Optimally solving Dec-POMDPs as Continuous-State MDPs: Theory and Algorithms

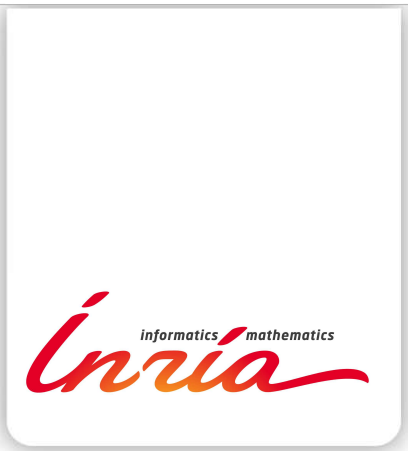
Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, François Charpillet

Optimally solving Dec-POMDPs as Continuous-State MDPs: Theory and Algorithms

Jilles Steve Dibangoye, Christopher Amato
Olivier Buffet, François Charpillet

RESEARCH REPORT
N° 8517
April 2014
Project-Team MAIA

ISSN 0249-6399 ISRN INRIA/RR--8517--FR+ENG



Optimally solving Dec-POMDPs as Continuous-State MDPs: Theory and Algorithms

Jilles Steve Dibangoye*, Christopher Amato†
Olivier Buffet*, François Charpillet*

Project-Team MAIA

Research Report n° 8517 — April 2014 — 77 pages

Abstract: Decentralized partially observable Markov decision processes (Dec-POMDPs) provide a general model for decision-making under uncertainty in cooperative decentralized settings, but are difficult to solve optimally (NEXP-Complete). As a new way of solving these problems, we introduce the idea of transforming a Dec-POMDP into a continuous-state deterministic MDP with a piecewise-linear and convex value function. This approach makes use of the fact that planning can be accomplished in a centralized offline manner, while execution can still be distributed. This new Dec-POMDP formulation, which we call an *occupancy MDP*, allows powerful POMDP and continuous-state MDP methods to be used for the first time. When the curse of dimensionality becomes too prohibitive, we refine this basic approach and present ways to combine heuristic search and compact representations that exploit the structure present in multi-agent domains, without losing the ability to eventually converge to an optimal solution. In particular, we introduce feature-based heuristic search that relies on feature-based compact representations, point-based updates and efficient action selection. A theoretical analysis demonstrates that our feature-based heuristic search algorithms terminate in finite time with an optimal solution. We include an extensive empirical analysis using well known benchmarks, thereby demonstrating our approach provides significant scalability improvements compared to the state of the art.

Key-words: Decentralized Control, Decentralized Partially Observable Markov Decision processes, Dec-POMDPs, automated planning, multi-agent systems, uncertainty

* MAIA project-team - firstname.lastname@inria.fr
† Massachussetts Institute of Technology, Cambridge, MA, USA – camato@csail.mit.edu

RESEARCH CENTRE NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Résolution optimale de Dec-POMDP à horizon fini comme des MDP à espace d'états continu: Théorie et algorithmes

Résumé : Les processus de décision markoviens partiellement observables décentralisés (Dec-POMDP) fournissent un modèle général pour la prise de décision dans l'incertain dans des cadres coopératifs décentralisés. En guise de nouvelle approche de résolution de ces problèmes, nous introduisons l'idée de transformer un Dec-POMDP en un MDP déterministe à espace d'états continu dont la fonction de valeur est linéaire par morceaux et convexe. Cette approche exploite le fait que la planification peut être effectuée d'une manière centralisée hors ligne, alors que l'exécution peut toujours être distribuée. Cette nouvelle formulation des Dec-POMDP, que nous appelons un *occupancy MDP*, permet pour la première fois d'employer de puissantes méthodes de résolution de POMDP et MDP à états continus. La malédiction de la dimensionalité devenant prohibitive, nous raffinons cette approche basique et présentons des façons de combiner la recherche heuristique et des représentations compactes qui exploitent la structure présente dans les domaines multi-agents, sans perdre la capacité de converger à terme vers une solution optimale. En particulier, nous introduisons une recherche heuristique qui repose sur des représentations compactes fondées sur des *features*, sur des mises-à-jour à base de points, et une sélection d'action efficace. Une analyse théorique démontre que nos algorithmes de recherche heuristique fondés sur des features se terminent en temps fini avec une solution optimale. Nous incluons une analyse empirique extensive utilisant des bancs d'essai bien connus, démontrant ainsi que notre approche améliore significativement le passage à l'échelle en comparaison de l'état de l'art.

Mots-clés : Contrôle réparti, processus de décision markoviens partiallement observables décentralisés, Dec-POMDP, planification automatique, systèmes multi-agents, incertain

Contents

1 Introduction

Decision making problems in sequential multiagent environments arise in many real-world situations. Examples include: exploration robots that must coordinate to perform experiments on an unknown planet (Zilberstein et al., 2002); rescue robots that, after a disaster, must safely find victims as quickly as possible (Paquet et al., 2010); distributed congestion control in a network (Winstein and Balakrishnan, 2013); or sensor networks where multiple sensors work jointly to perform a large-scale sensing task under strict power constraints (Jain et al., 2009). All these tasks require multiple decision makers, or agents, to coordinate their actions in order to achieve common long-term goals. Additionally, uncertainty is ubiquitous in these domains, both in the effects of actions and in the evolution of the actual system.

Markov decision processes (MDPs) address uncertainty in system dynamics, but assume centralization. Standard methods to solving MDPs, *e.g.,* linear and dynamic programming (Bellman, 1957; Puterman, 1994) and heuristic search (Barto et al., 1995; Hansen and Zilberstein, 2001; Smith and Simmons, 2006), are centralized during both the planning and the execution phases. When multiple agents are present, at plan-time, a single agent must have a global view of the underlying state of the entire system, and plan on behalf of its teammates. Every runtime step, this agent would transmit the appropriate action

that each agent must perform. These methods, collectively referred to as the *centralized planning for centralized control* paradigm, assume agents communicate with eachother each step with no delay or cost, either explicitly through messages or implicitly through observations. Unfortunately, in many practical applications agents are not permitted to observe the exact state of the system; rather each agent possesses only certain local, unshared observations, and acts without full knowledge of what others observe or plan to do. These characteristics have led to the development of a rich body of research on decentralized control.

The decentralized partially observable Markov decision process (Dec-POMDP) is a standard formulation for cooperative decision-making in sequential settings without instantaneous, free and noiseless communication (Bernstein et al., 2002; Nair et al., 2003). Over the past decade, there has been extensive research on solution methods for the Dec-POMDP model, using methodologies including *dynamic programming* (Hansen et al., 2004; Boularias and Chaib-draa, 2008; Amato et al., 2009), *linear programming* (Aras and Dutech, 2010) and *heuristic search* (Szer et al., 2005; Oliehoek et al., 2008, 2013). These approaches directly search for an optimal solution in the policy space, but quickly become intractable. This is not unexpected given the worst-case NEXP complexity of finite-horizon Dec-POMDPs (Bernstein et al., 2002). That is, as the number of decision steps (i.e., the problem horizon) increases the number of policies grows doubly exponentially, causing algorithms to quickly run out of either time or memory.

In heuristic search, algorithms use *heuristic value functions*; such functions yield a rough evaluation of the quality of each policy, and provide a ranking of alternative policies. In linear and dynamic programming, algorithms rely on the concept of *exact value functions*, which evaluates the sum of expected future rewards, as a function of the current policy and state. In either context, there are methods to compute value functions and converge to an optimal solution (Hansen et al., 2004; Szer et al., 2005; Oliehoek et al., 2008; Amato et al., 2009). Due to the doubly exponential growth in the number of policies, the practical application of these methods are somewhat limited. Methods have been developed to compress the policy space in an attempt to increase scalability(Boularias and Chaib-draa, 2008; Aras and Dutech, 2010; Oliehoek et al., 2013), but scalability remains limited for many problems.

While many Dec-POMDP algorithms assume that planning can be centralized as long as execution remains decentralized, current algorithms do not take full advantage of this assumption. As an alternative approach, we formulate a Dec-POMDP as a *continuous-state MDP* and show that the value function is piecewise-linear and convex. In this form, theory from POMDPs (Kaelbling et al., 1998) applies, allowing POMDP algorithms to produce optimal solutions to Dec-POMDPs. A wide range of POMDP algorithms, which have demonstrated significant scalability (Shani et al., 2013; Silver and Veness, 2010), can now be applied. We extend one such heuristic search algorithm (Smith and Simmons, 2004) to the Dec-POMDP case, but the curse of dimensionality results in very large state and action spaces for problems with a long planning horizon.

When the curse of dimensionality becomes too prohibitive, we refine this basic approach and present ways to combine classical heuristic search and compact representations, without losing the ability to converge to an optimal solution. To incorporate compact representations, we build on *feature-based dynamic programming* (Tsitsiklis and van Roy, 1996), which includes feature extraction and approximation methods. The main focus then is on efficient algorithms for designing and computing compact representations that can approximate piecewise-linear and convex functions arbitrarily well. In particular, we introduce feature-based heuristic search that relies on feature-based compact representations, point-based updates and efficient action selections. A theoretical analysis demonstrates that our feature-based heuristic search algorithms terminate in finite time with an optimal solution. This combination of POMDP theory and compact representations can greatly reduce the problem size and solution efficiency while retaining optimal solutions for Dec-POMDPs.

The remainder of this paper is structured as follows. We begin in Section 2 with a thorough background on Dec-POMDPs, solutions and value functions. Section 3 then demonstrates how to solve a Dec-POMDP as a continuous-state Markov decision process, and discusses various fundamental results. Next,

we describe in Section 4 feature-based compact representations that alleviate the curse of dimensionality. Thereafter, Section 5 presents our solution method, feature-based heuristic search, and its properties. We finally describe experimental results that compare our approach to the state of the art in Section 6. We include an extensive empirical analysis using well known benchmarks, demonstrating that our approach can produce solutions for significantly longer horizons than the current state of the art. Finally, we discuss some related issues, point to several key open questions, and end with concluding remarks.

2 Dec-POMDPs

This section introduces the basic components of decentralized partially observable Markov decision processes (Dec-POMDPs).

2.1 Problem Definition and Notation

Consider the problem in which multiple agents are faced with the task of influencing the behavior of a stochastic system as it evolves over time (see the two agent case in Figure 1). At every time step, each agent receives a private observation that gives (possibly) incomplete information about the current state of the system. Since the states are not observable, an agent cannot choose its actions based on the states. Instead, it can consider a complete history of its past actions and observations to choose an action. Actions produce two results: agents receive a common immediate reward, and the system evolves to a new state at the next time step according to a probability distribution conditioned on actions. At the next time step, agents face a similar problem again, but now the system may be in a different state. The goal of agents is to choose the sequence of actions based on these local observation sequences which cause the system to perform optimally with respect to some performance criterion (which we discuss below). The Dec-POMDP model formalizes these interactions between agents and the system. This paper formulates and solves this general decentralized stochastic control problem for a process that is to operate for only a finite planning horizon. A later paper will examine this general decentralized stochastic control problem for a process that is to operate into the indefinite future.

Definition 1. *A **Dec-POMDP** with $|I|$ agents is represented as a tuple $M \equiv (I, S, A, Z, p, o, r, b_0, T)$, where:*

- *I is a finite set of **agents** $i = 1, 2, \ldots, |I|$*
- *S is a finite set of n **states***
- *A^i is the finite set of agent i's **actions**. $A \equiv \otimes_i A^i$ is the finite set of joint actions*
- *Z^i is the finite set of agent i's **observations**. $Z \equiv \otimes_i Z^i$ is the finite set of joint observations*
- *$p = \{p^a \mid a \in A\}$ denotes the **transition model**. p^a is a $n \times n$ stochastic matrix, where $p^a(s, s')$ is the probability of ending up in state s' if the agents choose joint action a in state s*
- *$o = \{o^{a,z} \mid a \in A, z \in Z\}$ is the **observation model**. $o^{a,z}$ is a $n \times 1$ vector[1], where $o^{a,z}(s')$ is the probability of observing z if joint action a is performed and the resulting state is s'*
- *$r = \{r^a \mid a \in A\}$ is the **reward function**. r^a is a $1 \times n$ reward vector, where $r^a(s)$ is the bounded reward obtained by executing joint action a in state s*
- *b_0 is the initial probability distribution over states*
- *T is the number of decision **epochs** $t = 0, 1, \ldots, T-1$ (the problem horizon)*

We often use shorthand notation $p^{a,z}(s, s') \stackrel{\text{def}}{=} o^{a,z}(s')p^a(s, s')$, for all $s, s' \in S$, $a \in A$, and $z \in Z$, combining the transition and observation models.

To illustrate this problem and corresponding concepts, we introduce a very simple example, namely the multi-agent tiger problem (Nair et al., 2003).

[1] The observation vector is not a stochastic vector.

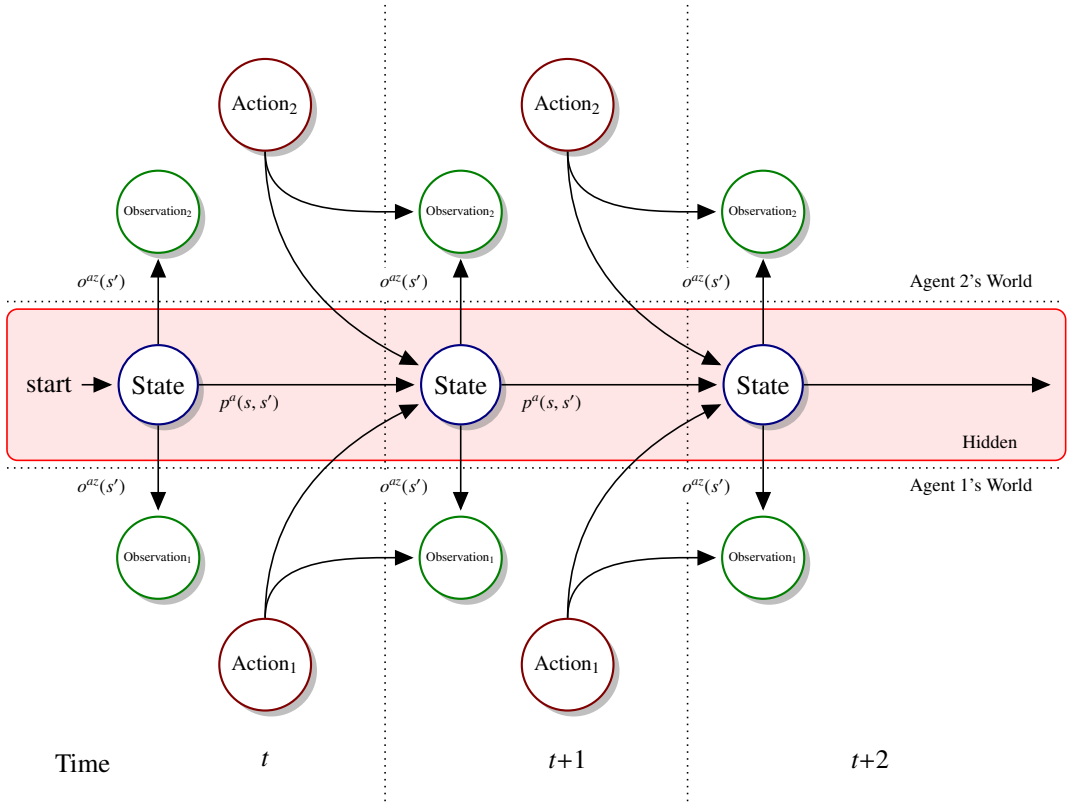


Figure 1: A graphical model of the two-agent Dec-POMDP model, where rewards are omitted

Example 1 (Multi-agent tiger — problem description)**.** *In the multi-agent tiger world, two agents stand in front of two closed doors. Behind one of the doors there is a hungry tiger, and behind the other door there is valuable treasure. The agents do not know the position of either. By listening, rather than simply opening one of the doors, the agents can gain information about the position of the tiger. But listening has a cost and is not entirely accurate (*i.e.*, it only reveals the correct information about the location of the tiger with some probability). Moreover, agents cannot communicate their observations to each other. At each step, each agent can independently either listen or open one of the doors. If one of the agents opens the door with the treasure behind it, they both get the reward. If either agent opens the door with the tiger, a penalty is incurred. However, if they both open the tiger door at the same time, they receive less penalty. The agents have to come up with policies for listening and opening doors based on the local observations. After a door is opened and the agents receive a reward or penalty, the problem starts over again and the tiger is randomly repositioned.*

We refer to the state of the multi-agent tiger world when the tiger is on the left as s_{LEFT} *and when it is on the right as* s_{RIGHT}*. The actions for each agent are* a_{LEFT}*,* a_{RIGHT}*, and* a_{LISTEN}*. There are only two possible observations for each agent (even after opening a door): to hear the tiger on the left* z_{LEFT} *or to hear the tiger on the right* z_{RIGHT}*. The reward function is defined as shown on Table 1.*

The transition and observation models can be described in detail as follows. The joint action $(a_{\text{LISTEN}}, a_{\text{LISTEN}})$ *does not change the state of the world. Any other joint action causes a transition to state* s_{LEFT} *with probability* 0.5 *and to state* s_{RIGHT} *with probability* 0.5 *— essentially resetting the problem. When the world is in*

actions of both agents	listens	opens good door	opens bad door
listens	−2	+9	−101
opens good door	+9	+20	−100
opens bad door	−101	−100	−50

Table 1: Reward function definition for the multi-agent tiger problem

state s_{LEFT}*, the joint action* $(a_{\text{LISTEN}}, a_{\text{LISTEN}})$ *results in observation* z_{LEFT} *for either agent with probability* 0.85 *and observation* z_{RIGHT} *with probability* 0.15*; conversely for state* s_{RIGHT}*. No matter what state the world is in, the other joint actions result in either observation with probability* 0.5*.*

This example illustrates the characteristics of the general optimal decentralized control in Dec-POMDPs. Due to uncertainty about the location of the tiger and the actions of the other agent, coordination in the multi-agent tiger problem is a difficult problem. We will use this example to help explain some of the concepts in this paper.

After specifying a T-step Dec-POMDP M, we would like agents to act in such a way as to maximize some common measure of long-term reward received. To achieve this goal, we need to combine agents' private information. The challenge in Dec-POMDP problem solving is thus in formulating each agent behavior so as to take into account (sufficiently well) the other agent behaviors. In this paper, we address this problem from the standpoint of the *(offline) centralized planning for decentralized control* paradigm, discussed in Section 2.1.1. This paradigm allows us to distinguish between the information available to the agents at the runtime (Section 2.1.2); and the information a centralized coordinator agent has at the plan-time (Section 2.1.3).

2.1.1 On the Use of Centralized Planning

A common assumption in many Dec-POMDP planning algorithms is that planning takes place in a centralized (offline) manner while actions are executed in a decentralized fashion (Hansen et al., 2004; Szer et al., 2005; Boularias and Chaib-draa, 2008; Oliehoek et al., 2008; Amato et al., 2009; Aras and Dutech, 2010; Oliehoek et al., 2013). A *centralized coordinator agent* generates separate policies for each agent and sends them out. Agents in turn execute them, as illustrated in Figure 2.

In this planning paradigm, we distinguish between two phases: (a) *plan-time* on the left side of Figure 2; and (b) *runtime* on the right side of Figure 2. At plan-time, a centralized coordinator agent computes a policy for each agent, specifying which action to perform for every possible action and observation histories that agent can experience (Hansen et al., 2004; Szer et al., 2005). Because the process takes place offline, the centralized coordinator agent knows the actions the agents would choose and the possible observations that result. Hence, the centralized coordinator agent has the knowledge about all private information of all agents. Once the plan-time terminates, each agent receives its private policy.

At runtime, however, agents are not allowed to share their actions and observations with each other. The only information an agent has about the state of the process is its past actions and observations. So, each agent proceeds by executing its policy based only on its past action and observation histories. In the following, we first present useful concepts about decentralized control and centralized planning.

2.1.2 Decentralized Control

At every runtime step, each agent chooses an action to be executed based on the actions executed and the observations received. A complete description of the behavior of that agent is called a *policy*. To better understand this concept, we first introduce the notions of private *histories* and *decision rules*.

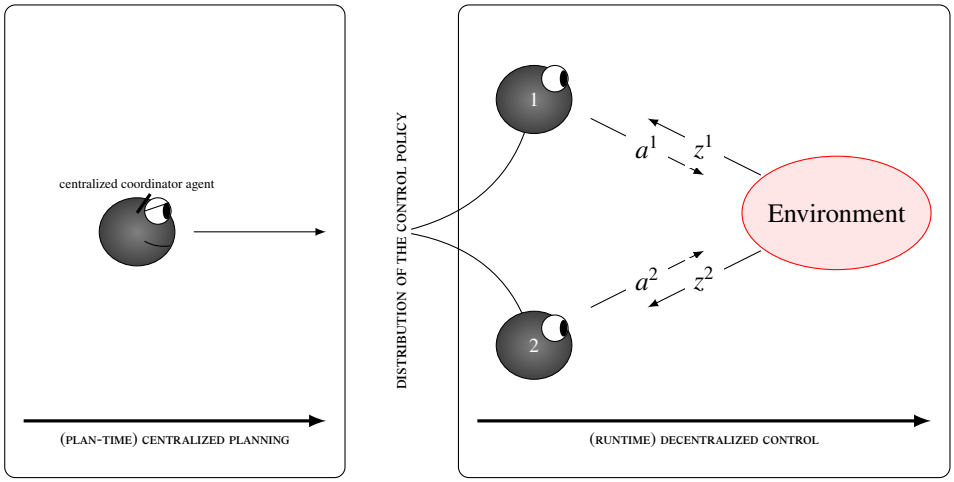


Figure 2: From (offline) centralized planning to decentralized control

Definition 2. *A t-step **history** of agent $i \in I$ is a length-t sequence of actions and observations, $\theta_t^i \stackrel{\text{def}}{=} (a_0^i, z_1^i, \ldots, z_{t-1}^i, a_{t-1}^i, z_t^i)$, where a_t^i and z_t^i denote actions and observations of agent $i \in I$ at runtime step $t \in \{0, 1, \ldots, T\}$.*

Any t-step history θ_t^i follows the recursion $\theta_t^i = (\theta_{t-1}^i, a_{t-1}^i, z_t^i)$ and initial history θ_0^i is empty. Let Θ_t^i be the set of all t-step histories of agent $i \in I$, namely the t-step history set. Then, the t-step history set follows the recursion $\Theta_t^i = \Theta_{t-1}^i \times A^i \times Z^i$, for all runtime steps $t \in \{1, 2, \ldots, T\}$; and the initial history set Θ_0^i is the singleton $\{\theta_0^i\}$.

Definition 3. *A t-step **decision rule** $d_t^i \colon \Theta_t^i \mapsto A^i$ prescribes agent $i \in I$ the action to be executed with certainty in each history $\theta_t^i \in \Theta_t^i$ at a specified runtime step $t \in \{0, 1, \ldots, T\}$.*

There exists many types of decision rules including: *randomized* and *deterministic* decision rules. In Dec-POMDPs, there exists optimal deterministic policies (Oliehoek et al., 2008; Puterman, 1994) and deterministic decision rules are simpler to implement and evaluate. For these reasons, we focus on deterministic decision rules, which we just call decision rules for the sake of simplicity. We further denote D_t^i to be the set of all t-step decision rules for agent $i \in I$ at runtime step $t \in \{0, 1, \ldots, T\}$, namely the t-step *decision rule set* of agent $i \in I$.

Definition 4. *A **policy** $\pi_{t_0:t_1}^i \stackrel{\text{def}}{=} (d_{t_0}^i, \ldots, d_{t_1}^i)$ is a sequence of decision rules for agent $i \in I$ from runtime step t_0 to runtime step t_1, where $0 \leq t_0 \leq t_1 \leq T$.*

Let $\Pi_{t_0:t_1}^i$ be the set all policies $\pi_{t_0:t_1}^i$ for agent $i \in I$, namely *the policy set* of agent $i \in I$. Each policy set is given by the cross-product of decision rule sets of agent $i \in I$ from the initial runtime step to the terminal runtime step. While policies provide a complete description of an agent behavior, they are memory demanding. They consist of decision rules that specify actions for all possible histories the agent may experienced. Fortunately, alternative descriptions of an agent behavior exists, including *policy trees* and *policy-tree functions*.

Definition 5. *A **policy tree** δ^i for agent $i \in I$ is formally defined as a tuple $(\mathcal{X}^i, \mathcal{A}^i, C^i)$ where:*

- *$\mathcal{X}^i$ is a set of all nodes x^i, where x_0^i is the root node;*
- *$\mathcal{A}^i(x^i)$ denotes the action $a^i \in A^i$ to be executed in node x^i;*

- $C^i(x^i, z^i)$ *defines the child node of* x^i *after perceiving observation* $z^i \in Z^i$.

We denote $\bar{\Pi}^i_{t_0:t_1}$ *the set of all policy trees* $\delta^i_{t_0:t_1}$ *of agent* $i \in I$ *from runtime step* t_0 *to runtime step* t_1.

A policy tree provides an agent with a behavioral strategy, which circumvents the overwhelming memory required to represent policies. In fact, a policy tree is a tree, where the root node specifies the first action the agent needs to perform. Then, after perceiving an observation, the agent transits to another node at the next level. At every runtime step, the agent faces the same problem of taking the action the current node prescribes, and making a transition to another node based on the observation received. Policies and policy trees provide an agent with behavioral strategies, but they remain significantly different. Policies prescribe a different behavior for each history the agent may experience. In policy trees, however, given the current node, the agent behavior is independent of the history. It is this conditional independence that makes policy trees different from policies.

Nonetheless, there is a close relationship between policies and policy trees. In fact, A policy can be defined as a mapping from histories to policy trees, namely a *policy-tree function*. To better understand this, let $\pi^i_{t_0:t_1}$ be a policy of agent $i \in I$ from runtime step t_0 to runtime step t_1, and $\Psi^i \colon \Theta^i_{t_0} \mapsto \bar{\Pi}^i_{t_0:t_1}$ its corresponding behavioral strategy mapping from histories to policy trees. For any arbitrary history $\theta^i_{t_0}$, we define policy tree $\Psi^i(\theta^i_{t_0}) = (\mathcal{X}^i, \mathcal{A}^i, C^i)$ as follows:

- $\mathcal{X}^i = \{x_{\theta^i} \mid \forall t \in \{t_0, \dots, t_1\}, \forall \theta^i \in \Theta^i_t\}$, where $x_{\theta^i_{t_0}}$ is the root node;
- $\mathcal{A}^i(x^i_{\theta^i})$ denotes the action $a^i = \pi^i_{t_0:t_1}(\theta^i)$ to be executed in node $x^i_{\theta^i} \in \mathcal{X}^i$;
- $C^i(x^i_{\theta^i}, z^i)$ defines the child node $x^i_{\theta^i a^i z^i}$ after perceiving observation $z^i \in Z^i$.

Clearly, a policy-tree function prescribes the agent the policy tree to be executed in each history. The policy-tree function has a natural interpretation, it is equivalent to the policy that prescribes actions to nodes. But it requires less memory in the sense that many histories may map to the same policy tree. Besides, it makes clear the connection between policies and policy trees. In the remainder of this paper, we will be using both policies and policy-tree functions interchangeably.

2.1.3 Centralized Planning

Section 2.1.2 focussed on the information each agent has at runtime. In this section, we focus on the information the centralized coordinator agent has about the process to be controlled at plan-time. At every plan-time step, he receives all possible observations each agent might perceive and chooses the action each agent will perform at runtime. The following extends agent concepts introduced in Section 2.1.2, *e.g.,* histories, decision rules, policies and policy trees, to team concepts.

Definition 6. *A t-step* ***joint history*** *is a length-t sequence of joint action and joint observations,* $\theta_t \stackrel{\text{def}}{=} (a_0, z_1, \dots, z_{t-1}, a_{t-1}, z_t)$*, where* a_t *and* z_t *denote joint actions and joint observations at plan-time step* $t \in \{0, 1, \dots, T\}$.

Joint history θ_t satisfies the recursion $\theta_t = (\theta_{t-1}, a_{t-1}, z_t)$. It also corresponds to a $|I|$-tuple of histories $\theta_t = (\theta^1_t, \theta^2_t, \dots, \theta^{|I|}_t)$, where θ^i_t denotes a private history of agent $i \in I$. For the sake of simplicity, we often denote joint history θ_t as a pair of histories (θ^i_t, θ^j_t), where $\theta^j_t \stackrel{\text{def}}{=} (\theta^1_t, \dots, \theta^{i-1}_t, \theta^{i+1}_t, \dots, \theta^{|I|}_t)$ is the joint history of agents $I \backslash \{i\}$. We further define Θ_t to be the set of all t-step joint histories at plan-time $t \in \{0, 1, \dots, T\}$, namely a t-step joint history set. A t-step joint history set follows the recursion $\Theta_t = \Theta_{t-1} \times A \times Z$ and initial joint history set $\Theta_0 = \{\theta_0\}$.

Definition 7. *A t-step* ***belief state*** $b_t(s) \stackrel{\text{def}}{=} Pr(s_t = s | b_0, \theta_t)$ *is defined as the posterior probability distribution of being in each state, conditional on t-step joint history* θ_t *and initial belief state* b_0*, for all* $t \in \{0, 1, \dots, T\}$.

A t-step belief state b_t results from taking joint action $a \in A$ and receiving joint observation $z \in Z$ starting in $(t-1)$-step belief state b_{t-1}: $\forall s' \in S$,

$$b_t(s') = \frac{o^{a,z}(s') \sum_{s \in S} p^a(s, s') \cdot b_{t-1}(s)}{\sum_{s' \in S} o^{a,z}(s') \sum_{s \in S} p^a(s, s') \cdot b_{t-1}(s)}. \quad (1)$$

In partially observable Markov decision processes, the belief state is sufficient for optimal decision making (Aström, 1965). In Dec-POMDPs, however, agents are not allowed to communicate their private information with each other at runtime. Consequently, agents are (typically) not able to calculate belief states at runtime, so the belief state cannot be used for optimal decision making in Dec-POMDPs. Nonetheless, belief states are crucial to calculate the expected rewards to be gained after executing a policy.

Before presenting team decision rules, policies, policy trees, and policy-tree functions, we first introduce the concept of separability.

Definition 8. *Let $f(\boldsymbol{x})$ be a function that maps K-tuple variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_K)$ to a K-tuple values $\boldsymbol{y} = (y_1, y_2, \ldots, y_K)$. We say that f is **separable** if and only if there exists functions $f_1, f_2, \ldots, f_K$ such that: $f(\boldsymbol{x}) = (f_1(x_1), f_2(x_2), \ldots, f_K(x_K))$.*

The concept of separability ensures team decision rules d_t, policies $\pi_{\Delta t}$, policy trees $\delta_{\Delta t}$, or policy-tree functions $\Psi_{\Delta t}$ are behavioral strategies that can be split into $|I|$-tuple of individual decision rules $(d_t^i)_{i \in I}$, policies $(\pi_{\Delta t}^i)_{i \in I}$, policy trees $(\delta_{\Delta t}^i)_{i \in I}$, or policy-tree functions $(\Psi_{\Delta t}^i)_{i \in I}$, respectively for all $t \in \{0, 1, \ldots, T\}$. As such, separable behavioral strategies that are formulated in a centralized manner can nonetheless be executed in a decentralized fashion that depends only on a given agent's past actions and observations and not those of the other agents.

We are now ready to define the total available information the centralized coordinator agent has at plan-time about the process to be controlled. Looking back at the centralized planning for decentralized control paradigm (Figure 2), it is worth noting that at plan-time the centralized coordinator agent has no access to either what specific observations agents will receive or what specific actions they will perform at runtime. It is this private information that agents receive, but the coordinator agent only knows the possible actions and observations as well as probabilities that each one will occur. In other words, the only information about the process available at plan-time is the separable decision rule agents will select at each specific runtime step. Thus, the total available information about the process to be controlled at runtime step t is a sequence of separable decision rules starting from the initial belief state. This information refers to as the *t-step information state*.

Definition 9. *The t-step **information state** $\iota_t \stackrel{\text{def}}{=} (b_0, \pi_{0:t-1})$ is a length-t sequence of separable decision rules starting with the initial belief state b_0, for all time step $t \in \{0, 1, \ldots, T\}$. Information state ι_t satisfies the recursion: $\iota_0 = (b_0)$, and $\iota_{t+1} = (\iota_t, d_t)$, for all t.*

2.1.4 Size of Information Sets

Sections 2.1.2 and 2.1.3 presented the information sets required during both the decentralized control or the centralized planning phases, including: history, decision rule, policy, policy trees, and policy-tree function sets. In this section, we emphasize on the prohibitive dimensions of the information sets we use throughout the paper. This also highlights the impetus for compact representations. Table 2 provides the worst case complexity required to enumerate elements in these information sets.

We distinguish between agent and team information sets. The agent sets are often exponential in the number of action and observation histories. The team sets all result by taking the cross-product over agents' sets. Since there are $|I|$ agents, the size of these sets are essentially the size of an agent's set raises at power of $|I|$. To cope with the fact that not all agents have the same set, we use A_* and Z_* to denote the

	Agent	Team
History Set	$\lvert Z^i\rvert^t \lvert A^i\rvert^t$	$\lvert Z\rvert^t \lvert A\rvert^t$
Decision Rule Set	$\lvert A^i\rvert^{\lvert Z^i\rvert^{t-1}\lvert A^i\rvert^{t-1}}$	$\lvert A_*\rvert^{\lvert I\rvert\lvert Z^*\rvert^{t-1}\lvert A_*\rvert^{t-1}}$
Policy Set	$\lvert A^i\rvert^{\frac{\lvert Z^i\rvert^t\lvert A^i\rvert^t-1}{\lvert Z^i\rvert\lvert A^i\rvert-1}}$	$\lvert A_*\rvert^{\frac{\lvert I\rvert\lvert Z^*\rvert^t\lvert A_*\rvert^t-1}{\lvert Z^*\rvert\lvert A_*\rvert-1}}$
Policy Tree Set	$\lvert A^i\rvert^{\frac{\lvert Z^i\rvert^t-1}{\lvert Z^i\rvert-1}}$	$\lvert A_*\rvert^{\frac{\lvert I\rvert\lvert Z^*\rvert^t-1}{\lvert Z^*\rvert-1}}$
Policy-Tree Function Set	$\lvert A^i\rvert^{\frac{\lvert Z^i\rvert^t\lvert A^i\rvert^t-1}{\lvert Z^i\rvert\lvert A^i\rvert-1}}$	$\lvert A_*\rvert^{\frac{\lvert I\rvert\lvert Z^*\rvert^t-1}{\lvert Z^*\rvert-1}}$

Table 2: Summary of the size table of t-step agent and team information sets. Sets A_* and Z^* denote the largest action A^i and observation Z^i sets, respectively.

largest action A_i and observation Z_i sets, respectively. Overall, it appears that the joint history set grows exponentially with the plan-time and the number of agents, but all separable decision rule, separable policy, separable policy tree, and information state sets increase doubly exponentially with increasing plan-time steps. In the remainder of this section, we provide details on individual agent history, decision rule, and policy sets, respectively.

The history set $\Theta_i^{0:t}$ contains as all possible t-length action and observation sequences. The number of t-length action and observation histories is

$$|\Theta_i^{0:t}| \quad = \quad |Z_i|^t |A_i|^t.$$

The t-th decision rule set D_i^t consists of t-th decision rule d_i^t mapping length-$(t-1)$ action and observation histories $\Theta_i^{0:t-1}$ to actions A_i. Since for each history in $\Theta_i^{0:t-1}$ we have $|A_i|$ alternative action candidates, the number of decision rules is

$$|D_i^t| = |A_i|^{|\Theta_i^{0:t-1}|} = |A_i|^{|Z_i|^{t-1}|A_i|^{t-1}}.$$

The t-th policy set $\Pi_i^{0:t}$ is a cross-product of decision rule sets. The total number of action and observation histories $|\cup_{\tau=0}^{t-1} \Theta_i^{0:\tau}|$ involved in a policy $\pi_i^{0:t}$ is

$$\begin{aligned} |\cup_{\tau=0}^{t-1} \Theta_i^{0:\tau}| \quad &= \quad \sum_{\tau=0}^{t-1} |\Theta_i^{0:\tau}| \\ &= \quad \sum_{\tau=0}^{t-1} |Z_i|^\tau |A_i|^\tau \\ &= \quad \frac{|Z_i|^t |A_i|^t - 1}{|Z_i||A_i| - 1}, \end{aligned}$$

which is exponential in t. Thus, the number of all possible t-th policies is

$$\begin{aligned} |\Pi_i^{0:t}| \quad &= \quad |A_i|^{|\cup_{\tau=0}^{t-1} \Theta_i^{0:\tau}|} \\ &= \quad |A_i|^{\frac{|Z_i|^t |A_i|^t - 1}{|Z_i||A_i| - 1}}, \end{aligned}$$

which is doubly exponential in t.

Consider now the size of the t-th policy tree set $\bar{\Pi}_i^{0:t}$. With $|Z_i|$-way branching at each level, the number of nodes in policy tree at level t is $|Z_i|^t$, so the number of nodes in the entire t steps to go police tree is

$$\begin{aligned} |\cup_{\tau=0}^{t-1} Z_i^\tau| \quad &= \quad \sum_{\tau=0}^{t-1} |Z_i|^\tau \\ &= \quad \frac{|Z_i|^t - 1}{|Z_i| - 1}, \end{aligned}$$

which is exponential in t. In order to specify a t-step police tree, one must select an action for each node from $|A_i|$ candidates, so the number of possible t-step policy trees is

$$|A_i|^{|\cup_{\tau=0}^{t-1} Z_i^\tau|} \quad = \quad |A_i|^{\frac{|Z_i|^t-1}{|Z_i|-1}},$$

which is doubly exponential. Although both policy and policy tree sets increase doubly exponentially with increasing run-time, the policy tree set is smaller than the policy set. For this reason, we use the policy tree to represent policies in practice.

2.2 Acting Optimally

In Markov decision theory, the reward, transition and observation functions depend on the history only through the current state and the action selected in that state; this property is also known as the *Markov assumption* (Puterman, 1994; Bellman, 1957). When the state is not observed (and history-dependent policies are used), the induced stochastic processes are no longer Markov. The reward, transition and observation probabilities become functions of states, actions and histories. For this reason, results in the Markov decision theory do not apply here. Many fundamental questions need to be investigated under deterministic and history-dependent separable policies. In this section, we discuss the optimality criterion we use throughout this paper.

2.2.1 Separable Policy Evaluation

One fundamental question follows. What is the expected value to be gained by executing a separable policy? We will demonstrate that it depends on the paths of states, joint actions and joint observations that separable policy traverses during the execution. These paths refer to as *sample paths* (Puterman, 1994). Clearly, each trajectory of state and action pairs $(s_{t_0}, a_{t_0}, \ldots, a_{t_1-1}, s_{t_1})$ yields a reward $\sum_{t=t_0}^{t_1} r^{a_t}(s_t)$. In calculating the sum of reward over all possible trajectories, we need to incorporate the probability of each of these trajectories, since state and action pairs are conditionally dependent on the executed separable policy. For this reason, we bind each trajectory with a corresponding sample path.

More formally, a sample path $\omega_{t_0:t_1} = (s_{t_0}, a_{t_0}, z_{t_0+1}, \ldots, a_{t_1-1}, z_{t_1}, s_{t_1})$ is a sequence of states, joint actions, and joint observations from plan-time step t_0 to plan-time step t_1. Each separable policy $\pi_{t_0:t_1}$ induces a probability distribution over sample paths, where the probability of any arbitrary sample path $\omega_{t_0:t_1}$ conditional on current information state ι_{t_0} is:

$$Pr(\omega_{t_0:t_1}|\pi_{t_0:t_1}, \iota_{t_0}) \quad \stackrel{\text{def}}{=} \quad Pr(s_{t_0}|\iota_{t_0}) \prod_{t=t_0}^{t_1-1} Pr(s_{t+1}, z_{t+1}|s_t, a_t) \cdot Pr(a_t|\pi_{t_0:t_1}, \omega_{t_0:t}). \tag{2}$$

Note that the action selection is determined implicitly through $Pr(a_t|\pi_{t_0:t_1}, \omega_{t_0:t})$ and the dynamics of the model M determine $Pr(s_{t+1}, z_{t+1}|s_t, a_t) = p^{a_t, z_{t+1}}(s_t, s_{t+1})$. Let $\pi_{t_0:t_1}$ be a separable policy, and $\theta_{t_0:t}$ the joint history in sample path $\omega_{t_0:t}$. Then, we have that $Pr(a_t|\pi_{t_0:t_1}, \omega_{t_0:t}) = 1$ if $\pi_{t_0:t_1}(\theta_{t_0:t}) = a_t$ and zero otherwise. As a consequence, (2) becomes:

$$Pr(\omega_{t_0:t_1}|\pi_{t_0:t_1}, \iota_{t_0}) \quad = \quad Pr(s_{t_0}|\iota_{t_0}) \prod_{t=t_0}^{t_1-1} p^{\pi_{t_0:t_1}(\theta_{t_0:t}), z_{t+1}}(s_t, s_{t+1}) \ , \tag{3}$$

Probability distribution $Pr(\omega_{t_0:t_1}|\pi_{t_0:t_1}, \iota_{t_0})$ is useful when defining the expected value to be gained by following separable policy $\pi_{t_0:t_1}$ from step t_0 onward. Overall, separable policy $\pi_{t_0:t_1}$ yields expected value given by:

$$\boldsymbol{E}\left[\sum_{t=t_0}^{t_1} r^{a_t}(s_t) \mid \pi_{t_0:t_1}, \iota_{t_0}\right] \quad = \quad \sum_{\omega_{t_0:t_1}} Pr(\omega_{t_0:t_1}|\pi_{t_0:t_1}, \iota_{t_0}) \left[\sum_{t=t_0}^{t_1} r^{a_t}(s_t)\right], \tag{4}$$

where $\boldsymbol{E}$ denotes the expectation with respect to $\pi_{t_0:t_1}$ and ι_{t_0}.

2.2.2 Optimality Criteria

A separable policy induces a distribution over sample paths which in turn induces a distribution over the sequences of rewards the agents receive. The goal of the centralized coordinator agent is to select a separable policy, which yields the largest reward sequence.

In this paper, we restrict our attention to finite-horizon cases where the optimality criterion is to maximize the expected sum of rewards. This quantity is usually referred to as the *return*.

Definition 10. *Let $\pi_{t:T-1}$ be a separable policy with respect to M. The **value function** $V_{M,\pi_{t:T-1}}$ denotes the expected cumulated rewards obtained if agents execute $\pi_{t:T-1}$ from runtime step t onward starting at information state ι_t:*

$$V_{M,\pi_{t:T-1}}(\iota_t) \stackrel{\text{def}}{=} \boldsymbol{E}\left[\sum_{k=t}^{T-1} r^{a_k}(s_k) \mid \pi_{t:T-1}, \iota_t\right]. \tag{5}$$

Value function $V_{M,\pi_{t:T-1}}$ refers to as a t-value function. This value function further satisfies the following recursion:$V_{M,\pi_{t:T-1}}(\iota_t) = V_{M,d_t}(\iota_t) + V_{M,\pi_{t+1:T-1}}(\iota_t, d_t)$. Note that quantity $V_{M,d_t}(\iota_t)$ is the immediate return of executing decision rule d_t at information state ι_t, and quantity $V_{M,\pi_{t+1:T-1}}(\iota_t, d_t)$ describes the future return of executing policy $\pi_{t+1:T-1}$ from runtime step $t+1$ onward starting at information state (ι_t, d_t). This recursion implies an efficient procedure for computing t-value functions. It is at the core of the Bellman's principle of optimality (Bellman, 1957).

2.2.3 Optimality Equations

The standard definitions of optimality equations in a T-step Dec-POMDP follows.

Definition 11. *Let $\Pi_{t:T-1}$ be the separable policy set with respect to M. For all $t \in \{0, 1, \ldots, T-1\}$, the **optimal t-value function** $V^*_{M,t}(\iota_t)$ at information state ι_t is $V^*_{M,t}(\iota_t) \stackrel{\text{def}}{=} \max_{\pi \in \Pi_{t:T-1}} V_{M,\pi}(\iota_t)$.*

Lemma 1. *Let D_t be the separable decision rule set with respect to M. For all $t \in \{0, 1, \ldots, T-1\}$, the **optimality equations** (or Bellman equations) at information state ι_t are:*

$$V^*_{M,t}(\iota_t) \stackrel{\text{def}}{=} \max_{d_t \in D_t} \left(V_{M,d_t}(\iota_t) + V^*_{M,t+1}(\iota_t, d_t)\right). \tag{6}$$

For $t = T$, we add a boundary condition $V^*_{M,T}(\cdot) \stackrel{\text{def}}{=} 0$.

Note that the optimal t-value function is written in terms of the optimal $(t+1)$-value function. An optimal separable policy can be directly extracted from the optimal value functions. Suppose $(V^*_{M,t})_{t \in \{0,1,\ldots,T-1\}}$ are solutions of the optimality equations (6) subject to the boundary condition, then it is clear that an optimal separable policy $\pi^*_{0:T-1} = (d^*_t)_{t \in \{0,1,\ldots,T-1\}}$ satisfies (Puterman, 1994; Bellman, 1957):

$$d^*_t \in \arg\max_{d_t \in D_t} \left(V_{M,d_t}(\iota_t) + V^*_{M,t+1}(\iota_t, d_t)\right). \tag{7}$$

This property implies that an optimal separable policy is found by first solving the optimality equations, and then for each runtime step choosing a separable decision rule that attains the maximum on the right hand side of (7) for $t \in \{0, 1, \ldots, T-1\}$.

3 Solving Dec-POMDPs as Continuous-State MDPs

Significant progress has been made in the size of problems solved as fully and partially observable Markov decision processes (MDPs and POMDPs). One reason for progress in MDPs has been the use of approximate dynamic programming and function approximation (Tsitsiklis and van Roy, 1996; de Farias and

Roy, 2003; Powell, 2007) to represent the state of the system and value functions more concisely. For POMDPs, efficient algorithms have been developed by recasting problems as belief MDPs that utilize probability distributions over states of the system, namely *belief states* (Smallwood and Sondik, 1973). This belief MDP is a continuous-state MDP with a piecewise linear and convex value function, allowing algorithms for POMDPs to scale to large problems while sometimes retaining performance bounds (Smith and Simmons, 2004; Shani et al., 2013). To take advantage of the advances in solvers for POMDPs and MDPs, we solve a Dec-POMDP by recasting it as a continuous-state MDP.

Section 3.1 shows that the state space of our continuous-state MDP consists of all reachable probability distributions over states of the system and histories of the agents (which we term *occupancy states*), and the action space consists of all separable decision rules mapping histories to actions. There are two fundamental results in this section.

The primary result is a demonstration that the occupancy state is sufficient for optimal planning in Dec-POMDPs. Then, we show in Section 3.1.4 that the optimal value functions are piecewise linear and convex functions of the occupancy states. Consequently, POMDP algorithms can for the first time be applied directly to Dec-POMDPs.

However, the curse of dimensionality significantly limits the scalability of POMDP algorithms. We further present in Section 3.3 standard classical planning algorithms including A* and LRTA*, but none of these algorithms overcome the curses of dimensionality. This highlights the necessity for compact representations of occupancy states, value functions and separable decision rules, as discussed in Sections 4.

3.1 Information State

At every plan-time step, the centralized coordinator agent selects the best separable decision rule for the current information state. We rank separable decision rules based on the expected value to be gained by executing a separable decision rule starting at the current information state. In calculating these expected values, we need to compute the probability of all possible sample paths conditional on the current information state. As the number of sample paths grows exponentially with increasing time steps, computing these conditional probabilities quickly becomes prohibitive. Given that this operation occurs at every time step, we suggest an alternative representation of the information state that can maintain these conditional probabilities over time steps.

3.1.1 Occupancy State

We introduce the concept of occupancy state as an alternative representation of the information state, that is able to maintain the conditional probabilities necessary for estimating the expected value to be gained by executing a separable decision rule starting at a given information state.

Definition 12. *The t-th* ***occupancy state*** *is defined as the posterior probability distribution of being in state s_t and joint history θ_t given information state ι_t,* i.e., $\eta_t(s_t, \theta_t) \stackrel{\text{def}}{=} Pr(s_t, \theta_t | \iota_t)$, $\forall t \in \{0, 1, \ldots, T\}$. *We denote $\triangle_t$ the t^{th}-step occupancy simplex, that is, the set of all possible t-th occupancy states.*

The occupancy state represents a predictive model about states the system may end up in and joint histories agents may experience given an information state. In other words, we only maintain state and joint history pairs that are reachable from under the information state.

A related concept is *the occupancy history* that is a probability distribution over the initial belief and joint histories: $\eta_t(\theta_t) \stackrel{\text{def}}{=} Pr(b_0, \theta_t | \iota_t)$. We use η_t to denotes occupancy histories by abuse of notation. Occupancy histories maintain only reachable joint histories, yet they preserve all the information we have

in the occupancy states:

$$\begin{aligned}\eta_t(s_t,\theta_t) &\stackrel{\text{def}}{=} Pr(s_t,\theta_t|\iota_t),\\ &= Pr(b_0,\theta_t|\iota_t)\cdot Pr(s_t|b_0,\theta_t), && \text{(conditional probability)}\\ &= \eta_t(\theta_t)\cdot b_t(s_t). && \text{(Definition 7)}\end{aligned}$$

Quantity b_t describes the belief about the state of the system conditional on joint history θ_t and initial belief state b_0. We will be using both concepts interchangeably.

It will be proved useful to reason about the local information an agent has about the state of the process. To this end, we introduce the notion of *local occupancy states* of agent $i \in I$. Intuitively, a local occupancy state of agent $i \in I$ is the projection of an occupancy state onto histories of the other agents and one history of agent $i \in I$. More formally, local occupancy state $\eta_t(\theta^i)$ from occupancy state η_t and history θ^i is defined at state s and history θ^j by: $\eta_t(\theta^i)(s,\theta^j) \stackrel{\text{def}}{=} \eta_t(s,\theta^i,\theta^j)$. It is worth noticing that local occupancy states are not probability distributions, *i.e.,* they do not necessary sum to one. The following demonstrates the sufficiency of the occupancy state with respect to the information state for optimal decision making in Dec-POMDPs.

3.1.2 Plan-Time Sufficient Statistic

We demonstrate that the occupancy state is a sufficient statistic at the plan-time, *i.e.,* a plan-time sufficient statistic. That is, given the occupancy state, no additional data about the current information state would provide any further information about the current state of the process at plan-time. The remainder of this section states a rather important theorem and provides a complete proof.

Theorem 1. *Let θ_t be a t-step joint history, for all time step $t \in \{0,1,\ldots,T\}$. Then*
*(a) Occupancy state $\eta_t = (Pr(s,\theta_t|\iota_t))_{s\in S,\theta_t\in\Theta_t}$ is a **plan-time sufficient statistic** of ι_t.*
(b) Occupancy state η_{t+1} depends on current occupancy state η_t and separable decision rule d_t:

$$\eta_{t+1}(s',(\theta_t,a_t,z_{t+1})) = \mathbf{1}_{\{a_t\}}(d_t(\theta_t))\sum_{s\in S}\eta_t(s,\theta_t)\cdot p^{a_t,z_{t+1}}(s,s'),$$

for any arbitrary $s' \in S$, $a_t \in A$, $z_{t+1} \in Z$ and $\theta_t \in \Theta_t$.

Proof. Part (a) follows directly from the definition of separable decision rules and the evaluation of separable decision rules. To demonstrate part (b), we need to show that the occupancy states describe a process that is Markov. In demonstrating this we also derive a procedure for updating the occupancy states. Let ι_t be our information state prior to plan-time t plus additional information that a particular separable decision rule was recorded. By definition 12, we make explicit the relation between the occupancy state and the information state as follows: for any arbitrary state s_t and joint history θ_t,

$$\eta_t(s_t,\theta_t) \stackrel{\text{def}}{=} Pr(s_t,\theta_t|\iota_t). \tag{8}$$

The substitution of (9) into (8) yields

$$\eta_t(s_t,\theta_t) = Pr(s_t,\theta_t|\iota_{t-1},d_{t-1}). \tag{9}$$

The expansion of the right-hand side of (9) over all states of the system at the end of plan-time $t-1$ produces

$$\eta_t(s_t,\theta_t) = \sum_{s_{t-1}\in S} Pr(s_{t-1},s_t,\theta_t|\iota_{t-1},d_{t-1}). \tag{10}$$

The expansion of the joint probability in (10) as the product of conditional probabilities results in

$$\eta_t(s_t,\theta_t) = \sum_{s_{t-1}\in S} Pr(a_{t-1}|\theta_{t-1},d_{t-1})\cdot Pr(s_t,z_t|s_{t-1},\theta_{t-1},\iota_{t-1},d_{t-1})\cdot Pr(s_{t-1},\theta_{t-1}|\iota_{t-1},d_{t-1}), \tag{11}$$

The first factor denotes the joint action a_{t-1} separable decision rule d_{t-1} prescribes at θ_{t-1}, since we assume throughout this paper separable decision rules are deterministic $Pr(a_{t-1}|\theta_{t-1}, d_{t-1}) \in \{0, 1\}$. In fact, $Pr(a_{t-1}|\theta_{t-1}, d_{t-1}) = 1$ if $\pi_{t-1}(\theta_{t-1}) = a_{t-1}$, otherwise $Pr(a_{t-1}|\theta_{t-1}, d_{t-1}) = 0$. So, $Pr(a_{t-1}|\theta_{t-1}, d_{t-1}) = \mathbf{1}_{\{a_{t-1}\}}(d_{t-1}(\theta_{t-1}))$, where $\mathbf{1}_F$ is an indicator function.

The second factor on the right-hand side of (11) is the transition probability:

$$\eta_t(s_t, \theta_t) \quad = \quad \mathbf{1}_{\{a_{t-1}\}}(d_{t-1}(\theta_{t-1})) \sum_{s_{t-1} \in S} p^{a_{t-1}, z_t}(s_{t-1}, s_t) \cdot Pr(s_{t-1}, \theta_{t-1}|\iota_{t-1}, d_{t-1}). \tag{12}$$

The last factor defines the prior occupancy state η_{t-1} at state s_{t-1} and joint history θ_{t-1}, which does not depend on the current separable decision rule d_{t-1}. Overall (10) becomes

$$\eta_t(s_t, \theta_t) \quad = \quad \mathbf{1}_{\{a_{t-1}\}}(d_{t-1}(\theta_{t-1})) \sum_{s_{t-1} \in S} p^{a_{t-1}, z_t}(s_{t-1}, s_t) \cdot \eta_{t-1}(s_{t-1}, \theta_{t-1}). \tag{13}$$

The important feature of (13) is that the calculation of the occupancy state after plan-time t requires only the occupancy state of the previous plan-time $t - 1$ and the current separable decision rule. Thus, the occupancy state summarizes all information gained prior to plan-time t and represents a sufficient statistic for the complete history of the planning process. In fact, (13) describes the possible transitions for a continuous-state Markov decision process in which states are occupancy states and actions are separable decision rules. For this planning process, the transitions are deterministic but the state space is continuous. □

3.1.3 Occupancy Markov Decision Processes

With Theorem 1 as a background, one can select a separable decision rule based on the current occupancy state instead of the information state. To make the relation between occupancy states and separable decision rules clear, consider the Markov decision process described by the occupancy states; we call it an *occupancy Markov decision process*.

Definition 13. *Let $\widehat{M} \equiv (\triangle, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P}, b_0, T)$ be the **occupancy Markov decision process** with respect to decentralized partially observable Markov decision process M, where:*

- *$\triangle = \cup_{t \in \{0,1,\ldots,T\}} \triangle_t$ is the occupancy simplex, where $\eta_0 = b_0$ is the initial occupancy state.*
- *$\boldsymbol{A} = \cup_{t \in \{0,1,\ldots,T\}} D_t$ is the separable decision rule set.*
- *$\boldsymbol{R}\colon \triangle \times \boldsymbol{A} \mapsto \mathbb{R}$ is a reward function: the reward at (η_t, d_t) is $\boldsymbol{R}(\eta_t, d_t) \stackrel{\text{def}}{=} V_{M,d_t}(\eta_t)$.*
- *$\boldsymbol{P}\colon \triangle \times \boldsymbol{A} \mapsto \triangle$ is a transition rule: next occupancy state $\eta_{t+1} \stackrel{\text{def}}{=} \boldsymbol{P}(\eta_t, d_t)$ given (η_t, d_t).*
- *T denotes the planning horizon.*

The following property is an immediate consequence of Theorem 1.

Corollary 1. *Let $\widehat{M}$ be the occupancy Markov decision process with respect to M. Then, optimal value functions of M are solutions of the optimality equations of $\widehat{M}$: for $t \in \{0, 1, \cdots, T-1\}$,*

$$V^*_{M,t}(\eta_t) \quad \stackrel{\text{def}}{=} \quad \max_{d_t \in \boldsymbol{A}} \left(\boldsymbol{R}(\eta_t, d_t) + V^*_{M,t+1}(\boldsymbol{P}(\eta_t, d_t)) \right). \tag{14}$$

For $t = T$, we add a boundary condition $V^*_{M,T}(\cdot) \stackrel{\text{def}}{=} 0$. Furthermore, the occupancy MDP $\widehat{M}$ is such that an optimal separable policy for it, together with the correct estimation of the occupancy states, will give rise to an optimal behavior for the original Dec-POMDP M.

Proof. The result is obviously true when $t = T$. Suppose now that the optimality equations (14) provide the optimal separable decision rules for $t + 1, t + 2, \cdots, T$ with respect to M. Then by using optimality equations (6), Theorem 1 and the induction hypothesis, we obtain:

$$\begin{aligned} V^*_{M,t}(\iota_t) \quad &= \quad \max_{d_t \in \boldsymbol{A}} \left(V_{M,d_t}(\iota_t) + V^*_{M,t+1}(\iota_t, d_t) \right), && \text{(optimality equations (6))} \\ &= \quad \max_{d_t \in \boldsymbol{A}} \left(\boldsymbol{R}(\eta_t, d_t) + V^*_{M,t+1}(\boldsymbol{P}(\eta_t, d_t)) \right). && \text{(Definition 13 and Theorem 1)} \end{aligned}$$

This proves the desired result. □

3.1.4 Piecewise-Linearity and Convexity

We are now ready to present one of the main results of this paper, that is, the piecewise-linearity and convexity of value functions solution of the optimality equations (14). To see where this piecewise-linear and convex structure comes from, imagine selecting a particular separable policy and evaluating its return starting from different occupancy states. Once a separable policy, a joint history and an initial state are specified, the future return is an independent function of the occupancy states. It is because of this conditional independence that the optimal value functions are linear in any arbitrary occupancy state.

Formally, let $\Psi(\pi, \cdot)$ be a mapping that specifies separable policy tree $\Psi(\pi, \theta_t)$ to be used given joint history θ_t and separable policy π. Define β_t^π to be a length-$n|\Theta_t|$ vector such that $\beta_t^\pi(\theta_t, s_t)$ is the expected value $V_{M,\Psi(\pi,\theta_t)}(s_t)$ of executing separable policy tree $\Psi(\pi, \theta_t)$ starting from state $s_t \in S$. Then, the value of executing separable policy π starting from occupancy state η_t is

$$V_{M,\pi}(\eta_t) \quad = \quad \sum_{s_t \in S} \sum_{\theta_t \in \Theta_t} \eta_t(s_t, \theta_t) \cdot \beta_t^\pi(s_t, \theta_t).$$

It will be useful, in the following exposition, to express this more compactly, *i.e.,* $V_{M,\pi}(\eta_t) = \langle \eta_t, \beta_t^\pi \rangle$. To build an optimal separable policy, we execute different separable policies from different initial occupancy states. Then, $V_{M,t}^*(\eta_t) = \max_{\pi \in \Pi_{t:T-1}} \langle \eta_t, \beta_t^\pi \rangle$. This definition of the optimal value function leads us to some important geometric insights into its form. We note that each separable policy induces function β^π that is linear in η_t, and $V_{M,t}^*$ is the *upper envelope* of this collection of functions. So, $V_{M,t}^*$ is a piecewise-linear and convex function of occupancy states.

Theorem 2. *Optimal value functions $(V_{M,t}^*)_{t \in \{0,1,\cdots,T\}}$ are piecewise-linear and convex functions of the occupancy states. Thus, for all $t \in \{0, 1, \cdots, T-1\}$, there exists a finite set of length-$n|\Theta_t|$ vectors Λ_t such that, for any arbitrary occupancy state $\eta_t \in \triangle$, we have:*

$$V_{M,t}^*(\eta_t) \quad = \quad \max_{\beta_t \in \Lambda_t} \langle \eta_t, \beta_t \rangle . \tag{15}$$

Proof. We show that (15) holds by induction. Since $V_{M,T}^*(\eta_T) = 0$ for all $\eta_T \in \triangle$, we have that $V_{M,T}^*(\eta_T) = \max_{\beta_T \in \Lambda_T} \langle \eta_T, \beta_T \rangle$, where $\beta_T(\cdot) = 0$ and $\Lambda_T = \{\beta_T\}$. Hence the property holds for $k = T$. Assume that the property holds for $k \geq t+1$, that is, $V_{M,k}^*(\eta_k) = \max_{\beta_k \in \Lambda_k} \langle \eta_k, \beta_k \rangle$. Now we want to prove the property for $k = t$. To this end, we first note that the immediate reward $\boldsymbol{R}(\eta_t, d_t)$ can be rewritten as a linear combination $\langle \eta_t, \beta_t^{d_t} \rangle$, where $\beta_t^{d_t}(s, \theta_t) \stackrel{\text{def}}{=} r^{d_t(\theta_t)}(s)$ denotes the immediate reward to be gained by executing d_t starting from state $s \in S$ and joint history $\theta_t \in \Theta_t$. Next, we note that the future reward $V_{M,t+1}^*(\boldsymbol{P}(\eta_t, d_t))$ can in turn be rewritten as the maximum over linear combinations $\max_{\beta_{t+1} \in \Lambda_{t+1}} \langle \eta_t p^{d_t}, \beta_{t+1} \rangle$, where p^{d_t} is a $(n \times |\Theta_t|) \times (n \times |\Theta_{t+1}|)$ transition matrix that transforms any t-th occupancy state into a $(t+1)$-th occupancy state. For any arbitrary states $s, s' \in S$, joint history $\theta_t \in \Theta_t$ and joint observation $z \in Z$, we have that $p^{d_t}(s, \theta_t; s', \theta_t, d_t(\theta_t), z) \stackrel{\text{def}}{=} p^{d_t(\theta_t), z}(s, s')$. With these linear transformations as a background, we successively obtain:

$$\begin{array}{rcll} V_{M,t}^*(\eta_t) & \stackrel{\text{def}}{=} & \max_{d_t \in A} \left(\boldsymbol{R}(\eta_t, d_t) + V_{M,t+1}(\boldsymbol{P}(\eta_t, d_t)) \right), & \text{(Corollary 1)} \\ & = & \max_{d_t \in A} \left(\langle \eta_t, \beta_t^{d_t} \rangle + \max_{\beta_{t+1} \in \Lambda_{t+1}} \langle \boldsymbol{P}(\eta_t, d_t), \beta_{t+1} \rangle \right), & \text{(Inductive Hypothesis)} \\ & = & \max_{d_t \in A} \max_{\beta_{t+1} \in \Lambda_{t+1}} \left(\langle \eta_t, \beta_t^{d_t} \rangle + \left\langle \eta_t p^{d_t}, \beta_{t+1} \right\rangle \right). & \text{(Rearranging Terms)} \end{array} \tag{16}$$

If we denote $(p^{d_t})^\top$ the transpose of p^{d_t}, (16) becomes

$$V_{M,t}^*(\eta_t) \quad = \quad \max_{d_t \in A} \max_{\beta_{t+1} \in \Lambda_{t+1}} \left(\langle \eta_t, \beta_t^{d_t} + \beta_{t+1} (p^{d_t})^\top \rangle \right).$$

Finally, if we let Λ_t be the set of all length-$n|\Theta_t|$ vectors $\beta_t \stackrel{\text{def}}{=} \beta_t^{d_t} + \beta_{t+1}(p^{d_t})^\top$ for all separable decision rules $d_t \in D_t$ and all vectors $\beta_{t+1} \in \Lambda_{t+1}$, then $V_{M,t}^*(\eta_t) = \max_{\beta_t \in \Lambda_t} \langle \eta_t, \beta_t \rangle$. □

3.2 Solving Occupancy MDPs as Belief MDPs

This section presents a basic approach to solving occupancy MDPs while exploiting the piecewise-linearity and convexity property of the optimal value functions.

We build upon the traditional finite-horizon POMDP decision theory (Smallwood and Sondik, 1973; Kaelbling et al., 1998; Pineau et al., 2006). The problem we confronted with POMDPs is to find a policy for selecting actions to be executed at each runtime step that maximizes some measure of performance. The key insight in finite-horizon POMDPs is that the optimal value functions are piecewise-linear and convex functions of belief states. It is possible to calculate piecewise-linear and convex value functions for the entire belief simplex through exact iterative updates (Smallwood and Sondik, 1973; Kaelbling et al., 1998; Zhang and Zhang, 2001). But the size of the value functions generated by exact updates is exponential in the horizon of the problem. As such, the exact computation of the value functions does not scale beyond small toy problems.

Alternatively, the value functions can be iteratively improved using update operations over specific belief states, known as point-based updates (Pineau et al., 2006; Smith and Simmons, 2004; Dibangoye et al., 2009b; Shani et al., 2013). Instead of computing the optimal value functions over the entire belief simplex, we can compute the value functions only over a finite subset of belief states (Lovejoy, 1991). As a result, the update operation is polynomial and the complexity of the value functions are bounded by the number of belief states of interest. This method yields good policies if an optimal value function over a subset of belief states may generalize well other belief states. To collect good belief subsets, point-based algorithms collect only belief states that are reachable from the starting belief state b_0 (Pineau et al., 2006). Techniques of this family differ on the method used to collect the belief states as well as on the order over which the value at those belief states are updated.

Interestingly, point-based approaches apply in any continuous-state MDP where the optimal value functions are piecewise-linear and convex function of states, *e.g.,* occupancy MDPs. Of this family, *heuristic search value iteration* (HSVI) algorithm terminates with an optimal solution (Smith and Simmons, 2004). In the remainder of this section, we show how to solve occupancy MDPs using point-based methods, especially using the HSVI algorithm. Section 3.2.1 describes alternative representations for piecewise-linear and convex value functions. Next, we discuss in Section 3.4 an efficient implementation of the greedy separable decision rule selection. Finally, we present in Section 3.2.2 HSVI for occupancy MDPs.

3.2.1 Standard PWLC Value Function Representations

We distinguish between two principal ways to represent piecewise-linear and convex value functions: *vector sets* and *point sets*. Figure 3 depicts these value function representations.

As illustrated in Figure 3(a), the vector set Λ_t denotes a set of length-$n|\Theta_t|$ vectors, and each vector β_t^π relies on a single separable policy π, so that the value at any arbitrary state s and joint history θ_t is given by $\beta_t^\pi(s,\theta) = V_{\Psi(\pi,\theta_t)}(s)$. The upper surface of vector set Λ_t is particularly well suited to representing a lower bound $\underline{\upsilon}_{M,t}$ of the optimal value function. As the algorithms proceed, the lower bound gets closer to the optimal value function $V^*_{M,t}$, and yields new vectors with values higher than that of previous vectors. Hence, the max operator in (15) guarantees the lower bound improved over regions of the state space where the new vectors dominate the previous vectors.

Unlike the vector set, the point set is more suitable for representing the upper-bound value functions. As illustrated in Figure 3(b), a point set Υ_t is a set of visited occupancy states along with their upper-bound values. The convex hull of point set Υ_t forms a t-step upper bound $\bar{\upsilon}_{M,t}$ of the optimal value function $V^*_{M,t}$. Then, the value of states that are not maintained in this point set are interpolated using a linear interpolation method (Hauskrecht, 2000; Smith, 2007). The linear interpolation method uses two different types of points: *corner* and *non-corner points*. The points that consist of degenerate occupancy states refer to

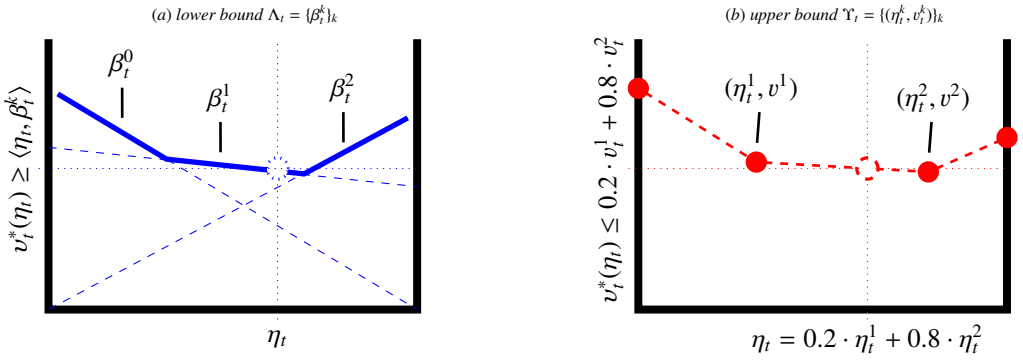


Figure 3: Graph (*a*) shows a set of hyperplanes (or vectors) β_t^0, β_t^1 and β_t^2 each of which relies one specific separable policy. The upper surface of the hyperplanes corresponds to a lower bound on the t-step optimal value function. Graph (*b*) illustrates a set of mappings from occupancy states to values including (η_t^1, v_t^1) and (η_t^2, v_t^2). The convex hull of this point set constitutes an upper bound on the t-step optimal value function.

as corner points, the other refer to as non corner points. Unfortunately, the complexity of this linear interpolation method increases with increasing points in Υ_t, thereby making this approach unfeasible in many domains. Alternative approaches to the linear interpolation method have been introduced in the literature (Hauskrecht, 2000; Armstrong-Crews and Veloso, 2008). Of this family, the *sawtooth* Algorithm 1 has demonstrated impressive performances on a number of large domains.

Algorithm 1: Sawtooth Projection

function SAWTOOTH(η_t, Υ_t)
  $y^0 \leftarrow \sum_{s \in S} \sum_{\theta \in \Theta_t} \eta_t(s, \theta) \cdot g^0(s, \theta)$
  forall the $l \in \{1, 2, \cdots, |\Upsilon_t|\}$ **do**
    $v^0 \leftarrow \sum_{s \in S} \sum_{\theta \in \Theta_t} \eta_t^l(s, \theta) \cdot g^0(s, \theta)$
    $\xi^l \leftarrow \min\{\eta_t(s, \theta) / \eta_t^l(s, \theta) | \forall s, \forall \theta, \eta_t^l(s, \theta) > 0\}$
    $y^l \leftarrow y^0 + (v_t^l - v^0) \cdot \xi^l$
  return $\min_l y^l$

Intuitively, sawtooth Algorithm 1 approximates the linear interpolation method, using a formula based on a single non-corner point (η_t^l, v_t^l) and a mapping g^0 from degenerate occupancy states to upper-bound values (corner points): for any occupancy state η_t, the upper-bound value satisfies:

$$\bar{v}_{M,t}(\eta_t) \quad = \quad \min_{l \in \{1, \ldots, |\Upsilon_t|\}} g^0(\eta_t) + (v_t^l - g^0(\eta_t^l)) \cdot \xi(\eta_t, \eta_t^l), \tag{17}$$

where $\xi(\eta_t, \eta_t^l) = \min\{\eta_t(s, \theta) / \eta_t^l(s, \theta) | \forall s, \forall \theta, \eta_t^l(s, \theta) > 0\}$ represents the *interpolation coefficient* of occupancy state η_t^l with respect to η_t, and $g^0(\eta_t) = \langle \eta_t, g^0 \rangle$. It is worth noticing that lower interpolation coefficients lead to weaker upper-bound values.

3.2.2 Heuristic Search Value Iteration

The goal of this section is to provide the reader with a methodology on how to extend point-based solvers to occupancy MDPs. In particular, we restrict our attention to the extension of HSVI for occupancy

MDPs. Due to the deterministic nature of occupancy MDPs, some of the contributions of HSVI do not apply, *e.g.,* the heuristic based on state uncertainty that is used to guide the state exploration is discarded. When applied to occupancy MDPs, HSVI appears more like an extension of the *learning real-time A** (LRTA*) algorithm (Korf, 1990) that takes advantage of the piecewise-linearity and convexity of the optimal value functions.

HSVI requires two-sided bounds, upper $(\bar{v}_{M,t})_{t\in\{0,1,\cdots,T\}}$ and lower $(\underline{v}_{M,t})_{t\in\{0,1,\cdots,T\}}$ bounds on the optimal value functions. In occupancy MDPs, it creates trajectories of states based only on upper bounds[2]. Each such trajectory starts with the initial state. HSVI always executes a greedy action with respect to the upper bounds, and then selects the next state based on this greedy action. Once the trajectory is finished, the upper and lower bounds are updated at each state, in the reverse order. HSVI is outlined in Algorithm 2.

Algorithm 2: The heuristic search value iteration algorithm for occupancy MDPs

```
function HSVI(M, (υ_{M,t})_{t∈{0,1,···,T}}, (v̄_{M,t})_{t∈{0,1,···,T}})
    while ¬STOP(η_0) do EXPLORE(η_0).

function STOP(η_t)
    return v̄_{M,t}(η_t) = υ_{M,t}(η_t).

function EXPLORE(η_t)
    if ¬STOP(η_t) then
        d_t^* ← arg max_{d_t} R(η_t, d_t) + v̄_{M,t+1}(η_t, d_t).
        EXPLORE(P(η_t, d_t^*)).
        update v̄_{M,t} and υ_{M,t} at η_t.
```

The trajectories are interrupted once they have reached a state η_t such that $\bar{v}_{M,t}(\eta_t) = \underline{v}_{M,t}(\eta_t)$, since there is no reason to expand an occupancy state whose optimal value is provably known (because the bounds coincide). Finally, it is often useful to prune lower and upper bounds to maintain concise representations, by removing either dominated vectors or points, respectively (Pineau et al., 2006; Smith, 2007).

Theorem 3. *When applied to T-step occupancy MDPs, HSVI terminates in bounded time with bounds that have their optimal values along at least one optimal path.*

Proof. The proof follows directly from the completeness and the optimality of HSVI in T-step POMDPs (Smith, 2007), since a T-step occupancy MDP can be viewed as a deterministic T-step POMDP where states are occupancy states and actions are separable decision rules. □

Key in the theoretical guarantees of HSVI is how lower and upper bounds are represented and updated. On the one hand, we assumed vectors $\beta \in \Lambda_t$ in lower bounds are exhaustive *look-up tables*, that is, mappings that assign one value to each state and joint history pair, which — at least in principle — are always possible for small state spaces and short planning horizons. On the other hand, the order in which states are updated is another key component of HSVI that speeds up the convergence. As the value of a state depends on the value of its successors, updating the successors prior to updating the current state may accelerate the convergence. The update of a state requires the action selection, which proceeds by

[2]In belief MDPs, the heuristic search value iteration algorithm creates trajectories of states based on both upper and lower bounds to cope with the uncertainty of the model. In occupancy MDPs, the model is deterministic so we only use the upper bound to guide the search towards optimistic states.

the exhaustive enumeration of all actions $d_t \in D_t$, that is, mappings that assign one action for *every*[3] history. For small action spaces and hence for short planning horizons, the update is affordable.

For longer planning horizons, however, the exhaustive lookup-table and action representations and selections quickly become impractical. The problem is not only the memory required for high-dimensional look-up tables and actions, but the time and data structures required to fill them accurately. In other words, the key issue is that of the compact representation of all look-up tables, occupancy states and decision rules. Unfortunately, HSVI (Algorithm 2) does not circumvent the curse of dimensionality, although the focused behavior starting with an initial state can reduce the storage requirement.

3.3 Solving Occupancy MDPs as Heuristic Search

Algorithms discussed above are algorithms for solving Markov decision problems. Although they also apply in deterministic cases, many of their components are unnecessary for solving deterministic MDPs. A widely used framework for deterministic problem solving in artificial intelligence is heuristic search. In this section, we describe heuristic search algorithms for solving occupancy MDPs that can take advantage of both the deterministic nature of the problem and the piecewise-linear and convex shape of the optimal value functions.

Heuristic search algorithms apply to state-space search problems defined by a set of states, a set of actions that map states to states, an initial state, and a set of goal states. The problem we face with heuristic search is to find a sequence of actions that maps the initial state to one goal state and possibly optimizes some measure of performance, *e.g.,* maximizing the return (Hart et al., 1968; Pearl, 1984; Korf, 1990). Heuristic search algorithms involve a systematic use of domain-specific knowledge in the form of heuristic evaluation functions to focus the search and solve search problems potentially much faster than uninformed search algorithms. In demonstrating the applicability of classical heuristic search to solving occupancy MDPs, we begin with a brief description of standard A* (Hart et al., 1968) and LRTA* (Korf, 1990) algorithms. Then, we describe how to transform A* and LRTA* so that they can fully take advantage of the underlying structure in occupancy MDPs.

3.3.1 A* For Occupancy MDPs

A* (Algorithm 3) involves two lists, an OPEN list and a CLOSED list, to manage the systematic search of a path from the initial state to a goal state that yields the maximum return. Initially, the OPEN list contains the initial state, and the CLOSED list is empty. At each iteration, the algorithm expands the most promising state in the OPEN list, moves it to the CLOSED list, and inserts its successor states into the OPEN list. So, the CLOSED list always maintains states the algorithm expanded, that is, the algorithm inserted their successor states into the OPEN list; and the OPEN list always keeps track of states the algorithm generated, but did not yet expand. The algorithm terminates when the OPEN list contains a goal state.

The order in which A* expands states depends on the heuristic evaluation function, which is written at any arbitrary state as follows: $f(\eta_t) = g(\eta_t) + h(\eta_t)$, where $g(\eta_t)$ is the past cumulated reward of the best path from the initial state to state η_t, and quantity $h(\eta_t)$ denotes a static heuristic estimate of $V^*_{M,t}(\eta_t)$, the optimal return from state η_t to a goal state. The behavior of A* depends to a large extend on the heuristic $h(\eta_t)$ that guides the search. If $h(\eta_t)$ is *admissible*, that is, if it never underestimates $V^*_{M,t}(\eta_t)$, and if states are expanded in order of decreasing $f(\eta_t)$, then the first goal state selected for expansion has necessarily been reached through an optimal path from the initial state, *i.e.,* a path that maximizes the return.

Due to the deterministic nature of occupancy MDPs, A* naturally applies in our setting. Yet, A* — as presented so far — is unable to take advantage of the piecewise-linearity and convexity property of the optimal value functions. To supplement traditional static heuristic h, we use the point set representation

[3] We virtually need to define decision rules over all possible histories to fill exhaustive look-up tables accurately, although only a few histories are reachable with respectively to the current occupancy state.

Algorithm 3: The A* algorithm for occupancy MDPs

```
function A*(M, η_0, h)
    g(η_0) ← 0 and f(η_0) ← h(η_0)
    OPEN ← {η_0} and CLOSED ← ∅
    while OPEN ≠ ∅ do
        η ← EXPAND(f, OPEN)
        if η is a goal state then return f
        foreach d ∈ D do
            g ← g(η) + R(η, d)
            if P(η, d) ∉ CLOSED or g > g(P(η, d)) then
                g(P(η, d)) ← g
                f(P(η, d)) ← g + h(P(η, d))
                if P(η, d) ∉ OPEN then OPEN ← OPEN ∪ {P(η, d)}

function EXPAND(f, OPEN)
    return arg min_{η∈OPEN} f(η)
```

of the upper bounds $(\bar{v}_t)_{t\in\{0,1,\cdots,T\}}$ employed in HSVI together with the sawtooth interpolation. Every time the algorithm inserts a new state into the OPEN list, it also adds a new point into the point set, thereby improving the heuristic as the algorithm proceeds. Relative to the standard A*, this modified A* algorithm may generate less states before finding an optimal path. This is mainly because it uses a heuristic that gets closer to the optimal value function as the search proceeds, whereas the standard A* maintains a static heuristic. When the number of generated states is reasonably manageable, maintaining the sawtooth representation is affordable and can result in significant improvement over the standard A*. For larger numbers of generated states, however, maintaining the sawtooth representation quickly becomes intractable. And hence, the apparent gain in the number of generated states may not fully benefit in the total computational time required to find an optimal path.

3.3.2 LRTA* For Occupancy MDPs

For difficult search problems, A* may take too long to find an optimal solution path, and anytime algorithms that find a sequence of improving solutions and eventually converge to an optimal solution can be more useful. One example of this family is the *Learning Real-Time A** (LRTA*) algorithm (Korf, 1990). Real-time algorithms are originally designed so that actions are executed in real-time, but they can nonetheless be applied offline during a planning phase.

The LRTA* Algorithm 4 proceeds in a similar vein to HSVI, but remains significantly different. It only maintains an upper bound on the optimal value function, and updates states forward as the trajectories are generated. In other words, it updates states in the order in which they are generated. This results in small improvements of the upper bound but much faster state updates since we only need to consider the value of the immediate successor given the selected action. The LRTA* algorithm assumes the heuristic is similar to that of the A* algorithm. This heuristic is represented using a look-up table, that is, a mapping from states to reals. Initially, this look-up table maps each state to a value that overestimates the state's optimal value. In contrast to A*, this heuristic is dynamic. That is, as the algorithm proceeds, values in the look-up table get closer to the optimal values along at least one optimal path. It provably terminates after bounded time with an optimal solution path.

The order in which the states are expanded depends on the heuristic evaluation function, which is

Algorithm 4: The learning real-time A* for occupancy MDPs

function LRTA*$(M, (\bar{v}_{M,t})_{t\in\{0,1,\cdots,T\}})$
  $best \leftarrow$ **nil**
  while $\neg$STOP(η_0) **do** EXPAND (η_0)

function STOP(η_t)
  if η_t *is a goal state and* $h(\eta_t) > f(best)$ **then** $best \leftarrow \eta_t$ **return** $best \neq$ **nil** *and* $h(\eta_t) \leq f(best)$

function EXPAND (η_t)
  if $\neg$STOP(η_t) **then**
    $d_t^* \leftarrow \arg\max_{d_t \in D_t} \boldsymbol{R}(\eta_t, d_t) + h(\eta_t, d_t)$
    update h at $\boldsymbol{P}(\eta_t, d_t^*)$
    EXPAND$(\boldsymbol{P}(\eta_t, d_t^*))$

written as follows: $f(\eta_t) = g(\eta_t) + h(\eta_t)$, where $g(\eta_t)$ is the past cumulated reward of the current path from the initial state to state η_t; quantity $h(\eta_t)$ denotes a dynamic heuristic estimate of $V^*_{M,t}(\eta_t)$, the optimal return from state η_t to a goal state. If h is admissible, the LRTA* algorithm finds a sequence of improving solutions and eventually converges to an optimal solution. As soon as a state is selected for expansion, it tests whether that state is a goal state; if the state is a goal state, it updates the current best solution path, thus providing a sequence of improved lower bounds on the optimal solution path. Next, the LRTA* algorithm tests whether the f-value of each newly-expanded state is larger than the current lower bound. If not, the trajectory that leads to the current state starting from the initial state is stopped, and none of its successors is expanded. Doing so permits the LRTA* algorithm to avoid the expansion of states that cannot lead to an improved solution path, and alleviates the memory requirements. Not surprisingly, the LRTA* algorithm applies in occupancy MDPs, but does not allow the heuristic to generalize over the entire state space. To take advantage of the piecewise-linearity and convexity of the optimal value functions, the sawtooth representation complements the heuristic evaluation function. As such, the values for visited states can generalize to other states that have not been visited yet, which results in a significant speed up of the convergence. Compared to the HSVI algorithm, this modified LRTA* algorithm requires much less memory per state. Where the HSVI algorithm needs to maintains both lower and upper bounds for each visited state, the LRTA* algorithm requires only the upper bound, thus resulting in non-negligible memory savings.

The LRTA* algorithm and its family of real-time search algorithms have the property to eventually converge to an optimal solution path, by producing successively better solution paths, especially when the number of actions is reasonably manageable. The learning rate of these algorithms, however, is significantly slowed down for larger state and action spaces. As a result, the LRTA* algorithm variants we presented here can only solve occupancy MDPs with short planning horizons, since the number of states and actions grows exponentially with the planning horizon. This highlights the impetus for compact representations of all states, actions, and value functions.

3.4 Constraint-Based Bellman Backups

This section addresses a key impediment in the scalability of dynamic programming algorithms: the selection of the best action for a given state. This is a core operation in every solution methods for solving occupancy Markov decision processes. To better understand this, consider value the optimal functions solutions of the Bellman optimality equations.

The optimal value functions $(V^*_{M,t})_{t\in\{0,1,\ldots,T-1\}}$ are computed over occupancy states by solving the Bell-

man optimality equations: for all $t \in \{0, 1, \ldots, T-1\}$,

$$V^*_{M,t}(\eta_t) \quad = \quad \max_{d_t \in D_t} \left(\boldsymbol{R}(\eta_t, d_t) + V^*_{M,t+1}(\boldsymbol{P}(\eta_t, d_t))\right).$$

Each single equation entails finding the best separable decision rule for a given occupancy state and deciding which value function to follow next. Such operation, refers to as the *Bellman backup*, can be performed using a brute force enumerative approach. Unfortunately, since the number of possible separable decision rule $|D_t|$ grows exponentially as time goes on, the brute force approach quickly runs out of time. In this following, we replace the brute force approach by a constraint-based approach, which computes the best separable decision rule while circumventing the explicit enumeration of all possible separable decision rules. To this end, it will prove useful to introduce the weighted constraint satisfaction problem (WCSP). That is a model that has emerged as a standard manner to formalize and solve many different combinatorial optimization problems.

Definition 14. *A **weighted constraint satisfaction problem** (WCSP) refers to a tuple (V, X, C) where:*

- *$V = \{V_1, \ldots, V_K\}$ is the set of K domains.*
- *$X = \{X_1, \ldots, X_K\}$ is the set of K variables, taking values from their domains.*
- *$C = \{c\}$ is the set of reward functions used to declare preferences among possible solutions.*

Each reward function $c \in C$ is defined over a subset of variables, $var(c) \subseteq X$, called the scope of c. The objective function f is defined as the sum of all reward functions: $f(X) = \sum_{c \in C} c(X_{var(c)})$.

In our WCSPs, variables correctly assigned receive finite rewards that express their degree of preference (the higher value the better preference) and variables not correctly assigned receive reward $-\infty$. The goal is to find a complete assignment of values to variables that maximizes the objective function. Next, we present the constraint-based approaches we use to perform Bellman backups. We make a distinction between the constraint-based approach for the lower-bound value functions and that of the upper-bound value functions.

3.4.1 Constraint-Based Lower-Bound Bellman Backups

In this section, we consider the problem of selecting the best separable decision rule given an occupancy state and a lower-bound value function: the *lower-bound Bellman backup operator*. More precisely, we introduce a constraint-based implementation of this operator that exploits the piecewise-linearity and convexity of the lower-bound value function. A formal definition of the lower-bound Bellman backup operator T follows.

Let $(\underline{\upsilon}_{M,t})_{t \in \{0,1,\ldots,T-1\}}$ be a lower-bound value function, and $\Lambda = \{\beta^1, \beta^2, \ldots, \beta^K\}$ a set of vector values that represents $(\underline{\upsilon}_{M,t})_{t \in \{0,1,\ldots,T-1\}}$.

Definition 15. *The **lower-bound Bellman backup operator** $T : \underline{\upsilon}_{M,t} \mapsto T\underline{\upsilon}_{M,t}$ is given by: $\forall \eta_t \in \triangle_t$,*

$$T\underline{\upsilon}_{M,t}(\eta_t) \quad = \quad \max_{d_t \in D_t, k \in \{1,2,\ldots,K\}} \left[\boldsymbol{R}(\eta_t, d_t) + \langle \boldsymbol{P}(\eta_t, d_t), \beta^k \rangle\right]. \tag{18}$$

In introducing our constant-based approach, we need to define an intermediate operator called *the lower-bound greedy operator*.

Definition 16. *The operator $G : \underline{\upsilon}_{M,t} \mapsto G\underline{\upsilon}_{M,t}$ is referred to as **the lower-bound greedy operator** and given by: $\forall \eta_t \in \triangle_t$, $G\underline{\upsilon}_{M,t}(\eta_t) = \arg\max_{d_t \in D_t, k \in \{1,2,\ldots,K\}} \left[\boldsymbol{R}(\eta_t, d_t) + \langle \boldsymbol{P}(\eta_t, d_t), \beta^k \rangle\right]$.*

The lower-bound greedy operator selects the greedy separable decision rule and next-step vector value given an occupancy state and a lower-bound value function. There is a close relationship between the lower-bound Bellman backup operator and the lower-bound greedy operator. The former operator improves the value of the lower-bound value function at a given occupancy state using the later.

Lemma 2. *The lower-bound Bellman backup operator* $T : \underline{\upsilon}_{M,t} \mapsto T\underline{\upsilon}_{M,t}$ *is given by:* $\forall \eta_t \in \triangle_t$,

$$T\underline{\upsilon}_{M,t}(\eta_t) \quad = \quad \boldsymbol{R}(\eta_t, d_t^*) + \langle \boldsymbol{P}(\eta_t, d_t^*), \beta_{t+1}^* \rangle, \tag{19}$$

where $(d_t^*, \beta_{t+1}^*) = G\underline{\upsilon}_{M,t}(\eta_t)$.

Proof. The proof follows directly from both the piecewise-linearity and convexity property of the lower-bound value function and the definitions of the lower-bound Bellman backup operator. □

This lemma formalizes the relationship between both the lower-bound Bellman backup operator and the lower-bound greedy operator. We are now ready to state the weighted constraint satisfaction problem that implements the lower-bound greedy operator.

Theorem 4. *Let* $\Lambda_{t+1} = \{\beta^1, \beta^2, \ldots, \beta^K\}$ *be a representation of the* $(t+1)$*-th lower-bound value function* $\underline{\upsilon}_{M,t+1}$*, and* η_t *an occupancy state. Then,* $G\underline{\upsilon}_{M,t}(\eta_t)$ *is a solution of the weighted constraint satisfaction problem* (V, X, C) *where:*

- $V = \{V_\theta, \mathcal{K} | \forall \theta \in \Theta \colon \eta_t(\theta) > 0\}$ *is the set of domains, especially* $V_\theta = A$ *denotes the set of mappings from joint history* θ *to any joint action, and* $\mathcal{K} = \{1, 2, \ldots, K\}$ *denotes the indexes of vector values in* Λ_{t+1}.
- $X = \{X_\theta, X_\mathcal{K} | \forall \theta \in \Theta \colon \eta_t(\theta) > 0\}$ *is the set of variables, in particular* X_θ *takes values in domain* V_θ*, the set of mappings from joint history* θ *to actions; and* $X_\mathcal{K}$ *takes values in* $\mathcal{K}$.
- $C = \{c_\theta | \forall \theta \in \Theta \colon \eta_t(\theta) > 0\}$ *is the set of reward functions* c_θ*, such that:*

$$c_\theta(X_\theta \mapsto a, X_\mathcal{K} \mapsto k) \quad = \quad \sum_s r^a(s) \cdot \eta_t(s, \theta) + \sum_z \sum_s \eta_t(s, \theta) \cdot p^{a,z}(s, s') \cdot \beta^k(s, \theta, a, z).$$

Proof. To prove this result, we start with the standard Bellman backup operator given occupancy state η_t and lower bound Λ_{t+1}:

$$\begin{aligned}
G\underline{\upsilon}_{M,t}(\eta_t) &= \arg\max_{d_t \in D_t, k \in \mathcal{K}} \left[\boldsymbol{R}(\eta_t, d_t) + \langle \boldsymbol{P}(\eta_t, d_t), \beta^k \rangle \right], \\
&= \arg\max_{d_t \in D_t, k \in \mathcal{K}} \sum_\theta \mathbf{1}_{\{d_t(\theta)\}}(a) \left(\sum_s r^a(s) \cdot \eta_t(s, \theta) + \sum_z \sum_s \eta_t(s, \theta) \cdot p^{a,z}(s, s') \cdot \beta^k(s, \theta, a, z) \right), \\
&= \arg\max_{d_t \in D_t, k \in \mathcal{K}} \sum_\theta \mathbf{1}_{\{d_t(\theta)\}}(a) \cdot c_\theta(X_\theta \mapsto a, X_\mathcal{K} \mapsto k), \\
&= \arg\max_{d_t \in D_t, k \in \mathcal{K}} f(d_t, \beta^k).
\end{aligned}$$

Which ends the proof. □

The problem of efficiently solving weighted constraint satisfaction problems goes beyond the scope of this paper. It is well known that WCSPs are in NP (Dechter, 2003). Fortunately, this problem is not new, so we rely on exact algorithms from the literature. In fact, there is a large literature on methods to solving WCSPs, exact methods include bucket elimination (Dechter, 1999), branch-and-bound algorithms (de Givry et al., 2005; Cooper et al., 2010); approximate algorithms also exists (Dechter, 1997; Dechter and Rish, 2003).

3.4.2 Constraint-Based Upper-Bound Bellman Backups

In this section, we consider the problem of selecting the best separable decision rule given an occupancy state and an upper-bound value function: the *upper-bound Bellman backup operator*. More precisely, we introduce a constraint-based implementation of this operator that exploit the piecewise-linearity and convexity of the upper-bound value function. A formal definition of the upper-bound Bellman backup operator H follows.

Definition 17. *The **upper-bound Bellman backup operator** $H\colon \bar{v}_{M,t} \mapsto H\bar{v}_{M,t}$ is given by: $\forall \eta_t \in \triangle_t$,*

$$H\bar{v}_{M,t}(\eta_t) \quad = \quad \max_{d_t \in D_t} \left[\boldsymbol{R}(\eta_t, d_t) + \textsc{sawtooth}(\boldsymbol{P}(\eta_t, d_t), \Upsilon_{t+1}) \right]. \tag{20}$$

Next, we reformulate the upper-bound Bellman backup operator to make explicit the parameters and operators involved. To this end, we consider the point set Υ_{t+1}, the sawtooth representation of the $(t+1)$-th upper bound value function $\bar{v}_{M,t+1}$.

Lemma 3. *The upper-bound Bellman backup operator $H\colon \bar{v}_{M,t} \mapsto H\bar{v}_{M,t}$ is given by: $\forall \eta_t \in \triangle_t$,*

$$H\bar{v}_{M,t}(\eta_t) \quad = \quad \max_{d_t \in D_t} \left[\boldsymbol{R}(\eta_t, d_t) + g^0(\boldsymbol{P}(\eta_t, d_t)) + \min_{l \in \mathcal{L}} \max_{\theta\colon \eta^l(\theta) > 0} \frac{\boldsymbol{P}(\eta_t, d_t)(\theta)}{\eta^l(\theta)} (v^l - g^0(\eta^l)) \right], \tag{21}$$

where g^0 denotes the initial upper-bound value function and $\mathcal{L} = \{1, 2, \ldots, |\Upsilon_{t+1}|\}$.

Proof. The following equalities hold by definition of operator T and sawtooth interpolation method:

$$H\bar{v}_{M,t}(\eta_t) \quad = \quad \max_{d_t \in D_t} \left[\boldsymbol{R}(\eta_t, d_t) + \textsc{sawtooth}(\boldsymbol{P}(\eta_t, d_t), \Upsilon_{t+1}) \right], \tag{22}$$

$$\quad = \quad \max_{d_t \in D_t} \left[\boldsymbol{R}(\eta_t, d_t) + g^0(\boldsymbol{P}(\eta_t, d_t)) + \min_{l \in \mathcal{L}} (v^l - g^0(\eta^l)) \cdot \xi(\boldsymbol{P}(\eta_t, d_t), \eta^l) \right]. \tag{23}$$

Quantity $\xi(\boldsymbol{P}(\eta_t, d_t), \eta^l)$ denotes the interpolation coefficient, where:

$$\xi(\eta, \eta^l) \quad = \quad \min_{s,\theta\colon \eta^l(s,\theta) > 0} \frac{\eta(s,\theta)}{\eta^l(s,\theta)}, \tag{24}$$

$$\quad = \quad \min_{\theta\colon \eta^l(\theta) > 0} \frac{\eta(\theta)}{\eta^l(\theta)}. \tag{25}$$

The last equality holds since the probability of being in state $s \in S$ given occupancy state η and joint history θ depend only upon the probability of have experienced joint history θ, that is, $\eta(\theta)$. In other words, if two occupancy states η' and η are such that $\eta'(\theta) > 0$ and $\eta(\theta) > 0$, then $\forall s \in S$, then we have that $\eta'(s,\theta) = \eta(s,\theta)$. The following equality follows by multiplying both side of the last equality by $(v^l - g^0(\eta^l))$ since $v^l \leq g^0(\eta^l)$:

$$\xi(\eta, \eta^l) \cdot (v^l - g^0(\eta^l)) \quad = \quad \max_{\theta\colon \eta^l(\theta) > 0} \frac{\eta(\theta)}{\eta^l(\theta)} \cdot (v^l - g^0(\eta^l)). \tag{26}$$

We conclude the proof by replacing $\xi(\eta, \eta^l) \cdot (v^l - g^0(\eta^l))$ by $\max_{\theta\colon \eta^l(\theta) > 0} \frac{\eta(\theta)}{\eta^l(\theta)} \cdot (v^l - g^0(\eta^l))$ in (23):

$$H\bar{v}_{M,t}(\eta_t) \quad = \quad \max_{d_t \in D_t} \left[\boldsymbol{R}(\eta_t, d_t) + g^0(\boldsymbol{P}(\eta_t, d_t)) + \min_{l \in \mathcal{L}} \max_{\theta\colon \eta^l(\theta) > 0} \frac{\boldsymbol{P}(\eta_t, d_t)(\theta)}{\eta^l(\theta)} \cdot (v^l - g^0(\eta^l)) \right]. \tag{27}$$

Which ends the proof. □

This lemma reveals that the upper-bound Bellman backup operator is a non-standard Bellman operator, since it involves both max and min operators. The upper-bound Bellman backup is an operator according to which separable decision rules are ranked on the basis of their worst-case return with respect to points in the point set. That is, a Wald's maximin model (Wald, 1939). Our weighted constraint satisfaction problem formulation (Definition 14) involves only a max operator. Thus, there is no single weighted constraint satisfaction problem that can implement the upper-bound Bellman backup operator.

Wald's maximin model can be reformulated as a mixed-integer linear program. With this insight as a background, one can recast the upper-bound Bellman backup operator into a mixed-integer linear program: $\forall \eta_t \in \triangle_t$,

$$\begin{aligned} H\bar{v}_{M,t}(\eta_t) &= \max_{d_t \in D_t, u \in \mathbb{R}} u && (28)\\ \text{s.t.} \quad & u \leq W(\eta_t, d_t, p^l),\ \forall l \in \mathcal{L}, && (29) \end{aligned}$$

where $W(\eta_t, d_t, p^l) = \boldsymbol{R}(\eta_t, d_t) + \textsc{sawtooth}(\boldsymbol{P}(\eta_t, d_t), \{p^l\})$. Unfortunately, solving this mixed-integer linear program remains an open problem. Instead of solving this problem, we suggest to split this optimization problem into $|\mathcal{L}|$ mixed-integer linear programs, one for each point in the point set. The mixed-integer linear program above can be reformulated as follows:

$$\begin{aligned} H\bar{v}_{M,t}(\eta_t) &= \max_{\ell \in \mathcal{L}} \max_{d_t \in D_t} W(\eta_t, d_t, p^\ell) && (30)\\ \text{s.t.} \quad & W(\eta_t, d_t, p^\ell) \leq W(\eta_t, d_t, p^l),\ \forall l \in \mathcal{L}\backslash\{\ell\}. && (31) \end{aligned}$$

Let $H^\ell \colon H^\ell \mapsto H^\ell \bar{v}_{M,t}$ be the upper-bound Bellman operator for parameter ℓ, such that:

$$\begin{aligned} H^\ell \bar{v}_{M,t}(\eta_t) &= \max_{d_t \in D_t} W(\eta_t, d_t, p^\ell) && (32)\\ \text{s.t.} \quad & W(\eta_t, d_t, p^\ell) \leq W(\eta_t, d_t, p^l),\ \forall l \in \mathcal{L}\backslash\{\ell\}. && (33) \end{aligned}$$

Then, the upper-bound Bellman operator H can be rewritten as follows:

$$H\bar{v}_{M,t}(\eta_t) = \max_{\ell \in \mathcal{L}} H^\ell \bar{v}_{M,t}(\eta_t). \quad (34)$$

In other words, quantity $H\bar{v}_{M,t}(\eta_t)$ is the maximum outcome among $H^\ell \bar{v}_{M,t}(\eta_t)$ for all $\ell \in \mathcal{L}$. Next, we introduce a constraint-based approach to solving $H^\ell \bar{v}_{M,t}(\eta_t)$ for each parameter $\ell \in \mathcal{L}$.

It is worth noticing that optimization problems $H^\ell \bar{v}_{M,t}(\eta_t)$ are mixed-integer linear programs, but the unconstrained variant of this problem, Equation (32), is a weighted constraint satisfaction problem. Unfortunately, solutions of that WCSP need not be feasible solutions with respect to constraints Equation (33). To take into account those constraints, we propose an iterative procedure, which involves solving a WCSP and thereafter checking for feasibility. The procedure terminates once a feasible solution is found, we will demonstrate this solution is a feasible solution with the highest return. At each iteration, our procedure adds the previous non feasible solutions into the previous WCSP as a *nogood*[4]. The weighted constraint satisfaction problems of interest follow.

Theorem 5. *The best separable decision rule d_t^ℓ at occupancy state η_t relative to $p^\ell \equiv (\eta^\ell, v^\ell)$ is the solution of the weighted constraint satisfaction problem (V, X, C^ℓ) where:*

- *$V = \{V_\theta, V_\ell | \forall \theta \in \Theta \colon \eta_t(\theta) > 0\}$ is the set of domains, especially $V_\theta = A$ denotes the set of mappings from joint history θ to any joint action, and $V_\ell = \{\forall \theta' \in \Theta \colon \eta^\ell(\theta') > 0\}$.*
- *$X = \{X_\theta, X_\ell | \forall \theta \in \Theta \colon \eta_t(\theta) > 0\}$ is the set of variables, in particular X_θ takes values in domain V_θ, the set of mappings from joint history θ to actions, and X_ℓ takes values in V_ℓ.*

[4] A nogood is a constraint, which forbids the selection of certain solutions.

- $C^\ell = \{c_\theta, nogood | \forall \theta \in \Theta \colon \eta_t(\theta) > 0\}$ *is the set of reward functions, such that reward function* c_θ *is given by:*

$$c_\theta(X_\theta \mapsto a, X_\ell \mapsto \theta') \quad = \quad \sum_s r^a(s) \cdot \eta_t(s,\theta) + g^0(\boldsymbol{P}(\eta_t, \theta, a)) + \frac{\boldsymbol{P}(\eta_t, \theta, a)(\theta')}{\eta^\ell(\theta')} \cdot (v^\ell - g^0(\eta^\ell)),$$

and nogood constraint is given by: $nogood(sol \mapsto (d_t, \theta')) = -\infty$*. for non feasible solutions* (d_t, θ')*, otherwise* $nogood(sol \mapsto (d_t, \theta')) = 0$*. Notice that* $\boldsymbol{P}(\eta_t, \theta, a)$ *describes the probability of all joint histories conditional on occupancy state* η_t*, joint history* θ*, and joint action* a*.*

The objective function is defined as the sum all constraints $f^\ell(X) = \sum_\theta c_\theta(X_{var(c_\theta)}) + nogood(X)$*.*

Proof. To prove this result, we need to demonstrate the objective function of this weighted constraint satisfaction problem corresponds to the separable decision rule d_t (and joint history θ') which maximize $H^\ell \bar{v}_{M,t}(\eta_t)$. Define quantity $W(\eta_t, d_t, \theta', p^\ell)$ as follows:

$$W(\eta_t, d_t, \theta', p^\ell) \quad \stackrel{\text{def}}{=} \quad \boldsymbol{R}(\eta_t, d_t) + g^0(\boldsymbol{P}(\eta_t, d_t)) + \frac{\boldsymbol{P}(\eta_t, d_t)(\theta')}{\eta^\ell(\theta')} \cdot (v^\ell - g^0(\eta^\ell)).$$

Given that functions $\boldsymbol{R}$, $\boldsymbol{P}$ and g^0 are linear and depend only upon the history, one can write the following with no risk of loosing important information:

$$\begin{aligned} W(\eta_t, d_t, \theta', p^\ell) \quad &\stackrel{\text{def}}{=} \quad \sum_\theta \left[\sum_s r^{d_t(\theta)}(s) \cdot \eta_t(s,\theta) + g^0(\boldsymbol{P}(\eta_t, \theta, d_t(\theta))) + \frac{\boldsymbol{P}(\eta_t, \theta, d_t(\theta))(\theta')}{\eta^\ell(\theta')} \cdot (v^\ell - g^0(\eta^\ell)) \right], \\ &= \quad \sum_\theta c_\theta(X_\theta \mapsto d_t(\theta), X_\ell \mapsto \theta'). \end{aligned}$$

Thus, one can rewritten $H^\ell \bar{v}_{M,t}(\eta_t)$ as follows:

$$\begin{aligned} H^\ell \bar{v}_{M,t}(\eta_t) \quad &= \quad \max_{d_t \in D_t, \theta' \in V_\ell} W(\eta_t, d_t, \theta', p^\ell) + \text{nogood}(d_t, \theta'), \\ &= \quad \max_{d_t \in D_t, \theta' \in V_\ell} f^\ell(d_t, \theta'). \end{aligned}$$

where nogood penalizes the unfeasible solutions, and f^ℓ denotes the objective function of our weighted constraint satisfaction problem. This ends the proof. □

In the remainder of this paper, we focus on how equivalence relations among histories provide us with lossless dimension reduction models that transform (*a*) any occupancy state into a compact occupancy state; and (*b*) any exhaustive look-up table into a compact look-up table. We also discuss incremental algorithms for automatically constructing these dimension reduction models. Finally, we present a novel search algorithm that takes advantage of compact representations to solve occupancy MDPs.

4 Compact Representations

Section 3 demonstrates that the information states and value functions can be represented in a vector space, the occupancy simplex, with no loss of important information. Unfortunately, the dimension of the occupancy state and of the value functions are proportional to the size of the joint history set, which increases exponentially with increasing time — making standard dynamic programming and heuristic search algorithms impractical. In this section, we propose dimension reduction models, based on equivalence relations among histories, which enable lossless compact representations.

The remainder of this section is organized as follows. We first present, in Section 4.1, the concept of history equivalence relations and practical implications. Section 4.2 describes probabilistic history equivalence relations that make possible lossless clustering of histories and lossless compact representations of occupancy states and separable decision rules. Next, we present, in Section 4.4, value-based history equivalence relations that ease lossless clustering of histories and lossless compact representations of value functions. Throughout this section, we show algorithmic benefits and drawbacks of feature-based dimension reduction models we use, and compares them to the original spaces.

4.1 History Equivalence Relations

Solving occupancy MDPs involves mapping an action to each reachable history to so that the return is as larger as possible. Since the number of histories grows exponentially with time, it quickly becomes intractable to map an action to each reachable history. For this reason, we want to group together histories that are *equivalent*. Equivalence relations have to be defined in such a way that important properties are preserved, *i.e.*, the ability to eventually find an optimal solution. Equivalence relations between histories can alternatively serve as a means to enhance the value function generalization. That is, to extrapolate values from one history to another one.

Definition 18. *Let E be an equivalence relation over individual histories, where individual histories of distinct agents cannot be equivalent.*[5] *For all agents $i \in I$, let $[\theta^i]_E$ be the set of histories in Θ^i that are equivalent to history θ^i with respect to equivalence relation E. We also refer to $[\theta^i]_E$ as the **equivalence class** of θ^i with respect to E.*

An equivalence relation between individual histories induces an equivalence relation between joint histories. Specifically, for any joint history $\theta = (\theta^1, \theta^2, \cdots, \theta^{|I|})$, the *equivalence class* of θ with respect to E is given by $[\theta]_E \stackrel{\text{def}}{=} ([\theta^1]_E, [\theta^2]_E, \cdots, [\theta^{|I|}]_E)$. Informally, an equivalence relation partitions the joint-history space Θ into K disjoint subsets $(\Theta_k)_{k\in\{1,2,\cdots,K\}}$, each of which represents an equivalence class with respect to E.

Definition 19. *An **individual-history label** ℓ^i in $\Theta^i_{k^i}$ is the 1^{st} individual history in the lexicographic order of individual histories in $\Theta^i_{k^i}$ of agent $i \in I$. A **joint-history label** $\ell = (\ell^i)_{i\in I}$ in subset $\Theta_{k=(k^i)_{i\in I}}$ is the $|I|$-tuple of individual-history labels, one per agent. We denote $\mathscr{L}^i$ and $\mathscr{L}$ the individual-history and joint-history label sets, respectively.*

Given equivalence relation E, any joint history $\theta \in \Theta$ belongs to a single partition Θ_k for some $k \in \{1, 2, \cdots, K\}$. For the sake of simplicity, we replace the identity of partition Θ_k by that joint-history label $\ell \in \Theta_k$. We are now ready to define *the transformation rule of an equivalence relation*.

Definition 20. *A **transformation rule** of equivalence relation E is a function $\zeta_E \colon \Theta' \mapsto \mathscr{L}$, defined by $\zeta_E(\theta) = \ell$, where $\Theta' \subseteq \Theta$, $\ell \in [\theta]_E$ and $\forall \theta' \in [\theta]_E$ we have that $\ell \preceq_{\text{LEX}} \theta'$.*

Remark 1. *It is crucial to define transformation rule $\zeta_E \colon \Theta' \mapsto \mathscr{L}$ of equivalence relation E as a function over a subset of the entire joint history set Θ. This permits us to represent ζ_E in a compact manner, which would not be the case if we allowed ζ_E to be defined over the entire joint history set. The choice of Θ' will come naturally in the algorithm.*

An equivalence relation between individual histories yields an equivalence relation between joint histories, which in turn yields an equivalence relation between: individual and separable decision rules; individual and separable policies; occupancy states; and value functions. The purpose of this section is to provide an efficient way to represent these concepts using equivalence relations between individual

[5] This is equivalent to considering one equivalence relation per agent.

histories. Before proceeding any further, we start with a general definition of a compact representation of any function defined over the entire joint history set Θ. In general, compact representations are lossy. In this paper, we design them so as to preserve the ability to eventually find an optimal solution.

Definition 21. *Let E be an equivalence relation between individual histories, and $g\colon \Theta \mapsto V$ be a function mapping joint histories $\theta \in \Theta$ to values $v \in V$. A **compact representation** $\hat{g}$ of function g consists of a triplet (g_E, ζ_E, v_E). That is, a function $g_E\colon \mathscr{L} \mapsto V$, a transformation rule of equivalence relation E, such that, for any joint history θ denoted $\zeta_E\colon \Theta' \mapsto \mathscr{L}$, and a default value $v_E \in V$, such that: for any arbitrary joint history $\theta \in \Theta$,*

$$\hat{g}(\theta) \quad = \quad \begin{cases} g_E(\zeta_E(\theta)) & \text{if } \theta \in \Theta', \\ v_E & \text{otherwise.} \end{cases}$$

The relation between g and $\hat{g}$ is explicit whenever $\theta \in \Theta'$, that is $g(\theta) = \hat{g}(\theta)$.

This definition lays the foundation of the compact representations based on equivalence relations for all decision rules, policies, value functions, and occupancy states. The following makes explicit how to compute and represent (g_E, ζ_E, v_E) such that $\hat{g}$ approximates arbitrarily accurately well g. In the following Sections 4.1.1, 4.1.2, and 4.1.3, we assume we know E and thus ζ_E, which depends only upon E. And hence, the only unknown is g_E. In Sections 4.2 and 4.4, we relax these assumptions, and present different equivalence relations together with associated transformation rules.

4.1.1 Compact Occupancy States

In this section, we introduce the concept of *compact occupancy state*, as a means of reducing the overwhelming memory requirement of full occupancy states, while preserving the ability to eventually find an optimal solution.

Definition 22. *Let E be an equivalence relation between individual histories, and η be any arbitrary occupancy state. A **compact occupancy state** $\hat{\eta}$ of η is based on a pair (η_E, ζ_E), which consists of a distribution of probabilities over state and joint-history label pairs η_E and a transformation rule $\zeta_E\colon \Theta' \mapsto \mathscr{L}$ of E, such that: for all state $s \in S$ and joint history $\theta \in \Theta$,*

$$\hat{\eta}(s, \theta) \quad = \quad \begin{cases} \eta_E(s, \zeta_E(\theta)) & \text{if } \theta = \zeta_E(\theta), \\ 0 & \text{otherwise,} \end{cases}$$

where $\eta_E(s, \ell) = \sum_{\theta \in [\ell]_E} \eta(s, \theta)$ for all state $s \in S$ and all joint-history label $\ell \in \mathscr{L}$.

This definition can be alternatively stated as follows. For any occupancy state η, we call compact occupancy state with respect to E, an occupancy state $\hat{\eta}$ based on (η_E, ζ_E), where η_E is defined over states and joint-history equivalence classes (joint-history labels). Intuitively, distribution η_E reassigns the probability mass of each equivalence class $[\theta]_E$ to joint-history label ℓ in $[\theta]_E$. Furthermore, compact occupancy states are full occupancy states where the probability mass is focused over a small number of state and joint-history pairs.

4.1.2 Compact Decision Rules and Policies

In this section, we extend the concept of compact representation to decision rules and policies. In particular, we define compact decision rules as mappings from histories to actions, that prescribe the same actions to equivalent histories; and we further define compact policies as sequences of compact decision rules.

Definition 23. *Let E be an equivalence relation between individual histories, and d^i an individual decision rule of agent $i \in I$. A **compact decision rule** $\hat{d}^i$ with respect to E and d^i is specified by a triplet (d^i_E, ζ_E, a^i) such that, for any individual history θ^i:*

$$\hat{d}^i(\theta^i) \quad = \quad \begin{cases} d^i_E(\zeta_E(\theta^i)) & \text{if } \theta \in \Theta', \\ a^i & \text{otherwise,} \end{cases}$$

where $d^i_E(\ell^i) = d^i(\ell^i)$ for any arbitrary individual-history label $\ell^i \in \mathscr{L}^i$.

This definition states that a compact decision rule is a full decision rule, except that histories that are equivalent with respect to a specified equivalence relation E, prescribe the same action. We denote $D^i_{t,E}$ the t-step compact decision rule set of agent i, and $D_{t,E}$ the t-step compact separable decision rule set $d_{t,E}$ with respect to E and $d_t \in D_t$. Notice that there might be different equivalence relations from one time step to another, or one agent to another. We further extend this concept to equivalent policies.

Definition 24. *Let $E \equiv (E_t)_{t \in \{0,\dots,T-1\}}$ be equivalence relations between individual histories, and π^i be an individual policy of agent $i \in I$. A **compact individual policy** $\hat{\pi}^i$, with respect to E and individual policy π^i, is given by a sequence of compact individual decision rules $\langle \hat{d}^i_0, \hat{d}^i_1, \dots, \hat{d}^i_{T-1} \rangle$ with respect to E and π^i.*

4.1.3 Compact Vectors and Points in Value Functions

We have demonstrated that piecewise linear and convex functions can be represented using either vector or point sets. In this section, we describe how to represent vectors or points in a compact way using compact representations based on an equivalence relation between individual histories.

Definition 25. *Let E be an equivalence relation between joint histories, and $\beta \in \mathbb{R}^{n|\Theta|}$ a vector value. A **compact vector** $\hat{\beta}$, with respect to E and β, is based on a triplet $(\beta_E, \zeta_E, \underline{v}_E)$. That is, a vector $\beta_E \in \mathbb{R}^{nK}$, a transformation rule $\zeta_E \colon \Theta' \mapsto \mathscr{L}$ of E, and a lower-bound value $\underline{v}_E$, such that:*

$$\hat{\beta}(s, \theta) \quad = \quad \begin{cases} \beta_E(s, \zeta_E(\theta)) & \text{if } \theta \in \Theta', \\ \underline{v}_E & \text{otherwise,} \end{cases}$$

where $\beta_E(s, \ell) = \min_{\theta \in [\ell]_E} \beta(s, \theta)$ for any arbitrary state $s \in S$ and joint-history label $\ell \in \mathscr{L}$.

Definition 26. *Let E be an equivalence relation between joint histories, and (non-corner) point $p \equiv (\eta, g)$ given by a full occupancy state η and a full function g. A **compact point** $\hat{p} \equiv (\hat{\eta}, \hat{g})$, with respect to E and p, consists of a compact occupancy state $\hat{\eta}$ and a compact function $\hat{g}$. The compact function $\hat{g}$, with respect to E and g, is based on a triplet $(g_E, \zeta_E, \bar{v}_E)$. That is, a function $g_E \colon \mathscr{L} \mapsto \mathbb{R}$, a transformation rule $\zeta_E \colon \Theta' \mapsto \mathscr{L}$ of E, and an upper-bound value $\bar{v}_E$, such that:*

$$\hat{g}(\theta) \quad = \quad \begin{cases} g_E(\zeta_E(\theta)) & \text{if } \theta \in \Theta', \\ \bar{v}_E & \text{otherwise,} \end{cases}$$

where $g_E(\ell) = \max_{\theta \in [\ell]_E} g(\theta)$ for any arbitrary joint-history label $\ell \in \mathscr{L}$.

These definitions are driven by how we actually use vector and point sets. Because we use vectors to represent lower bounds, it is natural the maintain in β_E the lowest value for each equivalence class. On the contrary, as we use points to represent upper bounds, g_E keeps track of the highest value for each equivalence class. This ensures that our compact representations preserve the ability to bound the optimal value function. Relative to standard full representations, our compact representations are weaker bounds over many occupancy states. This is mainly because the compact representation are lossy over some occupancy states. We will demonstrate that they nevertheless preserve the ability to eventually converge to the optimal value function.

Remark 2. *There is an important property we need to guarantee in order to ensure convergence. That is, our compact representations of lower and upper bounds need to improve over at least one occupancy state at each step of the algorithm, and to decline over no visited occupancy states.*

We are now ready to relax the assumptions we made so far. That is, we no longer assume that equivalence relation E and associated transformation rule ζ_E are given. The following presents two equivalence relations and their corresponding transformation rules: probabilistic equivalence (Section 4.2) and value-based equivalence (Section 4.4) relations. Before proceeding any further, we highlight the high-level goals of these equivalence relations

4.1.4 Equivalence Relation Families

In this paper, we focus on two families of equivalence relations: *behavioral* and *value-preserving* equivalence relations. The equivalence relations we target have to be defined in such a way that they belong to either of these relations. Each operation these relations involve over histories, including clustering of histories through equivalence relations, will restrict the space of policies and value functions of interest. For this reason, we define the concept of **optimality-preserving** operations, to characterize operations that preserve the ability to eventually find an optimal solution.

Definition 27. *Let E be an equivalence relation between individual histories. E is said to be a* ***behavioral equivalence relation*** *if and only if, for any agent $i \in I$ and any arbitrary pair of histories (of agent i) θ^i_a and θ^i_b that are equivalent with respect to E, it is optimality-preserving to prescribe the same future policy for θ^i_a and θ^i_b.*

The behavioral equivalence relations serve as a means to group together individual histories that can be mapped to the same policies. These relations are particularly useful to restrict our attention to the lower-dimensional space of separable decision rules and policies. Because separable decision rules and policies are selected based on occupancy states, behavioral equivalence relations yield also compact representations of occupancy states.

Definition 28. *Let E be an equivalence relation between joint histories. E is said to be a* ***value-preserving equivalence relation*** *if and only if, for any arbitrary pair of joint histories θ_a and θ_b that are equivalent with respect to E, it is optimality-preserving to constrain θ_a and θ_b to map to the same value.*

The value-preserving equivalence relations serve as a means to cluster together joint histories that can be mapped to the same values. These relations are particularly useful to provide lower-dimensional representation of high-dimensional value functions, using either compact vector or point sets. The value-preserving and behavioral equivalence relations are fundamentally different. The later essentially provides compact representations of mappings from histories to expected values; and the former produces compact representations for mappings from histories to actions. These relations remains, however, closely related. We will demonstrate that the behavioral equivalence that serves to build compact representation of a policy, can also serve as a value-preserving equivalence relation for the value function of that policy.

In the following, we present examples of behavioral equivalence relations, including local probabilistic and probabilistic m-truncation equivalence relations. Thereafter, we discuss a value-preserving equivalence relation called value-based equivalence relation.

To allow our algorithms to reason based on compact occupancy states instead of full occupancy states, we need to find equivalence relations that preserve optimality. Of this family, the probabilistic equivalence relation can preserve important information for optimal decision-making.

4.2 Local Probabilistic Equivalence

The probabilistic equivalence relation has been previously explored by Oliehoek et al. (2009). Roughly speaking, two histories are probabilistically equivalent if they have the same predictive model over states and histories of the other agents *conditional on the current occupancy state*[6].

Definition 29. *Histories θ_a^i and θ_b^i are **locally probabilistically equivalent** (LPE) with respect to occupancy state η if and only if, for any arbitrary state $s \in S$ and history $\theta^j \in \Theta^j$ of the other agents:*

$$Pr(s, \theta^j | \eta, \theta_a^i) \quad = \quad Pr(s, \theta^j | \eta, \theta_b^i).$$

We denote $[\theta]_{E_\eta}$ the probabilistically equivalent class of θ with respect to LPE E_η induced by η.

We often refer to this probabilistic equivalence relation as a *local probabilistic equivalence* because it clusters histories based only upon a single occupancy state. In fact, histories that are probabilistically equivalent with respect to one occupancy state need not be probabilistically equivalent with respect to another one. Therefore, each occupancy state η_a induces a local probabilistic equivalence relation E_a between joint histories, which in turn yields compact occupancy state η_{E_a} with respect to E_a and occupancy state η. Next, we illustrate this definition using the multi-agent tiger problem.

Example 2 (Multi-agent tiger — local probabilistic equivalence)**.** *Let us give two examples of equivalent individual histories in the multi-agent tiger problem:*

1. *Let us consider agent 1's individual histories:*

$$\theta_a^1 = (a_{\text{LISTEN}}, z_{\text{RIGHT}}, a_{\text{RIGHT}}, z_{\text{RIGHT}}) \text{ and}$$
$$\theta_b^1 = (a_{\text{RIGHT}}, z_{\text{RIGHT}}, a_{\text{LEFT}}, z_{\text{LEFT}}).$$

   Because they both end with an action opening a door, and thus reset the system without giving any information about the new state, they are probabilistically equivalent independently of the other individual histories and independently of the agents' policies.

2. *Let us consider agent 1's individual histories:*

$$\theta_a^1 = (a_{\text{LISTEN}}, z_{\text{LEFT}}, a_{\text{LISTEN}}, z_{\text{RIGHT}}) \text{ and}$$
$$\theta_b^1 = (a_{\text{LISTEN}}, z_{\text{RIGHT}}, a_{\text{LISTEN}}, z_{\text{LEFT}}).$$

   If, in addition, agent 2's policy consists in performing action a_{LISTEN} twice during the first two time steps (whatever its first observation), then these two histories are equivalent as they correspond to the same distribution over states and over agent 2's individual histories.

As illustrated by these two examples, some equivalences depend only on the structure of the Dec-POMDP at hand, while others also depend on the agents' (past) policies.

4.2.1 Theoretical Properties

The following provides sufficient conditions that ensure compact occupancy state η_E, with respect to a probabilistic equivalence relation E and η, is sufficient for optimal decision making in occupancy Markov decision processes. Such a compact occupancy state will be referred to as a *lossless compact occupancy state* with respect to E and the original occupancy state η.

[6] The original definition by (Oliehoek et al., 2009) involves information states; given that occupancy states are sufficient statistics for information states, our definition still holds.

Lemma 4. *Let η be an occupancy state, and E its corresponding local probabilistic equivalence relation between histories. Locally probabilistically equivalent histories, with respect to E and η, can be clustered while preserving optimality at occupancy state η.*

Proof. See (Oliehoek et al., 2009). □

In other words, each occupancy state η_a induces a local probabilistic equivalence E_a, which in turn yields a compact occupancy state η_{a,E_a}. Lemma 4 ensures that η_{a,E_a} is a lossless compact occupancy state with respect to η_a (and E_a). Thus, the optimal value for η_{a,E_a} is also the optimal value for η_a. However, for any other occupancy state η_b, there is no guarantee compact occupancy state η_{b,E_a} with respect to E_a is a lossless compact occupancy state. In the remainder of this paper, for any occupancy state η_a, we denote $\tilde{\eta}_a = \eta_{a,E_a}$, any lossless compact occupancy state with respect to η_a (and E_a). We further denote $\tilde{\Delta} \equiv \cup_{t\in\{0,\dots,T\}}\tilde{\Delta}_t$ the space of lossless compact occupancy states. Section 4.2.2 discusses an algorithm for calculating compact occupancy states.

Lemma 4 yields interesting derivatives related to the occupancy MDP planning. The primary simple yet important derivative is that, instead of planning over full occupancy states, one can — with no loss in generality — plan over lossless compact occupancy states.

Lemma 5. *Let $(V^*_{M,t})_{t\in\{0,1,\cdots,T\}}$ be solutions of the optimality equations. Then, for each $t \in \{0, 1, \cdots, T\}$, optimal value function $V^*_{M,t}$ depends on occupancy state η_t only through $\tilde{\eta}_t$.*

Proof. The proof follows directly from Lemma 4. □

This lemma suggests that the optimal value functions depend on joint histories only through joint-history labels. The following relates optimal value functions with optimal separable policies.

Lemma 6. *Any optimal separable policy depends on joint histories only upon joint-history labels.*

Proof. To better understand this property, recall that, given optimal value functions, an optimal separable policy is given by Equation (7):

$$\begin{aligned}
d^*_t &\in \arg\max_{d_t\in D_t}\Big(R(\iota_t, d_t) + V^*_{M,t+1}(F(\iota_t, d_t))\Big), \quad \text{information-state based selection}\\
&\in \arg\max_{d_t\in D_t}\Big(R(\eta_t, d_t) + V^*_{M,t+1}(\eta_{t+1})\Big), \quad \iota_t \text{ replaced by } \eta_t\\
&\in \arg\max_{d_t\in D_t}\Big(R(\tilde{\eta}_t, d_t) + V^*_{M,t+1}(\tilde{\eta}_{t+1})\Big), \quad \eta_t \text{ replaced by } \tilde{\eta}_t\\
&\in \arg\max_{d_t\in D_t}\Big(\textstyle\sum_{s,\ell}\beta^{d_t}_t(s,\ell)\cdot\tilde{\eta}_t(s,\ell) + \sum_{s',\ell'}\beta_{t+1}(s',\ell')\cdot\tilde{\eta}_{t+1}(s',\ell')\Big).
\end{aligned}$$

This ends the proof. □

Lemma 6 shows that there exists an optimal separable policy that depends on joint histories only upon joint-history labels. From the perspective of applications, we find it comforting that by restricting attention to compact policies, which are simple to implement and compute, we may achieve as large an expected total reward as if we used history-dependent policies. Doing so saves significant amounts of time and memory. Another important derivative of Lemma 4 is that the optimal value functions $(V^*_{M,t})_{t\in\{0,1,\cdots,T\}}$ remain piecewise linear and convex functions of lossless compact occupancy states. This leads to a lower dimension representation of the optimal value functions.

Theorem 6. *Let $(V^*_{M,t})_{t\in\{0,1,\cdots,T\}}$ be solutions of the optimality equations. Then, for each $t \in \{0, 1, \cdots, T\}$, value functions $V^*_{M,t}$ are piecewise-linear and convex functions of lossless compact occupancy states $\tilde{\Delta}_t$.*

Proof. The proof results directly from the observation that policies that are represented over joint-history labels can nonetheless be represented over the entire joint history space, since each joint-history label represents a partition of the entire history space and the union of these partitions defines the entire history set. Furthermore, the transformation from full occupancy states to compact occupancy states is a linear transformation. And thus, it preserves the piecewise linearity and convexity property. With these observations as a background, Theorem 2 concludes the proof. □

4.2.2 Computing Compact Occupancy States

In this section, we describe methods for calculating compact occupancy states based on the local probabilistic equivalence.

Algorithm 5: Calculating joint-history labels and compact occupancy states

function LABELS(η_a)
  foreach $i \in I$ **do**
    $\mathscr{L}^i \leftarrow \emptyset$ — initialize label set to empty set
    Θ^i — collect individual histories of agent i reachable from η_a
    while $\Theta^i \neq \emptyset$ **do**
      $\ell^i \leftarrow$ **LexMin**(Θ^i) — return the lexicographical minimal individual history
      $\mathscr{L}^i \leftarrow \mathscr{L}^i \cup \{\ell^i\}$ — add individual history (or label) ℓ^i into label set $\mathscr{L}^i$
      $\Theta^i \leftarrow \Theta^i \backslash [\ell^i]_{E_a}$ — remove all individual histories that are PE with ℓ^i
  return $\otimes_{i \in I} \mathscr{L}^i$

function COMPACTOCCUPANCYSTATE(η_a, η)
  $\mathscr{L} \leftarrow$ LABELS(η_a)
  foreach $s \in S$ *and* $\ell \in \mathscr{L}$ **do**
    $\eta_{E_a}(s, \ell) \leftarrow \sum_{\theta \in [\ell]_{E_a}} \eta(s, \theta)$
  return η_{E_a}

We now present methods in Algorithm 5 that: (*a*) extract joint-history labels; and (*b*) compute compact occupancy states relative to a local probabilistic equivalence. Method LABELS (Algorithm 5) extracts joint-history labels for any specified occupancy state η. For each agent $i \in I$, LABELS groups all together histories that are probabilistically equivalent with respect to η, which creates intermediate sets of histories $(\mathscr{L}^i)_{i \in I}$. Next, it creates the set of joint-history labels $\otimes_{i \in I} \mathscr{L}^i$, where each joint-history label $\ell = (\ell^i)_{i \in I}$ includes one individual history label ℓ^i from each label set $\mathscr{L}^i$. To speed up these operations, the following lemma provides us with a simple criterion to identify probabilistically equivalent histories based only upon local occupancy states.

Lemma 7. *Two histories θ^i_a and θ^i_b are **locally probabilistically equivalent** w.r.t. η iff $\eta(\theta^i_a) \propto \eta(\theta^i_b)$.*

Proof. The proof results directly from Definition 29 and that of a local occupancy state. □

Method COMPACTOCCUPANCYSTATE calculates compact occupancy state η_{E_a} with respect to E_a and occupancy state η. COMPACTOCCUPANCYSTATE reassigns the probability mass $\eta(s, \theta)$ into $\eta_{E_a}(s, \ell)$, for any arbitrary state $s \in S$ and joint history $\theta \in \Theta$ and joint-history label $\ell \in [\theta]_{E_a}$.

As previously discussed, a compact occupancy state is also an occupancy state. Hence, by planning only over compact occupancy states, we are still planning over occupancy states. These occupancy states differ from the original occupancy states only in the fact that they are sparser, that is, the probability mass is concentrated only over a smaller number of state and joint history pairs. This may lead to faster convergence to an optimal solution. This is mainly because many full occupancy states share the same lossless compact occupancy states, which enhances the generalization of the value function over the entire full occupancy state space, and hence the convergence. To further enhance the generalization of the value function, we now present the transformation rule of the local probabilistic equivalence relation.

4.2.3 Transformation Rule of Local Probabilistic Equivalence

Intuitively, the transformation rule aims at converting any full occupancy state, vector value, or decision rule into a compact occupancy state, vector value, or decision rule, respectively. In this section, we provide an explicit definition of the transformation rule associated to the local probabilistic equivalence relation. The transformation rule ζ_E of equivalence relation E provides a mapping from individual histories to individual equivalence classes induced by E. The compactness of the representation that results from equivalence relation E is strongly related to the compactness of its transformation rule ζ_E. This highlights the necessity of finding a transformation rule that possesses itself a compact representation. This representation can be anything: a logical proposition, an equation, an algorithm, etc.

Algorithm 6: Transformation Rule Algorithm for the LPE

```
function TRANFORMATION-RULE(η̃_t, θ^i_t)
    foreach ℓ^j_t ∈ 𝓛^j_t and s ∈ S do
        η̂_t(s, ℓ^j_t, θ^i_t) ← Pr(s, ℓ^j_t, θ^i_t | η_0)
    foreach ℓ^i_t ∈ 𝓛^i_t do
        if η̂_t(θ^i_t) ∝ η̃_t(ℓ^i_t) then return ℓ^i_t
```

We use the *transformation rule for LPE* Algorithm 6 to represent the transformation rule of the local probabilistic equivalence relation. The algorithm is parametrized by a lossless compact occupancy state. It returns the individual label (of the lossless compact occupancy state) that corresponds to the input individual history of a specified agent. To do so, the algorithm computes the predictive model over states and histories of the other agents conditional on the initial occupancy state and the input individual history. This model is thereafter compared to that of the individual labels of the specified agent. The algorithm returns the individual label that is locally probabilistically equivalent to the specified individual history. Lemma 7 guarantees this procedure returns the individual label that represents the individual equivalence class of the input individual history.

As already discussed, the transformation rule serves as a means to generalize the occupancy states, decision rules, and even value functions beyond the scope of the labels used to define the compact representation. It is nonetheless worth noticing that the projection of the LPE is somewhat time consuming. This is mainly because it requires computing the predictive model other states and the labels of the other agents. This operation is often time consuming for local probabilistic equivalence relations that involve a large number of states, labels, or agents. Another probabilistic equivalence relation, namely the global probabilistic equivalence, mitigates the time complexity of the local probabilistic equivalence relation. It further enhances the generalization of the value function.

4.3 Global Probabilistic Equivalence

In this section, we define the notion of *global probabilistic equivalence* between histories, in a similar vein of the local probabilistic equivalence. Intuitively, two histories are globally probabilistically equivalent if they share the same probabilistic model over all possible states of the system and histories of the other agents regardless the current information state.

Definition 30. *Two individual histories θ^i_a and θ^i_b are **globally probabilistically equivalent** (GPE) if and only if, for any reachable occupancy state η, we have that $\eta(\theta^i_a) \propto \eta(\theta^i_b)$.*

We refer to this relation as a *global probabilistic equivalence* since it holds over multiple different occupancy states. This contrasts with the local probabilistic equivalence, which assumes the equivalence

between histories holds only with respect to a specified occupancy state. An example of global probabilistic equivalence relation follows.

Example 3 (Multi-agent tiger — global probabilistic equivalence)**.** *By definition, globally probabilistically equivalent histories should not depend on past policies. So, from our previous example, only non-contextual equivalences between individual histories will be kept in a global probabilistic equivalence relation. Another trivial global probabilistic equivalence relation is the identity. That is, a relation that assumes each individual history is only equivalent to itself.*

Calculating a global probabilistic equivalence relation is both time and memory expensive and often unnecessary. In practice, we only need a global probabilistic equivalence over a small subset of occupancy states, those that are visited during the planning stage.

4.3.1 Probabilistic Truncation Equivalence

The following introduces the *probabilistic truncation equivalence* relation, which restricts the global probabilistic equivalence to a small subset of occupancy states $\underline{\triangle} \subseteq \triangle$. Before proceeding any further, we start with a few notations and definitions. The probabilistic truncation equivalence assumes all important information about the process to be controlled lies in the m last actions and observations agents have experienced, where $m \in \{0, 1, \dots, T\}$. Obviously, this assumption is always true for at least $m = T$. A more formal definition follows.

Definition 31. *The* ***m-truncation*** *(of history $(a_i^0, z_i^1, \cdots, a_i^{t-1}, z_i^t)$) is the m last actions and observations:*

$$\textsc{trunc}((a_i^0, z_i^1, \cdots, a_i^{t-1}, z_i^t), m) \stackrel{\text{def}}{=} \begin{cases} (a_i^0, z_i^1, \cdots, a_i^{t-1}, z_i^t) & \text{if } m \geq t, \\ (a_i^{t-m+1}, z_i^{t-m+1}, \cdots, a_i^{t-1}, z_i^t) & \text{otherwise.} \end{cases} \tag{35}$$

Histories θ_a^i and θ_b^i are m-truncation equivalent if and only if $\textsc{trunc}(\theta_a^i, m) = \textsc{trunc}(\theta_b^i, m)$. We further denote $\theta_{t:m}$ the m-truncation associated to t-step history θ_t.

We now introduce definitions and notations we use when dealing with m-truncations and associated decision rules, policies and value functions.

Definition 32. *An* ***individual t-step m-truncation decision rule*** *$d_{t:m}^i$ of agent i is a mapping from individual t-step m-truncations to individual actions. A* ***separable t-step m-truncation decision rule*** *$d_{t:m}$ is an N-tuple of individual t-step m-truncation decision rules $(d_{t:m}^1, \dots, d_{t:m}^N)$.*

Definition 33. *A t-step* ***m-truncation occupancy state*** *$\eta_{t:m}$ is a distribution of probabilities over state and t-step m-truncation pairs. We denote $\textsc{trunc}(\eta_t, m) = \eta_{t:m}$ the m-truncation occupancy state associated to occupancy state η_t. We further denote $\triangle_{t:m}$ the t-step m-truncation occupancy state space, and $\triangle^{[m]} = \cup_{t=0}^{T} \triangle_{t:m}$ the space of all m-truncation occupancy states.*

Following this idea, we note that a separable m-truncation policy consists of separable m-truncation decision rules, and m-truncation value functions are mappings from m-truncation occupancy states to reals. Next, we discuss sufficient conditions that enable optimal decision-making using m-truncation policies and value functions. To this end, we group together m-truncation and global probabilistic equivalence relations in order to take advantage of lower memory requirements of the former, while preserving the behavioral equivalence of the later.

Definition 34. *Let $\underline{\triangle}$ be a subset of the occupancy state space $\triangle$. We say that histories are* ***probabilistically m-truncation equivalent*** *with respect to $\underline{\triangle}$ if and only if, for any occupancy state η in $\underline{\triangle}$ and E the local probabilistic equivalence with respect to η, histories that are m-truncation equivalent are also locally probabilistically equivalent with respect to E and η.*

In other words, histories that are *probabilistically m-truncation equivalent* with respect to $\underline{\triangle}$ are: (*a*) *m*-truncation equivalent with one another; and (*b*) locally probabilistically equivalent with respect to any occupancy state $\eta \in \underline{\triangle}$. Next, we present theoretical implications of the probabilistic *m*-truncation equivalence. The probabilistic *m*-truncation equivalence relation may be lossy. For this reason, parameter *m* is adjusted with respect to the set of occupancy states.

4.3.2 Theoretical Properties

Let us make explicit the relationship between local and global probabilistic equivalence relations. Then, we will discuss the relationship between global probabilistic equivalence and *m*-truncation probabilistic equivalence relations.

Lemma 8. *The following properties hold:*
(a) Globally probabilistically equivalent histories are also locally probabilistically equivalent.
(b) Locally probabilistically equivalent histories need not be globally probabilistically equivalent.

Proof. The proof results immediately from Definition 29 and Definition 30. □

Lemma 8 permits us to extend results that hold for local probabilistic equivalence to global probabilistic equivalence. For instance, given a global probabilistic equivalence E, any optimal separable policy depends on joint histories only upon joint-history equivalence classes with respect to E. From the perspective of histories, the global probabilistic equivalence is a weaker version of the local probabilistic equivalence. This is mainly because histories that are GPE are also LPE, but histories that are LPE need not be GPE. It is nevertheless questionable whether or not such a property extends to occupancy states.

Lemma 9. *The following properties hold:*
(a) Occupancy states that are GPE, are also behavioral equivalent.
(b) Occupancy states that are LPE, need not be behavioral equivalent.
(c) Occupancy states with lossless compact occupancy states that are GPE, are behavioral equivalent.

Proof. Let E and E' be GPE and LPE relations, respectively. If we let occupancy states η and η' be GPE w.r.t. E, then $\eta_E = \eta'_E$. In addition, from Lemma 8, E preserves optimality at η and η', since E is a GPE. And hence, from Definition 27 we know that η and η' are behavioral equivalent w.r.t. E. This proves property (*a*). However, if we have $\eta_{E'} = \eta'_{E'}$, there is no guarantee either $\eta_{E'}$ or $\eta'_{E'}$ are lossless compact occupancy states of η and η', respectively. Thus, there is no guarantee E' preserves optimality at η and η'. This proves property (*b*). But if we have $\tilde{\eta}_E = \tilde{\eta}'_E$, then η and η' are behavioral equivalent, as $\tilde{\eta}_E$ and $\tilde{\eta}'_E$ constitute from Lemma 8 lossless compact occupancy states for η and η', respectively. □

Lemma 9 presents criteria for checking whether or not occupancy states are behavioral equivalent. These criteria are necessary for enhancing the generalization of the value functions over the entire occupancy states. From the perspective of occupancy states, a GPE relation remains a weaker equivalence relation than an LPE relation. From the value generalization perspective, the global probabilistic equivalence can generalize values from one occupancy state to any other behavioral equivalent occupancy states. A similar argument holds using the local probabilistic equivalence, but establishing the behavioral equivalence using the local probabilistic equivalence is much harder.

The properties that hold for general probabilistic equivalence relations also hold for probabilistic truncation equivalence relations. To better understand this, consider subset of occupancy states $\underline{\triangle} \subset \triangle$, the *m*-truncation probabilistic equivalence relation with respect to $\underline{\triangle}$, and the *m*-truncation occupancy set $\underline{\triangle}^{[m]}$ with respect to occupancy set $\underline{\triangle}$. The following ensures the planning only over *m*-truncation occupancy set $\underline{\triangle}^{[m]}$ instead of $\underline{\triangle}$ preserve the optimality.

Lemma 10. *Let E be a probabilistic m-truncation equivalence relation with respect to $\underline{\triangle}$. It is optimality preserving to plan only over $\underline{\triangle}^{[m]}$ instead of $\underline{\triangle}$.*

Proof. The proof results directly from Theorem 6 and Lemma 8. □

Lemma 10 suggests that given a probabilistic m-truncation equivalence relation with respect to $\underline{\triangle}$, separable policies and value functions depend on occupancy states in $\underline{\triangle}$ only upon m-truncation occupancy states in $\underline{\triangle}^{[m]}$.

4.3.3 Computing m-Truncation Occupancy States

Algorithm 7: Calculating joint-history labels and m-truncation occupancy states.

function LABELS(η, m)
  foreach $i \in I$ **do**
    $\mathscr{L}^i \leftarrow \emptyset$ and Θ^i — collect individual histories of agent i reachable from η_a
    while $\Theta^i \neq \emptyset$ **do**
      $\ell^i \leftarrow$ **LexMin**(Θ^i) — return the lexicographical minimal individual history
      $\mathscr{L}^i \leftarrow \mathscr{L}^i \cup \{\text{TRUNC}(\ell^i, m)\}$ — add individual history (or label) ℓ^i into label set $\mathscr{L}^i$
      $\Theta^i \leftarrow \Theta^i \backslash [\ell^i]_E$ —*remove all individual histories that are PTE with* ℓ^i
  return $\otimes_{i \in I} \mathscr{L}^i$

function COMPACTOCCUPANCYSTATE(η, m)
  $\mathscr{L} \leftarrow$ LABELS(η, m)
  foreach $s \in S$ *and* $\ell \in \mathscr{L}$ **do**
    $\eta_E(s, \ell) \leftarrow \sum_{\theta \in [\ell]_E} \eta(s, \theta)$
  return η_E

In this section, we discuss how to compute m-truncation occupancy states given a specified full occupancy state and a probabilistic m-truncation equivalence relation. The algorithms start with parameter $m_t = 1$, one parameter for each time step $t \in \{0, 1, \ldots, T-1\}$.

We present the transformation rule of the probabilistic m-truncation equivalence relation. Algorithm 7, developed for the probabilistic m-truncation equivalence, is similar to that of the local probabilistic equivalence (Algorithm 5). These algorithms differ only in the criteria they use to group individual histories. Here, we group individual histories that are probabilistically m-truncation equivalent with one another. It is likely that the specified m is not sufficient to distinguish every individual histories. Indeed, in a compact occupancy state, two individual labels may be m-truncation equivalent, but not locally probabilistically equivalent. In this case, we say that m does not capture the structure of the problem. To handle this problem, we suggest an algorithm that adjusts parameter m as the algorithm explores the search space, that is, the space of occupancy states.

The transformation rule (Algorithm 8) proceeds as follows. Whenever the main algorithm meets a novel occupancy state η, we update parameter m with respect to η, see routine PARAMETER (Algorithm 8). To do this, we check whether all individual histories in η are probabilistic m-truncation equivalent with one another. Otherwise, we increment parameter m, and iterate the procedure, until no more increment is necessary. One can maintain either a global parameter m, which is consistent with all visited occupancy states; or maintain local parameters m, each of which is consistent with the associated occupancy state. For this reason, we present two projection methods.

In the case we use the local parameter associated with each occupancy state, the compact representation consists in pair (η_E^{ref}, m). Given compact occupancy state η_E^{ref}, we extract the set of individual labels $\mathscr{L}^i$. Thereafter, we compare m-truncation $\text{TRUNC}(\theta^i, m)$ of input individual history θ^i to labels in $\mathscr{L}^i$. If no matches exists, we return individual history θ^i. Otherwise, we return $\text{TRUNC}(\theta^i, m)$.

Algorithm 8: The transformation rule using PTE

function PARAMETER(m, η^{ref})
  foreach $i \in I$ **do**
    Θ^i — collect individual histories of agent i reachable from η.
    foreach $\theta^i_a, \theta^i_b \in \Theta^i$ **do**
      $bool_a \leftarrow \text{TRUNC}(\theta^i_a, m) = \text{TRUNC}(\theta^i_b, m)$ — check m-truncation equivalence
      $bool_b \leftarrow [\theta^i_a]_{\eta^{\text{ref}}} = [\theta^i_b]_{\eta^{\text{ref}}}$ — check probabilistic equivalence
      if $bool_a \wedge \neg bool_b$ **then** $m \leftarrow m + 1$
  return m

function PROJECTION((η^{ref}_E, m), θ^i)
  $\mathscr{L}^i$ — collect individual labels of agent i reachable from η^{ref}_E.
  if $\text{TRUNC}(\theta^i, m) \in \mathscr{L}^i$ **then return** $\text{TRUNC}(\theta^i, m)$ **else return** *any label* $\ell^i \in \mathscr{L}^i$

function PROJECTION($(\mathscr{L}^i, m)$, θ^i)
  if $\text{TRUNC}(\theta^i, m) \in \mathscr{L}^i$ **then return** $\text{TRUNC}(\theta^i, m)$ **else return** θ^i

In the case we use the global parameter associated with all visited occupancy states, the compact representation consists in pair $(\otimes_{i\in I}\mathscr{L}^i, m)$. Given any input individual history θ^i, we check whether its m-truncation is part of $\mathscr{L}^i$, in which case, we return $\text{TRUNC}(\theta^i, m)$. Otherwise, we return individual history θ^i.

4.4 Value-Based Equivalence

The purpose of probabilistic equivalence relations was to find a small number of state features that are sufficient to maintain important information about the state of the process, while preserving the optimality. The value-based equivalence relation follows a different yet complementary goal. That is, to find a small number of state features that are sufficient for accurately representing lower and upper bounds, and hence preserving the ability to eventually find an optimal solution.

4.4.1 Value-Based Equivalence

We introduce value-based equivalence relations among joint histories, which make easy the generalization of values and, even more importantly, make possible compact representations of upper and lower bounds. Roughly speaking, two joint histories are value-equivalent if they have the same value for a specified mapping from state and joint history pairs to reals.

Definition 35. *Let β be a mapping from state and joint history pairs to reals, and θ_a and θ_b be two joint histories. We say that θ_a and θ_b are **value-equivalent** with respect to β if and only if $\beta(s, \theta_a) = \beta(s, \theta_b)$, for any state $s \in S$. Such a relation is referred to as a **value-based equivalence relation** between joint histories.*

4.4.2 Connections Between Probabilistic and Value-Based Equivalence Relations

It is worth noticing that, if we let E be a value-based equivalence relation with respect to β, and $\hat{\eta}$ be the compact occupancy state with respect to E and η, both occupancy states $\hat{\eta}$ and η yield the same value with respect to β. This is mainly because β exhibits some redundancies, that is, many joint histories have the same value. To better understand this observation, notice that for joint histories θ_a and θ_b that are probabilistically equivalent with respect to occupancy state η, there exists a separable policy π that is

maximal at η such that $\Psi(\pi, \theta_a) = \Psi(\pi, \theta_b)$, and thus for any state $s \in S$ we have $V_{\Psi(\pi,\theta_a)}(s) = V_{\Psi(\pi,\theta_b)}(s)$. In other words, for joint histories θ_a and θ_b that are probabilistically equivalent with respect to η, there exists a function β^π maximal at η such that θ_a and θ_b are also value-equivalent with respect to β^π. But the opposite is not always true, that is, joint histories that are value-equivalent need not be probabilistically equivalent.

We denote $\tilde{\beta}$ the **lossless compact mapping** of β with respect to value-based equivalence relation E based on β, such that $\tilde{\beta}$ is specified by $(\beta_E, \zeta_E, \underline{v}_E)$. The following lemma summarizes the relationship between value-based equivalence and local probabilistic equivalence.

Lemma 11. *Let E_β be the value-based equivalence relation with respect to mapping β. Value functions depend on full occupancy states and full hyperplanes only upon lossless occupancy states and lossless hyperplanes:*

$$V^*_{M,t}(\eta) \quad = \quad \max_{\pi=\langle d_t,\dots,d_{T-1}\rangle} \sum_s \sum_\ell \tilde{\beta}^\pi(s,\ell) \cdot \tilde{\eta}_{E_{\beta^\pi}}(s,\ell),$$

for any arbitrary t-step occupancy state η.

Proof. The proof follows directly from Definition 35:

$$\begin{aligned} V^*_{M,t}(\eta) &= \max_{\pi=\langle d_t,\dots,d_{T-1}\rangle} \sum_s \sum_\theta \beta^\pi(s,\theta) \cdot \eta(s,\theta), \\ &= \max_{\pi=\langle d_t,\dots,d_{T-1}\rangle} \sum_s \sum_\theta \tilde{\beta}^\pi(s,\theta) \cdot \tilde{\eta}(s,\theta), \\ &= \max_{\pi=\langle d_t,\dots,d_{T-1}\rangle} \sum_s \sum_\ell \tilde{\beta}^\pi(s,\ell) \cdot \tilde{\eta}_{E_{\beta^\pi}}(s,\ell). \end{aligned}$$

This ends the proof. □

This lemma underlines the connection between probabilistic and value-based equivalence relations. Indeed, the probabilistic equivalence induces the value-based equivalence. This connection builds on the relationship between occupancy states and value functions as mentioned in Lemma 4. To make explicit this relationship, we introduce the concept of *support occupancy state* for each vector value. The idea is to provide an alternative equivalence relation to build compact representations for vector values.

Definition 36. *Let E_η be a probabilistic equivalence relation induced by occupancy state η. Let E_β be a value-based equivalence relation induced by vector value β. Occupancy state η is referred to as a **support (occupancy state)** for vector value β if and only if histories that are equivalent with respect to E_η are also equivalent with respect to E_β.*

Consider support occupancy state η for vector value β. Let E_η be a probabilistic equivalence relation induced by occupancy state η. Let E_β be a value-based equivalence relation induced by vector value β. Thus, we can interchangeably use either E_η or E_β in order to build a compact representation for vector value β.

Lemma 12. *If we let η_t be a t^{th}-step occupancy state, and $\hat{\pi} = \langle \hat{d}_t, \hat{d}_{t+1}, \dots, \hat{d}_{T-1}\rangle$ be a separable policy such that $\sum_s \sum_\theta \beta^{\hat{\pi}}_t(s,\theta) \cdot \tilde{\eta}_t(s,\theta) = \max_{\pi=\langle d_t,d_{t+1},\dots,d_{T-1}\rangle} \sum_s \sum_\theta \beta^\pi_t(s,\theta) \cdot \tilde{\eta}_t(s,\theta)$, then η_t can serve as the support occupancy state of hyperplane $\beta^{\hat{\pi}}_t$.*

Proof. The proof follows directly from Definition 36 and Lemma 11. If we let E_η and E_β be probabilistic and value-based equivalence relations induced by occupancy state η and hyperplane β, and further assume

η is a support occupancy state of β, then both compact hyperplanes β_{E_β} and β_{E_η} of β yield the same values for any arbitrary occupancy state η'. That is,

$$\sum_s \sum_\ell \beta_{E_\eta}(s,\ell) \cdot \eta'_{E_\eta}(s,\ell) \quad = \quad \sum_s \sum_{\ell'} \beta_{E_\beta}(s,\ell') \cdot \eta'_{E_\beta}(s,\ell').$$

This ends the proof. □

It is worth noticing that histories that are value-based equivalent with respect to E_β, need not be probabilistically equivalent with respect to E_η. For this reason we say that E_η is a weaker value-based equivalence relation than E_β.

Next, we present a novel algorithm, namely *the feature-based heuristic search value iteration* (FB-HSVI) algorithm, that synthesizes all the contributions of this paper including: the transformation of Dec-POMDPs into occupancy-state MDPs; the piecewise-linearity and convexity property of the optimal value functions; and the feature-based compact representations of all occupancy states, separable decision rules, and value functions.

5 Feature Based Heuristic Search

This section presents a family of algorithms we call feature-based heuristic search algorithms. These algorithms complement A*, LRTA*, or HSVI algorithms to exploit: the piecewise-linearity and convexity; and the feature-based compact representations of all occupancy states, decision rules, and vector values. In particular, this section demonstrates how to accurately fill the upper and lower bound value functions so as to ensure they prescribe an optimal separable policy, starting with initial information state η_0.

5.1 Feature-Based Heuristic Search Value Iteration

The problem is no longer that of the generalization from examples. The key issue is that of making the search for better approximate value functions as efficiently as possible. Fortunately, efficient search methods, collectively known as *heuristic search*, have been extensively studied in artificial intelligence (Korf, 1990). So there is no need to re-invent totally new methods. To a large extent we need only combine piecewise-linear and convex approximation architectures with existing heuristic search methods. The idea of combining piecewise-linear and convex approximation architectures and heuristic search is not completely new, the HSVI algorithm is a notable example.

We present the feature-based heuristic search value iteration (FB-HSVI) algorithm. FB-HSVI builds upon LRTA* (Korf, 1990) and HSVI (Smith and Simmons, 2004). Similarly to HSVI, it proceeds by creating trajectories of states, based on upper $\bar{\upsilon}_{M,t}$ and lower $\underline{\upsilon}_{M,t}$ bounds on the optimal value function $V^*_{M,t}$, for all $t \in \{0, 1, \cdots, T\}$. Each such trajectory starts from the initial state. It always executes the best action specified by the upper bound, and then selects the next state, which results from the current state and the next action. When the trajectory is finished the states are updated. As it proceeds, the bounds become tighter and the trajectories improve. It terminates after a finite number of trials and returns lower-bound value functions arbitrary closer to the optimal value functions.

FB-HSVI remains fundamentally different from HSVI in many aspects. Unlike HSVI, it replaces the high-dimensional joint history space by a lower dimensional feature set — providing us with compact representations for states, actions and vectors. Such compact representations often weaken lower and upper bound value functions over unvisited states. Yet, they preserve the ability to eventually finding the optimal value functions over visited states. FB-HSVI also replaces the brute-force action selection by a much efficient selection procedure. It recasts the action selection into a weighted constraint satisfaction problem (WCSP) (Schiex et al., 1995; Bistarelli et al., 1997; Dechter, 2003), which permits us to circumvent the exhaustive enumeration of all possible actions as suggested so far. Finally, it exploits the

Algorithm 9: The FB-HSVI Algorithm.

function `FB-HSVI()`
  initialize $\underline{\upsilon}_{M,t}$ and $\bar{\upsilon}_{M,t}$ for all $t \in \{0, \cdots, T-1\}$.
  while $\neg$***Stop***$(\eta_0, 0)$ **do** `Explore`$(\eta_0, 0)$

function `Explore`(η_t, g_t)
  $\tilde{\eta}_t \leftarrow$ **Compact**(η_t).
  if $\neg$***Stop***$(\tilde{\eta}_t, g_t)$ **then**
    $d_t^* \in \arg\max_{d_t} \boldsymbol{R}(\tilde{\eta}_t, d_t) + \bar{\upsilon}_{t+1}(\boldsymbol{P}(\tilde{\eta}_t, d_t))$.
    Update $\bar{\upsilon}_{M,t}$.
    `Explore`$(\boldsymbol{P}(\tilde{\eta}_t, d_t^*), \boldsymbol{R}(\tilde{\eta}_t, d_t^*) + g_t)$.
    Update $\underline{\upsilon}_{M,t}$.
  return g_t

function `Stop`(η_t, g_t)
  if $\bar{\upsilon}_{M,t}(\eta_t) > \underline{\upsilon}_{M,t}(\eta_t)$ **then return** $g_t + \bar{\upsilon}_{M,t}(\eta_t) \leq \underline{\upsilon}_{M,t}(\eta_0)$
  return *true*

deterministic nature of the occupancy MDP. In fact, it uses stopping criteria from classical planning, *e.g.,* LRTA* (Korf, 1990). This circumvents the need to explore unnecessary regions of the search space.

These ideas together provide FB-HSVI with the ability to: first, scaling up to occupancy MDPs of unprecedented size; second, enhancing the generalization of the value functions over unvisited states; and speeding up the convergence to approximate value functions that can prescribe an optimal separable policy. Next, we describe lower and upper bounds, their initialization, and update rules.

5.1.1 Lower-Bound Value Functions

This section describes the lower-dimensional representation we use for lower-bound value functions. We build upon the vector set representation and exploit the probabilistic and value-based equivalence relations. Where the classical representation involves a set of full vector values, we use feature-based compact vector values.

Definition 37. *The t-step* ***lower-bound value function*** $\underline{\upsilon}_{M,t}$ *is given by a vector set* Λ_t*, where each vector* $\hat{\beta}_t$ *consists of a triplet* $(\beta_{t,E}, \zeta_E, \underline{\upsilon}_t)$*, such that, for any occupancy state* η_t*, we have:*

$$\begin{aligned} \underline{\upsilon}_{M,t}(\eta_t) &= \max_{\hat{\beta}_t \in \Lambda_t} \langle \eta_t, \hat{\beta}_t \rangle, \\ \hat{\beta}_t(s,\theta) &= \begin{cases} \beta_{t,E}(\zeta_E(\theta)) & \text{If } \theta \in \Theta', \\ \underline{\upsilon}_t & \text{Otherwise.} \end{cases} \end{aligned}$$

where $\langle \eta_t, \hat{\beta}_t \rangle = \sum_s \sum_\theta \eta_t(s,\theta) \cdot \hat{\beta}_t(s,\theta)$.

Remark 3. *For any arbitrary feature-based compact vector value* $\hat{\beta}_t \in \Lambda_t$*, and any arbitrary occupancy state* η_t*, we have* $\langle \eta_t, \hat{\beta}_t \rangle \leq V^*_{M,t}(\eta_t)$.

Relative to full vector values, feature-based compact vector values yield accurate values for a subset of occupancy states, especially for occupancy states that serve as support occupancy states. It is worth noticing that feature-based compact vector values provide elsewhere lower bounds weaker than that of full vector values. The feature-based compact vector values can nonetheless accurately represent optimal value functions, but may require more vectors than the classical representation. A more formal proof follows in Section 5.2, but we start with Algorithm 10 for initialization, evaluation $\boldsymbol{e}^{\text{LB}}$, and update $\boldsymbol{u}^{\text{LB}}$, respectively.

Initialization. We initialize the lower-bound value function, especially parameter $(\underline{v}_t)_{t\in\{0,1,\ldots,T-1\}}$ of every vector $\hat{\beta}_t \equiv (\beta_{t,E}, \zeta_E, \underline{v}_t)$, using a trivial lower-bound value. This lower-bound value corresponds to the minimum expected reward to be gained starting from runtime t onward.

Evaluation e^{LB}. The evaluation of the lower-bound value function at occupancy state η_t is given by $e^{\text{LB}}(\eta_t)$, and consists of the maximum scalar product $\langle \eta_t, \hat{\beta}_t \rangle$ between occupancy state η_t and each vector $\hat{\beta}_t \in \Lambda_t$, for all $t \in \{0, 1, \ldots, T-1\}$.

Algorithm 10: Lower bound initialization, evaluation and update operators

function INITIALIZATION()
  $\underline{v}_t \leftarrow (T-t)\min_{s\in S}\min_{a\in A}\ r^a(s) \quad \forall t \in \{0, 1, \cdots, T\}$

function $e^{\text{LB}}(\eta_t)$
  return $\max_{\hat{\beta}_t\in\Lambda_t}\ \langle \eta_t, \hat{\beta}_t \rangle$

function $u^{\text{LB}}(\hat{\eta}_t, \hat{d}_t)$
  $\hat{\beta}^*_{t+1} \leftarrow \arg\max_{\hat{\beta}_{t+1}\in\Lambda_{t+1}} \langle F(\hat{\eta}_t, \hat{d}_t), \hat{\beta}_{t+1} \rangle$.
  $\mathscr{L} \leftarrow$ joint-history labels reachable from $\hat{\eta}_t$
  for $s \in S$ *and* $\ell \in \mathscr{L}$ **do**
    $\beta_{t,E_{\eta_t}}(s,\ell) \leftarrow r^{\hat{d}_t(\ell)}(s) + \sum_{s'\in S}\sum_{z\in Z} p^{\hat{d}_t(\ell),z}(s,s') \cdot \hat{\beta}^*_{t+1}(\ell, \hat{d}_t(\ell), z)(s')$.
  return $\Lambda_t \cup \{(\beta_{t,E_{\eta_t}}, \zeta_{E_{\eta_t}}, \underline{v}_t)\}$

Update u^{LB}. Finally, the update of the lower-bound value function after selecting separable decision rule d_t in occupancy state η_t is given by sub-routine $u^{\text{LB}}(\eta_t, d_t)$. This operation consists in a three-step method. First, we select an intermediate vector $\hat{\beta}^*_{t+1}$ that is maximal at $\boldsymbol{P}(\eta_t, d_t)$. Next, we create the compact vector $\hat{\beta}_t$ that is the sum of vector β^{d_t} and vector $\hat{\beta}^*_{t+1}$. Finally, we add the compact representation $(\beta_{t,E_{\eta_t}}, \zeta_{E_{\eta_t}}, \underline{v}_t)$ of vector $\hat{\beta}_t$ into vector set Λ_t. It is worth noticing that E_{η_t} can be either the local probabilistic equivalence relation or the probabilistic truncation equivalence relation induced by occupancy state η_t.

5.1.2 Upper-Bound Value Functions

In the following, we describe the lower-dimensional representation we use for the upper-bound value functions. We build upon the point-set representation and exploit the probabilistic and value-based equivalence relations. Where classical representation involves a set of full state/value pairs, we use a set of feature-based compact state/value pairs instead.

Definition 38. *The t-step **upper-bound value function** $\bar{v}_{M,t}$ is given by a point set Υ_t, including corner and non-corner points. Each point consists in a pair $(\hat{\eta}_t \equiv (\eta_{t,E}, \zeta_E), v_t)$ of a t-step compact occupancy state and a value, such that, $V^*_{M,t}(\hat{\eta}_t) \leq v_t$.*

Relative to full state/value pairs, feature-based compact state/value pairs yield accurate values for a subset of occupancy states, especially for those that are equivalence to occupancy states in the point set. It is worth noticing that feature-based compact occupancy states provide elsewhere upper bounds weaker than that of full state/value pairs. The feature-based compact points can nonetheless accurately represent optimal value functions, but again may require more points than the classical point-set representation. A more formal proof follows in Section 5.2, but we start with Algorithm 11 for initialization, evaluation e^{UB}, and update u^{UB}, respectively.

Initialization. The t-step initial upper-bound value function $\bar{v}_{M,t}$ consists of corner points. We group these points into mapping β_t^0, initially $\beta_t^0(s,\theta) = V_{M,\hat{\pi}}(s)$ for any state s and joint history θ, where $\hat{\pi}$ denotes an optimal policy for a relaxation of the original problem.

Evaluation e^{UB}. The evaluation of any occupancy state η_t is based on the sawtooth extrapolation-interpolation method over all non-corner points in Υ_t. The difference with the classical sawtooth extrapolation-interpolation is that, the input occupancy state η_t is transformed into a feature-based compact occupancy state $\eta_{t,E}$ using transformation rule ζ_E^l for non-corner point $((\eta_{t,E}^l, \zeta_E^l), v_t^l)$. This permits η_t to get closer to $\eta_{t,E}^l$ and sometimes provides tighter upper-bound values.

Algorithm 11: Upper bound initialization, evaluation and update operators

function INITIALIZATION(π^{mdp})
  for $s \in S$ *and* $\theta \in \Theta_t$ **do** $\beta_t^0(s,\theta) \leftarrow V_{M,\pi^{\text{mdp}}}(s)$.

function SAWTOOTH(η_t)
  $y_0 \leftarrow \sum_s \sum_\theta \eta_t(s,\theta) \cdot \beta_t^0(s,\theta)$
  forall the $l \in \{1, 2, \cdots, L_t\}$ **do**
    $v^0 \leftarrow \sum_s \sum_\ell \eta_{t,E}^l(s,\ell) \cdot \beta_t^0(s,\ell)$
    $\xi^l \leftarrow \min\{\eta_{t,E}(s,\ell)/\eta_{t,E}^l(s,\ell) | \forall s, \forall \ell, \eta_{t,E}^l(s,\ell) > 0\}$
    $y_l \leftarrow y_0 + (v^l - v^0) \cdot \xi^l$
  return $\min_{l \in \{0,\ldots,L_t\}} y_l$

function e^{UB}(η_t)
  return SAWTOOTH(η_t)

function u^{UB}($\hat{\eta}_t$)
  $\hat{d}_t \leftarrow \arg\max_{d_t} \mathbf{R}(\hat{\eta}_t, d_t) + V_{M,t+1}(\mathbf{P}(\hat{\eta}_t, d_t))$ — select greedy separable decision rule
  return $\Upsilon_t \cup \{(\hat{\eta}_t, \mathbf{R}(\hat{\eta}_t, \hat{d}_t) + V_{M,t+1}(\mathbf{P}(\hat{\eta}_t, \hat{d}_t)))\}$

Update u^{UB}. The upper-bound update rule is a three-step algorithm. First, it selects the greedy separable decision rule $\hat{d}_t$ based on the current feature-based compact occupancy state $\hat{\eta}_t$. Next, it computes the upper-bound value at occupancy state $\hat{\eta}_t$ based on separable decision rule $\hat{d}_t$. Finally, it add state/valie pair $(\hat{\eta}_t, \hat{d}_t)$ into the point set. In the case, this point is a corner point, we update mapping β_t^0; otherwise, we add it into the point set Υ_t. It is worth noticing that the selection of the greedy separable decision rule can be made much faster by replacing the brute force selection by a constraint-based optimization method.

5.2 Theoretical Analysis

In this section, we discuss two important theoretical properties of FB-HSVI: convergence to an optimal solution, and an online bound of the current solution.

5.2.1 Optimality Preserving Transformations

In this section, we demonstrate that our lower and upper bound representations, initialization, evaluation, and update transformations preserve the optimality.

We start by demonstrating that the representations we use are valid lower and upper bound representations. That is, they are guarantee to always lower bound (respectively upper bound) the optimal value function at any occupancy states.

Lemma 13. *Let $\boldsymbol{e}^{\text{LB}}\colon (\triangle \mapsto \mathbb{R}) \times \triangle \mapsto \mathbb{R}$ be the lower-bound evaluation rule. The t-step value function $\underline{\upsilon}_{M,t}$ is a lower-bound value function of the optimal value function $V^*_{M,t}$ over the entire t-step occupancy space $\triangle_t$.*

Proof. To prove this result, we consider any arbitrary occupancy state $\eta_t \in \triangle_t$ and show that the following inequality holds $\boldsymbol{e}^{\text{LB}}(\underline{\upsilon}_{M,t}, \eta_t) \leq V^*_{M,t}(\eta_t)$. Consider the set of support occupancy states $\underline{\triangle}_t = \{\eta^l_t\colon l \in \{1,\dots,L\}\}$ associated with vector values in $\Lambda_t = \{\beta^l_t\colon l \in \{1,\dots,L\}\}$, where η^l_t is the support occupancy state of β^l_t. Let $p_{\underline{\triangle}_t}(\eta_t)$ be the projection of η_t onto $\underline{\triangle}_t$, that is there exists $(c_l)_{l\in\{1,\dots,L\}}$, such that $p_{\underline{\triangle}_t}(\eta_t) = \sum_l c_l \cdot \eta^l_t$, where $c_l \geq 0$, and $\sum_l c_l = 1$. Thus, occupancy state η_t can be rewritten $\eta_t = p_{\underline{\triangle}_t}(\eta_t) + p_{\triangle\setminus\underline{\triangle}_t}(\eta_t)$. Intuitively, by convexity lower bound at η_t lower bounded by the sum of lower bounds of $p_{\underline{\triangle}_t}(\eta_t)$ and $p_{\triangle\setminus\underline{\triangle}_t}(\eta_t)$. By construction, the lower bound at $p_{\underline{\triangle}_t}(\eta_t)$ is given by Λ_t, whereas lower bound at $p_{\triangle\setminus\underline{\triangle}_t}(\eta_t)$ is given by trivial lower bound v_t. This is mainly because, we know by hypothesis that $\sum_s \sum_\theta \eta^0_t(s,\theta) \cdot \underline{\upsilon}_t \leq V^*_{M,t}(\eta^0_t)$ and $\boldsymbol{e}^{\text{LB}}(\underline{\upsilon}_{M,t}, \eta^l_t) \leq V^*_{M,t}(\eta^l_t)$, for all $l \in \{1, 2, \cdots, L\}$. Thus, we have:

$$
\begin{array}{lcll}
V^*_{M,t}(\eta_t) & \geq & c_0 \sum_s \sum_\theta \eta^0_t(s,\theta) \cdot \underline{\upsilon}_t + \sum_{l=1}^L c_l \cdot \boldsymbol{e}^{\text{LB}}(\underline{\upsilon}_{M,t}, \eta^l_t), & \text{(by convexity)} \\
& \geq & c_0 \sum_s \sum_\theta \eta^0_t(s,\theta) \cdot \underline{\upsilon}_t + \max_{\beta_t} \sum_{l=1}^L c_l \cdot \sum_{s,\theta} \beta_t(s,\theta) \cdot \eta^l_t(s,\theta), & \text{(max operator)} \\
& = & c_0 \sum_s \sum_\theta \eta^0_t(s,\theta) \cdot \underline{\upsilon}_t + \max_{\beta^l_t} \sum_{s,\theta} \beta^l_t(s,\theta) \cdot (\eta_t - c_0 \cdot \eta^0_t)(s,\theta), & \text{(rearranging terms)} \\
& \geq & \max_{\beta^l_t} \sum_{s,\theta} \beta^l_t(s,\theta) \cdot \eta_t(s,\theta), & (\underline{\upsilon}_t \leq \beta^l_t(s,\theta), \forall s,\theta) \\
& = & \boldsymbol{e}^{\text{LB}}(\underline{\upsilon}_{M,t}, \eta_t). &
\end{array}
$$

This ends the proof. □

This lemma ensures the value function (Definition 37) represents a lower bound of the optimal value function.

Lemma 14. *Let $\underline{\upsilon}_{M,t}\colon \triangle_t \mapsto \mathbb{R}$ be a lower-bound value function of the optimal value function $V^*_{M,t}\colon \triangle_t \mapsto \mathbb{R}$, and Λ_t be the associated vector set. Let $\boldsymbol{u}^{\text{LB}}\colon (\triangle \mapsto \mathbb{R}) \times \triangle \times D \mapsto (\triangle \mapsto \mathbb{R})$ be the lower-bound update rule. Then update rule $\boldsymbol{u}^{\text{LB}}$ transforms any lower-bound value function $\underline{\upsilon}_{M,t}$ into a lower-bound value function, for all $t \in \{0, 1, \cdots, T-1\}$.*

Proof. Let η_t be any arbitrary occupancy state. Let d^*_t be a greedy separable decision rule given η_t and $\bar{\upsilon}_{M,t+1}$. Let β_{η_t} be the vector value, update rule $\boldsymbol{u}^{\text{LB}}$ adds to Λ_t.

In demonstrating this property, we prove that for any t-step occupancy state y, we have $\underline{\upsilon}_{M,t}(y) \leq \boldsymbol{e}^{\text{LB}}(\boldsymbol{u}^{\text{LB}}(\eta_t, d^*_t), y) \leq V^*_{M,t}(y)$. To demonstrate $\underline{\upsilon}_{M,t}(y) \leq \boldsymbol{e}^{\text{LB}}(\boldsymbol{u}^{\text{LB}}(\underline{\upsilon}_{M,t}, \eta_t, d^*_t), y)$, we note that:

$$
\begin{array}{lcll}
\boldsymbol{e}^{\text{LB}}(\boldsymbol{u}^{\text{LB}}(\underline{\upsilon}_{M,t}, \eta_t, d^*_t), y) & = & \max_{\beta_t \in \Lambda_t \cup \{\beta_{\eta_t}\}} \langle y, \beta_t \rangle, & \text{(definitions of } \boldsymbol{u}^{\text{LB}} \text{ and } \boldsymbol{e}^{\text{LB}}) \\
& \geq & \max_{\beta_t \in \Lambda_t} \langle y, \beta_t \rangle, & \text{(definition of max)} \\
& = & \underline{\upsilon}_{M,t}(y). &
\end{array}
$$

This shows that $\boldsymbol{u}^{\text{LB}}(\eta_t, d^*_t)$ dominates $\underline{\upsilon}_{M,t}$. To demonstrate $\boldsymbol{e}^{\text{LB}}(\boldsymbol{u}^{\text{LB}}(\eta_t, d^*_t), y) \leq V^*_{M,t}(y)$, we distinguish between the lower-bound value function before and after the update:

$$
\begin{array}{lcll}
\boldsymbol{e}^{\text{LB}}(\underline{\upsilon}_{M,t}, y) & = & \underline{\upsilon}_{M,t}(y), & \text{(definition of } \boldsymbol{e}^{\text{LB}}) \\
& \leq & V^*_{M,t}(y), & \text{(definition of lower bound)}
\end{array}
$$

$$
\begin{array}{lcll}
\langle y, \beta_{\eta_t} \rangle & = & \boldsymbol{R}(y, d^*_t) + \underline{\upsilon}_{t+1}(\boldsymbol{P}(y, d^*_t)), & \text{(definition of } \beta_{\eta_t}) \\
& \leq & \boldsymbol{R}(y, d^*_t) + V^*_{M,t+1}(\boldsymbol{P}(y, d^*_t)), & \text{(definition of lower bound)} \\
& \leq & \max_{d_t \in \boldsymbol{A}} \boldsymbol{R}(y, d_t) + V^*_{Mt+1}(\boldsymbol{P}(y, d_t)), & \text{(definition of max)} \\
& \leq & V^*_{M,t}(y). & \text{(optimality equations)}
\end{array}
$$

Together these results establish that $\boldsymbol{u}^{\text{LB}}(x, d_t^*)$ is upper bounded by $V_{M,t}^*$. □

This lemma demonstrates the update rule does not weaken the lower bound value function over the entire occupancy states. Although it does not ensure lower bound improvement, it can improve the lower bound value function over a small subset of occupancy states. A formal proof of convergence to an optimal value function follows in Section 5.2.2.

Lemma 15. *Let $\boldsymbol{e}^{\text{UB}}\colon (\triangle \mapsto \mathbb{R}) \times \triangle \mapsto \mathbb{R}$ be the upper-bound evaluation rule. The t-step value function $\bar{\upsilon}_{M,t}$ is a upper-bound value function of the optimal value function $V_{M,t}^*$ over the entire t-step occupancy space $\triangle_t$, for all $t \in \{0, 1, \ldots, T-1\}$.*

Proof. To prove this property, we need to show that for any arbitrary occupancy state η_t, inequality $\boldsymbol{e}^{\text{UB}}(\bar{\upsilon}_{M,t}, \eta_t) \geq V_{M,t}^*(\eta_t)$ holds. To this end, notice that, η_t can be rewritten as a convex combination between any compact occupancy state η_t^l in Υ_t and a positive vector η_t^0, such that, $\eta_t = \xi_l \cdot \eta_t^l + \eta_t^0$. But we know by hypothesis that $\langle \eta_t, \beta_t^0 \rangle \geq V_{M,t}^*(\eta_t)$ and $\beta_t^l \geq V_{M,t}^*(\eta_t^l)$, where (η_t^l, β_t^l) constitutes a non-corner point in Υ_t. Due to convexity,

$$\begin{array}{rcll} V_{M,t}^*(\eta_t) & \leq & \xi_l \cdot \beta_t^l + \sum_s \sum_\theta \eta_t^0(s,\theta) \cdot \beta_t^0(s,\theta), & \text{(convexity)} \\ & = & c_l \cdot \beta_t^l + \sum_s \sum_\theta (\eta_t(s,\theta) - \xi_l \cdot \eta_t^l(s,\theta)) \cdot \beta_t^0(s,\theta), & \text{(replacing } \eta_t^0\text{)} \end{array}$$

In the last expression, ξ_l is the only unknown variable. To get the best upper-bound value, we wish to find the maximum value of ξ_l, that is consistent with our assumptions. Especially, we need to find ξ_l such that: $\eta_t^0(s,\theta) \geq 0$ and $\eta_t^l(s,\theta) \geq 0$, for any state s and joint history θ. Since $\eta_t^0(s,\theta) = \eta_t(s,\theta) - \xi_l \eta_t^l(s,\theta)$, we obtain expression: $\xi_l = \min_{s,\theta}\{\eta_t(s,\theta)/\eta_t^l(s,\theta) \colon \eta_t^l(s,\theta) > 0\}$. This ends the proof. □

This lemma ensures the value function (Definition 38) represents an upper bound of the optimal value function.

Lemma 16. *Let $\bar{\upsilon}_{M,t}\colon \triangle_t \mapsto \mathbb{R}$ be a upper-bound value function of the optimal value function $V_{M,t}^*\colon \triangle_t \mapsto \mathbb{R}$, and Υ_t be the associated vector set. Let $\boldsymbol{u}^{\text{UB}}\colon (\triangle \mapsto \mathbb{R}) \times \triangle \times D \mapsto (\triangle \mapsto \mathbb{R})$ be the upper-bound update rule. The update operator $\boldsymbol{u}^{\text{UB}}$ transforms any upper-bound value function $\bar{\upsilon}_{M,t}$ into an improved upper-bound value function.*

Proof. In demonstrating this result, we need to prove that for any arbitrary t-step occupancy states x and y, the following holds: $V_{M,t}^*(y) \leq \boldsymbol{e}^{\text{UB}}(\boldsymbol{u}^{\text{UB}}(x), y) \leq \bar{\upsilon}_{M,t}(y)$. Let (x, β_t^x) be the point that update $\boldsymbol{u}^{\text{UB}}(\bar{\upsilon}_{M,t}, x)$ produces. Let β_t^y be the value that results from the sawtooth interpolation with respect to non-corner point (x, β_t^x), corner point set β_t^0 and occupancy state y. Thus, we have:

$$\begin{array}{rcll} \boldsymbol{e}^{\text{UB}}(\boldsymbol{u}^{\text{UB}}(x), y) & = & \min\left\{\boldsymbol{e}^{\text{UB}}(y), \beta_t^y\right\}, & \text{(definitions of } \boldsymbol{e}^{\text{UB}} \text{ and } \boldsymbol{u}^{\text{UB}}\text{)} \\ & \leq & \boldsymbol{e}^{\text{UB}}(y), & \text{(definition of operator min)} \\ & = & \bar{\upsilon}_{M,t}(y). & \end{array}$$

This shows that $\bar{\upsilon}_{M,t}$ dominates $\boldsymbol{u}^{\text{UB}}(x)$. To show inequality $V_{M,t}^*(y) \leq \boldsymbol{e}^{\text{UB}}(\boldsymbol{u}^{\text{UB}}(x), y)$, we distinguish between the upper-bound value function before and after the update:

$$\begin{array}{rcll} \boldsymbol{e}^{\text{UB}}(y) & = & \bar{\upsilon}_{M,t}(y), & \text{(definition of } \boldsymbol{e}^{\text{UB}}\text{)} \\ & \geq & V_{M,t}^*(y), & (\bar{\upsilon}_{M,t} \text{ dominates } \upsilon_t^*) \\ & & & \\ \beta_t^y & \geq & V_{M,t}^*(y). & \text{(definition of } \beta_t^y\text{)} \end{array}$$

Together these results establish that $\boldsymbol{u}^{\text{UB}}(x)$ dominates $V_{M,t}^*$. □

This lemma demonstrates the upper bound update rule does not weaken the upper bound value function over the entire occupancy states. However, this lemma does not ensure upper bound improvement, it can nonetheless improve the upper bound value function over a small subset of occupancy states. A formal proof of the convergence to an optimal value function follows.

5.2.2 Convergence of FB-HSVI

We start by establishing the stopping criteria FB-HSVI uses to terminate the search. Let $r_{M,0}(\eta_t)$ be the sum of reward cumulated throughout the path from the initial state η_0 to information state η_t. We call admissible evaluation function, quantity $f(\eta_t) = r_{M,0}(\eta_t) + \bar{v}_{M,t}(\eta_t)$, which gives an upper bound on the reward of any solution path that is an extension of the current best path to information state η_t. We further denote $\underline{v}_{M,0}(\eta_0)$ the reward of the current best solution. Then, $\underline{v}_{M,0}(\eta_0)$ is a lower bound on the reward of an optimal solution. Clearly, there is no reason to expand an information state that has an f-reward (*i.e.,* upper bound) less than or equal to the current lower bound, $\underline{v}_{M,0}(\eta_0)$, since it cannot lead to an improved solution. Thus, we have the following convergence test for FB-HSVI: the best solution found so far is optimal if there are no unexpanded information states on the search frontier with an f-reward lower than $\underline{v}_{M,0}(\eta_0)$. Another useful stopping criterion relies on the fact that there is no reason to expand an information state η_t that has an upper bound equal to its current lower bound, $\bar{v}_{M,t}(\eta_t) = \underline{v}_{M,t}(\eta_t)$, since it already yields its optimal value.

To demonstrate the theoretical guarantees of FB-HSVI, we introduce the notions of *finished* and *not finished* information states. An information state is finished if either stopping criterion holds, otherwise it is not finished. Clearly, all information states η_T at time T are finished.

Theorem 7. *The FB-HSVI algorithm always terminates after a finite number of trials and the last solution it finds is optimal.*

Proof. First we show that the algorithm cannot terminate before an optimal solution is found. Suppose that the algorithm terminates before finding an optimal solution which has reward f^*. The sequence of lower bounds used during the planning phase is $l_0, l_1, \cdots, l_k$, where l_0 is initial lower bound before any solution is found, l_1 is the reward of the first solution found, and l_k that of the last solution found. We know by hypothesis that, $l_0 < l_1 < \cdots < l_k < f^*$, where the last inequality holds under the assumption that FB-HSVI terminates with a suboptimal solution.

Now consider an optimal path $\eta^0, \eta^1, \cdots, \eta^T$ leading from the initial information state to a terminal information state. Under the assumption that this optimal path was not found, there must be some information state η along this path that was generated but not expanded. That is only possible if $f(\eta) \leq l_k$. But by the admissibility of f, we know that $f(\eta) \geq f^*(\eta)$ and therefore $f(\eta) \geq f^*(\eta) > l_k$, for any arbitrary $m \in \{0, 1, \cdots, k\}$. From this contradiction, it follows that FB-HSVI cannot terminate before an optimal solution is found.

Then we show that FB-HSVI trials will cause at least one information state that is not finished to become finished. Suppose FB-HSVI has not yet terminate and a trial is executed. Let the final information states encountered during the forward expansion be η_t and η_{t+1}. Given that the trial terminates at η_{t+1}, we know that before the trial, η_t was not finished but η_{t+1} was finished. Because η_{t+1} results from the action selection at η_t, we know that η_t will be finished after it is updated: either η_{t+1} yields its optimal value, then η_t will also yield its optimal value after it is updated; or η_{t+1} has an f-reward lower than $\underline{v}_{M,0}(\eta_0)$, then η_t will also have an f-reward lower than $\underline{v}_{M,0}(\eta_0)$ after it is updated. Thus, executing a trial cause information state η_t which was not finished to become finished.

Finally, we show that FB-HSVI terminates after a finite number of trials. To this end, we note that the search graph of the FB-HSVI is a tree, with a fixed branching factor $|D_t|$ at depth $t \in \{0, 1, \cdots, T\}$. By hypothesis, only information states that appear in depth $t < T$ are not finished. Thus, the total number of information states $|\cup_{t=0}^{T-1} \mathcal{I}_t|$ at all depths up to T is

$$|\cup_{t=0}^{T-1} \mathcal{I}_t| \quad = \quad |A^*|^{\frac{|I||Z^*|^T|A^*|^T-1}{|Z^*||A^*|-1}} \quad \text{(see Table 2).} \tag{36}$$

Given that at least one information state becomes finished after each trial, thus the initial information state η_0 must become finished after at most $|\cup_{t=0}^{T-1} \mathcal{I}_t|$ trials, causing FB-HSVI to terminate. □

Another important property of FB-HSVI is that it refines both the upper and lower bound value functions throughout the algorithm. Since information state expansions are interleaved with updates, FB-HSVI offers an anytime solution. Cutting off FB-HSVI trials at anytime, we know that the difference between the current best solution and the optimal solution is bounded.

Theorem 8. *At anytime throughout the FB-HSVI algorithm, the current solution is within ε of the optimal solution, where $\varepsilon = \bar{\upsilon}_{M,0}(\eta_0) - \underline{\upsilon}_{M,0}(\eta_0)$.*

Proof. Formally, the regret of executing a separable policy π instead of an optimal separable policy π^* is written as follows:

$$\begin{aligned}
regret(\pi) &\overset{\text{def}}{=} V_{M,\pi^*}(\eta_0) - V_{M,\pi}(\eta_0), && \text{(definition of } regret\text{)}\\
&= V_{M,\pi^*}(\eta_0) - \underline{\upsilon}_{M,0}(\eta_0), && \text{(definition of } \underline{\upsilon}_{M,0}(\eta_0)\text{)}\\
&= V_{M,\pi^*}(\eta_0) - \underline{\upsilon}_{M,0}(\eta_0) + (\bar{\upsilon}_{M,0}(\eta_0) - \bar{\upsilon}_{M,0}(\eta_0)), && \text{(add zero)}\\
&= (V_{M,\pi^*}(\eta_0) - \bar{\upsilon}_{M,0}(\eta_0)) + (\bar{\upsilon}_{M,0}(\eta_0) - \underline{\upsilon}_{M,0}(\eta_0)), && \text{(rearranging terms)}\\
&\leq \bar{\upsilon}_{M,0}(\eta_0) - \underline{\upsilon}_{M,0}(\eta_0). && (V_{M,\pi^*}(\eta_0) \leq \bar{\upsilon}_{M,0}(\eta_0))
\end{aligned}$$

So, whenever FB-HSVI is interrupted its current solution is within ϵ of the optimal solution. □

6 Experiments

This section empirically demonstrates and validates the importance of our contributions: (*a*) the reformulation of decentralized stochastic control problems into a deterministic continuous-state Markov decision process; (*b*) the reformulation of the action selection into a weighted constraint satisfaction problem; (*c*) the piecewise linearity and convexity property of the optimal value function; and (*d*) probabilistic and value-based loss-less compact representations. The first goal of these experiments is to establish that together these contributions yield a scalable feature-based heuristic search algorithmic framework. This is accomplished by showing feature-based heuristic search algorithms can successfully solve all problems from the literature over unprecedented planning horizons. We also demonstrate that the feature-based heuristic search algorithms outperform all existing exact algorithms on all tested domains.

The remainder of this section is organized as follows. First, we introduce the experimental setup we use throughout this section. Next, we compare different feature-based heuristic search algorithms on a variety of domains from the literature. We also compare these feature-based heuristic search algorithms to existing exact algorithms from the literature. We further compare approximate variants of our feature-based heuristic search algorithms to previous approximate methods. Finally, we study the impact of the number of agents over the performance of the feature-based heuristic search algorithms.

6.1 Core Algorithmic Components

As discussed throughout this paper, there are many key components that can affect the performance of feature-based heuristic search value iteration algorithms, such as the (upper and lower) bound representations, bound update methods, the backup implementation, the history compression, the value generalization, or the initial upper bound heuristic. Feature-based heuristic search value iteration algorithms differ on one or more of these components. Table 3 provides a detailed mapping between the algorithms — as adapted from Section 3 — and their collection of algorithmic components. We define each algorithmic component in detail below.

Backup Implementation Methods. Recall that the backup operation consists of selecting greedy separable decision rule d_t^* given current occupancy state η_t and value function $\upsilon_{M,t+1}$. There is one simple

Algorithm	Backup	Bound Shape	Compression	Generalization
LRTA* (Korf, 1990)	BRUTE FORCE	TABULAR	NONE	NO
LRTA*-LPE	WCSP	POINT SET	LPE	YES
LRTA*-TPE	WCSP	POINT SET	TPE	YES
HSVI (Smith and Simmons, 2004)	BRUTE FORCE	VECTOR / POINT SET	NONE	YES
HSVI-LPE	WCSP	VECTOR / POINT SET	LPE	YES
HSVI-TPE	WCSP	VECTOR / POINT SET	TPE	YES

Table 3: A review of the selected feature-based heuristic search algorithms, where we identify the algorithmic components they use.

method for selecting a greedy separable decision rule, namely the *brute-force* method. This approach enumerates and evaluates all possible separable decision rules and pick one with the highest return: $d_t^* = \arg\max_{d_t} \boldsymbol{R}(\eta_t, d_t) + \upsilon_{M,t+1}(\boldsymbol{P}(\eta_t, d_t))$. Another method, we call the *weighted constraint satisfaction* approach, builds upon the reformulation of the brute-force method into a set of weighted constraint optimization problems. This method applies only if we allow the bounds to generalize values from one occupancy state to other occupancy states, using the SAWTOOTH representation for the upper bound and a set of vectors for the lower bound.

Bound Shapes. Once the greedy separable decision rule has been selected, the feature-based heuristic search algorithms need to choose where and how to store this information. There are a number of important considerations at this stage. First, the feature-based heuristic search algorithms need to choose the representation of their bounds among: tabular, vector-set, or point-set representations. Next, they also need to select the order in which they want the updates to be performed. The order in which updates are performed actually depends on the underlying algorithm we choose between LRTA* and HSVI. The LRTA* algorithm updates the upper bounds forward, following the order in which occupancy states are generated. The HSVI algorithm updates both the upper and lower bounds backward, that is, in the reverse order in which occupancy states are generated.

Occupancy State Compression Methods. The ability to maintain a concise representation of the occupancy states is another important feature of our algorithms. We distinguish between two key compact representations for the occupancy states, including: the *local probabilistic equivalence* relation (LPE) and the *truncation probabilistic equivalence* relation (TPE).

Value Generalization. By recasting decentralized partially observable Markov decision processes as occupancy Markov decision processes, we allow values of occupancy states to generalize from one another. The value generalization is possible only when we exploit the piece-wise linearity and convexity of the optimal value function. As such value functions that are represented as tabular functions cannot generalize values from one occupancy state to another.

6.2 Domains

In recent years, the decentralized stochastic control community has evolved a set of benchmark domains, each motivated by an interesting realistic application, such as knowledge discovery, space exploration, robot navigation, or network communication. We selected many of these benchmarks, with the goal of spanning the range of properties that may affect the performance of a feature-based heuristic search value iteration algorithm. Below, we review the selected domains and their properties, for further details see Appendix A.

	domain M parameters				$\|\Pi_{0:t}\|$ for different t		
	N	$\|S\|$	$\|A^i\|$	$\|Z^i\|$	2	5	10
Dec-Tiger	2	2	3	2	6561	3.43×10^{30}	1.39×10^{977}
Mabc	2	4	2	2	256	1.84×10^{19}	3.23×10^{616}
Grid-Small	2	16	5	2	390625	5.42×10^{44}	3.09×10^{1431}
Recycling-Robots	2	4	3	2	6561	3.43×10^{30}	1.39×10^{977}
Box Pushing	2	100	4	5	3.34×10^{7}	5.23×10^{940}	1.25×10^{2939746}
Mars Rovers	2	256	6	8	1.69×10^{14}	1.88×10^{7285}	$2.57 \times 10^{238723869}$

Table 4: Domain parameters and maximum number of separable policies per planning horizons.

6.3 Empirical Analysis of our Algorithms

All our feature-based heuristic search algorithms (Table 4) were implemented in the same framework, using identical basic operations, such as occupancy state and value function updates, and separable decision rule selection through point-based backups. The weighted constraint satisfaction problems were solved using the ToulBar2 toolbox[7]. A time limit was set to 1000ms.

6.4 Comparing to other planners

In this section, we provide insights on how the FB-HSVI algorithm compares to other exact and approximate solvers. For each domain in our selection of benchmarks, we compare the FB-HSVI algorithm to previous Dec-POMDP solvers, including: exact solvers — GMAA*-ICE (Oliehoek et al., 2013), IPG (Amato et al., 2009), MILP (Aras and Dutech, 2010), and LPC (Boularias and Chaib-draa, 2008); and approximate solvers — PBIP (Dibangoye et al., 2009a), and CBDP (Kumar and Zilberstein, 2010). IPG and LPC perform dynamic programming, GMAA*-ICE performs heuristic search and MILP is a mixed integer linear programming method. PBIP and CBDP are point-based approximate dynamic programming algorithms for solving finite-horizon Dec-POMDPs. Results for GMAA*-ICE (provided by Matthijs Spaan), IPG, MILP, LPC, PBIP, and CBDP were conducted on different machines. Because of this, the timing results are not directly comparable to the other methods, but are likely to only differ by a small constant factor.

6.4.1 Comparing to other exact planners

Table 5 shows performance results for exact algorithms. For each algorithm we reported the computation time, which includes the time to compute heuristic values when appropriate since all algorithms do not use the same heuristics. We also reported the best expected cumulative reward $V_{M,0}(\eta_0)$ at the initial occupancy state. Table 5 clearly shows that the FB-HSVI algorithm allows for significant improvement over the state-of-the-art solvers: for all tested benchmarks we provide results for longer horizons than have been solved previously (the bold entries).

There are several reasons for the FB-HSVI algorithm's performance: first, we search in the space of policies mapping lower-dimensional features to actions, whereas the other exact solvers search in the space of policies mapping full histories to actions; we use a value function mapping occupancy states to reals allowing it to generalize the value function over unvisited occupancy states whereas all other solvers use value functions mapping partial policies to reals; finally, we replace the brute-force backup by an efficient weighted constraint satisfaction approach.

The FB-HSVI algorithm performs best when the domain possesses a structure that results in a compact memory-bounded parametric value function, as in the recycling robot and mabc domains. For the

[7] ToulBar2 is an open source WCSP solver, for further details see https://mulcyber.toulouse.inra.fr/projects/toulbar2/.

T	MILP	LPC	IPG	ICE	**FB-HSVI**	$V_{0,M}(\eta_0)$
MULTI-AGENT TIGER						
2	–	0.17	0.32	0.01	**0.03**	−4.00
3	4.9	1.79	55.4	0.01	**0.40**	5.2908
4	72	534	2286	108	**1.36**	4.8027
5				347	**9.65**	7.0264
6					**24.42**	10.381
7					**33.11**	**9.9935**
8					**41.21**	**12.217**
9					**58.51**	**15.572**
10					**65.57**	**15.184**
20						**28.75**
30						**51.90**
40						**67.09**
50						**80.66**
100						**170.90**
RECYCLING ROBOT						
2	–	–	0.30	36	**0.01**	7.000
3	–	–	1.07	36	**0.10**	10.660
4	–	–	42.0	72	**0.30**	13.380
5	–	–	1812	72	**0.34**	16.486
10					**0.52**	31.863
30					**1.13**	93.402
70					**2.13**	216.47
100					**2.93**	**308.78**
MEETING IN A 3X3 GRID						
2	–	–	–	–	**0.03**	0.0
3	–	–	–	–	**0.04**	0.133
4	–	–	–	–	**0.79**	0.432
5	–	–	–	–	**1.30**	0.894
6					**1.88**	**1.491**
7					**2.55**	**2.19**
8					**18.06**	**2.96**
9					**24.39**	**3.80**
10					**34.42**	**4.68**
20					**291.1**	**14.35**
30					**456.6**	**24.33**
40						**34.33**
50						**44.32**
100						**94.24**
BROADCAST CHANNEL						
2	–	–	–	–	**0.02**	2
3	–	–	–	–	**0.22**	2.99
4	–	–	–	–	**0.32**	3.89
5	–	–	–	–	**0.33**	4.79
10					**0.78**	9.29
30					**14.0**	27.42
50					**41.7**	45.50
100					**473.3**	90.76
1000						**903.28**

T	MILP	LPC	IPG	ICE	**FB-HSVI**	$V_{0,M}(\eta_0)$
GRID SMALL						
2	0	–	0	36	**0**	0.37
3	0.65	–	0.18	36	**0.1**	0.91
4	1624	–	4.09	1512	**0.73**	1.55
5		–	77.4	242605	**1.39**	2.24
6					**3.51**	2.97
7					**8.30**	3.71
8					**42.2**	**4.47**
9					**69.03**	**5.23**
10					**581.2**	**6.03**
20						**13.93**
30						**21.25**
40						**28.55**
50						**36.11**
100						**75.92**
COOPERATIVE BOX-PUSHING						
2	–	–	1.07	36	**0.1**	17.600
3	–	–	6.43	540	**0.457**	66.081
4	–	–	1138	2596	**0.622**	98.593
5					**5.854**	**107.72**
6					**10.7**	**120.67**
7					**24.96**	**156.42**
8					**28.97**	**191.22**
9					**184.3**	**210.27**
10					**293.7**	**223.74**
20						**458.10**
30						**636.28**
40						**774.14**
50						**1134.70**
100						–
MARS ROVERS						
2	–	–	83	1.0	**0.10**	5.80
3	–	–	389	1.0	**0.23**	9.38
4				103	**0.47**	10.18
5					**0.82**	**13.26**
6					**3.97**	**18.62**
7					**5.81**	**20.90**
8					**22.8**	**22.47**
9					**26.5**	**24.31**
10					**62.7**	**26.31**
20						**52.13**
30						**78.09**
40						**103.52**
50						**128.95**
100						**249.92**

Table 5: Experiments comparing the computation times (in seconds) of all exact solvers. Time limit violations are indicated by " ", and "–" indicate unknown values.

remaining benchmarks, the optimal solution often requires prohibitive memory, which limits the ability to scale to larger planning horizons while preserving optimality. In contrast to previous exact solvers, the FB-HSVI algorithm has an advantage since it is an anytime algorithm. That is, at any time, the algorithm can always provide a satisfactory solution that meets computation time and memory constraints while providing provable online bounds as depicted Figure 4. More precisely, if we interrupt the FB-HSVI algorithm at any time (or trial as depicted in Figure 4) before the convergence to an optimal solution, the algorithm is still able to return a good solution. We note that for planning horizon $T = 10$, the algorithm finds the optimal solution after only 10 trials, the remaining trials aim at providing performance guarantees. In the following, we exploit the anytime property of the FB-HSVI algorithm to compute approximate solutions for larger planning horizons.

6.4.2 Comparing to other approximate planners

In this section, we use a variant of the FB-HSVI algorithm that optimizes value functions for a fixed memory K over a single trial. We refer to this variant as the K-FB-HSVI algorithm. Clearly, this is an approximate version of the FB-HSVI algorithm. We compare the K-FB-HSVI algorithm to state-of-the-art approximate and memory-bounded algorithms PBIP and CBDP using parameters suggested by the authors. These experiments are two-folds. Our primary objective is to see how the memory K impacts the computational costs, as well as expected values. Next, we also want to gain insights on how the

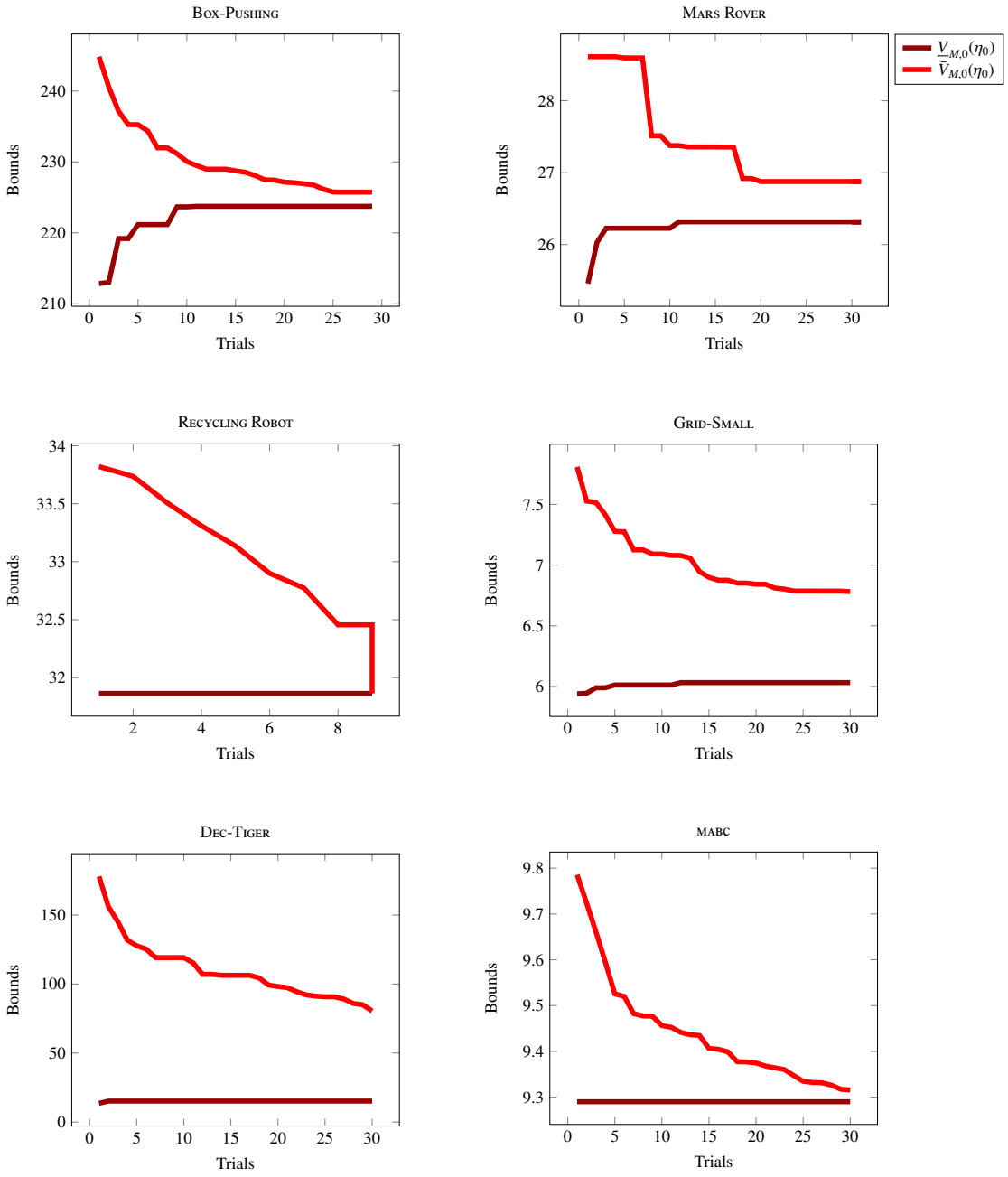


Figure 4: Anytime performance of the FB-HSVI algorithm over our selection of benchmarks and planning horizon $T = 10$. By interrupting the FB-HSVI at any trial (here the 30th trial) before the termination, the algorithm can still provide a near-optimal solution.

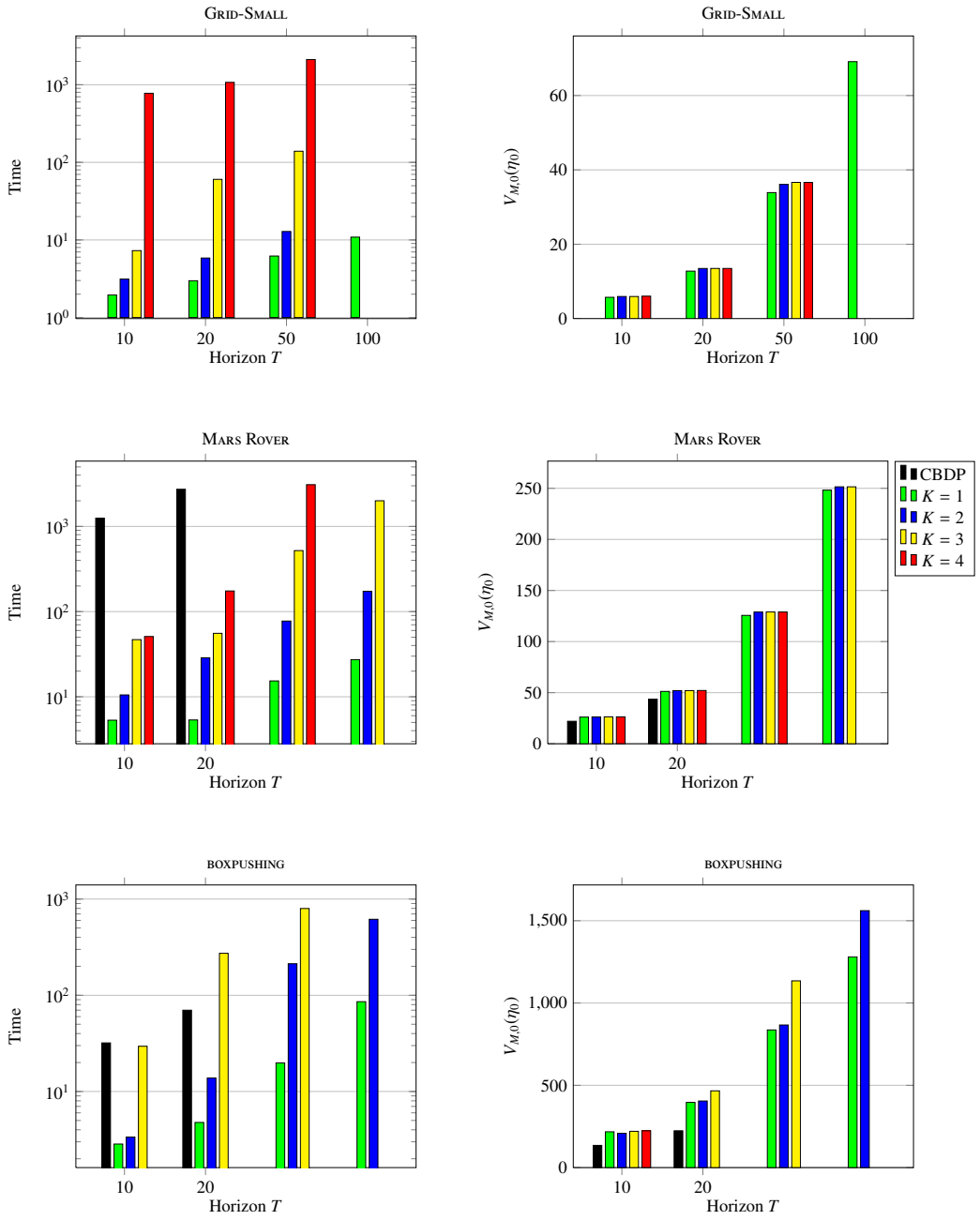


Figure 5: The performance of K-FB-HSVI and CBDP algorithms over our selection of benchmarks and larger planning horizons.

K-FB-HSVI algorithm compares with the state-of-the-art approximate algorithms. Figure 5 reports the computational cost and the expected values of the K-FB-HSVI algorithm for different values of parameter $K \in \{1, 2, 3, 4\}$. Overall, it appears that the computational costs increases exponentially with increasing

parameter K, and the expected values also increase accordingly, but at a slower rate. The GRID-SMALL and MARS-ROVER domains are of particular interest, since the expected values are nearly identical no matter the value of parameter K. In the BOX-PUSHING domain, however, the expected value increases significantly as K grows. So, the influence of parameter K over the expected values is domain-sensitive. Interestingly, we note that the K-FB-HSVI algorithm is superior to memory-bounded dynamic programming algorithms. For the larger benchmarks we tested, the PBIP algorithm quickly runs out of time. For the BOX-PUSHING domain, the PBIP algorithm takes 19000 seconds, whereas the CBDP algorithm takes only 32 seconds. For the MARS-ROVER domain at horizon $T = 20$, the CBDP algorithm takes 2729 seconds, whereas the 3-FB-HSVI algorithm is ten times faster, the 2-FB-HSVI algorithm is 200 times faster, and the 2-FB-HSVI algorithm is 800 times faster. Furthermore, all K-FB-HSVI algorithms ($K \in \{1, 2, 3\}$) yield expected values higher than that from the CBDP algorithm.

6.4.3 Choosing a heuristic to guide exploration

The first set of experiments explores the performance of two different heuristic methods we use to guide exploration towards optimal solutions. We distinguish between the heuristics based on the underlying MDP and POMDP problems of the Dec-POMDP to be solved, and we call these heuristics MDP and POMDP heuristics, respectively. POMDP heuristic functions were computed using the Point-based Value Iteration (PBVI) algorithm, with belief states being selected using the selection heuristic with parameter $\delta = 0.001$ (Pineau et al., 2006). Empirically, the results are not very sensitive to parameter δ. MDP heuristic functions, however, were computed using the Backward Induction algorithm (Bellman, 1957). The objective of these experiments is to see if the overall conclusions on how the choice of the heuristic method affects the total computation time to find an optimal solution, as well as seeing if this is domain depend.

Throughout these experiments, we report (in Figure 7) the time and value pairs for first generating the heuristic function and then using it within the FB-HSVI algorithm. Overall, while the POMDP heuristic method is likely to produce tighter heuristic functions, this advantage does not fully manifest in the total computation time to find an optimal solution. Over all benchmarks and long planning horizons, the MDP heuristic is more beneficial than the POMDP heuristic method with respect to the total computation time. To better understand how the heuristics affect the total computation time, we also report the time required to compute the heuristic function, the initial value of that heuristic, and the number of vectors (functions mapping states to reals) involved in each heuristic functions in Figures 6, 8, and 9, respectively.

The primary observation (Figure 6) is that the MDP heuristic for all tested domains and planning horizons ($h \leq 10$) provides significant savings over the POMDP heuristic, solving MDPs takes less than one second for all benchmarks, whereas approximate POMDP solutions quickly run out of time for medium-sized planning horizons. For instance, the time to compute POMDP solutions for both the GRIDSMALL and BOXPUSHING domains quickly exceeds the time limit: GRIDSMALL's POMDP heuristic function at horizon $h = 5$ takes within 100000 seconds; and BOXPUSHING's POMDP heuristic function at horizon $h = 7$ takes within 1500 seconds. This is not surprising given that POMDP solution methods require time exponential with respect to the planning horizon.

Next, we compare (Figure 8) the initial values $\bar{V}_{M,0}(b_0)$ for MDP and POMDP heuristic functions over all tested domains. We note that the initial values $\bar{V}_{M,0}(b_0)$ of using either heuristic functions are very close to one another for all but the DEC-TIGER domain. The returns are almost identical for MARS-ROVER, BOX-PUSHING, and RECYCLING-ROBOT domains. This is mainly because these domains are jointly fully observable. That is, the current joint observation reveals the true state of the system. The returns are within a small value from one another for GRID-SMALL and BROADCAST-CHANNEL domains since beliefs are close to degenerate beliefs: at horizon $h = 10$, the POMDP heuristic function is about $\bar{V}_{M,0}(b_0) = 9.29$ whereas the MDP heuristic function is about $\bar{V}_{M,0}(b_0) = 9.78$. However, the heuristic values significantly differ for the DEC-TIGER domain: at horizon $h = 5$, the POMDP heuristic function is about $\bar{V}_{M,0}(b_0) = 26.81$

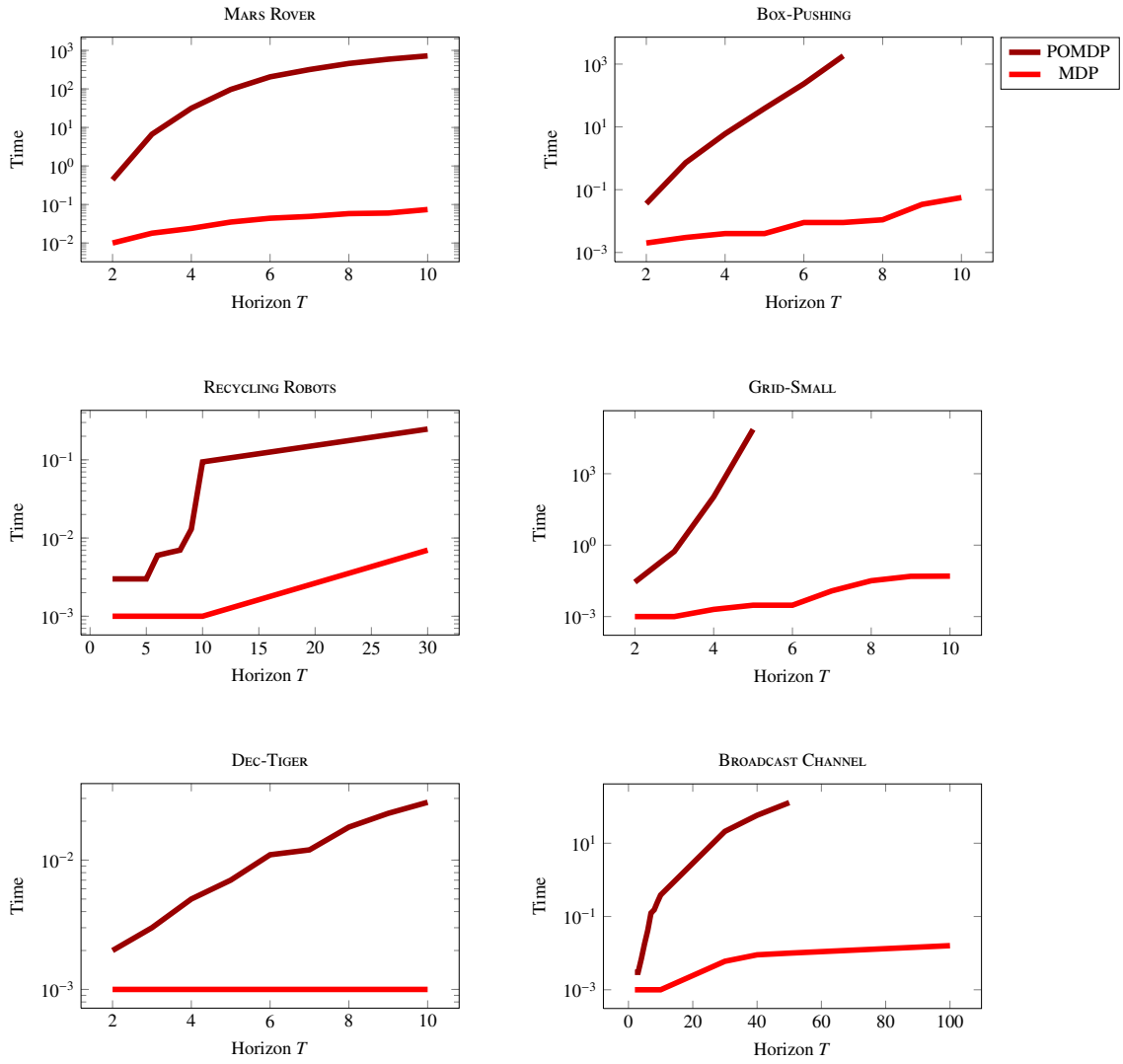


Figure 6: Comparison of heuristic methods to guide the search towards optimal solutions. All graphs shows the time (seconds) on the y-axis given the various numbers of planning horizons on the x-axis until convergence or timeout.

whereas the MDP heuristic function is about $\bar{V}_{M,0}(b_0) = 100.0$; and at horizon $h = 10$, the POMDP heuristic function is about $\bar{V}_{M,0}(b_0) = 68.73$ whereas the MDP heuristic function is about $\bar{V}_{M,0}(b_0) = 200.0$. It is worth noticing that for the Dec-Tiger domain both heuristic methods provide poor upper bounds to guide exploration, though the POMDP heuristic function is tighter than the MDP heuristic function.

Finally, our experiments (Figure 9) show that the POMDP method requires much more memory to store its heuristic functions for all domains but the Recycling-Robot benchmark. This domain is a transition- and observation-independent decentralized Markov decision process (Becker et al., 2004). As such, both the underlying MDP and POMDP are identical and so are their optimal value functions. The

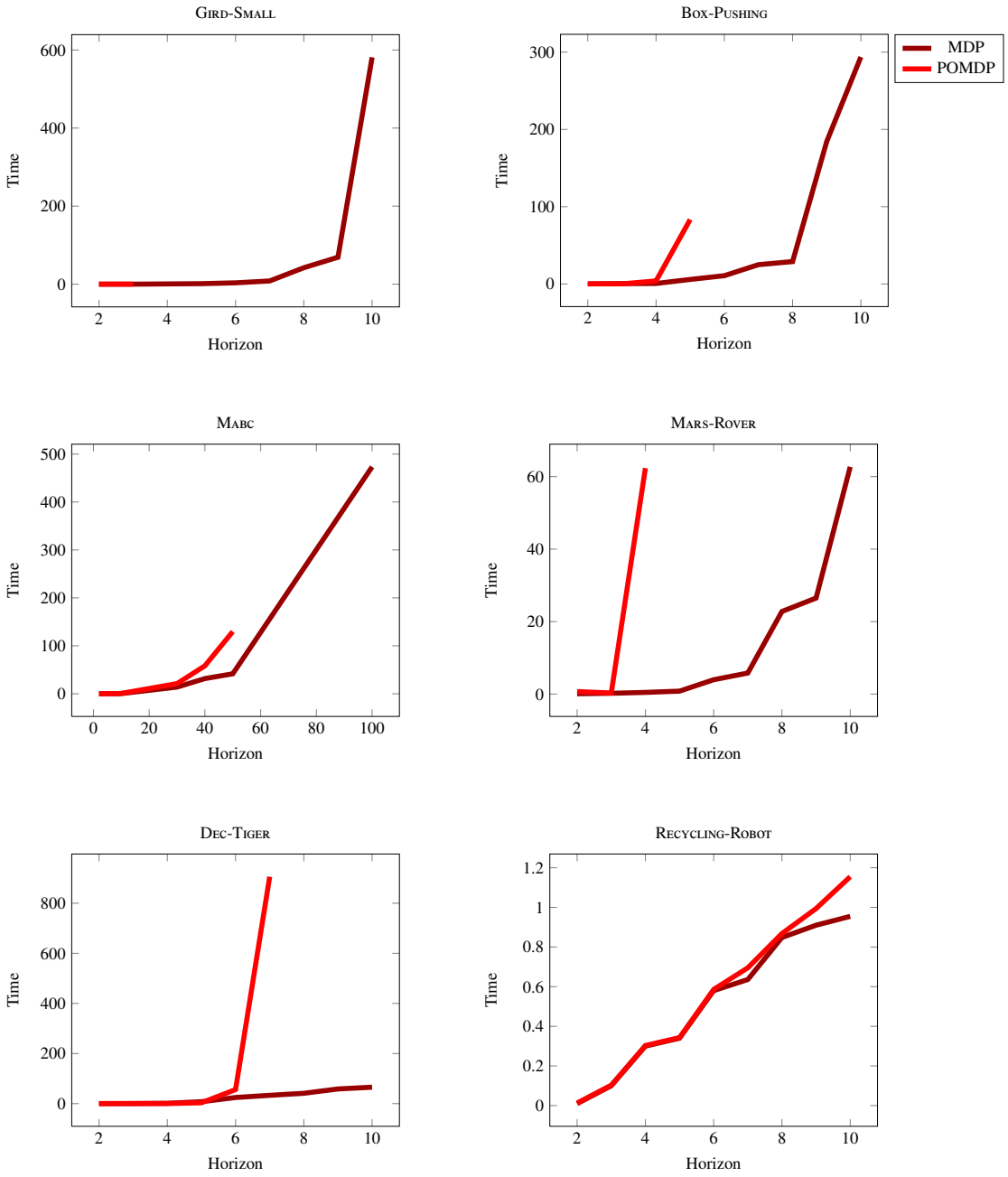


Figure 7: Comparison of time and horizon pairs for first generating the heuristic functions and then using it in the FB-HSVI algorithm. Omitted values correspond to instances where the computation time exceeds the limits.

POMDP heuristic functions for GRID-SMALL and BOX-PUSHING domains require more memory. For GRID-SMALL at planning horizon $h = 5$, the size of the MDP heuristic function is about 3 orders of magnitude

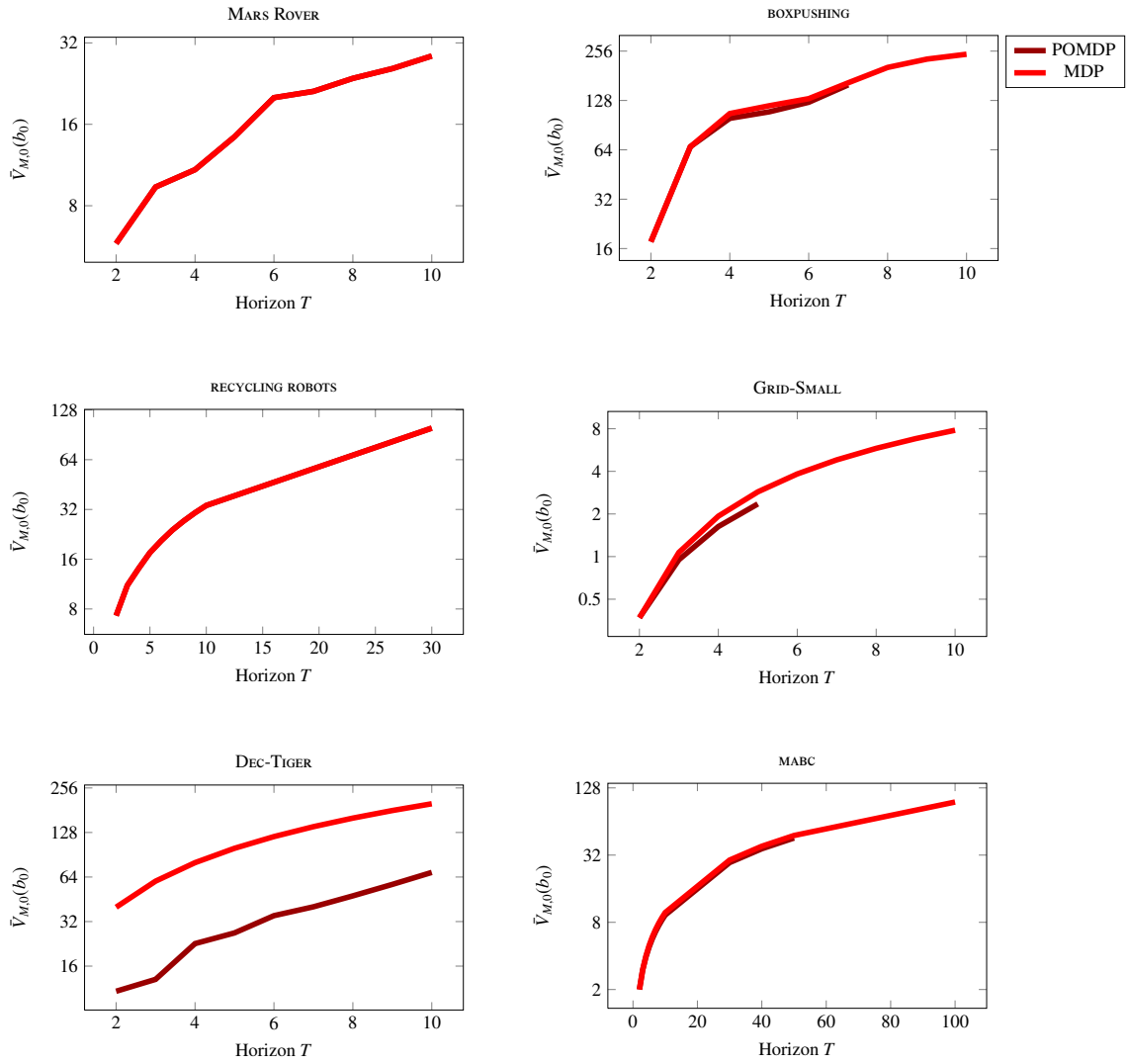


Figure 8: Comparison of heuristic methods to guide the search towards optimal solutions. All graphs shows the return on the y-axis given the various number of planning horizons on the x-axis until convergence or timeout.

less than that of the POMDP heuristic function, which requires about thousands of vector values whereas the MDP heuristic function requires only 5 vector values. Similarly, for BOX-PUSHING at planning horizon $h = 7$, the POMDP heuristic function requires about hundreds of vector values whereas the MDP heuristic function requires only 7 vector values. Though the remaining domains yield more concise POMDP heuristic functions, the MDP heuristic method yields much more compact heuristic functions.

Overall, while the POMDP heuristic method is likely to produce better initial heuristic functions to guide exploration, this advantage will not fully manifest in the total computation time required to solve the problem at hand. The primary reason is that the time required to compute such heuristic functions is non negligible. Furthermore, due to the size of the resulting heuristic functions, the updates of the

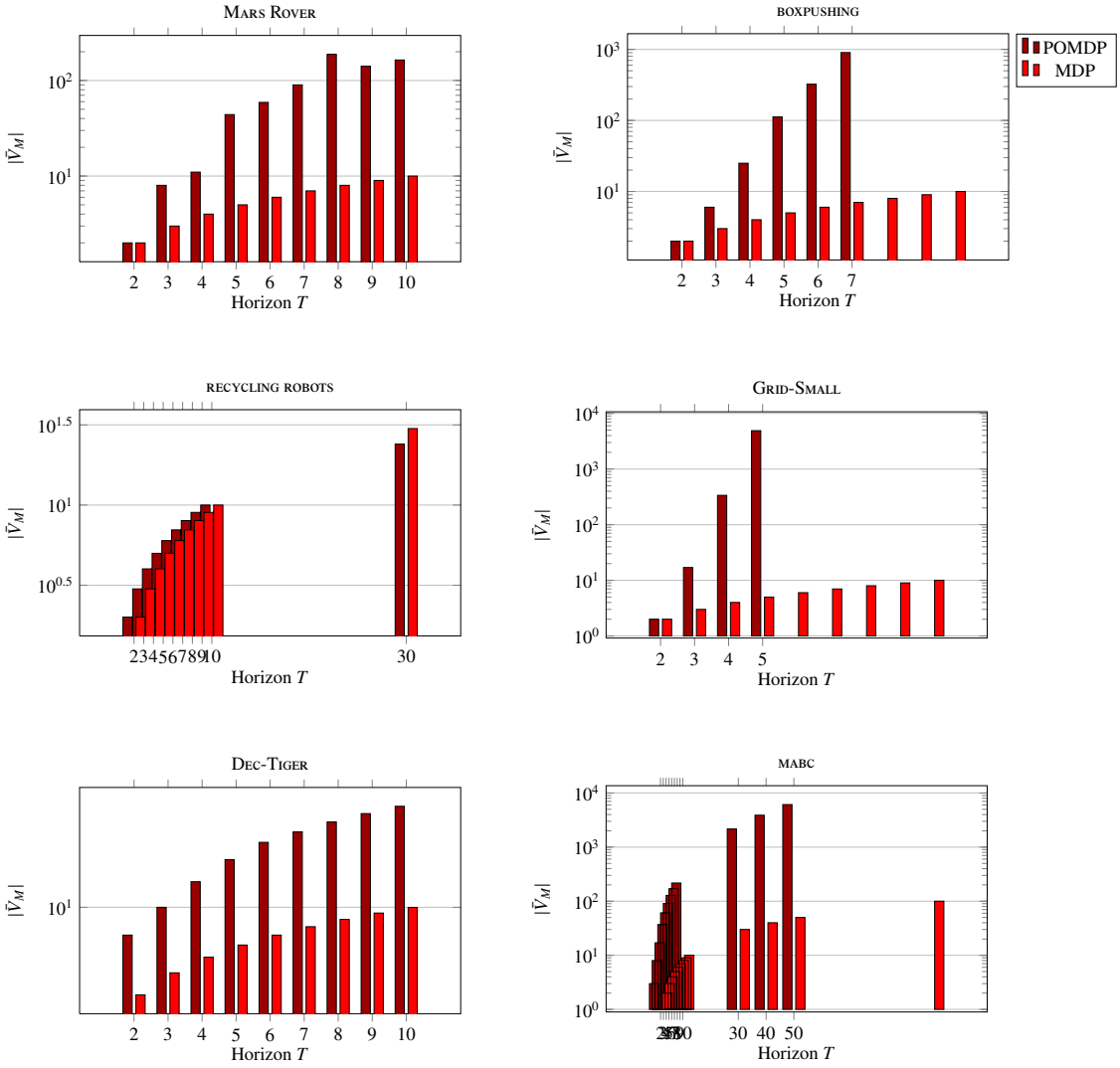


Figure 9: Comparison of heuristic methods to guide the search towards optimal solutions. All graphs shows the size of the heuristic functions on the y-axis given the various number of planning horizons on the x-axis until convergence or timeout.

upper bounds will involve additional computational costs, and thus limits the practical usefulness for all but the smallest planning horizons and domains. We make the observation that tighter heuristic functions (for example the POMDP heuristic functions) come with non negligible computation time. As a result, we chose to guide exploration for our experiments using the cheapest heuristic methods (*e.g.,* the MDP heuristic method). It is worth noticing that this choice is different from that of previous work on exact solution methods for Dec-POMDPs (Oliehoek et al., 2008, 2013), which choose the tighter heuristic functions but neglect the computational costs involved by such heuristics. In fact, the computation time required to compute tighter heuristic functions aside, algorithms using these heuristics are likely to speed up their convergence rate for short planning horizons. But as the planning horizon increases, the updates

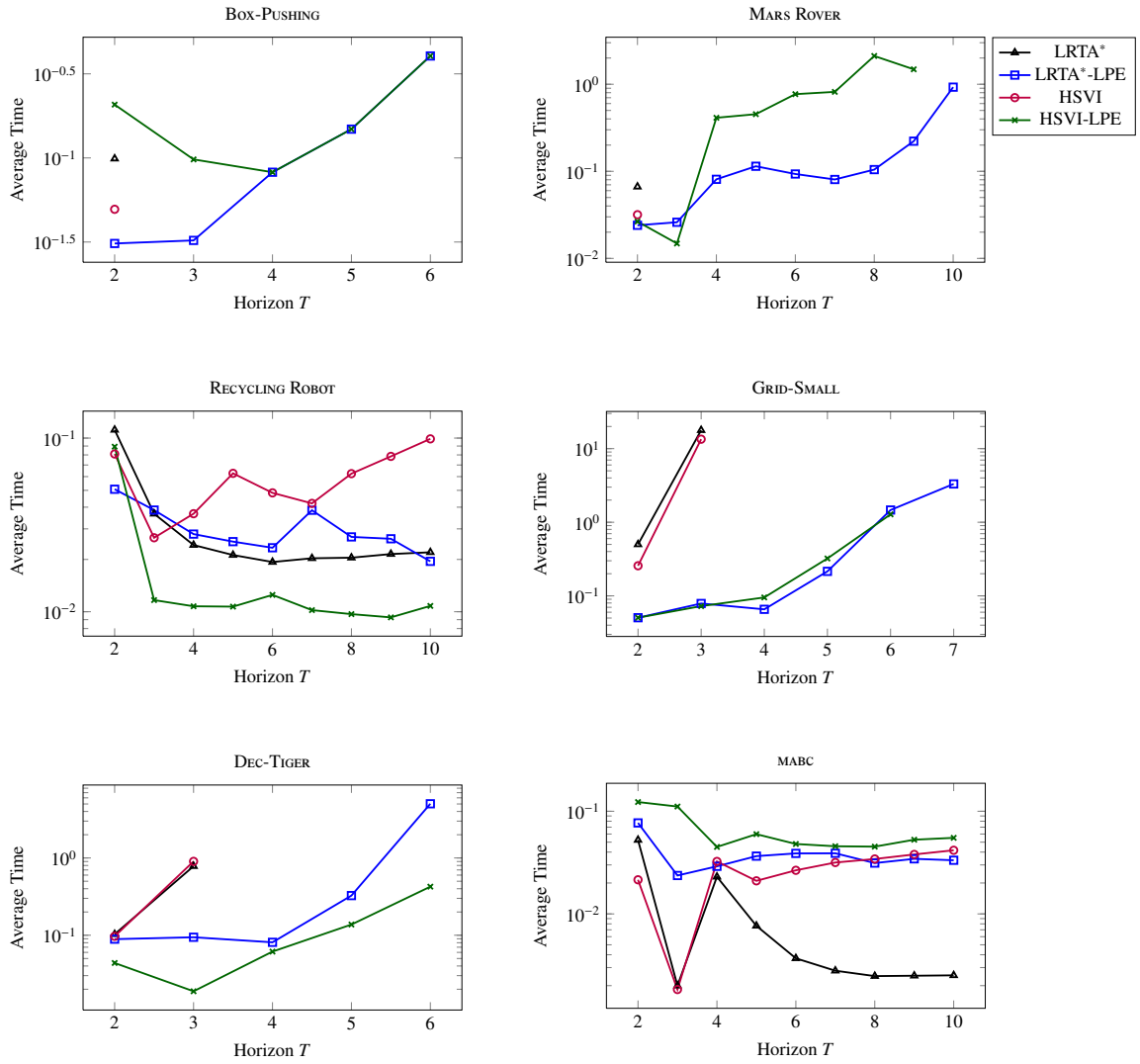


Figure 10: Comparison of the brute-force and the weighted constraint satisfaction implementations for selecting separable decision rules. All graphs show the average computation time (seconds) to perform a single point-based backup on the y-axis, given the various planning horizons on the x-axis until convergence or timeout. Missing values correspond to approaches that run out of time.

of both upper and lower bounds quickly becomes intractable.

6.4.4 Choosing a method for point-based backups

The second set of experiments explores the performance of our novel implementation of the selection of decentralized decision rules over our selection of benchmarks, using the weighted constraint satisfaction reformulation instead of the classical brute force implementation. The objective is to demonstrate how using either implementation affects the speed of separable decision rule selection, as well as seeing

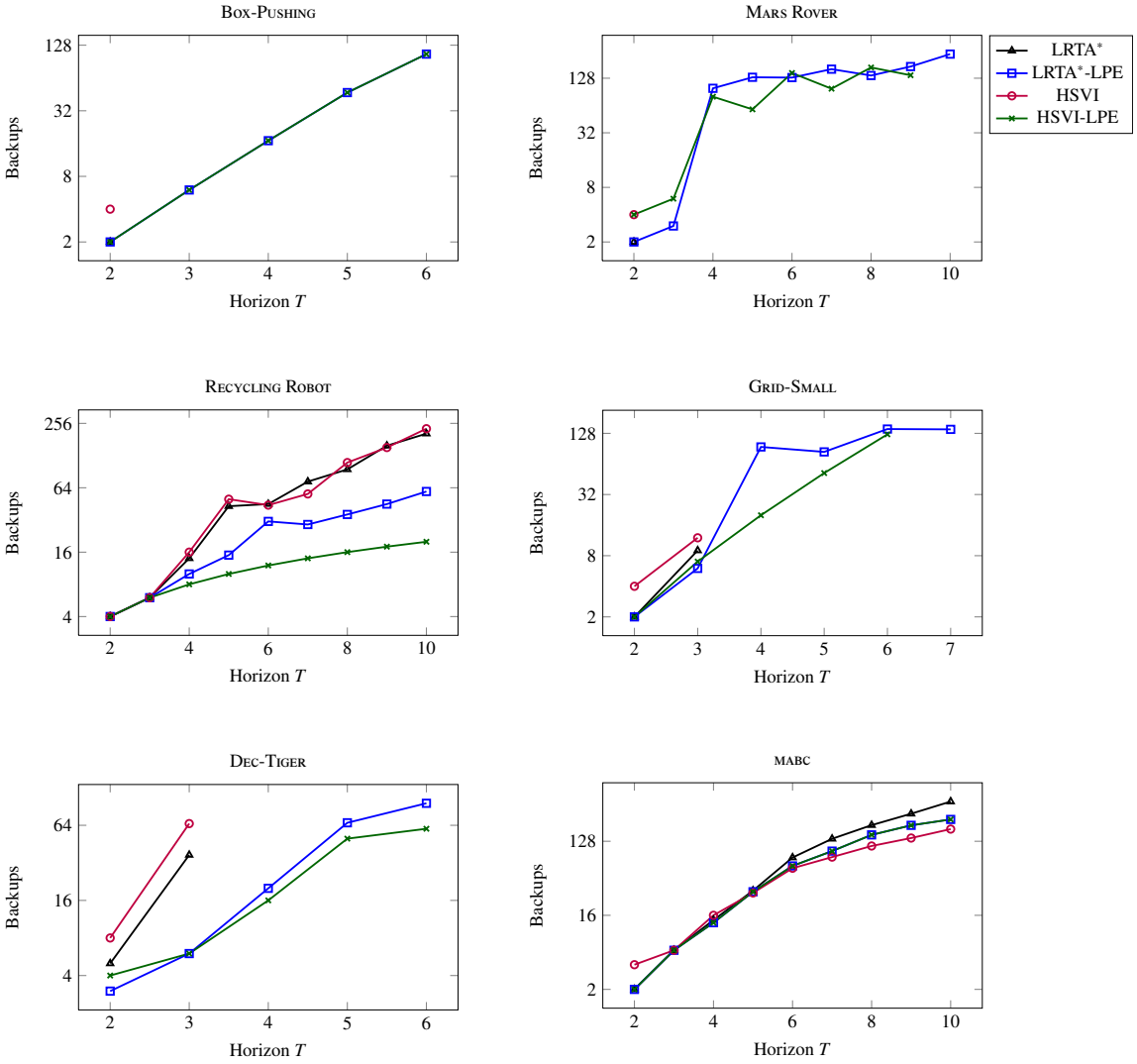


Figure 11: Comparison of the brute-force and the weighted constraint satisfaction implementations for selecting separable decision rules at a specified occupancy state through algorithm LRTA*, LRTA*-LPE, HSVI, and HSVI-LPE. All graphs show the number of point-based backups on the y-axis given the various numbers of planning horizons on the x-axis until convergence or timeout. Missing values correspond to approaches that run out of time.

whether this superiority is domain-dependent. In particular, we want to demonstrate the computational advantages of the weighted constraint satisfaction implementation over the brute-force implementation in terms of execution time and ability to improve the overall performance of our algorithms. To this end, we compare algorithms LRTA*, LRTA*-LPE, HSVI, and HSVI-LPE. We show that LRTA*-LPE and HSVI-LPE updates are more than 2 orders of magnitude of speedup for some settings. We further show they are about an order of magnitude faster on average, and can find solutions for planning horizons by more than a factor 5. Next, we study the reason of this superiority by explicitly comparing the actual

average computation time required for performing a single selection of a decentralized decision rule over various planning horizons. We show that, as we increase the planning horizon, the computational load of performing this selection using the brute-force implementation becomes too prohibitive. In contrast, the weighted constraint satisfaction implementation can still allow algorithms to solve the problems with medium-sized planning horizons.

Figure 10 shows a comparison of brute force implementation and the weighted constraint satisfaction implementations on various planning horizons and our selection of benchmarks. Clearly, algorithms that use the weighted constraint satisfaction implementation (*i.e.,* LRTA*-LPE and HSVI-LPE) provide significant savings over those that use the brute-force implementation (*i.e.,* LRTA* and HSVI) for all but the smallest planning horizons and benchmarks. For GRID-SMALL at planning horizon $T = 3$, the weighted constraint satisfaction implementation is about 340 times faster than the brute force implementation, which takes 169 seconds, whereas the weighted constraint satisfaction implementation takes 0.47 seconds. Furthermore, the weighted constraint satisfaction implementation can scale up the planning horizons to 7 whereas the brute-force implementation cannot scale beyond horizon 2 for all but the smallest benchmarks. Figure 11 shows how the number of backups impacts the speed of convergence to an optimal solution over various planning horizons for the same selection of algorithms. For the smallest domains, algorithms that use the brute-force implementation can scale up. This is mainly because decentralized decision rules in these domains have small and bounded dimensionality, as further discussed in Section 6.4.5. Overall, it appears that algorithms that use the weighted constraint satisfaction implementation require less backups than those that use the brute-force implementation. This is mainly because the weighted constraint satisfaction implementation exploits the piecewise linearity and convexity property of the optimal value function, which in turn improves the speed of convergence. For DEC-TIGER at planning horizon $T = 3$, the brute-force implementation requires up to 10 times more backups than the weighted constraint satisfaction implementation. This later observation also favors the algorithms that exploit the weighted constraint satisfaction reformulation as a means of accelerating the convergence rate towards optimal solutions.

6.4.5 Choosing a method to keep information concise

In the following, we investigate how to maintain concise information about the occupancy states, decision rules, and value functions. Clearly, algorithms that use compression methods provide significant savings in the number of maintained histories over those that do not (*e.g.,* LRTA* and HSVI). This is mainly because the number of generated histories grows in the worst case exponentially with planning horizons. For RECYCLING ROBOT at horizon $T = 5$, algorithms that use our compression methods maintain 30 orders of magnitude less histories than algorithms that do not. For this reason, our experiments investigate the efficiency of lossless compression methods in terms of their ability to maintain concise data and speed up convergence towards an optimal solution. The goal is to see if we can draw overall conclusions about how the choice of the compression method affects the size of the data and the speed of convergence. To this end, we compare lossless compression methods LPE (via LRTA*-LPE and HSVI-LPE) and TPE (via LRTA*-TPE and HSVI-TPE) over our selection of benchmarks and various planning horizons.

In general, the speed of convergence increases with increasing efficiency of the compression methods. So we want to measure the number of histories — we denote $|H|$ — that is explicitly maintained through the search for both methods, as a way to gain insights on the efficiency of compression methods. Quantity $|H|$ is of importance as all occupancy states, decision rules, and value functions are mapping from reachable histories to some reals. We also want to demonstrate that when $|H|$ is potentially high, using compression methods can help merging redundant histories, reducing the dimension of the search space, and thus speeding up the convergence towards an optimal solution.

Figures 12 and 13 report $|H|$ for LRTA*-LPE, HSVI-LPE, LRTA*-TPE, and HSVI-TPE algorithms over our selection of benchmarks and various planning horizons. We observe that TPE yields more con-

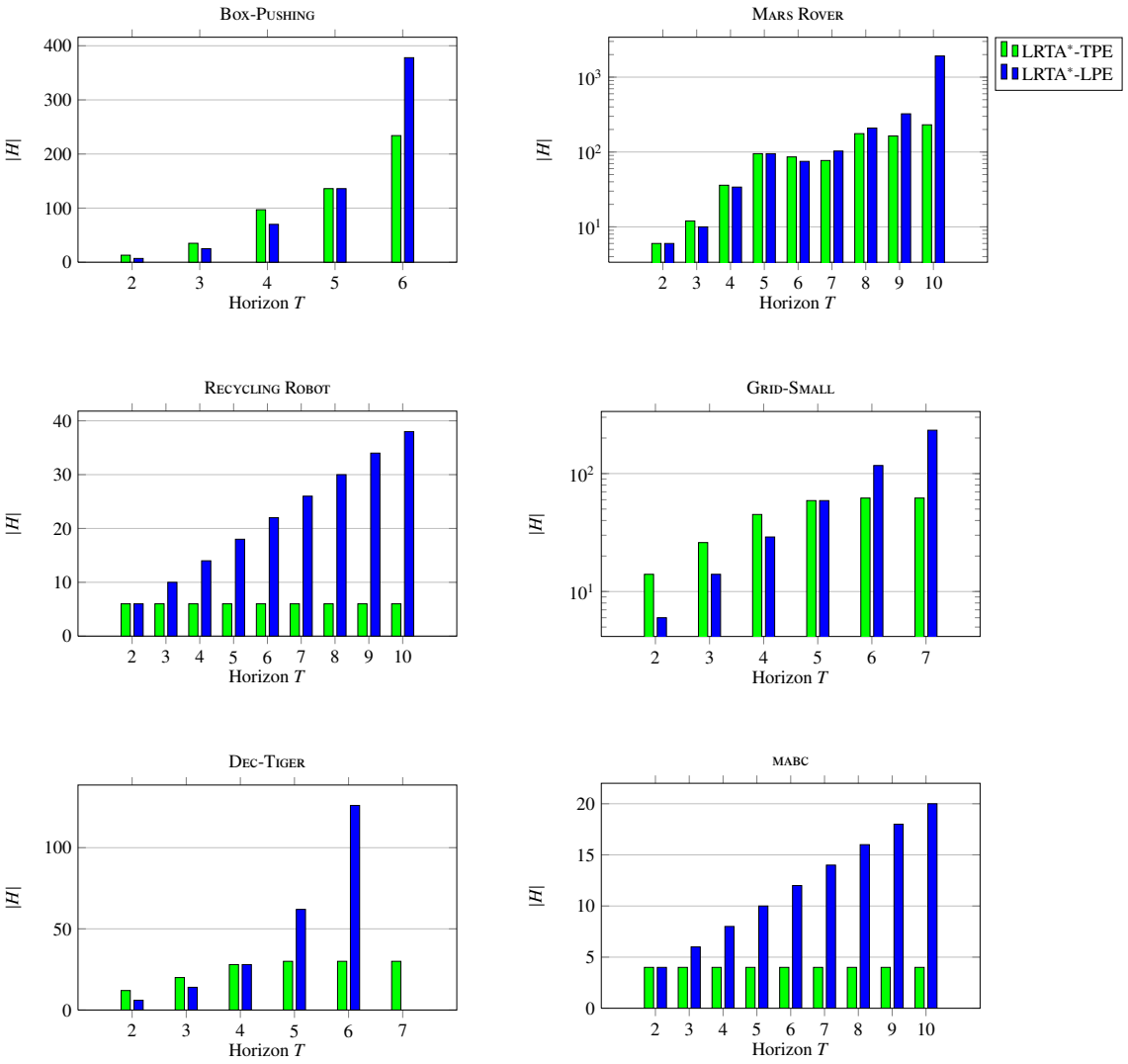


Figure 12: Comparison of compression methods to maintain concise data through LRTA-LPE and LRTA*-TPE. All graphs shows the memory requirements until convergence or time exceeds in the y-axis given the various number of planning horizons in x-axis.

cise value functions, occupancy states, and decision rules than LPE over most benchmarks and planning horizons. In particular, we notice that in many tested domains, there is a bounded number of histories that is sufficient for representing optimal value functions. TPE often succeeds in identifying this memory-bounded parametric space of histories, resulting in more concise value functions, whereas LPE fails. In the Recycling Robot domain for example, TPE yields no more than 6 histories for all horizons whereas LPE maintains up to 38 different histories, a number that keeps growing as the planning horizon increases. In fact, the Recycling-Robot domain allows a memory-bounded parametric space (the space of joint observations) that is sufficient for representing the optimal value functions (Dibangoye et al., 2013). That is, the most recent observation is sufficient to summarize all past local histories of each agent. In the

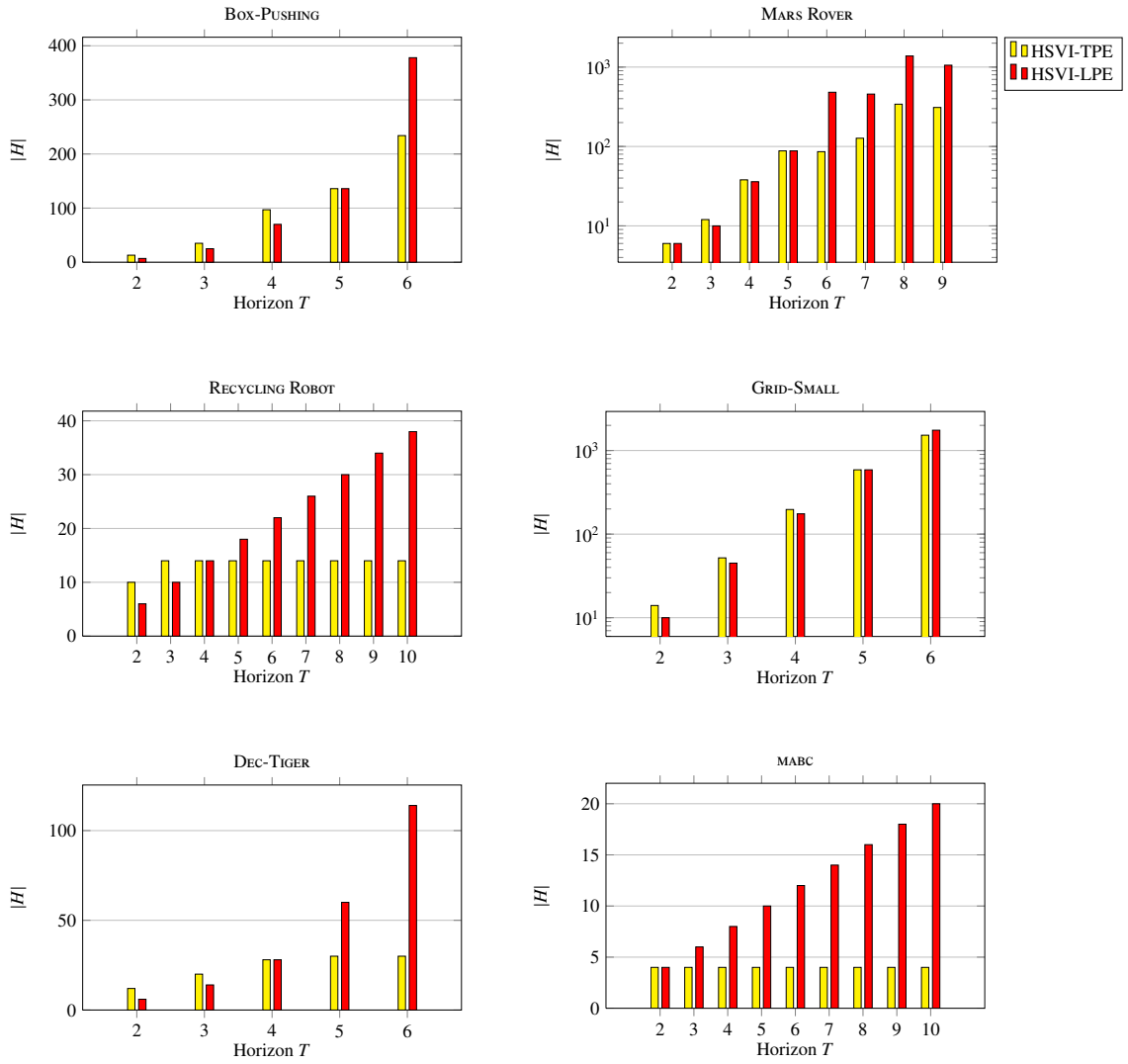


Figure 13: Comparison of compression methods to maintain concise data through HSVI-LPE and HSVI-TPE. All graphs show the memory requirements until convergence or timeout on the y-axis given the various planning horizons on the x-axis.

Broadcast-Channel domain, TPE yields no more than 4 histories for all horizons whereas LPE produces up to 20 different histories. Again, these results are due to the underlying structure of the Broadcast-Channel domain. In such a scenario, the future states of the world are conditionally independent of the histories, such problems are often referred as *unobservable Markov decision processes* (Madani et al., 2003). Hence, TPE can forget about histories, and reason only upon states. Another domain of interest is Dec-Tiger. In this problem, TPE produces no more than 30 histories for all horizons whereas those LPE maintains up to 126 different histories. Our assumption is that there always exists an optimal separable policy that is periodic (with period 3) for the Dec-Tiger domain. In other words, there exists an optimal separable policy that depends on histories only upon the most recent three action-observation pairs.

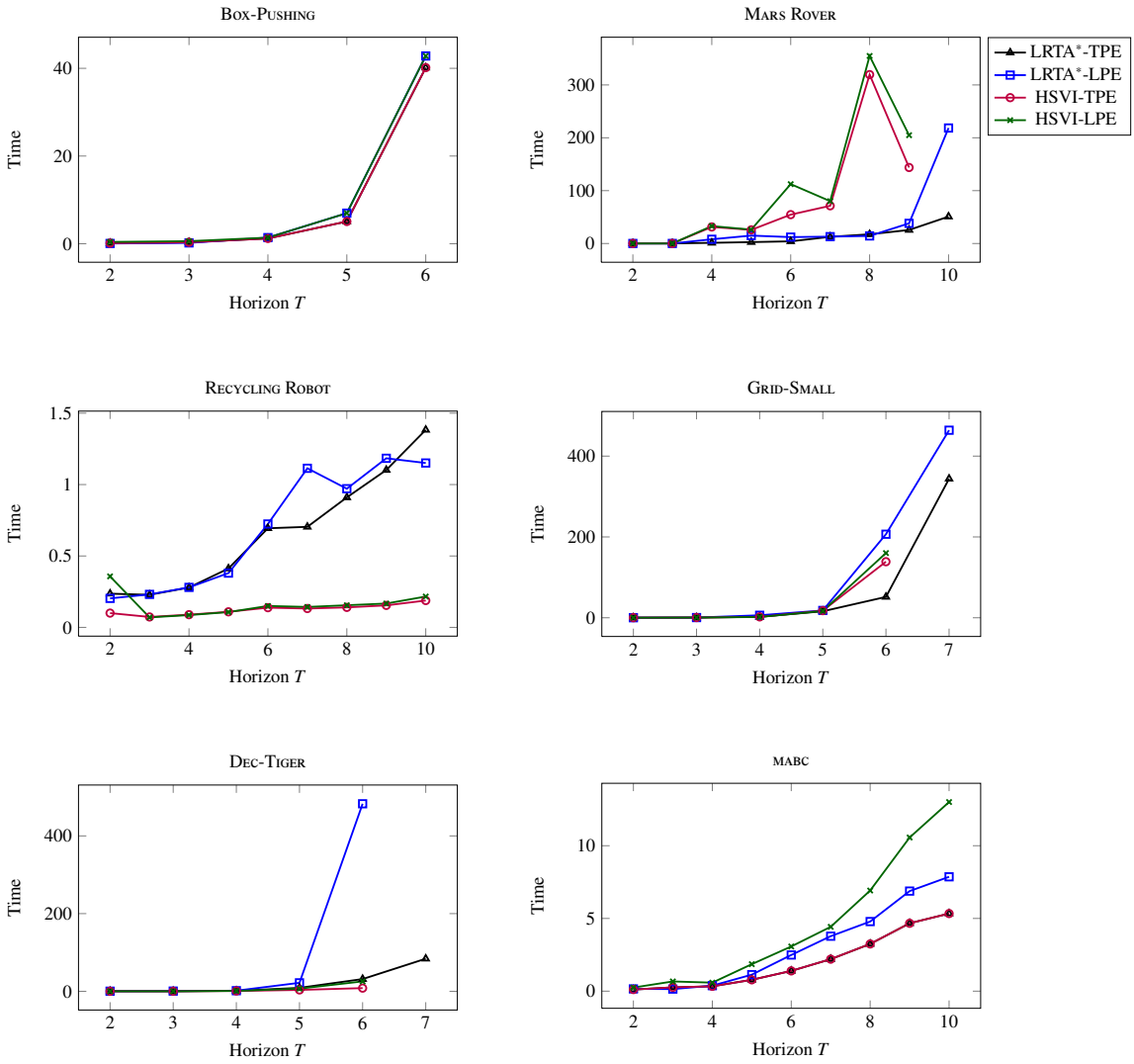


Figure 14: Comparison of compression methods to maintain concise data through LRTA$^*_1$, LRTA*-TPE, HSVI-LPE, and HSVI-TPE. All graphs show the time required to convergence (in seconds) on the y-axis given the various planning horizons on the x-axis.

Though LPE fails to capture the memory-bounded parametric space when it is present, value functions can still generalize from one occupancy state to another via the value-based compression method, at the expense of computational overhead, preserving fast convergence rates. We investigate this intuition next.

For the next set of experiments, we emphasize the scalability of TPE-based algorithms by explicitly comparing the actual computation time required to find an optimal solution. These experiments somewhat mitigate the importance of having a concise value function. Though the number of histories maintained is crucial, it does not tell the whole story. We show that compression methods that fail identifying the memory-bounded parametric space of histories can still demonstrate impressive performances, at the expense of being able to quickly generalize values from one occupancy state to another. Figure 14 reports

the time required to find an optimal solution for LRTA*-LPE, HSVI-LPE, LRTA*-TPE, and HSVI-TPE algorithms over our selection of benchmarks and various planning horizons. Overall, the convergence times for TPE-based algorithms are slightly faster than for LPE-based algorithms over all benchmarks and planning horizons. For the DEC-TIGER domain at horizon $T = 6$, LRTA*-TPE (31 seconds) is about 16 times faster than LRTA*-LPE (482 seconds); and for the BROADCAST CHANNEL at planning horizon $T = 10$, HSVI-TPE (5 seconds) is 2 times faster than HSVI-LPE (13 seconds). In many other tested domains, however, the superiority TPE-based algorithms is challenged. This is unexpected given that TPE yields more concise value functions. To better understand this, consider first the RECYCLING-ROBOT domain. It appears that both HSVI-TPE and HSVI-LPE have roughly the same convergence speed for various planning horizons, but they maintain a significantly different number of histories. Similar results appear for most domains over short planning horizons. So the gain of identifying a memory-bounded parametric space of histories does not fully manifest in the total computation time. There are many fundamental reasons that explain this counterintuitive observation. For either lossless compression method, the following operations have the same computation cost: (*a*) point-based backups; (*b*) occupancy state and bound updates. The point-based backups consists of solving weighted constraint satisfaction problems, which map clusters of local histories to actions. Since the number of clusters from either compression method is identical, solving weighted constraint satisfaction problems incurs the same computational cost. Moreover, the value functions can still generalize from one occupancy state to another for either method using value-based equivalence relation, at the expense of recasting the occupancy state based on another one. Recasting an occupancy state given another occupancy state incurs additional overhead when using TPE since $|H|$ is often larger. When the cost of recasting occupancy states is negligible, it is worth using LPE.

Figure 14 also compares HSVI-based algorithms and LRTA*-based algorithms over our selection of benchmarks and planning horizons. There is no clear superiority of either approach. LRTA*-based algorithms are faster than HSVI-based algorithms over BROADCAST-CHANNEL, GRID-SMALL , BOX-PUSHING and MARS-ROVER domains; whereas HSVI-based algorithms are faster than LRTA*-based algorithms over DEC-TIGER and RECYCLING-ROBOT domains. The reason is mainly because these approaches use different stopping criteria, which results in different exploration strategies. It is not clear which exploration strategy is superior, but using the HSVI strategy incurs non-negligible overhead since algorithms need to maintain both upper and lower bounds, and perform backups for both bounds. Furthermore, HSVI-based algorithms seem to generate longer trajectories of occupancy states than LRTA*-based algorithms. This mainly because HSVI stopping criterion require the difference between bounds at a given point to be zero, which is very hard to meet. As the trajectories get very long, the number of histories to maintain also increases. This partially explains why LRTA*-based algorithms have a little better compression than HSVI-based algorithms. However, when the additional computational cost required to maintain the lower-bound value functions and generate longer trajectories is negligible, the HSVI strategy is efficient. As the planning horizon increases, backing up both bounds becomes too prohibitive for all but the smallest benchmarks. This explains why HSVI-based algorithms fail to scale up to larger planning horizons. The FB-HSVI algorithm attempts to exploit the strength of both approaches, while circumventing prohibitive operations including the backups of the lower-bound value functions. Instead, it updates the lower-bound value functions based on the greedy separable decision rule, resulting in a weaker bound but faster updates.

6.4.6 Scaling to More Agents

Our feature-based heuristic search algorithms have shown impressive ability to scale up to larger planning horizons (by Dec-POMDP standards). Yet, it is still questionable whether they can deal with larger numbers of agents. To this end, we extend to multiple agent the well-known recycling robot scenario — originally introduced in the (Sutton and Barto, 1998) book, and then extended to the multi-agent case by (Amato et al., 2007). Our extensions were modeled after the single agent recycling robot problem

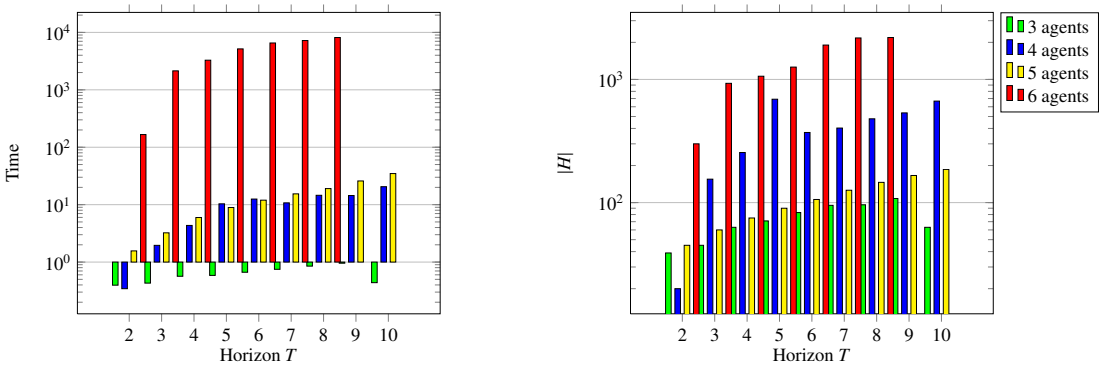


Figure 15: Performance of the FB-HSVI algorithm for increasing number of agents over the n-AGENT RECYCLING-ROBOT problem, where n denotes the number of agents involved in the domain.

described in (Sutton and Barto, 1998). Given n such models, each of which is associated to a single agent, we construct decentralized Markov decision processes. This is achieved by randomly choosing a number of state-action-state tuples where agents need to collaborate. To ensure collaboration we map these tuples to randomly selected values for either reward or transition models. This tied together the n single-agent models to a certain degree.

Figure 15 provides the opportunity to evaluate the scalability of the FB-HSVI algorithm with respect to the number of agents. While it is (at least in principle) possible to apply the FB-HSVI algorithm to any Dec-POMDP with a large number of agents, in practice, there is no reason to believe that a generic algorithm would scale up to larger planning horizons. Indeed, our results over the n-AGENT RECYCLING-ROBOT domain show that for short planning horizon $T \leq 10$, the FB-HSVI algorithm was able to find an optimal solution, but convergence time increases exponentially with planning horizons. The graph at the left hand side of Figure 15 provides a partial explanation. As the number of histories increases exponentially with increasing number of agents. So, even for problems where agents have only two local observations, the memory and computation time required to find an optimal solution are often prohibitive for generic algorithms including the FB-HSVI algorithm.

7 Conclusion

This paper describes a general methodology for solving either exactly or approximately decentralized stochastic control problems, and more specifically finite-horizon decentralized control of partially observable Markov decision processes and their subclasses. Our methodology, namely the *centralized planning for decentralized control approach*, builds upon the following principal steps. The first step recasts the decentralized stochastic control problem at hand into an equivalent centralized fully observable Markov decision process. Next, we identify a concise statistic — *the occupancy state* — that represents the state of the resulting fully observable MDP, which we call the *occupancy MDP*. We further demonstrate the optimal value functions of occupancy MDPs are piecewise linear and convex functions of the occupancy states. Finally, we present algorithms to find an optimal or approximate separable policy.

Along with our methodology we introduce a family of generic exact and approximate algorithms, called *the feature-based heuristic search value iteration* algorithms (FB-HSVI), for solving finite-horizon occupancy MDPs. This family of algorithms involves many components including: (*a*) efficient updates; (*b*) concise state representations; and (*c*) value function generalization. The main contributions pertaining to the feature-based heuristic search value iteration algorithms are summarized below.

Scalability. We believe the family of feature-based heuristic search value iteration algorithms is a major step towards scalable exact and approximate solutions for decentralized stochastic control problems. This is achieved by identifying a concise representation for the occupancy state — distributions of probabilities over states and joint histories — through the use of equivalence relations between histories.

Improved backups. Another aspect of the improved performance of this family of algorithms relies on a novel implementation of the backup operator. We replace the classical brute-force implementation for performing backups by weighted constraint satisfaction problems, which are easier to solve. We demonstrate that both upper and lower bounds can be updated by solving a series of weighted constraint satisfaction problems.

Value function generalization. The last aspect of the improved scalability builds upon the value function generalization. This is achieved by demonstrating that occupancy Markov decision processes have value functions that are piecewise linear and convex functions of occupancy states. We show that concise representations for both lower and upper bounds retain the piecewise linearity and convexity property of the optimal value functions throughout the planning stage.

Anytime performance. The family of the feature-based heuristic search value iteration algorithms are trial-based algorithms. These algorithms alternate between the generation of a separable policy and the update of the current best separable policy. As the algorithms proceed, the current (best) separable policy is improved at the expense of increased computational time. The algorithms can be terminated either when a satisfactory solution is attained, or when allocated planning time is exceeded. In either case, these algorithms can always provide online performance bounds on the returned solution illustrating how far from the optimal solution the current solution is.

Approximate memory-bounded variants. The feature-based heuristic search value iteration algorithms may require an intractable amount of time to generate an optimal solution for some problems. For this reason, we suggest a memory-bounded variant where histories map to a bounded parametric space. As the number of parameters increases, the solution improves at the expense of increased computational time. The trade-off can be controlled by adjusting the parametric space, with a *memory* parameter. The memory corresponds to the number of past actions and observations to be retained in the histories. This approximate counterpart of the exact feature-based heuristic search value iteration algorithms opens a novel direction of research for more scalable algorithms solving decentralized stochastic control problems.

While we have demonstrated the ability to solve decentralized stochastic control problems which are larger than those previously solved, many practical applications are much larger than domains experimented in this paper. Though we explored the idea of projecting state and history pairs into a concise feature space while preserving optimality, often the resulting feature space remains prohibitively huge. This is of concern since the number of states and histories impact all occupancy states, separable decision rules, and value functions. Maintaining these objects for large feature spaces is prohibitive. This highlights the necessity of addressing the scalability issue of the feature-based heuristic search value iteration algorithms through more concise (and possibly lossy) feature spaces. In that direction, we would like to explore using occupancy states over observation histories (rather than action-observation histories), which were shown to be sufficient (along with action-observation histories) simultaneous to this work (Oliehoek, 2013).

Many attempts to address the scalability issues in decentralized stochastic control rely on tractable subclasses. For instance, we have already shown that occupancy states over just states (and not agent histories) can be used in transition- and observation-independent decentralized Markov decision processes

to greatly increase scalability (Dibangoye et al., 2012, 2013). We also applied this theory and algorithm to decentralized stochastic control problems with separable assumptions, for example networked distributed partially observable Markov decision processes (Nair et al., 2005), allowing us to scale up to larger number of agents (Dibangoye et al., 2014). We plan to explore applying our methodology and feature-based heuristic search value iteration algorithms to decentralized stochastic control problems where agents have separable dynamics or rewards, as well as domains with delayed communications (Ooi and Wornell, 1996; Grizzle et al., 1981; Oliehoek and Spaan, 2012), as a means of reducing the memory burden.

A secondary (but no less important) issue concerning the scalability issues in decentralized stochastic control pertains to efficient inference methods to update occupancy states and value functions at the planning stage. We think occupancy states together with graphical models could help exploit the locality of interaction among agents statically (as in (Nair et al., 2005; Kumar and Zilberstein, 2009)) or dynamically (as in (Canu and Mouaddib, 2011)), and hence improve inference mechanisms. This is a critical issue as the number of occupancy states necessary to obtain a good solution may be exponential in the planning horizon. So, techniques that can efficiently update both occupancy states and value functions are of great importance.

We dedicated a small portion of our empirical study to a comparative analysis of our approximate memory-bounded algorithms to previous approximate solvers (Section 6.4.2). This includes evaluating a selection of the largest benchmarks from the literature for bounded and lossy parametric spaces. Preliminary performance results have demonstrated that this is a promising direction of research towards truly scalable algorithms for decentralized stochastic control problems. Similar preliminary experiments in the case of infinite-horizon decentralized partially observable Markov decision processes have yielded significant performance improvements (MacDermed and Isbell, 2013). Intuitively, these approaches consist of searching for a separable policy in a subset of the full separable policy space, at the expense of weaker performance. The challenge therefore is to provide a theoretical bound on the error of the approximation introduced in such frameworks. Another orthogonal research direction consists of exploiting these suboptimal solutions to guide exploration towards good solutions.

Overall, the feature-based heuristic search value iteration algorithms have already pushed the envelop in terms of decentralized stochastic control problems that can be solved within reasonable time and memory. We hope our general methodology and algorithms will play a leading role in further research on theory and scalable algorithms for decentralized stochastic control problems.

8 Acknowledgements

We would like to thank Matthijs Spaan for providing results for GMAA*-ICE as well as Frans Oliehoek and the reviewers for their helpful comments. Research supported in part by AFOSR MURI project #FA9550-09-1-0538.

A Domains

A.1 Multi-agent tiger problem

Probably the most widely used problem for single agent partially observable Markov models is the tiger problem introduced in (Kaelbling et al., 1998). A multi-agent version of this problem was introduced in (Nair et al., 2003). This domain has been extensively discussed in Section 2. The challenge of this problem is to determine how long would agents stand and listen before opening a door. This actually depends on the accumulated confidence in partial information they receive upon listening. In fact, as the consequences of meeting the tiger are made less dire, the agents would need not to listen too much before choosing to open a door. In contrast, as the reliability of listening is made worse, agents would need to

listen much more before choosing to open a door. In this scenario, we build the intuition that the agents' policies depend only upon three most recent action-observation pairs.

A.2 Multi-access broadcast channel

Figure 16: The multi-access broadcast channel benchmark.

Another standard problem, depicted in Figure 16, is an idealized model of a multi-access broadcast channel, originally proposed by Hansen et al. (2004) and adapted from (Ooi and Wornell, 1996). In this scenario, two agents are controlling a message channel on which only one message per time step can be sent. Agents have the same goal of maximizing the global throughput of the channel. After taking an action, each agent receives an observation signal about a possible collision. In contrast to general Dec-POMDPs, the future states of the world, which correspond to the presence of a message to send, are conditionally independent of observations. Hence, observations are not sufficient to determine the global state of the world. However, if (instantaneous and noise-free) communications were allowed, the future states of the world would only depend upon the present state. This sets the multi-access broadcast channel domains apart from general Dec-POMDPs.

A.3 Meeting under uncertainty (/Grid Small)

This problem was first presented in (Bernstein et al., 2002). It is a simplified version of a real-life problem of multi-robot planning (Suzuki and Yamashita, 1999).

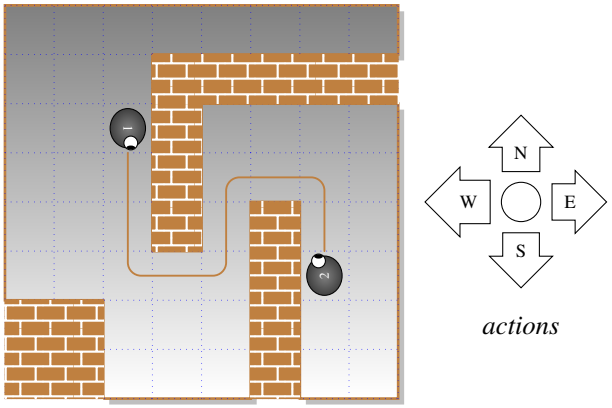


Figure 17: A meeting-grid under uncertainty scenario on an 8×8 grid.

In this scenario, two agents want to meet as soon as possible on a two-dimensional grid. Each agent's possible actions include moving north, south, west, east and staying at the same place. The actions of a

given agent do not affect the other agents. After taking an action, each agent can sense some information, which in this case corresponds to its own location. Here, each agent's own partial information is insufficient to determine the global state of the world. This is mainly because agents are not permitted to explicitly communicate their individual locations with each other. However, if this (instantaneous and noise-free) communication was allowed the agents' pieces of information together would reveal the true state of the world, (i.e., the agents' joint location). The presence of this "*joint full observability*" property distinguishes this subclass of Dec-POMDPs from general multi-agent systems. More generally, in Dec-POMDPs, the agents' pieces of information together can map to multiple different states of the world. As a consequence, decisions depend on full histories. In this scenario, however, since both transitions and observations are not affected by the other agents, each agent's decision depends only upon the most recent observations (i.e., the agent's present locations) (Becker et al., 2004).

A.4 Cooperative Box-pushing

The cooperative box-pushing problem is a well-known robotics problem introduced by (Kube and Zhang, 1997). In this domain, two or more agents have to cooperate to move a big object (the box) that they could not move on their own. Even though the robots cannot communicate with each other, they have to achieve a certain degree of coordination to move the box at all. Figure 18 shows a sample problem in a grid-world domain.

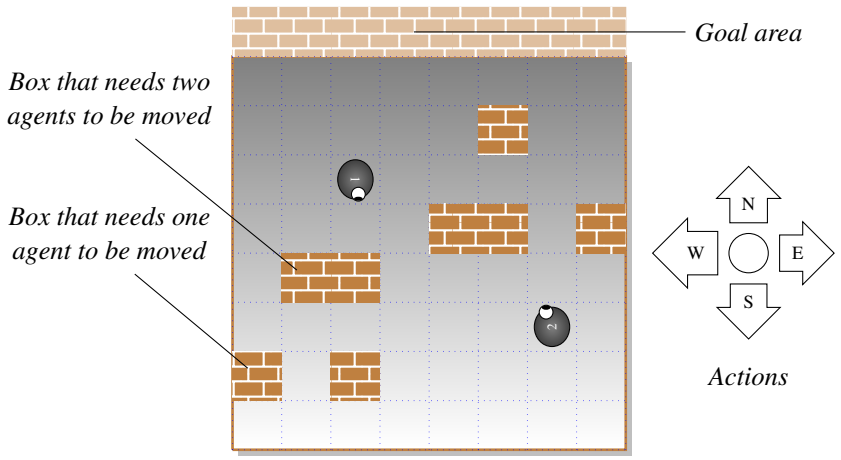


Figure 18: The box-pushing problem in the grid world.

This problem is very flexible in the sense that it can easily be modified to different sizes and problem complexities by varying the problem parameters. The system states of this problem are all tuples of possible positions of the two agents and the boxes. Each agent has 4 actions: turn left, turn right, move forward and stay. If an agent is facing a box that it can move on its own and selects the action move forward, the box is pushed one grid cell into the direction of the agent's movement and the agent also moves one step forward. If the agent moves against a wall or against a larger box that it cannot push by itself it just stays in place. However, if both agents push against the large box at the same time, the large box moves by one grid cell as do the agents. To model an uncertain environment we assume that each agent's actions are only successful with probability 0.9 and with probability 0.1 it simply stays in place.

After every time step each agent gets one out of five possible observations deterministically describing the situation of the environment in front of the agent: empty field, wall, other agent, small box, large box.

The reward function is designed such that the agents benefit from cooperation. Every time step the

agents spend in the environment, they get a negative reward of −0.1 per agent. For each agent that bumps into a wall or a box it cannot move, they receive a penalty of −5. For each small box that reaches the goal area they get a reward of 10. If the agents cooperatively push the large box into the goal area they get a reward of 100.

A.5 Recycling Robots

We already discussed the recycling robot benchmark in Section 6.4.6.

A.6 Navigation problems

Another family of domains is that of navigation scenarios (Melo and Veloso, 2011). The reason for using navigation scenarios is that the decentralized stochastic control models appear particularly appealing for modeling multi-robot navigation problems. Furthermore, for this class the results can easily be visualized and interpreted. In each of the navigation scenarios, two robots must reach one specific state. In the environments, the goal for each robot is marked with a cross and the robots each depart from the other robot's goal state, in an attempt to increase the possibility of interaction. Each robot has four possible motion actions that move the agent in the corresponding direction with probability 0.8 and fail with the probability 0.2, leaving the state of the agent unchanged. When the two robots stand in the same cell simultaneously, they both get a penalty of −20. Upon reaching the corresponding goal, the agents receive a reward of +1.

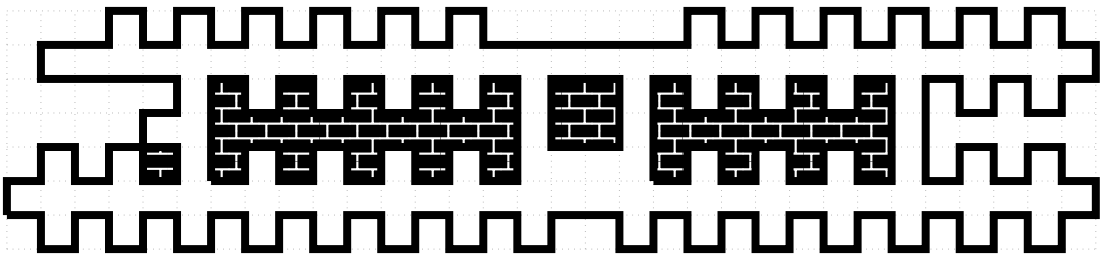


Figure 19: The building of the Carnegie Mellon University.

References

Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2418–2424, 2007.

Christopher Amato, Jilles S. Dibangoye, and Shlomo Zilberstein. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

Raghav Aras and Alain Dutech. An investigation into mathematical programming for finite horizon decentralized POMDPs. *Journal of Artificial Intelligence Research*, 37:329–396, 2010.

Nicholas Armstrong-Crews and Manuela M. Veloso. An approximate algorithm for solving oracular POMDPs. In *IEEE International Conference on Robotics and Automation*, pages 3346–3352, 2008.

Karl J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1–2):81–138, 1995.

Raphen Becker, Shlomo Zilberstein, Victor R. Lesser, and Claudia V. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22: 423–455, 2004.

Richard E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 1957.

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), 2002.

Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

Abdeslam Boularias and Brahim Chaib-draa. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pages 20–27, 2008.

Arnaud Canu and Abdel-Illah Mouaddib. Collective decision under partial observability - a dynamic local interaction model. In *IJCCI (ECTA-FCTA)*, pages 146–155, 2011.

Martin C. Cooper, Simon de Givry, M. Sanchez, Thomas Schiex, Matthias Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010.

Daniela Pucci de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

Simon de Givry, Federico Heras, Matthias Zytnicki, and Javier Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 84–89, 2005.

Rina Dechter. Bucket elimination: a unifying framework for processing hard and soft constraints. *Constraints*, 2(1):51–55, 1997.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2): 41–85, 1999.

Rina Dechter. Constraint optimization. In *Constraint Processing*, pages 363–397. Morgan Kaufmann, San Francisco, 2003.

Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003.

Jilles S. Dibangoye, Abdel-Illah Mouaddib, and Brahim Chaib-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 569–576, 2009a.

Jilles S. Dibangoye, Guy Shani, Brahim Chaib-draa, and Abdel-Illah Mouaddib. Topological order planner for POMDPs. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1684–1689, 2009b.

Jilles S. Dibangoye, Christopher Amato, and Arnaud Doniec. Scaling up decentralized MDPs through heuristic search. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 217–226, 2012.

Jilles S Dibangoye, Christopher Amato, Arnaud Doniec, and François Charpillet. Producing efficient error-bounded solutions for transition independent decentralized MDPs. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems*, 2013.

Jilles S Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. Exploiting separability in multi-agent planning with continuous-state mdps. In *Proceedings of the Thirteenth International Conference on Autonomous Agents and Multiagent Systems*, page to appear, 2014.

Jessy W. Grizzle, Steven I. Marcus, and Kai Hsu. Decentralized control of a multiaccess broadcast network. In *Decision and Control including the Symposium on Adaptive Processes, 1981 20th IEEE Conference on*, volume 20, pages 390–391, 1981. doi: 10.1109/CDC.1981.269554.

Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

Manish Jain, Matthew E. Taylor, Milind Tambe, and Makoto Yokoo. DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 181–186, 2009.

Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

C. R. Kube and H. Zhang. Task modelling in collective robotics. *Autonomous Robots*, 4(1):53–72, 1997.

Akshat Kumar and Shlomo Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 561–568, 2009.

Akshat Kumar and Shlomo Zilberstein. Point-based backup for decentralized POMDPs: complexity and new algorithms. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 1315–1322, 2010.

William S. Lovejoy. Computationally Feasible Bounds for Partially Observed Markov Decision Processes. *Operations Research*, 39(1), 1991.

Liam C. MacDermed and Charles Isbell. Point based value iteration with optimal belief compression for Dec-POMDPs. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 100–108, 2013.

Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147:5–34, 2003.

Francisco S. Melo and Manuela M. Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789, 2011.

Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 705–711, 2003.

Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 133–139, 2005.

Frans A. Oliehoek. Sufficient plan-time statistics for decentralized POMDPs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2013.

Frans A. Oliehoek and Matthijs T.J. Spaan. Tree-Based Solution Methods for Multiagent POMDPs with Delayed Communication. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos A. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 577–584, 2009.

Frans A. Oliehoek, Matthijs T. J. Spaan, Christopher Amato, and Shimon Whiteson. Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *Journal of Artificial Intelligence Research*, 46:449–509, 2013.

James M. Ooi and Gregory W. Wornell. Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proc. of the 35th IEEE Conference on Decision and Control*, volume 1, pages 293–298. IEEE, 1996.

Sébastien Paquet, Brahim Chaib-draa, Patrick Dallaire, and Dany Bergeron. Task allocation learning in a multiagent environment: Application to the robocuprescue simulation. *Multiagent and Grid Systems*, 6(4):293–314, 2010.

Judea Pearl. Some recent results in heuristic search theory. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(1): 1–13, 1984.

Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.

Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007. ISBN 0470171553.

Matrin L. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. Wiley-Interscience, Hoboken, New Jersey, 1994.

Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann, 1995.

Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Journal of Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

David Silver and Joel Veness. Monte-carlo planning in large POMDPs. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172, 2010.

Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

Trey Smith. *Probabilistic Planning for Robotic Exploration*. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2007.

Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 520–527, Arlington, Virginia, United States, 2004.

Trey Smith and Reid G. Simmons. Focused real-time dynamic programming for mdps: Squeezing more out of a heuristic. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*, pages 1227–1232, 2006.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, March 1999. ISSN 0097-5397.

Daniel Szer, François Charpillet, and Shlomo Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 568–576, 2005.

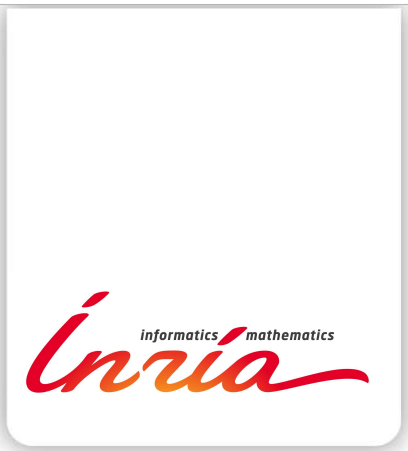
John N. Tsitsiklis and Benjamin van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1-3):59–94, January 1996. ISSN 0885-6125.

Abraham Wald. Contributions to the Theory of Statistical Estimation and Testing Hypotheses. *The Annals of Mathematical Statistics*, 10:299–326, 1939.

Keith Winstein and Hari Balakrishnan. TCP ex Machina: Computer-Generated Congestion Control. In *SIGCOMM*, Hong Kong,, August 2013.

Nevin Lianwen Zhang and Weihong Zhang. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.

Shlomo Zilberstein, Richard Washington, Daniel S. Bernstein, and Abdel-Illah Mouaddib. Decision-theoretic control of planetary rovers. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents*, pages 270–289, London, UK, 2002. Springer-Verlag.



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399