \frac{\Pr(XX \mid s_0)\Pr(s_0)}{\Pr(XX|s_0)\Pr(s_0)+\Pr(XX|s_1)\Pr(s_1)} = \frac{\frac{1}{2}\cdot\frac{1}{2}}{\frac{1}{2}\cdot\frac{1}{2}+\frac{1}{6}\cdot\frac{1}{2}} = \frac{3}{4},$$

$$\Pr(s_0 \mid YY) = \frac{\Pr(YY \mid s_0)\Pr(s_0)}{\Pr(YY)} = \frac{\Pr(YY \mid s_0)\Pr(s_0)}{\Pr(YY|s_0)\Pr(s_0)+\Pr(YY|s_1)\Pr(s_1)} = \frac{\frac{1}{4}\cdot\frac{1}{2}}{\frac{1}{4}\cdot\frac{1}{2}+\frac{1}{12}\cdot\frac{1}{2}} = \frac{3}{4},$$

$$\Pr(s_1 \mid YY) = 1 - \Pr(s_0 \mid YY) = 1 - \Pr(s_0 \mid XX) = \Pr(s_1 \mid XX).$$

Hence, the observation histories XX and YY induce the same belief over the states. We also have to show that conditioning just on the suffixes $\cdot X$ and $\cdot Y$ (where the $\cdot$ denotes the 'forgotten' first observation) induces the same belief over the states. For this, we compute $\Pr(\cdot X \mid s_0) = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$ and $\Pr(\cdot X \mid s_1) = \frac{1}{3} \cdot \frac{1}{2} + \frac{2}{3} \cdot \frac{7}{8} = \frac{3}{4}$. Hence, $\Pr(\cdot Y \mid s_0) = 1 - \Pr(\cdot X \mid s_0) = \frac{1}{4}$ and $\Pr(\cdot Y \mid s_1) = 1 - \Pr(\cdot X \mid s_1) = \frac{1}{4}$. Again using Bayes' rule, we see that

$$\Pr(s_0 \mid \cdot X) = \frac{\Pr(\cdot X \mid s_0)\Pr(s_0)}{\Pr(\cdot X)} = \frac{\Pr(\cdot X \mid s_0)\Pr(s_0)}{\Pr(\cdot X \mid s_0)\Pr(s_0) + \Pr(\cdot X \mid s_1)\Pr(s_1)} = \frac{\frac{3}{4}\cdot\frac{1}{2}}{\frac{3}{4}\cdot\frac{1}{2}+\frac{3}{4}\cdot\frac{1}{2}} = \frac{1}{2},$$

$$\Pr(s_0 \mid \cdot Y) = \frac{\Pr(\cdot Y \mid s_0)\Pr(s_0)}{\Pr(\cdot Y)} = \frac{\Pr(\cdot Y \mid s_0)\Pr(s_0)}{\Pr(\cdot Y \mid s_0)\Pr(s_0) + \Pr(\cdot Y \mid s_1)\Pr(s_1)} = \frac{\frac{1}{4}\cdot\frac{1}{2}}{\frac{1}{4}\cdot\frac{1}{2}+\frac{1}{4}\cdot\frac{1}{2}} = \frac{1}{2},$$

$$\Pr(s_1 \mid \cdot Y) = 1 - \Pr(s_0 \mid \cdot Y) = 1 - \Pr(s_0 \mid \cdot X) = \Pr(s_1 \mid \cdot X).$$

Hence, also the suffixes $\cdot X$ and $\cdot Y$ induce the same belief over the state. Hence, we conclude that we cluster XX and YY according to Equation (4.2).

We also note that $\Pr(XY \mid s_0) = 0$*, so* $\Pr(s_0 \mid XY) = 0$ *and* $\Pr(s_1 \mid XY) = 1$*, so* XY *is in its own cluster. Finally, we can compute that* $\Pr(YX \mid s_0) = \frac{1}{4}$ *and* $\Pr(YX \mid s_1) = \frac{7}{12}$*, so* $\Pr(s_0 \mid YX) = \frac{3}{10}$ *and* $\Pr(s_1 \mid YX) = \frac{7}{10}$*, so also* YX *is in its own cluster. Now suppose that the agent chooses actions* $\{XX, YY\} \mapsto a_1$ *and* $XY \mapsto a_0$ *and* $YX \mapsto a_0$*.*

By incrementality, the observation histories XXX *and* YYX *should be clustered. Since we use windows of size* $k = 2$*, this means that the suffixes* $\cdot XX$ *and* $\cdot YX$ *should be clustered. We now show that this is not the case, since they do not induce the same belief over the states. We compute*

$$\begin{aligned}
\Pr(\cdot XX \mid s_0) &= \Pr(XXX \mid s_0) + \Pr(YXX \mid s_0) \\
&= \Pr(XX \mid s_0)O(X \mid a_1, s_0) + \Pr(YX \mid s_0)O(X \mid a_0, s_0) = \tfrac{1}{2} \cdot 1 + \tfrac{1}{4} \cdot \tfrac{1}{2} = \tfrac{5}{8}, \\
\Pr(\cdot YX \mid s_0) &= \Pr(XYX \mid s_0) + \Pr(YYX \mid s_0) \\
&= \Pr(XY \mid s_0)O(X \mid a_0, s_0) + \Pr(YY \mid s_0)O(X \mid a_1, s_0) = 0 + \tfrac{1}{4} \cdot 1 = \tfrac{1}{4}, \\
\Pr(\cdot XX \mid s_1) &= \Pr(XXX \mid s_1) + \Pr(YXX \mid s_1) \\
&= \Pr(XX \mid s_1)O(X \mid a_1, s_1) + \Pr(YX \mid s_1)O(X \mid a_0, s_1) = \tfrac{1}{6} \cdot \tfrac{1}{2} + \tfrac{7}{12} \cdot \tfrac{1}{3} = \tfrac{5}{18}, \\
\Pr(\cdot YX \mid s_1) &= \Pr(XYX \mid s_1) + \Pr(YYX \mid s_1) \\
&= \Pr(XY \mid s_1)O(X \mid a_0, s_1) + \Pr(YY \mid s_1)O(X \mid a_1, s_1) = \tfrac{1}{6} \cdot \tfrac{1}{3} + \tfrac{1}{12} \cdot \tfrac{1}{2} = \tfrac{7}{72}.
\end{aligned}$$

From this we finally compute using Bayes' rule that

$$\begin{aligned}
\Pr(s_0 \mid \cdot XX) &= \frac{\Pr(\cdot XX \mid s_0)\Pr(s_0)}{\Pr(\cdot XX)} = \frac{\frac{5}{8} \cdot \frac{1}{2}}{\frac{5}{8} \cdot \frac{1}{2} + \frac{5}{18} \cdot \frac{1}{2}} = \frac{9}{13}, \\
\Pr(s_0 \mid \cdot YX) &= \frac{\Pr(\cdot YX \mid s_0)\Pr(s_0)}{\Pr(\cdot YX)} = \frac{\frac{1}{4} \cdot \frac{1}{2}}{\frac{1}{4} \cdot \frac{1}{2} + \frac{7}{72} \cdot \frac{1}{2}} = \frac{18}{25},
\end{aligned}$$

which shows that the suffixes $\cdot XX$ *and* $\cdot YX$ *are not equivalent according to Equation* (4.2)*. Hence, we conclude that using Equation* (4.2) *does not yield an incremental clustering.*

A.2.3 Forget operator is well-defined (Lemma 4.6)

Lemma 4.6 (Well-definedness of forget operator)**.** *Fix a clustering* C *up to stage* $t-1$*. Then there exists a* finest *clustering* C^F *up to stage* t *agreeing with* C *up to stage* $t-1$*, which is incremental at stage* $t-1$ *and coarser than sliding* k*-window memory.*

Proof. Fix an agent i and let $\equiv_{C'_{i,t}}$ denote the equivalence relation corresponding to a partition $C'_{i,t}$. Let $\equiv_k$ denote the equivalence relation corresponding to sliding k-window memory.

Let $\mathcal{C}$ be the collection of all clusterings C' agreeing with C up to stage $t-1$ which are incremental at stage $t-1$ and coarser than sliding k-window memory. Note that the clustering which agrees with C up to stage $t-1$ and has only one cluster in stage t satisfies this, so $\mathcal{C}$ is non-empty. Define the relation $\equiv_F$ by

$$o_i^{[1,t]} \equiv_F \tilde{o}_i^{[1,t]} \iff \forall C' \in \mathcal{C} : o_i^{[1,t]} \equiv_{C'_{i,t}} \tilde{o}_i^{[1,t]}.$$

Then $\equiv_F$ is an equivalence relation inducing a partition $C^F_{i,t}$. Let C^F be the clustering agreeing with C up to stage $t-1$, extended with the partitions $C^F_{i,t}$ for $i \in \mathcal{D}$ as clustering in stage t.

If $o_i^{[1,t]} \equiv_k \tilde{o}_i^{[1,t]}$, then $o_i^{[1,t]} \equiv_{C'_{i,t}} \tilde{o}_i^{[1,t]}$ for all $C' \in \mathcal{C}$ (since each C' is coarser), so also $o_i^{[1,t]} \equiv_F \tilde{o}_i^{[1,t]}$, so $C^F_{i,t}$ is coarser than sliding k-window memory.

Now we show incrementality. Let $o_i^{[1,t-1]} \equiv_{C_{i,t-1}} \tilde{o}_i^{[1,t-1]}$ and let $o \in o_i$ be an observation, then

$$\forall C' \in \mathcal{C} : \left(o_i^{[1,t-1]} \cdot o\right) \equiv_{C_{i,t}} \left(\tilde{o}_i^{[1,t-1]} \cdot o\right)$$

since each C' is incremental at stage $t-1$. Hence, we have $\left(o_i^{[1,t-1]} \cdot o\right) \equiv_F \left(\tilde{o}_i^{[1,t-1]} \cdot o\right)$, so C^F is also incremental at stage $t-1$, as required.

Finally, from the definition of $\equiv_F$ it directly follows that C^F is finer than each C', so C^F is the finest clustering which is incremental at stage $t-1$ and coarser than sliding k-window memory. □

A.2.4 Suffix equivalence is an equivalence relation (Lemma 4.7)

Lemma 4.7. *Suffix-equivalence $\sim$ is an equivalence relation.*

Proof. Note that weak equivalence $\sim_w$ is clearly an equivalence relation. Throughout the proof, fix an agent i and a stage t.

We first show that $\sim$ is reflexive. Let $o_i^{[\ell_t,t]}$ be given. We need to show that $o_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$. Since the largest common suffix of $o_i^{[\ell_t,t]}$ and $o_i^{[\ell_t,t]}$ is $o_i^{[\ell_t,t]}$, we only have to show that $o_i^{[\ell_t,t]} \sim_w o_i^{[\ell_t,t]}$, which holds since $\sim_w$ is reflexive.

We now show that $\sim$ is symmetric. Let $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ with $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ be given. Let $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ be the largest common suffix of $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$. Hence, $o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$ for all $\hat{o}_i^{[\ell_t,t]}$ such that $o_i^{[m,t]} = \hat{o}_i^{[m,t]}$. In particular, we have $o_i^{[\ell_t,t]} \sim_w \tilde{o}_i^{[\ell_t,t]}$. By symmetry and transitivity of $\sim_w$ this implies $\tilde{o}_i^{[\ell_t,t]} \sim_w o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$, so $\tilde{o}_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$. Since $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ this implies $\tilde{o}_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$ for all $\hat{o}_i^{[\ell_t,t]}$ with $\tilde{o}_i^{[m,t]} = \hat{o}_i^{[m,t]}$. Hence, $\tilde{o}_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$, showing symmetry.

Finally, we show transitivity. Let $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ and $\bar{o}_i^{[\ell_t,t]}$ with $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]} \sim \bar{o}_i^{[\ell_t,t]}$ be given. Let $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ be the largest common suffix of $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ and let $\tilde{o}_i^{[m',t]} = \bar{o}_i^{[m',t]}$ be the largest common suffix of $\tilde{o}_i^{[\ell_t,t]}$ and $\bar{o}_i^{[\ell_t,t]}$. If $m \geq m'$, then $o_i^{[m,t]} = \tilde{o}_i^{[m,t]} = \bar{o}_i^{[m,t]}$. We have $o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$ for all $\hat{o}_i^{[\ell_t,t]}$ such that $o_i^{[m,t]} = \hat{o}_i^{[m,t]}$. Since $o_i^{[m,t]} = \bar{o}_i^{[m,t]}$ is a common suffix of $o_i^{[\ell_t,t]}$ and $\bar{o}_i^{[\ell_t,t]}$, this shows that $o_i^{[\ell_t,t]} \sim \bar{o}_i^{[\ell_t,t]}$.

If $m < m'$, then we can use the symmetry of $\sim$ to obtain $\bar{o}_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$. This reverses m and m', so now we can use the previous case to conclude that $\bar{o}_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$. Again using symmetry, we get $o_i^{[\ell_t,t]} \sim \bar{o}_i^{[\ell_t,t]}$. Hence, we conclude that transitivity holds.

Since $\sim$ is reflexive, symmetric and transitive, we conclude that $\sim$ is an equivalence relation. □

A.2.5 Clusters in suffix-equivalence correspond to a suffix (Lemma 4.8)

Lemma 4.8. *Let c_i^t be a cluster in clustered sliding window memory. Then there exists a suffix $o_i^{[m,t]}$ such that c_i^t contains precisely the suffixes with suffix $o_i^{[m,t]}$.*

Proof. Consider the largest common suffix for each pair of suffixes $o_i^{[\ell_t,t]}, \tilde{o}_i^{[\ell_t,t]}$ in c_i^t and take a shortest $o_i^{[m,t]}$ among them. We prove by contradiction that each suffix in c_i^t has this suffix $o_i^{[m,t]}$. Suppose that there exists a suffix that does not have suffix $o_i^{[m,t]}$. Take this suffix and another suffix that does have suffix $o_i^{[m,t]}$. Then these two suffixes have a shorter largest common suffix than $o_i^{[m,t]}$, contradiction. Hence, all suffixes in c_i^t have suffix $o_i^{[m,t]}$.

Hence, it remains to prove that c_i^t contains all suffixes with suffix $o_i^{[m,t]}$. Fix two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ in c_i^t with largest common suffix $o_i^{[m,t]}$. Then $o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$ for all $\hat{o}_i^{[\ell_t,t]}$ with suffix $o_i^{[m,t]}$. If $\bar{o}_i^{[\ell_t,t]}$ is any suffix with suffix $o_i^{[m,t]}$, then we hence have $o_i^{[\ell_t,t]} \sim_w \bar{o}_i^{[\ell_t,t]}$. By symmetry and transitivity of $\sim_w$ this implies $\bar{o}_i^{[\ell_t,t]} \sim_w o_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$, so $\bar{o}_i^{[\ell_t,t]} \sim_w \hat{o}_i^{[\ell_t,t]}$ for all $\hat{o}_i^{[\ell_t,t]}$ such that $\bar{o}_i^{[m,t]} = \hat{o}_i^{[m,t]}$. Hence, $\bar{o}_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$, which shows that $\bar{o}_i^{[\ell_t,t]} \in c_i^t$, so c_i^t contains precisely the suffixes with suffix $o_i^{[m,t]}$. □

A.2.6 Approximate equivalence is an equivalence relation (Lemma 4.13)

Lemma 4.13. *Approximate equivalence $\approx$ is an equivalence relation.*

Proof. Fix $p_{\max}$. Reflexivity of $\approx$ follows directly from the reflexivity of $\sim$.

We now show that $\approx$ is symmetric. Let $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ with $o_i^{[\ell_t,t]} \approx \tilde{o}_i^{[\ell_t,t]}$ be given. Let $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ be their largest common suffix. Then $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ or $\Pr(o_i^{[m,t]}) \leq p_{\max}$. By symmetry of $\sim$ this implies $\tilde{o}_i^{[\ell_t,t]} \sim o_i^{[\ell_t,t]}$ or $\Pr(\tilde{o}_i^{[m,t]}) \leq p_{\max}$, so $\tilde{o}_i^{[\ell_t,t]} \approx o_i^{[\ell_t,t]}$, showing symmetry.

We now show that $\approx$ is transitive. Let three suffixes $o_i^{[\ell_t,t]}$, $\hat{o}_i^{[\ell_t,t]}$ and $\bar{o}_i^{[\ell_t,t]}$ with $o_i^{[\ell_t,t]} \approx \hat{o}_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]} \approx \bar{o}_i^{[\ell_t,t]}$ be given. Let $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ be the largest common suffix of $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$

and let $\tilde{o}_i^{[m',t]} = \bar{o}_i^{[m',t]}$ be the largest common suffix of $\tilde{o}_i^{[\ell_t,t]}$ and $\bar{o}_i^{[\ell_t,t]}$. Since $\approx$ is symmetric we can assume without loss of generality that $m \geq m'$, as in the proof of Lemma 4.7. Then we have $o_i^{[m,t]} = \tilde{o}_i^{[m,t]} = \bar{o}_i^{[m,t]}$, so if $o_i^{[m'',t]} = \bar{o}_i^{[m'',t]}$ is the largest common suffix of $o_i^{[\ell_t,t]}$ and $\bar{o}_i^{[\ell_t,t]}$, then $m'' \leq m$. If $\Pr(o_i^{[m,t]}) \leq p_{\max}$, then also $\Pr(o_i^{[m'',t]}) \leq p_{\max}$, so $o_i^{[\ell_t,t]} \approx \bar{o}_i^{[\ell_t,t]}$. Otherwise, we have $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$. Hence, the cluster containing $o_i^{[\ell_t,t]}$ contains all suffixes with suffix $o_i^{[m,t]}$ by Lemma 4.8. Since $o_i^{[m,t]} = \bar{o}_i^{[m,t]}$, it then also contains $\bar{o}_i^{[m,t]}$, so $o_i^{[m,t]} \sim \bar{o}_i^{[m,t]}$ and hence $o_i^{[m,t]} \approx \bar{o}_i^{[m,t]}$. Hence, we conclude that transitivity holds and hence that $\approx$ is an equivalence relation. □

A.2.7 Clusters in approximate equivalence correspond to a suffix (Lemma 4.14)

Lemma 4.14. *Let c_i^t be a cluster in probability-based clustering. Then there exists a suffix $o_i^{[m,t]}$ such that c_i^t contains precisely the suffixes with suffix $o_i^{[m,t]}$.*

Proof. Consider the largest common suffix for each pair of suffixes $o_i^{[\ell_t,t]}, \tilde{o}_i^{[\ell_t,t]}$ in c_i^t and take a shortest $o_i^{[m,t]}$ among them. As in the proof of Lemma 4.8, it then follows that all suffixes in c_i^t have suffix $o_i^{[m,t]}$. Fix two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ in c_i^t with largest common suffix $o_i^{[m,t]}$. If $\Pr(o_i^{[m,t]}) > p_{\max}$, then we must have $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$. By Lemma 4.8, it then follows that c_i^t contains all clusters with suffix $o_i^{[m,t]}$. We now consider the case $\Pr(o_i^{[m,t]}) \leq p_{\max}$. Let any suffix $\hat{o}_i^{[\ell_t,t]}$ with suffix $o_i^{[m,t]}$ be given. Then the largest common suffix of $o_i^{[\ell_t,t]}$ and $\hat{o}_i^{[\ell_t,t]}$ is $o_i^{[m',t]}$ for some $m' \leq m$. Then we have $\Pr(o_i^{[m',t]}) \leq \Pr(o_i^{[m,t]}) \leq p_{\max}$, so $o_i^{[\ell_t,t]} \approx \hat{o}_i^{[\ell_t,t]}$, which shows that $\hat{o}_i^{[\ell_t,t]} \in c_i^t$. Hence, also in this case c_i^t contains precisely the suffixes with suffix $o_i^{[m,t]}$. □

A.3 Queue Pruning (Lemma 8.1)

Lemma 8.1. *Assume that $L \geq n|C_{i,t}|$ for all $0 \leq t \leq h-1$ and all $i \in \mathcal{D}$. Let $\boldsymbol{\varphi}$ be a partial policy and let $\boldsymbol{\varphi}'$ be a child of $\boldsymbol{\varphi}$ in the small-step search tree. Then $prog(\boldsymbol{\varphi}') \geq prog(\boldsymbol{\varphi}) + 1$.*

Proof. Assume that $\boldsymbol{\varphi}$ has specified an action for all clusters of length $\sigma(\boldsymbol{\varphi}) - 1$, for i out of n agents, and for c out of $|C_{i+1,\sigma(\boldsymbol{\varphi})}|$ clusters of agent $i+1$, and let p be the probability that the LOH of agent $i+1$ is in one of the first c clusters. Define $\sigma(\boldsymbol{\varphi}'), i', c'$, and p' similarly for $\boldsymbol{\varphi}'$.

If $c < |C_{i+1,\sigma(\boldsymbol{\varphi})}| - 1$, then $\boldsymbol{\varphi}'$ just specifies an action for an additional cluster for agent $i+1$, so $\sigma(\boldsymbol{\varphi}') = \sigma(\boldsymbol{\varphi})$, $i' = i$, $c' = c+1$ and $p' \geq p$. Then

$$\begin{aligned} prog(\boldsymbol{\varphi}') &= \sigma(\boldsymbol{\varphi}') \cdot L + i' \cdot \frac{L}{n} + c' + p' \left(\frac{L}{n} - \left| C_{i'+1,\sigma(\boldsymbol{\varphi}')} \right| \right) \\ &\geq \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + (c+1) + p \left(\frac{L}{n} - \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| \right) \\ &= prog(\boldsymbol{\varphi}) + 1. \end{aligned}$$

If $c = |C_{i+1,\sigma(\boldsymbol{\varphi})}| - 1$, then $\boldsymbol{\varphi}'$ specifies the last action for agent $i+1$. If $i+1 < n$, then $\sigma(\boldsymbol{\varphi}') = \sigma(\boldsymbol{\varphi})$, $i' = i+1$, $c' = 0$ and $p' = 0$. Then

$$\begin{aligned} prog(\boldsymbol{\varphi}') &= \sigma(\boldsymbol{\varphi}') \cdot L + i' \cdot \frac{L}{n} = \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + \frac{L}{n} \\ &= \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| + \left(\frac{L}{n} - \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| \right) \\ &\geq \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + (c+1) + p \left(\frac{L}{n} - \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| \right) \\ &= prog(\boldsymbol{\varphi}) + 1. \end{aligned}$$

Finally, if $c = |C_{i+1,\sigma(\boldsymbol{\varphi})}| - 1$ and $i+1 = n$, then $\sigma(\boldsymbol{\varphi}') = \sigma(\boldsymbol{\varphi}) + 1$, $i' = 0$, $c' = 0$ and $p' = 0$. Then

$$\begin{aligned} prog(\boldsymbol{\varphi}') &= \sigma(\boldsymbol{\varphi}') \cdot L = \sigma(\boldsymbol{\varphi}) \cdot L + (n-1) \cdot \frac{L}{n} + \frac{L}{n} \\ &= \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| + \left(\frac{L}{n} - \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| \right) \\ &\geq \sigma(\boldsymbol{\varphi}) \cdot L + i \cdot \frac{L}{n} + (c+1) + p \left(\frac{L}{n} - \left| C_{i+1,\sigma(\boldsymbol{\varphi})} \right| \right) \\ &= prog(\boldsymbol{\varphi}) + 1. \end{aligned}$$

In each case, we use that $L \geq n|C_{i,t}|$ when using that the weight $\left(\frac{L}{n} - \left|C_{i+1,\sigma(\boldsymbol{\varphi})}\right|\right)$ for the probability is nonnegative. We conclude that $prog(\boldsymbol{\varphi}') \geq prog(\boldsymbol{\varphi}) + 1$. □

B Further Empirical Results

In this appendix, we present further tables with empirical results for PF-MAA* and TR-MAA*.

	1,$Q_{\mathsf{MDP},1}$,20		2,$Q_{\mathsf{MDP},1}$,100		3,$Q_{\mathsf{MDP},1}$,100		2,$Q_{\mathsf{MDP},2}$,10^3		3,$Q_{\mathsf{MDP},2}$,10^4		3,$Q_{\mathsf{maxr},3}$,10^4	
h	value	time	value	time	value	time	value	time	value	time	value	time
DecTiger												
6	−10.68	<1	10.38	1	10.38	1	10.38	2	10.38	5	10.38	1
7	−11.55	<1	8.38	1	8.37	1	8.38	3	7.45	8	9.99	8
8	−13.55	<1	6.38	1	6.58	1	7.25	3	5.05	10	8.36	15
9	−16.68	<1	15.57	1	10.37	1	15.57	4	9.16	11	15.57	11
10	−18.68	<1	13.57	1	11.66	1	13.57	4	8.92	13	15.18	16
12	−22.68	<1	20.76	1	8.49	1	20.76	5	7.58	15	20.76	17
15	−28.68	<1	25.95	1	7.14	2	25.95	5	6.74	19	25.95	22
20	−38.68	1	27.14	2	−0.09	2	28.01	6	7.02	26	29.12	30
30	−58.68	1	51.91	2	−18.34	2	51.91	6	3.20	39	51.91	32
40	−78.68	1	65.48	2	−32.23	3	65.48	8	2.60	52	67.09	41
50	−98.68	1	79.05	2	−47.97	3	79.92	8	8.86	65	81.03	51
100	−198.68	1	169.30	2	−131.09	4	169.30	13	15.43	123	170.91	83
Grid												
4	2.176	<1	2.232	1	2.225	1	2.241	2	2.242	2	2.242	3
5	2.916	<1	2.915	1	2.854	1	2.969	4	2.970	4	2.970	6
6	3.616	1	3.664	1	3.623	2	3.715	13	3.717	12	3.717	23
7	4.319	1	4.414	2	4.352	2	4.471	22	4.474	29	4.474	61
8	5.023	1	5.168	2	5.140	2	5.248	35	5.229	45	5.234	109
9	5.727	1	5.925	2	5.898	2	6.032	49	6.032	73	5.999	218
10	6.431	1	6.684	2	6.661	3	6.817	67	6.821	97	6.764	286
12	7.840	1	8.203	3	8.195	3	8.391	99	8.414	146	8.302	444
15	9.953	1	10.485	3	10.507	4	10.754	147	10.810	268	MO	488
20	13.475	2	14.288	4	14.357	4	14.693	234	14.808	435	MO	488
50	34.606	3	37.105	7	37.461	8	MO	445	MO	483	MO	519
100	69.825	4	75.133	11	75.966	14	MO	435	MO	469	MO	515
BoxPushing												
4	98.17	<1	98.59	<1	98.59	<1	98.59	1	98.59	2	98.59	2
5	107.65	<1	107.73	<1	107.73	<1	107.71	2	107.73	3	107.69	5
6	114.40	<1	115.90	1	115.63	1	119.74	3	120.00	6	120.68	10
7	150.09	<1	155.25	1	155.99	1	155.35	4	155.99	5	155.91	12
8	186.03	<1	187.09	1	191.20	1	187.09	4	191.20	8	191.23	14
9	203.06	1	203.87	1	210.26	1	203.87	4	210.26	8	210.26	15
10	218.32	1	219.12	1	221.12	1	219.12	4	223.70	9	224.28	16
12	265.11	1	266.77	1	282.70	1	270.59	6	283.79	12	284.15	17
15	326.87	1	325.52	1	342.99	2	324.88	5	346.95	16	349.18	19
20	421.35	1	423.19	2	466.34	2	420.94	6	471.64	22	474.97	23
30	625.60	2	612.03	2	718.76	3	619.01	7	720.68	66	721.28	31
40	844.82	2	811.27	3	963.02	4	815.93	8	964.54	99	965.09	39
50	991.02	2	977.79	3	1207.67	4	1012.03	9	1209.26	161	1209.80	47
100	1905.81	3	1892.52	4	2431.40	6	1993.41	14	MO	344	2433.51	87
FireFight ($n_h = 3$)												
4	−6.626	<1	−6.672	1	−6.586	1	−6.626	2	−6.579	2	−6.579	2
5	−7.096	<1	−7.294	1	−7.148	2	−7.095	5	−7.073	5	−7.073	7
6	−7.379	<1	−7.661	1	−7.673	2	−7.196	3	−7.176	3	−7.176	24
10	−7.196	<1	−7.196	1	−7.255	1	−7.196	2	−7.176	1	−7.176	24
100	−7.196	1	−7.196	1	−7.256	2	−7.196	2	−7.176	2	−7.176	24
2000	−7.196	5	−7.196	5	−7.256	6	−7.196	7	−7.176	7	−7.176	25

Table 8: Results for each configuration of PF-MAA* listed separately. MO and TO denote memout (>16GB) and timeout (>600s).

	1,$Q_{\mathsf{MDP},1}$,20		2,$Q_{\mathsf{MDP},1}$,100		3,$Q_{\mathsf{MDP},1}$,100		2,$Q_{\mathsf{MDP},2}$,10^3		3,$Q_{\mathsf{MDP},2}$,10^4		3,$Q_{\mathsf{maxr},3}$,10^4	
h	value	time	value	time	value	time	value	time	value	time	value	time
GRID3X3												
5	0.90	1	0.90	1	0.90	1	0.90	2	0.90	2	0.90	7
6	1.49	1	1.49	1	1.49	1	1.49	4	1.49	5	1.49	13
7	2.17	1	2.17	1	2.17	1	2.19	8	2.19	11	2.18	16
8	2.93	1	2.96	2	2.96	2	2.97	12	2.97	18	2.92	24
9	3.75	1	3.80	2	3.80	2	3.81	17	3.81	31	3.70	29
10	4.60	1	4.67	2	4.67	2	4.68	23	4.68	37	4.50	39
12	6.50	2	6.50	2	6.50	2	6.53	32	6.53	61	6.13	51
15	9.40	2	9.39	3	9.39	3	9.42	50	9.42	90	8.61	77
20	14.34	2	14.33	3	14.33	4	14.37	78	14.37	150	12.76	118
30	24.32	2	24.32	5	24.32	5	24.35	131	24.35	313	21.07	192
40	34.32	3	34.32	6	34.32	7	34.35	195	34.35	499	29.39	255
50	44.32	3	44.32	7	44.32	9	44.35	233	TO	600	37.70	344
100	94.32	4	94.32	12	94.32	15	94.35	354	TO	600	79.26	522
MARS												
6	16.98	1	16.99	1	16.99	1	18.62	3	18.62	4	18.62	5
7	20.60	1	20.60	1	20.60	1	20.90	3	20.90	4	20.90	10
8	20.57	1	20.65	1	20.41	1	22.48	4	22.48	7	21.13	13
9	23.92	1	24.02	2	24.02	2	24.31	6	24.32	8	20.99	14
10	23.58	1	24.44	2	24.44	2	26.31	7	26.31	20	20.80	14
12	30.94	1	31.01	2	31.00	2	32.09	7	32.05	25	20.40	15
15	33.66	1	34.23	2	-	3	37.72	9	37.47	47	19.80	18
20	49.08	1	44.87	3	-	3	49.37	17	49.15	79	18.80	23
30	74.54	2	75.15	4	-	2	77.47	28	77.09	197	16.80	37
40	95.39	2	96.06	5	-	3	100.02	34	94.21	255	31.40	93
50	116.35	2	117.03	6	-	2	122.56	46	117.33	293	29.40	299
100	221.14	3	221.75	11	-	3	234.07	95	MO	396	TO	600
HOTEL												
100	502.2	1	502.2	1	502.2	1	502.2	5	502.2	34	502.2	30
500	2502.2	2	2502.2	4	2502.2	4	2502.2	36	MO	39	2502.2	209
1000	5002.2	3	5002.2	9	5002.2	9	MO	34	MO	35	5002.2	569
2000	10002.2	7	10002.2	28	10002.2	28	MO	35	MO	35	TO	600
RECYCLING												
100	308.79	1	308.79	2	308.79	2	308.79	9	308.79	52	308.79	44
500	1539.56	2	1539.56	4	1539.56	4	1539.56	30	1539.56	302	1539.56	231
1000	3078.02	2	3078.02	6	3078.02	6	3078.02	70	TO	600	TO	600
2000	6154.94	4	6154.94	14	6154.94	14	6154.94	211	TO	600	TO	600
BROADCAST												
100	90.74	1	90.56	1	90.56	1	90.76	4	90.75	29	90.76	25
500	452.69	1	450.56	3	450.56	3	452.72	19	452.73	198	452.05	156
1000	905.09	2	900.56	5	900.56	5	905.18	47	MO	369	903.17	422
2000	1809.87	4	1800.56	14	1800.56	14	1810.07	134	MO	374	TO	600
FIREFIGHT ($n_h = 4$)												
4	−14.127	1	−14.121	3	−14.107	3	−14.121	11	−14.107	11	−14.107	18
5	−17.081	1	−17.132	4	−17.085	5	−16.878	73	−16.848	67	TO	600
6	−19.495	1	−19.708	5	−19.692	5	−19.092	316	−19.092	310	TO	600
10	−22.995	1	−23.001	4	−22.929	4	−22.995	403	TO	600	TO	600
12	−23.980	2	−23.847	5	−23.974	6	−23.831	440	TO	600	TO	600
15	−24.419	2	−24.424	7	−25.097	11	−24.392	466	TO	600	TO	600
20	−24.818	3	−24.891	10	−26.422	17	TO	600	TO	600	TO	600
30	−25.053	5	−25.144	19	−27.490	32	TO	600	TO	600	TO	600
40	−25.087	6	−25.212	27	−28.060	46	TO	600	TO	600	TO	600
50	−25.092	8	−25.222	33	−28.284	59	TO	600	TO	600	TO	600

Table 9: Results for each configuration of PF-MAA* listed separately. MO and TO denote memout (>16GB) and timeout (>600s). For MARS, with $k = 3$ and $L = 100$ there are more clusters than iterations for $h \geq 15$, and hence the solver does not provide a result.

	FB-HSVI	GA-FSC	2,Q_{MDP},10^3		3,Q_{MDP},10^4		PF-MAA* (f)		PF-MAA*		random	upper
h	value	value	value	time	value	time	value	time	value	time	value	value
DecTiger												
6	10.38	10.38	10.38	1	10.38	4	10.38	1	10.38	8	−277.33	10.38
7	9.99		8.38	1	8.38	4	8.38	2	9.99	18	−323.56	9.99
8	12.22	8.00	7.25	2	6.38	5	6.58	2	8.36	29	−369.78	12.22
9	15.57		15.57	2	15.57	6	15.57	2	15.57	26	−416.00	15.57
10	15.18	13.57	13.57	2	13.57	7	13.57	2	15.18	33	−462.22	15.18
12			20.76	2	20.76	8	20.76	3	20.76	38	−554.67	20.76
15			25.95	2	25.95	9	25.95	3	25.95	46	−693.33	25.95
20	28.75		28.01	2	27.14	9	27.14	4	29.12	61	−924.44	38.28
30	51.90		51.91	2	51.91	12	51.91	5	51.91	77	−1386.67	59.47
40	67.09		65.48	3	65.48	15	65.48	5	67.09	100	−1848.89	80.35
50	80.66		79.92	3	79.05	19	79.05	6	81.03	124	−2311.11	101.32
100	170.90	169.30	169.30	4	169.30	33	169.30	8	170.91	220	−4622.22	206.44
Grid												
4	2.24	2.24	2.240	1	2.242	2	2.232	2	2.242	6	0.684	2.242
5	2.97	2.97	2.960	1	2.957	3	2.916	3	2.970	15	0.928	2.970
6	3.71	3.72	3.696	1	3.689	4	3.664	4	3.717	48	1.176	3.717
7	4.47		4.441	2	4.429	5	4.414	4	4.474	111	1.425	4.474
8	5.23		5.190	2	5.170	5	5.168	5	5.248	189	1.675	5.328
9	6.03		5.942	2	5.913	6	5.925	6	6.032	340	1.924	6.237
10			6.695	2	6.657	6	6.684	6	6.821	449	2.174	7.174
12			8.202	2	8.146	6	8.203	7	8.414	689	2.674	9.092
15			10.464	3	10.382	8	10.507	8	10.810	902	3.424	12.045
20			14.234	3	14.108	10	14.357	10	14.808	1158	4.674	17.031
50		40.49	36.858	5	36.463	21	37.461	18	MO	1447	12.174	47.031
100			74.564	7	73.721	43	75.966	29	MO	1419	24.674	97.031
BoxPushing												
4	98.59	98.17	98.59	1	98.59	2	98.59	1	98.59	4	−1.69	98.59
5	107.72		107.73	1	107.73	2	107.73	1	107.73	9	−2.67	107.74
6	120.67		119.11	1	119.78	3	115.90	2	120.68	19	−3.73	120.69
7	156.42		155.74	1	155.99	3	155.99	2	155.99	21	−4.84	156.71
8	191.22	189.05	186.86	1	191.07	5	191.20	2	191.23	26	−5.97	191.29
9	210.27		203.87	1	208.48	5	210.26	3	210.26	26	−7.12	210.35
10	223.74	216.98	218.33	1	220.35	4	221.12	3	224.28	29	−8.30	224.51
12			260.64	2	270.01	7	282.70	3	284.15	36	−10.67	284.74
15			310.47	2	316.76	11	342.99	4	349.18	40	−14.31	350.12
20	458.10	468.15	402.46	2	393.19	16	466.34	5	474.97	51	−20.46	476.21
30	636.28		579.44	2	497.20	38	718.76	7	721.28	104	−32.91	722.66
40	774.14		766.74	3	560.54	49	963.02	9	965.09	146	−45.42	966.85
50	1134.70	1201.05	949.75	3	599.58	68	1207.67	9	1209.80	217	−57.94	1211.96
100		2420.26	1864.49	5	651.03	112	2431.40	14	2433.51	445	−120.55	2437.45
FireFight ($n_h = 3$)												
4		−6.58	−6.702	1	−6.579	2	−6.586	2	−6.579	6	−9.026	−6.579
5		−7.07	−7.095	1	−7.073	3	−7.096	3	−7.073	17	−10.849	−7.070
6		−7.18	−7.458	1	−7.176	3	−7.379	4	−7.176	29	−12.556	−7.176
10			−7.612	1	−7.176	3	−7.196	2	−7.176	27	−18.413	−7.176
100			−7.612	1	−7.176	3	−7.196	4	−7.176	28	−40.028	−7.176
2000			−7.612	5	−7.176	6	−7.196	15	−7.176	38	−40.130	−7.176

Table 10: Lower bounds, i.e. the value of the policies obtained with different policy-finding algorithms. MO denotes memout (>16GB), TO denotes timeout (>600s).

	FB-HSVI	GA-FSC	2,Q_{MDP},10^3		3,Q_{MDP},10^4		PF-MAA* (f)		PF-MAA*		random	upper
h	value	value	value	time	value	time	value	time	value	time	value	value
GRID3X3												
5	0.89		0.90	1	0.90	1	0.90	3	0.90	12	0.02	0.90
6	1.49		1.49	1	1.49	4	1.49	3	1.49	21	0.03	1.49
7	2.19		2.19	2	2.19	10	2.17	4	2.19	35	0.05	2.19
8	2.96		2.97	3	2.97	15	2.96	5	2.97	54	0.07	2.97
9	3.80		3.80	4	3.80	19	3.80	4	3.81	77	0.09	3.81
10	4.68		4.67	5	4.68	25	4.67	5	4.68	99	0.12	4.69
12			6.51	5	6.51	33	6.50	6	6.53	144	0.16	6.55
15			9.40	7	9.41	56	9.40	7	9.42	217	0.24	9.44
20	14.35		14.35	9	14.35	87	14.34	9	14.37	347	0.36	14.39
30	24.33		24.33	15	24.34	194	24.32	12	24.35	636	0.61	24.38
40	34.33		34.33	19	34.33	315	34.32	15	34.35	950	0.85	34.38
50	44.32		44.33	26	44.33	448	44.32	19	44.35	1176	1.10	44.38
100	94.24		94.33	60	MO	530	94.32	31	94.35	1475	2.34	94.38
MARS												
6	18.62		16.99	2	17.84	7	16.99	3	18.62	12	−8.58	18.62
7	20.90		20.71	2	20.71	7	20.60	3	20.90	18	−9.86	20.90
8	22.47		21.59	2	21.59	9	20.65	4	22.48	24	−11.11	22.48
9	24.31		24.11	3	24.13	11	24.02	4	24.32	28	−12.34	24.32
10	26.31		24.63	3	24.63	15	24.44	4	26.31	41	−13.56	26.32
12			31.12	3	31.12	16	31.01	5	32.09	47	−15.98	33.64
15			34.33	4	35.25	21	34.23	6	37.72	74	−19.59	39.48
20	52.13		45.30	6	46.09	40	49.08	7	49.37	119	−25.63	52.71
30	78.09		74.95	7	74.62	94	75.15	7	77.47	263	−37.76	79.31
40	103.52		95.76	9	93.72	266	96.06	10	100.02	382	−49.89	105.86
50	128.95		116.73	17	111.22	440	117.03	10	122.56	638	−62.02	132.41
100	249.92		221.70	30	MO	401	221.75	17	234.07	1091	−122.67	264.85
HOTEL												
100			502.2	3	502.2	26	502.2	3	502.2	70	−184.6	502.2
500			2502.2	25	MO	36	2502.2	9	2502.2	285	−1095.7	2502.2
1000			MO	31	MO	30	5002.2	21	5002.2	638	−2234.6	5002.2
2000			MO	28	MO	28	10002.2	62	MO	670	−4512.4	10002.2
RECYCLING												
100	308.78	308.79	308.79	4	308.79	38	308.79	5	308.79	106	47.36	308.79
500			1539.56	19	1539.56	232	1539.56	9	1539.56	562	229.32	1539.56
1000			3078.02	52	TO	600	3078.02	15	3078.02	1270	456.77	3078.02
2000			6154.94	174	TO	600	6154.94	32	6154.94	1411	911.68	6154.94
BROADCAST												
100	90.76	90.29	90.76	2	90.75	27	90.74	4	90.76	57	28.62	90.76
500		450.29	452.72	16	452.73	191	452.69	7	452.73	373	141.53	452.74
1000	903.28	900.29	905.16	42	MO	335	905.09	12	905.18	838	282.68	905.20
2000			1810.02	124	MO	338	1809.87	31	1810.07	1108	564.98	1810.13
FIREFIGHT ($n_h = 4$)												
4			−14.121	1	−14.107	3	−14.107	7	−14.107	40	−16.424	−14.107
5			−17.091	2	−17.049	7	−17.081	11	−16.848	740	−20.580	−16.835
6			−19.550	2	−19.599	9	−19.495	12	−19.092	1226	−24.746	−19.039
10			−23.932	3	−24.455	17	−22.929	9	−22.995	1603	−41.422	−22.807
12			−24.760	4	−25.913	27	−23.847	13	−23.831	1640	−49.741	−23.392
15			−25.723	4	−27.349	41	−24.419	20	−24.392	1666	−62.172	−23.589
20			−26.969	5	−28.587	60	−24.818	30	TO	1800	−82.746	−23.639
30			−27.954	7	−30.645	97	−25.053	56	TO	1800	−123.321	−23.639
40			−28.264	9	−31.442	137	−25.087	79	TO	1800	−163.135	−23.639
50			−28.349	10	−31.782	180	−25.092	100	TO	1800	−202.201	−23.639

Table 11: Lower bounds, i.e. the value of the policies obtained with different policy-finding algorithms. MO denotes memout (>16GB), TO denotes timeout (>600s).

	FB-HSVI	RS-MAA* (f)		RS-MAA*		TR-MAA* (f)		TR-MAA*		lower	MDP
h	value	value	time	value	time	value	time	value	time	value	value
DecTiger											
6	10.39	10.38	<1	10.38	<1	10.38	2	10.38	1	10.38	120.00
7	10.00	9.99	2	9.99	2	13.39	120	9.99	2	9.99	140.00
8	12.23	12.22	2	12.22	3	21.24	120	12.22	23	12.22	160.00
9	15.58	15.57	8	15.57	9	27.42	120	16.32	818	15.57	180.00
10	15.19	15.30	120	15.18	594	34.32	120	20.23	914	15.18	200.00
12		20.76	120	20.76	772	48.44	120	27.87	698	20.76	240.00
15		27.49	120	25.95	1800	69.62	120	39.03	974	25.95	300.00
20		38.28	120	38.28	262	104.90	120	57.02	272	29.12	400.00
30		59.47	120	59.47	253	175.49	120	91.55	550	51.91	600.00
40		80.35	120	80.35	243	246.08	120	127.00	280	67.09	800.00
50		101.32	120	101.32	261	316.66	120	161.55	552	81.03	1000.00
100		206.44	120	206.44	260	669.55	120	337.00	266	170.91	2000.00
Grid											
4	2.25	2.242	1	2.242	1	2.242	2	2.242	2	2.242	2.865
5	2.98	2.970	5	2.970	5	2.970	4	2.970	13	2.970	3.834
6	3.72	3.717	72	3.717	82	3.717	12	3.717	52	3.720	4.820
7	4.48	TO	120	MO	455	4.549	120	4.474	241	4.474	5.814
8	5.24	TO	120	MO	492	5.458	120	5.328	681	5.248	6.811
9	6.04	TO	120	MO	499	6.393	120	6.237	775	6.032	7.809
10		TO	120	MO	513	7.349	120	7.174	714	6.821	8.809
12		TO	120	MO	512	9.300	120	9.092	788	8.414	10.808
15		TO	120	MO	451	12.269	120	12.045	864	10.810	13.808
20		TO	120	MO	523	17.258	120	17.031	941	14.808	18.808
50		TO	120	MO	490	47.257	120	47.272	1415	40.490	48.808
100		TO	120	MO	440	97.257	120	MO	1196	75.966	98.808
BoxPushing											
4	98.60	98.59	11	98.59	12	98.59	12	98.59	11	98.59	106.43
5	107.73	107.74	120	107.74	1800	107.77	120	107.74	1800	107.73	118.77
6	120.68	120.73	120	120.69	921	120.78	120	120.69	916	120.68	131.19
7	156.43	156.73	120	156.71	480	156.82	120	156.71	460	156.42	163.94
8	191.23	191.29	120	191.29	412	191.31	120	191.31	198	191.23	203.42
9	210.28	210.35	120	210.35	549	210.61	120	210.41	576	210.27	228.75
10	223.75	TO	120	224.51	872	226.79	120	224.62	265	224.28	244.85
12		TO	120	MO	606	285.58	120	284.74	240	284.15	302.73
15		TO	120	MO	605	353.50	120	350.12	624	349.18	377.12
20		TO	120	MO	624	479.12	120	476.21	259	474.97	511.13
30		TO	120	MO	644	725.74	120	722.66	271	721.28	777.61
40		TO	120	MO	632	971.26	120	966.85	314	965.09	1042.03
50		TO	120	MO	645	1217.23	120	1211.96	362	1209.80	1306.24
100		TO	120	MO	637	2446.86	120	2437.45	645	2433.51	2628.14
FireFight ($n_h = 3$)											
4		−6.579	1	−6.579	1	−6.579	3	−6.579	2	−6.579	−4.282
5		−7.070	9	−7.070	10	−7.070	7	−7.070	58	−7.070	−4.342
6		−7.176	63	−7.176	70	−7.176	3	−7.176	97	−7.176	−4.355
10		TO	120	MO	614	−7.176	4	−7.176	192	−7.176	−4.363
100		TO	120	MO	619	−7.176	17	MO	1798	−7.176	−4.363
2000		TO	120	MO	745	TO	120	MO	1460	−7.176	−4.363

Table 12: Upper bounds on the value of optimal policies, obtained with algorithms that find such bounds. TO and MO denote timeout (>120s for the fast configuration) and memout (>16GB).

	FB-HSVI	RS-MAA* (f)		RS-MAA*		TR-MAA* (f)		TR-MAA*		lower	MDP
h	value	value	time	value	time	value	time	value	time	value	value
GRID3X3											
5	0.90	0.90	1	0.90	1	0.90	3	0.90	3	0.90	0.94
6	1.50	1.49	9	1.49	8	1.49	10	1.49	12	1.49	1.58
7	2.20	2.19	120	2.19	209	2.19	120	2.19	205	2.19	2.31
8	2.97	2.97	120	2.97	469	2.97	120	2.97	490	2.97	3.12
9	3.81	3.82	120	3.81	437	3.81	120	3.81	508	3.81	3.98
10	4.69	4.70	120	4.70	486	4.70	120	4.69	515	4.68	4.88
12		6.55	120	6.55	489	6.55	120	6.55	517	6.53	6.76
15		9.45	120	9.45	484	9.45	120	9.44	601	9.42	9.67
20	14.36	14.40	120	14.39	574	14.40	120	14.39	714	14.37	14.63
30	24.34	24.39	120	24.38	797	24.39	120	24.38	943	24.35	24.62
40		34.39	120	34.38	662	TO	120	34.38	1197	34.35	34.62
50		44.39	120	44.38	733	TO	120	44.38	1482	44.35	44.62
100		94.39	120	94.38	1115	TO	120	TO	1800	94.35	94.62
MARS											
6	18.63	18.62	3	18.62	3	18.62	4	18.62	5	18.62	20.07
7	20.91	20.90	5	20.90	5	20.90	6	20.90	22	20.90	21.17
8	22.48	22.48	33	22.48	34	22.48	23	22.48	33	22.48	23.65
9	24.32	24.32	120	24.32	72	24.32	8	24.32	44	24.32	25.70
10	26.32	26.32	120	26.32	1800	26.36	120	26.32	344	26.31	28.61
12		33.85	120	33.64	1800	33.96	120	33.85	807	32.09	36.11
15		39.56	120	39.48	365	39.63	120	39.49	1529	37.72	43.02
20		52.86	120	52.71	748	52.94	120	52.77	687	52.13	57.52
30		79.50	120	79.50	195	79.62	120	79.31	720	78.09	86.65
40		106.16	120	106.16	212	106.32	120	105.86	683	103.52	115.85
50		132.82	120	132.82	249	133.04	120	132.41	742	128.95	144.97
100		287.46	120	265.73	374	266.28	120	264.85	1279	249.92	288.97
HOTEL											
100		502.2	3	502.2	3	502.2	3	502.2	8	502.2	502.8
500		2502.2	12	2502.2	13	2502.2	9	2502.2	31	2502.2	2502.8
1000		5002.2	25	5002.2	27	5002.2	16	5002.2	63	5002.2	5002.8
2000		10002.2	53	10002.2	58	10002.2	38	10002.2	129	10002.2	10002.8
RECYCLING											
100	308.79	308.79	1	308.79	1	309.43	120	308.79	6	308.79	328.37
500		1539.56	23	1539.56	25	1549.81	120	1541.49	278	1539.56	1637.46
1000		3078.02	47	3078.02	52	3100.25	120	3083.30	289	3078.02	3273.83
2000		6155.16	120	6154.94	324	6201.15	120	6166.91	283	6154.94	6546.55
BROADCAST											
100	90.77	90.76	2	90.76	2	90.83	120	90.76	94	90.76	95.56
500		452.74	27	452.74	28	457.46	120	456.89	101	452.73	476.61
1000		905.20	120	905.20	175	915.70	116	914.50	105	905.18	952.93
2000		TO	120	1810.13	319	1832.19	120	1829.73	106	1810.07	1905.56
FIREFIGHT ($n_h = 4$)											
4		−14.107	13	−14.107	14	−14.107	26	−14.107	15	−14.107	−9.943
5		TO	120	−16.835	599	−16.835	64	MO	1262	−16.848	−11.067
6		TO	120	MO	795	−19.039	118	MO	1321	−19.092	−11.759
10		TO	120	MO	763	−22.807	120	MO	1051	−22.929	−12.424
12		TO	120	MO	756	−23.392	120	MO	1288	−23.831	−12.451
15		TO	120	MO	728	−23.589	120	MO	1319	−24.392	−12.458
20		TO	120	MO	771	−23.639	120	MO	1224	−24.818	−12.459
30		TO	120	MO	763	−23.547	120	MO	1334	−25.053	−12.459
40		TO	120	MO	785	−23.482	120	MO	1079	−25.087	−12.459
50		TO	120	MO	778	−20.648	120	MO	1241	−25.092	−12.459

Table 13: Upper bounds on the value of optimal policies, obtained with algorithms that find such bounds. TO and MO denote timeout (>120s for the fast configuration) and memout (>16GB).

C Contribution Statement

We first list the key new ideas introduced in this thesis compared to the two conference papers [Koops *et al.*, 2023; Koops *et al.*, 2024] on which this thesis is (partially) based:

- The simpler definition of lossless clustering for sliding window memory, using a new insight that we proved in Lemma 4.9.
- The idea of regret computations introduced in Section 5.6.2.
- The faster precomputation of the Q_{BG} heuristic (Section 6.1).
- The result that $Q_{1,\infty}$ and Q_{POMDP} are equal (Theorem 7.3).
- The aperiodic terminal reward heuristics (Section 9.1.2).

Detailed overview. Finally, we give a detailed overview of the relation between this thesis and the two papers [Koops *et al.*, 2023; Koops *et al.*, 2024].

- Section 1 (the introduction) was written specifically for the thesis.
- Section 2 (on POMDPs) was written specifically for the thesis.
- Section 3 (on Dec-POMDPs): Section 3.1 and 3.7 incorporate material from Section 2 of Koops *et al.* [2023] and Section 2 of Koops *et al.* [2024]. The remainder of this section was written specifically for the thesis.
- Section 4 (on clustering) is based on Section 3 and Appendix A of Koops *et al.* [2024].
  - Section 4.1: the definitions and some surrounding text are from Section 3 of Koops *et al.* [2024]; the examples and the paragraph on storing incremental clusterings are new.
  - Section 4.2 is a revised version of material from Appendix A of Koops *et al.* [2024].
  - Section 4.3 is written specifically for this thesis, but incorporates some material based on Appendix A of Koops *et al.* [2024].
  - Section 4.4 is an expanded version of material from Section 3 of Koops *et al.* [2024].
  - Section 4.5 is a revised/expanded version of material from Section 3 and Appendix A of Koops *et al.* [2024]. Lemma 4.9 is new, which leads to simpler definition of lossless clustering of sliding window memory than presented in Koops *et al.* [2024].
  - Section 4.6 is an expanded version of material from Section 3 and Appendix A of Koops *et al.* [2024]. In particular, Lemma 4.16 is new.
- Section 5 (on MAA*) is primarily based on Section 3 of Koops *et al.* [2023] and Section 4 of Koops *et al.* [2024]. Section 5.1, 5.2, 5.3 and 5.4 are written specifically for this thesis, although Section 5.2 incorporates some material based on Section 3 of Koops *et al.* [2023]. Section 5.5 is a revised/combined version of Section 3 of Koops *et al.* [2023] and Section 4 of Koops *et al.* [2024]; the detailed comparison to GMAA*-ICE is new. The idea presented in Section 5.6.1 was first mentioned in Section 6 of Koops *et al.* [2023], but has been worked out in more detail here. The idea presented in Section 5.6.2 is new.
- Section 6 (on heuristics): Section 6.1, 6.2 and 6.4 are newly written for this thesis, and the faster precomputation of the Q_{BG} heuristic is a novel contribution in this thesis. The general framework introduced in Section 6.3 was first introduced in Section 6 of Koops *et al.* [2024]. However, the information function and the application to precomputed heuristics are new.

- Section 7 (on RS-MAA*) is based on Koops *et al.* [2023]. Specifically, Section 7.1 is based on Section 4 of Koops *et al.* [2023], but is rewritten to define heuristics in terms of the general framework introduced in Section 6.3. Section 7.2 is based on Section 5 and 6 of Koops *et al.* [2023], but the result that $Q_{1,\infty}$ and Q_{POMDP} are equal (Theorem 7.3) is new. Finally, Section 7.3 is a revised version of a paragraph from Section 6 of Koops *et al.* [2023].
- Section 8 (on PF-MAA*) is a revised version of Section 5 and Appendix C and D of Koops *et al.* [2024]. However, we provide a more thorough discussion of properties which make a heuristic useful for policy-finding in Section 8.2.
- Section 9 (on TR-MAA*) is based on Section 6 and Appendix D of Koops *et al.* [2024]. However, the terminal reward heuristics (Section 9.1.1) are formalized and explained in much more detail in the thesis. The aperiodic terminal reward heuristics (Section 9.1.2) are a novel contribution in this thesis.
- Section 10 (the empirical evaluation) is based on Section 7 of Koops *et al.* [2023] and Section 7 and Appendix E of Koops *et al.* [2024].
  - Section 10.1: The overview table and the description of the benchmarks are written specifically for this thesis, while the remainder is from Koops *et al.* [2023].
  - Section 10.2 is written specifically for this thesis, although some experiments for GMAA*-ICE we also done for Koops *et al.* [2023].
  - Section 10.3 based on Section 7 of Koops *et al.* [2023], but the experiments for the regret computations are new.
  - Section 10.4 is a revised version of Section 7 and Appendix E of Koops *et al.* [2024].
  - Section 10.5 is a revised version of Section 7 and Appendix E of Koops *et al.* [2024], but we present new results using the aperiodic terminal reward heuristics and adapt the discussion accordingly.
- Section 11 (on related work) is a revised/expanded version of the related work section of Koops *et al.* [2024], with some additions from the related work from Koops *et al.* [2023].
- Section 12 (the conclusion) was written specifically for this thesis.
- Appendix A (proofs): Appendix A.1, A.2.1, A.2.2 and A.2.7 are new, while Appendix A.2.3, A.2.4, A.2.5, A.2.6 and A.3 are from Appendix A and C of Koops *et al.* [2024].
- Appendix B (further empirical results) is similar to the tables provided in Appendix E of Koops *et al.* [2024], but contain new results for the aperiodic terminal reward heuristics.
- Appendix C was written specifically for this thesis.

Figures: Figure 1, 2 and 5 are based on a figure provided to me by dr. Thiago D. Simão. In addition, dr. Thiago D. Simão also provided Figure 10 (which was included in Koops *et al.* [2023]). The other figures have been produced by myself.