Approximate Dec-POMDP Solving Using Multi-Agent A*

Wietze Koops[1] , **Sebastian Junges**[1] and **Nils Jansen**[2,1]
[1]Radboud University, Nijmegen, The Netherlands
[2]Ruhr-University Bochum, Germany
{wietze.koops, sebastian.junges}@ru.nl, n.jansen@rub.de

Abstract

We present an A*-based algorithm to compute policies for finite-horizon Dec-POMDPs. Our goal is to sacrifice optimality in favor of scalability for larger horizons. The main ingredients of our approach are (1) using clustered sliding window memory, (2) pruning the A* search tree, and (3) using novel A* heuristics. Our experiments show competitive performance to the state-of-the-art. Moreover, for multiple benchmarks, we achieve superior performance. In addition, we provide an A* algorithm that finds upper bounds for the optimum, tailored towards problems with long horizons. The main ingredient is a new heuristic that periodically reveals the state, thereby limiting the number of reachable beliefs. Our experiments demonstrate the efficacy and scalability of the approach.

1 Introduction

Decentralized partially observable Markov decision processes (Dec-POMDPs) formalize multi-agent decision-making under stochastic dynamics and partial observability. They are, for instance, suitable to model bandwidth allocation [Hemmati *et al.*, 2015] and maintenance problems [Bhustali and Andriotis, 2023]. The decision problem underlying solving Dec-POMDPs exactly or ϵ-optimally is NEXP-hard [Bernstein *et al.*, 2002; Rabinovich *et al.*, 2003]. This paper explores a model-based approach for approximate solving of finite-horizon Dec-POMDPs. Concretely, our approach finds policies that obtain a high value on a given Dec-POMDP and bounds on the value achieved by an optimal policy. The proximity between the achieved values and the bounds shows that the policies empirically perform very well.

Small-step MAA*. A prominent line of work for solving Dec-POMDPs for finite horizons builds upon multi-agent A* (MAA*) [Szer *et al.*, 2005]. The crux of these algorithms is to search through the space of all joint policies in a search space where we incrementally fix the decisions of individual agents. To alleviate a doubly-exponential out-degree with growing horizons, *small-step* MAA* [Koops *et al.*, 2023] makes these decisions for every observation history sequentially, yielding very deep search trees that have a limited out-degree. However, so far, MAA* is used to compute exact solutions by iteratively refining the upper bound provided by an admissible heuristic. In this paper, we show that MAA* provides a competitive foundation for algorithms that find good but not necessarily optimal policies. Likewise, MAA* is also a solid foundation for computing non-trivial (and potentially tight) upper bounds. The result is an algorithm that scales to much higher horizons than (exact) MAA*-based algorithms. Below, we briefly give our perspective on both lower and upper bounds before outlining the main *technical ingredients* (TIs).

Lower bounds by finding policies. To find good policies fast, we limit the space of policies by considering policies that are independent of old observations (TI1) and we limit the number of partial policies that we may explore at any level of the A* search tree (TI2). Since we already limit the number of policies expanded, it is not essential to use a tight heuristic for this. We show that MAA* can also find good policies fast, using heuristics that are generally not tight (TI3).

Proving upper bounds. When using MAA* with any admissible heuristic, the highest heuristic value is an upper bound for the optimal value. However, the only heuristic that scales to the horizons for which we find lower bounds is Q_{MDP}, which treats the Dec-POMDP as a fully-observable, centralized MDP. Other, tighter heuristics cannot handle the horizons we are interested in. In this paper, we introduce novel heuristics (TI4) which are less tight but more scalable than the heuristics previously used for solving Dec-POMDPs.

Technical ingredient 1: Clustering with sliding-window memory. A major challenge in finding good policies for large horizons is the fact that the optimal policy may be exponentially large. Lossless incremental *clustering* [Oliehoek *et al.*, 2009] helps to determine that an optimal policy may, w.l.o.g., take the same decision based on different observation histories. However, even with clustering, computing a policy generally requires determining an action for exponentially many different observation histories. In this work, we consider clustering for a specific subclass of policies. In particular, we consider sliding window policies that only depend on the most recent k observations, which empirically are often the most relevant observations. We develop a lossless clustering for sliding window memory, which clusters these windows with no additional loss of optimal policy value, compared to using sliding window memory.

Technical ingredient 2: Pruning the queue. A*-based algorithms explore partial policies in order of the heuristic value. Especially if the heuristic is not tight, this may lead to a breadth-first-like exploration of policies, effectively preventing exploring policies that are defined for longer horizons. We therefore prevent considering partial policies by pruning nodes in the search tree. We use a hard cap for every stage of the search tree while ensuring that we do eventually expand a complete (i.e., non-partial) policy.

Technical ingredient 3: Loose heuristics. In any A*-variant, policies are analysed in the order suggested by the heuristic. A good heuristic is thus one which leads early on to a policy with a high value. While tight heuristics do ensure this property, also loose heuristics can have a similar property. Empirically, it is often near-optimal to greedily optimize for the next few steps. Concretely, our heuristic considers only the next few time steps using a Dec-POMDP. The potential reward after these few steps is estimated using an MDP or even by assuming that the maximal reward is constantly achieved.

Technical ingredient 4: Scalable and tight heuristics for upper bounds. To prove upper bounds on the optimal value, we need tight admissible heuristics. Common heuristics relax the restrictions of a decentralized, partially observable setting and assume the problem to be centralized and/or fully observable. However, full state-observability yields a heuristic that is not sufficiently tight, and relaxing the setting to a (centralized) POMDP does not avoid expensive computations as it requires considering exponentially many beliefs in the horizon. Inspired by the idea of only sharing information after some delay, we present an admissible heuristic that *periodically* reveals state information. This yields a trade-off between the horizons over which one must reason about partial information and the tightness of the heuristic. The heuristic can be computed on all benchmarks we selected.

Contributions. To summarize, our technical advancements on clustering, heuristics, and pruning outlined above together yield a pair of algorithms that find good policies as well as upper bounds for Dec-POMDPs for horizons more than an order of magnitude larger than for which exact Dec-POMDP solving is possible. In particular, for the BOXPUSHING benchmark with horizons up to 100, we find policies with values that are only 1% smaller than our upper bounds.[1]

1.1 Related Work

This work builds on a series of works on A* algorithms for solving Dec-POMDPs, culminating in the exact algorithms GMAA*-ICE [Oliehoek *et al.*, 2013] and RS-MAA* [Koops *et al.*, 2023]. 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. Instead of only adding the k best children, our algorithm adds all children and prunes them when necessary (TI2). 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.

Related work in the general A* literature includes Cazenave [2010], which proposed an idea similar to small-step MAA* in a general setting. 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.

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. Another line of work is on algorithms that use dynamic programming (DP) [Hansen *et al.*, 2004; Seuken and Zilberstein, 2007; Carlin and Zilberstein, 2008; Kumar and Zilberstein, 2009; Dibangoye *et al.*, 2009; Amato *et al.*, 2009]. Although these DP algorithms are bottom-up rather than top-down, they also use pruning to limit the number of policies (per time step). In addition, they use techniques reminiscent of clustering, to avoid spending too much time on optimizing for unlikely observations.

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.

There is also significant work on model-free reinforcement learning (RL) algorithms [Kraemer and Banerjee, 2016; Bono *et al.*, 2018; Dibangoye and Buffet, 2018; Mao *et al.*, 2020]. 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.

2 Problem Statement

We briefly recap Dec-POMDPs [Oliehoek and Amato, 2016] following the notation of Koops *et al.* [2023]. In particular, $\Delta(X)$ denotes the set of distributions over a finite set X.

Definition 1 (Dec-POMDP)**.** *A* Dec-POMDP *is a tuple* $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{O}}, b, T, R, O \rangle$ *with a set* $\mathcal{D} = \{1, \dots, n\}$ *of* n agents*, a finite set* $\mathcal{S}$ *of* states*, a set* $\boldsymbol{\mathcal{A}} = \times_{i \in \mathcal{D}} \mathcal{A}_i$ *of* joint actions*, and a set* $\boldsymbol{\mathcal{O}} = \times_{i \in \mathcal{D}} \mathcal{O}_i$ *of* joint observations*, where* $\mathcal{A}_i$ *and* $\mathcal{O}_i$ *are finite sets of* local actions *and* local observations *of agent* i*. The* transition function $T \colon \mathcal{S} \times \boldsymbol{\mathcal{A}} \to \Delta(\mathcal{S})$ *defines the* transition probability $\Pr(s' \mid s, \mathbf{a})$*,* $b \in \Delta(\mathcal{S})$ *is the* initial belief*,* $R \colon \mathcal{S} \times \boldsymbol{\mathcal{A}} \to \mathbb{R}$ *is the* reward *function, and the* observation function $O \colon \boldsymbol{\mathcal{A}} \times \mathcal{S} \to \Delta(\boldsymbol{\mathcal{O}})$ *defines the* observation probability $\Pr(\mathbf{o} \mid \mathbf{a}, s')$.

A Dec-POMDP describes a system whose state changes stochastically at every stage. The initial state is s^0 with probability $b(s^0)$. At each stage t, each agent i takes a local action a_i^t, resulting in a joint action $\mathbf{a}^t = \langle a_1^t, \dots, a_n^t \rangle$ and a reward $r^t = R(s^t, \mathbf{a}^t)$. The next state is s^{t+1} with probability $\Pr(s^{t+1} \mid s^t, \mathbf{a}^t)$. Finally, a joint observation $\mathbf{o}^{t+1}$ is

[1] Supplementary material and source code are available at https://arxiv.org/abs/2405.05662 and https://zenodo.org/records/11160648.

drawn with probability $\Pr(\mathbf{o}^{t+1} \mid \mathbf{a}^t, s^{t+1})$, and each agent i receives their local observation o_i^{t+1}.

We write $o_i^{[v,t]} = o_i^v \dots o_i^t$ for the local observations of agent i between stage v and t. If the next local observation is $o \in \mathcal{O}_i$, then $o_i^{[v,t]} \cdot o$ denotes the local observations between stage v and $t+1$. We write $\tau_i = o_i^{[1,t]}$ for the *local observation history* (LOH) of agent i, and $\boldsymbol{\tau} = \mathbf{o}^1 \dots \mathbf{o}^t$ 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.

Agents can choose their action based on all their past observations, i.e. based on their LOH. A *local policy* for agent i maps LOHs for that agent to a local action, formally: $\pi_i \colon \mathcal{O}_i^{\leq h-1} \to \mathcal{A}_i$. A *joint policy* is a tuple of local policies $\boldsymbol{\pi} = \langle \pi_1, \cdots, \pi_n \rangle$, and Π denotes the set of all joint policies. We will often refer to a joint policy simply as a *policy*.

Given a policy $\boldsymbol{\pi}$, we define the value of executing this policy in the initial belief b over a horizon h as:

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

where s^t and $\mathbf{a}^t$ are the state and joint action at stage t. Finally, in all probabilities, we implicitly also condition on the past policy and the initial belief.

The goal of the agents is to maximize the expected reward. An optimal policy is a policy $\boldsymbol{\pi}^* \in \arg\max_{\boldsymbol{\pi}' \in \Pi} V_{\boldsymbol{\pi}'}(b,h)$. The aim of this paper is to find a policy $\boldsymbol{\pi}$ with a high value, as well as upper bounds for the optimal value.

Problem statement: Given a Dec-POMDP and a horizon h, find a policy $\boldsymbol{\pi}$ and an upper bound $U(b,h)$ s.t.

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

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

Sliding window memory. We search the space of *sliding k-window memory* policies that depend only on the last k observations. Windows that end at stage t start at stage $\ell_t = \max(t-k, 0) + 1$. A policy π_i has *sliding window memory*, if $o_i^{[\ell_t,t]} = \tilde{o}_i^{[\ell_t,t]}$ implies $\pi_i\big(o_i^{[1,t]}\big) = \pi_i\big(\tilde{o}_i^{[1,t]}\big)$.

3 Clustering

Abstractly, our algorithm searches over policies that map LOHs to actions. To limit the search space, we adopt clustering [Oliehoek *et al.*, 2009], i.e., the idea that policies assign the same action to LOHs that belong to the same cluster of LOHs. Firstly, we introduce formally sliding k-window memory. Then we introduce *clustered* sliding k-window memory, which aims to merge existing clusters further without inducing a loss in policy value.

Definition 2. *A* clustering *is a partition $C_{i,t}$ of $\mathcal{O}_i^t$ for each stage $0 \leq t \leq h-1$ and each agent $i \in \mathcal{D}$.*

Definition 3. *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 $o_i^{[\ell_t,t]}$.

Cluster policies. We now introduce *cluster policies*. Write $C_i = \bigcup_{t=0}^{h-1} C_{i,t}$ for the set of agent i's clusters. Let $C \colon \bigcup_{i \in \mathcal{D}} \bigcup_{t=0}^{h-1} \mathcal{O}_i^t \to \bigcup_{i \in \mathcal{D}} C_i$ be the map that assigns an LOH 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 $\pi_i(\tau_i) = \pi_i^C(C(\tau_i))$ for each LOH τ_i. We denote the set of all cluster policies $\boldsymbol{\pi}^C = \langle \pi_1^C, \dots, \pi_n^C \rangle$ by Π^C. The value of a cluster policy $\boldsymbol{\pi}^C$ is the value of the corresponding policy $\boldsymbol{\pi} = \langle \pi_1, \dots, \pi_n \rangle$, where each π_i is the local policy corresponding to π_i^C. Note that a best cluster policy corresponding to sliding k-window memory is not necessarily optimal.

3.1 Clustered Sliding Window Memory

We extend the so-called lossless clustering [Oliehoek *et al.*, 2009] to sliding window memory. We require our clustering to be *incremental*: if two LOHs are clustered, then their extensions by the same observation are also clustered together.

Definition 4. *Let $\equiv_{C_{i,t}}$ be the equivalence relation induced by the partition $C_{i,t}$. A clustering is* incremental *at stage t if*

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

A clustering is incremental if it is incremental at each stage.

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 $C_{i,t}$. We call C' finer than C if C is coarser than C'.

Definition 5. *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.

We call a clustering lossless, if it is lossless with respect to a trivial (i.e. no) clustering.

We define clustered sliding window memory recursively, stage by stage. We write $\mathbf{c}_{\neq i}^{t-1}$ for the tuple consisting of the cluster of each agent except agent i in stage $t-1$. We write $F(\mathbf{c}_{\neq i}^{t-1}, \mathbf{o}_{\neq i}^t)$ for the resulting tuple of clusters of the other agents that we get in stage t after applying sliding window memory (Def. 3). Formally, these are the clusters in the finest clustering which is incremental at stage $t-1$ and coarser than sliding window memory clustering.

Definition 6. *Two suffixes $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$ are* belief-equivalent*, written $o_i^{[\ell_t,t]} \sim_b \tilde{o}_i^{[\ell_t,t]}$, if for all $v \in \{\ell_t, \dots, t\}$,*

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

For $v = \ell_t$, this definition states that the joint belief over the states and the clusters of the other agents is the same for $o_i^{[\ell_t,t]}$ and $\tilde{o}_i^{[\ell_t,t]}$. Using a result of Hansen *et al.* [2004] yields:

Lemma 1. *If a clustering is incremental, coarser than sliding k-window memory and finer than belief-equivalence, it is lossless w.r.t. sliding k-window memory.*

Only demanding that these beliefs are equal for $v = \ell_t$ in Def. 6 does not give an incremental clustering (we give an example in the supplementary material). Instead, we need that the beliefs are equal 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. Def. 6 alone is *not* sufficient to establish incrementality of the clustering. However, when ensuring that each cluster contains precisely the LOHs with common suffix $o_i^{[m,t]}$, we can prove incrementality. We define these clusters using the following equivalence relation:

Definition 7. *Consider suffixes $o_i^{[\ell_t,t]}$, $\tilde{o}_i^{[\ell_t,t]}$ with largest common suffix $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$. They are* 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 $o_i^{[\ell_t,t]} \sim_b \hat{o}_i^{[\ell_t,t]}$.*

In this definition, we allow $m > t$, in which case the suffix is empty, and we cluster all LOHs of agent i together.

We prove that $\sim$ is indeed an equivalence relation in the supplementary material. We write $[o_i^{[\ell_t,t]}]$ for the equivalence class of $o_i^{[\ell_t,t]}$. With this in hand, we can define the clustering corresponding to clustered sliding window memory.

Definition 8. *The clustering corresponding to clustered sliding window memory with window size k consists of the sets defined by $C_{i,t} = \left\{ [o_i^{[\ell_t,t]}] \;\middle|\; o_i^{[\ell_t,t]} \in \mathcal{O}_i^{\min\{t,k\}} \right\}$.*

Under this notion of equivalence, identical extensions of equivalent clusters are equivalent, which implies:

Lemma 2. *Clustered sliding window memory is incremental.*

Lemma 1 and 2 together imply:

Theorem 1. *Clustered sliding window memory is lossless with respect to sliding window memory.*

Probability-based clustering. To further limit the number of clusters, we can also cluster suffixes to a smaller common suffix together if the probability corresponding to the smaller common suffix is still smaller than some threshold $p_{\max}$. Formally, we consider two suffixes $o_i^{[\ell_t,t]}$, $\tilde{o}_i^{[\ell_t,t]}$ with largest common suffix $o_i^{[m,t]} = \tilde{o}_i^{[m,t]}$ to be *approximately equivalent* if $o_i^{[\ell_t,t]} \sim \tilde{o}_i^{[\ell_t,t]}$ or $\Pr(o_i^{[m,t]}) \leq p_{\max}$, and define the clusters to be the equivalence classes of this equivalence relation.

4 Small-Step Multi-Agent A*

Our algorithm searches for a policy by incrementally fixing actions, for one clustered LOH at the time. Specifically, our algorithm uses small-step MAA* [Koops *et al.*, 2023], which in turn builds on MAA* [Szer *et al.*, 2005]. We present the algorithm explicitly using clusters. Small-step MAA* explores clusters in a fixed order: first by stage, then by agent, and then according to a given order of the clusters.

Definition 9. *Given a total order $\preceq_{i,t}$ on $C_{i,t}$ for $t < h$ and $i \in \mathcal{D}$, we define an order $\preceq$ on $\bigcup_{i \in \mathcal{D}} C_i$ by $c_i^t \preceq \tilde{c}_j^{t'}$ iff*

$$\big(t < t'\big) \text{ or } \big(t = t' \wedge i < j\big) \text{ or } \big(t = t' \wedge i = j \wedge c_i^t \preceq_{i,t} \tilde{c}_j^{t'}\big).$$

We call this order the expansion order.

In MAA* and its derivatives, we incrementally construct policies. The intermediate policies are called *partial*. In particular, a *local partial policy* is a partial function $\varphi_i \colon C_i \to \mathcal{A}_i$. A *partial policy* is a tuple $\boldsymbol{\varphi} = \langle \varphi_1, \dots, \varphi_n \rangle$ such that the local partial policies φ_i are defined on precisely the clusters of agent i among the first d clusters in the expansion order for some d. Let Φ be the set of all partial policies. 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:

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

where $\varphi_i(c_i) = \bot$ if $\varphi_i(c_i)$ is not defined. The *extensions of* $\boldsymbol{\varphi}$ are the fully specified cluster policies extending $\boldsymbol{\varphi}$, $E(\boldsymbol{\varphi}) = \left\{ \boldsymbol{\pi} \in \Pi^C \mid \boldsymbol{\varphi} <_E \boldsymbol{\pi}^C \right\}$. Using this, we can define the small-step search tree:

Definition 10. *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. A* expands nodes in a search tree guided by a heuristic $Q \colon \Phi \to \mathbb{R}$ [Russell and Norvig, 2020]. The A* algorithm keeps a priority queue of open nodes, and in each step, it expands the open node with the highest heuristic value by adding all children of that node to the queue. The algorithm terminates once a leaf is selected as node with the highest heuristic value. The algorithm finds a best clustered policy $\boldsymbol{\pi}^C$ if the heuristic is *admissible*, i.e. an upper bound:

$$Q(\boldsymbol{\varphi}) \;\geq\; \max_{\boldsymbol{\pi}^C \in E(\boldsymbol{\varphi})} V_{\boldsymbol{\pi}^C}(b, h),$$

and matches the value for fully specified policies $\boldsymbol{\pi}^C$, i.e. $Q(\boldsymbol{\pi}^C) = V_{\boldsymbol{\pi}^C}(b, h)$ for fully specified policies $\boldsymbol{\pi}^C$.

The algorithm is exact if in addition the clustering is lossless, i.e. $\max_{\boldsymbol{\pi} \in \Pi} V_{\boldsymbol{\pi}}(b, h) = \max_{\boldsymbol{\pi} \in \Pi^C} V_{\boldsymbol{\pi}}(b, h)$.

5 Policy-Finding Multi-Agent A*

In MAA*, to find the optimal policy, one must expand all nodes in the queue whose heuristic value exceeds the value of the optimal policy. As a result, heuristics should be as tight as possible to limit the number of nodes expanded.

With *policy-finding multi-agent A** (PF-MAA*), we aim to find good policies fast. PF-MAA* applies small-step MAA* with clustered sliding window memory. To ensure timely termination, we limit the number of policies expanded by pruning the priority queue. Furthermore, it is not essential that the A*-heuristics for policy finding are tight. In this context, a useful heuristic is a heuristic that overestimates the value of good policies more than the value of bad policies, thereby steering the algorithm towards the good policies. To allow covering a larger fragment of the search space within a given time limit, the heuristics should also be easy to compute.

Finding good policies using A*. In 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. To do this, we define a progress measure *prog*, and prune policies with a low progress. That is, let N denote the number of policies already expanded. We expand partial policy $\boldsymbol{\varphi}$ if $\mathit{prog}(\boldsymbol{\varphi}) \geq N$ and prune it otherwise.

Our progress measure *prog* satisfies two design goals. First, it limits the number of policies expanded up to *each* level in the search tree, to ensure that the queue always contains a policy which is not pruned. This implies that we find a fully specified policy. Second, it should prune the least promising partial policies. For this, note that admissible 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. Hence, *prog* should increase with the number of actions specified and the probability of the clusters for which an action is specified.

These design goals lead to the following progress measure. Consider 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 $i+1$. Let p be the probability that the LOH of agent $i+1$ is in one of the first c clusters. Then we define the *progress* of $\boldsymbol{\varphi}$ as

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

We assume that $L \geq n|C_{i,t}|$ for all $t < h$ and all $i \in \mathcal{D}$. In the supplementary material, we show that this progress measure indeed ensures that PF-MAA* finds a fully specified policy within $h \cdot L$ policy expansions, and give further intuition.

Maximum reward heuristic. We use the following simple *maximum reward heuristic* $Q_{\mathsf{maxr},r}(\boldsymbol{\varphi})$, where $r > \sigma(\boldsymbol{\varphi})$. 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. Although this heuristic is clearly not tight in general, it is still useful when finding lower bounds: search guided by this heuristic is essentially a local search, choosing policies yielding good reward over the next $r - \sigma(\boldsymbol{\varphi})$ stages.

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 MDP value for the remaining $h - r$ stages. Formally, if $Q_{\mathsf{MDP}}(s, h')$ is the optimal value of the corresponding MDP with initial state s and horizon h', then the terminal reward corresponding to a joint belief b computed from the joint observation history at stage r is

$$\textstyle\sum_{s \in \mathcal{S}} b(s) \cdot Q_{\mathsf{MDP}}(s, h - r). \tag{1}$$

We then take the weighted average over all joint beliefs to compute the terminal reward.

Horizon reduction. Heuristic values are computed recursively, similarly to the small-step MAA* implementation RS-MAA*. 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 above.

6 Terminal Reward Multi-Agent A*

Among the heuristics used so far in the literature, only the MDP heuristic [Littman *et al.*, 1995] can be computed effectively for large horizons. This heuristic reveals the state to the agents in each step, and thereby overapproximates the value of a belief node drastically. The POMDP heuristic [Szer *et al.*, 2005; Roth *et al.*, 2005] is tighter as it assumes that each agent receives the full joint observation, but the state information is not revealed. The recursive heuristics considered by Koops *et al.* [2023] reveal the joint observation only once during planning. Both heuristics are too expensive to compute for large horizons on a variety of benchmarks (as remarked by Oliehoek *et al.* [2013] for the POMDP heuristic).

We propose a computationally more tractable alternative that periodically reveals the state. This leads to a new family of admissible heuristics, which we call *terminal reward heuristics*, which are empirically tighter than the POMDP heuristic, but computationally cheaper. Applying small-step MAA* with lossless clustering and this heuristic yields *terminal reward multi-agent A** (TR-MAA*). With TR-MAA*, we aim to find a tight upper bound for the value of an optimal policy, which is also scalable.

Generalized policies. To define the heuristics, we introduce a more general type of policy. Formally, a *generalized local policy* $\pi_i \colon (\boldsymbol{\mathcal{O}} \times \mathcal{S})^{\leq h-1} \to \mathcal{A}_i$ maps histories of states and joint observations to a local action. A *generalized (joint) policy* is a tuple $\langle \pi_1, \ldots, \pi_n \rangle$ of generalized local policies[2]. Let Π_{gen} denote the set of all generalized policies. As before, a generalized policy $\boldsymbol{\pi}$ *extends* a partial policy $\boldsymbol{\varphi}$, denoted by $\boldsymbol{\varphi} <_E \boldsymbol{\pi}$, if π_i ignores the state information and agrees with φ_i on all OHs corresponding to an LOH for which φ_i specifies the action. Formally, $\boldsymbol{\varphi} <_E \boldsymbol{\pi}$ iff $\forall i \in \mathcal{D}. \forall (\boldsymbol{\tau}, \boldsymbol{s}) \in (\boldsymbol{\mathcal{O}} \times \mathcal{S})^{\leq h-1}. \, \varphi_i(\tau_i) \in \{\bot, \pi_i(\boldsymbol{\tau}, \boldsymbol{s})\}$. 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 $\boldsymbol{\varphi}$.

Terminal reward heuristic. We write the Dec-POMDP optimization problem over generalized policies as

$$\begin{aligned} &\max_{\boldsymbol{\pi} \in \Pi_{\mathsf{gen}}} V_{\boldsymbol{\pi}}(b, h) \\ &\quad \text{subject to} \quad \tau_i = \widetilde{\tau}_i \;\text{ implies }\; \pi_i(\boldsymbol{\tau}, \boldsymbol{s}) = \pi_i(\widetilde{\boldsymbol{\tau}}, \widetilde{\boldsymbol{s}}) \\ &\qquad \text{for all } (i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \in \mathcal{I}, \text{ with} \end{aligned} \tag{2}$$

$$\mathcal{I} = \bigcup_{v=0}^{h-1} \left\{(i, \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}}, \boldsymbol{s}, \widetilde{\boldsymbol{s}}) \mid i \in \mathcal{D},\ \boldsymbol{\tau}, \widetilde{\boldsymbol{\tau}} \in \boldsymbol{\mathcal{O}}^v,\ \boldsymbol{s}, \widetilde{\boldsymbol{s}} \in \mathcal{S}^v\right\}.$$

Using $\mathcal{I}$, we quantify over all agents and pairs of traces. The condition $\tau_i = \widetilde{\tau}_i$ selects the pairs that agent i cannot distinguish. On these pairs, the agent has to take the same action. To compute a heuristic $Q(\boldsymbol{\varphi})$, instead of maximizing over

[2]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.

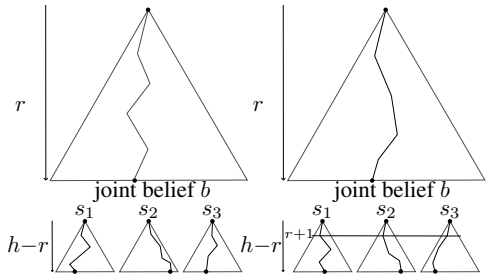

Figure 1: Revealing the state for a joint belief b at stage r over three states s_1, s_2, s_3. On the left the state is revealed at stage r, on the right it is revealed at stage $r+1$. In the latter case, the three policies are forced to take same action in stage 0.

Π_{gen} in Eq. (2), we maximize over the generalized extensions $E_{\mathsf{gen}}(\boldsymbol{\varphi})$ of $\boldsymbol{\varphi}$. Moreover, we relax some of these constraints, i.e. we only consider a subset $\mathcal{I}' \subseteq \mathcal{I}$ of the constraints. Denote the corresponding heuristic by $Q_{\mathcal{I}'}(\boldsymbol{\varphi})$. Since relaxing constraints can only increase the maximum, we get:

Theorem 2. *$Q_{\mathcal{I}'}$ is admissible for all $\mathcal{I}' \subseteq \mathcal{I}$.*

It remains to explain which constraints to relax to obtain a heuristic which is easier to compute. This is illustrated in Figure 1 on the left. We propose to relax, what we call, *late constraints*, i.e., 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. The terminal reward is a Dec-POMDP value, with horizon $h-r$ and the revealed state as initial belief. This is similar to Eq. (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. Choosing a larger r typically yields a tighter, but more expensive 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 1 on the right. 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 $Q(s, \mathbf{a}, h-r)$ 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 model that the state information cannot be used in stage 0, we now compute

$$\max_{\mathbf{a}\in\boldsymbol{\mathcal{A}}} \sum_{s\in\mathcal{S}} b(s) \cdot Q(s, \mathbf{a}, h-r) \qquad (3)$$

as a heuristic for the terminal reward corresponding to the joint belief b revealed in stage r. Taking the maximum over the actions outside of the weighted sum in Eq. (3) 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). In practice, since computing $Q(s, \mathbf{a}, h-r)$ is relatively expensive, we first compute an MDP heuristic for the Dec-POMDP with horizon $h-r$, and only compute $Q(s, \mathbf{a}, h-r)$ for the actions $\mathbf{a} \in \boldsymbol{\mathcal{A}}$ which based on the MDP values can possibly attain the maximum in Eq. (3).

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 1: Hyperparameters for PF-MAA*.

Horizon reduction. TR-MAA* also applies a horizon reduction as explained in Sec. 5. For TR-MAA*, the terminal reward is a heuristic computed as explained in the previous paragraph. If $h > r$, we also apply the horizon reduction to compute a heuristic for the root node.

7 Empirical Evaluation

This section provides an empirical evaluation of PF-MAA* (Sec. 5) and TR-MAA* (Sec. 6). As baseline, we provide reference values from the state-of-the-art ϵ-optimal solver FB-HSVI [Dibangoye *et al.*, 2014; 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][3], and upper bounds found by RS-MAA* [Koops *et al.*, 2023]. For RS-MAA*, we give the best results among the heuristics $Q_{1,\infty}$, $Q_{25,\infty}$, $Q_{200,\infty}$ and $Q_{200,3}$.

Implementation. We base our implementation on the data structures and optimizations of RS-MAA* [Koops *et al.*, 2023][4]. Notably, we handle the last agent's last stage efficiently and abort heuristic computations after $M = 200$ steps.

Setup. All experiments ran on a system with an Apple M1 Ultra using the PyPy environment. 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. We ran TR-MAA* with a limit of 120 seconds and a limit of 1800 seconds, respectively. We used a global 16GB memory limit. We validated PF-MAA* values reported in this paper with a separate code base. We gave both RS-MAA* and TR-MAA* the values found by PF-MAA*.

Hyperparameter selection. For PF-MAA*, the main hyperparameters are the window size k, which of the heuristics $Q_{\mathsf{maxr},r}$ or $Q_{\mathsf{MDP},r}$ to use, the depth r of the heuristic, and the iteration limit L per stage. We report the configurations that we used in Table 1. Setting $r \geq 4$ or $k \geq 4$ is not feasible for all benchmarks. For TR-MAA*, we use the heuristic from Section 6 with $r = 3$ and $r = 5$ respectively. In each case, the Q_3 heuristic is used to solve Dec-POMDPs.

Benchmarks. We used the standard benchmarks from the literature: DECTIGER [Nair *et al.*, 2003], FIREFIGHTING [Oliehoek *et al.*, 2008b] (3 fire levels, 3 houses), GRID with two observations [Amato *et al.*, 2006], BOXPUSHING [Seuken and Zilberstein, 2007], GRID3X3 [Amato *et al.*,

[3]The implementations for these solvers are not available, we copy the achieved results from the respective papers.

[4]See https://zenodo.org/records/11160648.

h	FB-HSVI	GA-FSC	PF-MAA*			random	upper
			Q_{MDP}	fast	quality		
DECTIGER							
50	80.7		79.9	79.1	**81.0**	−2311.1	101.3
100	**170.9**	169.3	169.3	169.3	**170.9**	−4622.2	206.4
GRID							
20			14.23	14.36	**14.69**	4.67	17.13
50		**40.49**	36.86	37.46	MO	12.17	47.21
BOXPUSHING							
20	458.1	468.1	402.5	466.3	**475.0**	−20.5	476.4
50	1134.7	1201.0	949.8	1207.7	**1209.8**	−57.9	1218.4
100		2420.3	1864.5	2431.4	**2433.5**	−120.5	2453.4
MARS							
50	**128.9**		116.7	117.0	122.6	−62.0	132.8
100	**249.9**		221.7	222.0	234.1	−122.7	265.7

Table 2: Lower bounds, i.e. the value of the policies obtained with different policy-finding algorithms. MO denotes memout (>16GB). A full version is available in the supplementary material.

h	FB-HSVI	RS-MAA*		TR-MAA*		lower	MDP
		fast	quality	fast	quality		
GRID							
7	4.48	TO	MO	**4.47**	4.49	4.47	5.81
20		TO	MO	**17.13**	MO	14.69	18.81
50		TO	MO	**47.21**	MO	40.49	48.81
BOXPUSHING							
20		TO	MO	481.2	**476.4**	475.0	511.1
50		TO	MO	1227.4	**1218.4**	1209.8	1306.2
100		TO	MO	2469.7	**2453.4**	2433.5	2628.1
MARS							
50		**132.8**	**132.8**	136.5	136.9	128.9	145.0
100		287.5	**265.7**	273.4	276.5	249.9	289.0

Table 3: 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). A full version is available in the supplementary material.

2009], MARS [Amato and Zilberstein, 2009], HOTEL [Spaan and Melo, 2008], RECYCLING [Amato *et al.*, 2007], and BROADCAST [Hansen *et al.*, 2004].

7.1 Results

Before we discuss the results, we present an overview of the most interesting data in Tables 2 and 3, giving results for the lower and upper bound, respectively. Further data is given in the supplementary material. Each table presents results for various benchmarks and different horizons h. In Table 2, 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*[5]. In Table 3, 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.

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.

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 Sec. 5 than using Q_{MDP} (for the same number of iterations L; see Table 1). We note that Q_{MDP} is the only heuristic in the literature that scales up to high horizons on all benchmarks.

[5]We took these bounds from either the literature or our methods, see the supplementary material for details.

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.

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 243.97 for $h = 50$ and $h = 100$ respectively.

A scalable upper bound. On all benchmarks, we provide an upper bound for horizons up to 100 which is better than Q_{MDP}. This is an improvement over RS-MAA*, which only reaches horizon 10 on BOXPUSHING and horizon 6 on GRID. In fact, 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 9 times closer to the optimum than the MDP value.

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.

Timings. PF-MAA* is generally faster than required by the time limit. For instance, BOXPUSHING $h = 100$ takes 9 and 401 seconds combined, respectively. For $r = 5$, TR-MAA* often reaches its memory limit within 300 seconds.

8 Conclusion

We presented novel methods for finding good policies for Dec-POMDPs and for finding upper bounds on the optimal value. Together, this allows us to find policies with values that are at least 99% of the optimal value, on horizons an order of magnitude higher than for which exact solving is possible, as well as giving upper bounds for high horizons the first time. The main advancements leading to this result are clustered sliding window memory, pruning the priority queue and several new A*-heuristics. Further research includes investigating different methods for clustering observation histories.

Acknowledgements

We would like to thank the anonymous reviewers for their useful comments. This work has been partially funded by the ERC Starting Grant DEUCE (101077178), the NWO Veni grant ProMiSe (222.147), and the NWO grant PrimaVera (NWA.1160.18.238).

Contribution Statement

Wietze Koops is the primary designer of the algorithm and implemented it and evaluated the algorithms. The other authors contributed discussions, ideas, and shaped the description of the algorithm.

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. Springer, 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.

[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.

[Carlin and Zilberstein, 2008] Alan Carlin and Shlomo Zilberstein. Value-based observation compression for DEC-POMDPs. In *AAMAS*, pages 501–508, 2008.

[Cazenave, 2010] Tristan Cazenave. Partial move A*. In *22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 25–31. IEEE, 2010.

[Dibangoye and Buffet, 2018] Jilles Steeve Dibangoye and Olivier Buffet. Learning to act in continuous Dec-POMDPs. In *JFPDA*. HAL, 2018.

[Dibangoye *et al.*, 2009] Jilles Steeve Dibangoye, Abdel-Illah Mouaddib, and Brahim Chaib-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *AAMAS*, pages 569–576, 2009.

[Dibangoye *et al.*, 2014] 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.*, 2016] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. Optimally solving Dec-POMDPs as continuous-state MDPs. *JAIR*, 55:443–497, 2016.

[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.

[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.

[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.

[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.

[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.

[Littman *et al.*, 1995] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, pages 362–370, 1995.

[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.

[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 *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.*, 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.

[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, 2007] Sven Seuken and Shlomo Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *UAI*, pages 344–351. AUAI Press, 2007.

[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.

[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.