Delft University of Technology

Point-Based Value Iteration for Finite-Horizon POMDPs

Walraven, Erwin; Spaan, Matthijs T.J.

Publication date
2019

Document Version
Final published version

Published in
The Journal of Artificial Intelligence Research

Citation (APA)
Walraven, E., & Spaan, M. T. J. (2019). Point-Based Value Iteration for Finite-Horizon POMDPs. *The Journal of Artificial Intelligence Research*, *65*, 307-341. https://www.jair.org/index.php/jair/article/view/11324

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Journal of Artificial Intelligence Research 65 (2019) 307-341 Submitted 01/2019; published 07/2019

Point-Based Value Iteration for Finite-Horizon POMDPs

Erwin Walraven E.M.P.WALRAVEN@TUDELFT.NL
Matthijs T. J. Spaan M.T.J.SPAAN@TUDELFT.NL
Delft University of Technology,
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Abstract

Partially Observable Markov Decision Processes (POMDPs) are a popular formalism for sequential decision making in partially observable environments. Since solving POMDPs to optimality is a difficult task, point-based value iteration methods are widely used. These methods compute an approximate POMDP solution, and in some cases they even provide guarantees on the solution quality, but these algorithms have been designed for problems with an infinite planning horizon. In this paper we discuss why state-of-the-art point-based algorithms cannot be easily applied to finite-horizon problems that do not include discounting. Subsequently, we present a general point-based value iteration algorithm for finite-horizon problems which provides solutions with guarantees on solution quality. Furthermore, we introduce two heuristics to reduce the number of belief points considered during execution, which lowers the computational requirements. In experiments we demonstrate that the algorithm is an effective method for solving finite-horizon POMDPs.

1. Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a framework for planning under uncertainty in partially observable environments (Kaelbling, Littman, & Cassandra, 1998). POMDPs have been applied in several real-world applications, such as spoken dialog systems (Williams & Young, 2007) and assistance for people with dementia (Boger, Poupart, Hoey, Boutilier, Fernie, & Mihailidis, 2005). The framework has been extended to various settings, such as planning for decentralized agents (Oliehoek & Amato, 2016) and planning subject to constraints (Walraven & Spaan, 2018).

POMDPs can be solved to optimality using exact value iteration algorithms (Cassandra, Littman, & Zhang, 1997; Walraven & Spaan, 2017). However, solving POMDPs to optimality is PSPACE-complete (Papadimitriou & Tsitsiklis, 1987) and most research on POMDPs focuses on scalable approximation techniques nowadays. Most notably, point-based value iteration algorithms (Pineau, Gordon, & Thrun, 2003) have proven to be successful for computing POMDP solutions. This led to a variety of algorithms, such as randomized point-based value iteration (Spaan & Vlassis, 2005) and point-based methods with guarantees on convergence to optimality (Kurniawati, Hsu, & Lee, 2008; Smith & Simmons, 2005; Poupart, Kim, & Kim, 2011). Furthermore, several algorithms compute upper bounds on solution quality, which enables assessment of the quality of the computed policy.

In several planning domains it is natural to assume a finite horizon without discounting of reward. For example, in domains where utility should be maximized in the next 24 hours, one would be interested in the reward collected in the next 24 hours, and the utility afterwards is no longer relevant. Such situations occur in, e.g., smart energy grids, where

©2019 AI Access Foundation. All rights reserved.

charging providers solve planning problems with a finite horizon for electric vehicles (Qi, Xu, Shen, Hu, & Song, 2014), and unit commitment (Morales-España, Latorre, & Ramos, 2013) requires planning under uncertainty for creating finite-horizon schedules for power generators. Another example is condition-based maintenance in which the condition of, e.g., a machine is partially observable while performing planning for maintenance (Byon & Ding, 2010). A common characteristic of existing point-based POMDP algorithms is that they have been designed for problems with an infinite time horizon. This raises the question whether and how state-of-the-art infinite-horizon POMDP algorithms can be used for solving finite-horizon problems without discounting, while providing guarantees on solution quality and convergence. It turns out that the algorithms do not easily generalize to finite time horizons, and in most cases the algorithms can only be used for infinite-horizon POMDPs that include a discount factor that is strictly less than 1. This means that existing algorithms cannot be applied directly, and this shows that finite-horizon POMDPs require tailored point-based algorithms.

1.1 Contributions

In this paper we present a new solution approach for POMDPs with a finite time horizon. We introduce and evaluate a point-based value iteration algorithm that is suitable for solving finite-horizon POMDPs without discounting of reward. To be more specific, our contributions are the following.

First, we provide an extensive overview of strategies for solving finite-horizon POMDPs using existing algorithms. This discussion shows that state-of-the-art point-based value iteration algorithms have several limitations in finite-horizon settings, and it also shows that there is a need for tailored value iteration algorithms for solving finite-horizon POMDPs.

Second, we present the new finite-horizon point-based value iteration algorithm FiVI. This algorithm unifies several ideas that have been developed for solving infinite-horizon POMDPs, and the algorithm enables us to solve finite-horizon POMDPs while having guarantees on solution quality and convergence.

Third, we present additional heuristics which further improve the performance of FiVI. In particular, we describe a technique to reduce the number of backups that is executed by the algorithm. Additionally, we present a technique to improve the efficiency of value upper bound updates during the execution of FiVI.

Fourth, we execute several experiments which demonstrate the efficacy of FiVI when solving POMDPs with a finite time horizon. The experiments show that FiVI is an attractive approach for such problems, and it confirms that our additional heuristics have positive influence on the performance of the algorithm. Finally, the experiments show that FiVI typically finds solutions with a better value lower bound and a smaller gap compared to other strategies for finite-horizon problems.

1.2 Outline

Our paper is structured as follows. In Section 2 we introduce Partially Observable Markov Decision Processes and mathematical concepts that are typically used when computing solutions. In Section 3 we discuss why state-of-the-art value iteration algorithms for POMDPs cannot be applied to finite-horizon problems. In Section 4 we present FiVI, which is our

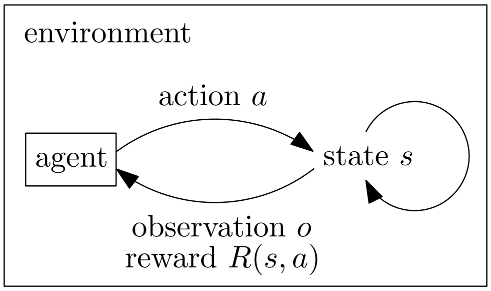

Figure 1: POMDP agent interacting with the environment

point-based value iteration algorithm that is suitable for solving finite-horizon problems. We further extend our FiVI algorithm in Section 5, which presents our backup and update strategies. In Section 6 we describe our experimental results, and in Section 7 we discuss our conclusions and directions for future work.

2. Partially Observable Markov Decision Processes

In this section we introduce Partially Observable Markov Decision Processes and we provide an overview of concepts used in solution algorithms. We consider a Partially Observable Markov Decision Process (Kaelbling et al., 1998), which models an agent that interacts with an uncertain environment. A POMDP can be defined using the tuple $M = \langle S, A, O, T, \Omega, R, b_1 \rangle$. The sets S, A and O contain a finite number of states, actions and observations, respectively. The function $T : S \times A \times S \rightarrow [0, 1]$ defines the stochastic state transitions. After executing action $a \in A$ in state $s \in S$, the state changes stochastically to state $s' \in S$ with probability $T(s, a, s') = P(s'|s, a)$. The function $R : S \times A \rightarrow \mathbb{R}$ represents the reward function, such that the reward $R(s, a)$ is received after executing action $a \in A$ in state $s \in S$. The function $\Omega : A \times S \times O \rightarrow [0, 1]$ represents the observation function. Instead of observing the state s' directly, the agent receives observation $o \in O$ with probability $\Omega(a, s', o) = P(o|a, s')$. The interaction between the agent and the environment is visualized in Figure 1. The agent executes action a, after which it receives observation o and reward $R(s, a)$. Here it is important to note that it does not receive information about the state s itself.

2.1 Belief States and Belief Updates

In fully-observable MDPs the environment state provides a Markovian signal based on which the agent can make optimal decisions. However, in POMDPs an observation does not provide sufficient information to make optimal decisions. All executed actions and observations encountered in the past can affect the knowledge the agent has about the current state, and hence a notion of memory is necessary to define an optimal decision making policy.

For POMDPs a Markovian planning signal can be defined using belief states b rather than actual states s. A belief state b is a vector of length $|S|$ defining the probability $b(s)$ that the current environment state is s. In other words, the vector characterizes the current belief of the agent regarding the actual environment state. A belief state is a sufficient statistic for the full history of actions and observations, and therefore there are no other representations which provide the agent with more information about the history. In a POMDP it is assumed that the agent has an initial belief b_1 before it starts with action execution. Several POMDP algorithms exploit this assumption while solving a POMDP, which we further discuss later.

While interacting with the environment the agent updates its belief b. After executing action a and observing o, the resulting belief b_a^o is defined using Bayes' rule:

$$b_a^o(s') = \frac{P(o|a,s')}{P(o|b,a)} \sum_{s \in S} P(s'|s,a) b(s), \tag{1}$$

where $P(o|b,a)$ corresponds to the probability to observe o after executing action a in belief b. This probability is calculated as follows:

$$P(o|b,a) = \sum_{s' \in S} P(o|a,s') \sum_{s \in S} P(s'|s,a) b(s), \tag{2}$$

and in the belief update equation this term serves as a normalizing constant. The notion of belief is used while interacting with the environment, but it is also used to characterize POMDP solutions, as discussed in the next two sections for both infinite-horizon problems and finite-horizon problems.

2.2 Infinite-Horizon Problems

Infinite-horizon POMDPs are used in problem domains where control policies have to be executed infinitely long. As an example, we consider an elevator control problem (Crites, 1996), in which the control policy needs to ensure that people get moved to the right floor in a short amount of time. From a decision making point of view this problem is partially observable, because pressing the button to request an elevator does not provide information about the destination floor. Short-term performance is important because passengers do not want to wait too long, and the notion of a finite horizon is not suitable because new passengers can arrive at any point in time in the future. For infinite-horizon problems with discounting the following optimality criterion is used:

$$E\left[\sum_{t=1}^{\infty} \gamma^{t-1}\, r_t\right], \tag{3}$$

in which a discount factor $0 \leq \gamma < 1$ is used as a weight for the reward. The discount factor ensures that short-term reward is considered more important than reward received much later in time.

The solution of an infinite-horizon POMDP is a policy $\pi : \Delta(S) \rightarrow A$ mapping beliefs to actions, in which $\Delta(S)$ denotes the continuous set of probability distributions over S. Similar to infinite-horizon MDPs, the aim is to maximize the expected sum of discounted rewards. For a given policy π the expected discounted reward $V^{\pi}(b)$ collected when executing π starting from b is defined as:

$$V^{\pi}(b) = E_{\pi}\left[\sum_{k=1}^{\infty} \gamma^{k-1} R(b_k, \pi(b_k)) \;\middle|\; b_1 = b\right], \tag{4}$$

where $R(b_t, \pi(b_t)) = \sum_{s \in S} R(s, \pi(b_t))b_t(s)$ denotes the expected reward when executing $\pi(b_t)$ in belief b_t.

For the optimal policy π^* it holds that $V^{\pi^*}(b) \geq V^{\pi}(b)$ for each $b \in \Delta(S)$ and for all policies π. Similar to MDPs it satisfies the Bellman optimality equation:

$$V^{\pi^*}(b) = \max_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o \in O} P(o|b,a)V^{\pi^*}(b_a^o) \right]. \tag{5}$$

The optimal policy π^* corresponding to this value function is defined as:

$$\pi^*(b) = \arg\max_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o \in O} P(o|b,a)V^{\pi^*}(b_a^o) \right]. \tag{6}$$

The value functions introduced in this section provide a conceptual characterization of an optimal value function and the corresponding optimal policy.

Although discounting can be justified from an application point of view in several domains, in many cases it is only used for mathematical convenience (Hansen, 2007). Discounting can be convenient because it ensures that the sum of an infinite number of rewards, as shown in Equation 3, becomes equivalent to the sum of a finite number of rewards. This means that the expectation becomes a well-defined and finite sum, and it means that it becomes possible to solve infinite-horizon problems by considering only a finite number of time steps in the future. In the next section we discuss modeling of finite-horizon problems which do not involve discounting.

2.3 Finite-Horizon Problems

Finite-horizon POMDPs are used in domains where a policy is executed during a finite number of time steps. As an example we consider an electric vehicle (EV) charging provider which optimizes day-to-day operations based on finite-horizon forecasts of, e.g., electricity price and charging demand. In such domains it can be the objective to charge a fleet of EVs as cheap as possible while accounting for the uncertainty in arrival time and demand. It is natural to compute a policy which maximizes the expected sum of reward:

$$E\left[\sum_{t=1}^{h} r_t \right], \tag{7}$$

in which we intentionally count from $t = 1$ rather than $t = 0$, such that there are h steps in total. Although discounting can be applied if there is a finite number of time steps, we focus in this paper on finite-horizon problems that do not include a discount factor.

In the finite-horizon case the solution is a non-stationary policy $\pi : \{1, \ldots, h\} \times \Delta(S) \rightarrow A$, which maps beliefs and time steps to actions, and it maximizes the expected sum of rewards received by the agent. A policy can be seen as a plan which enables the agent to perform its task in the best possible way, and its quality can be evaluated using a value function $V^{\pi} : \{1, \ldots, h\} \times \Delta(S) \rightarrow \mathbb{R}$. The value $V^{\pi}(t, b)$ denotes the expected sum of rewards that the agent receives when following policy π starting from belief b at time t, and it is defined as:

$$V^{\pi}(t,b) = E_{\pi}\left[\sum_{t'=t}^{h} R(b_{t'}, \pi(t', b_{t'})) \;\middle|\; b_t = b \right], \tag{8}$$

where $b_{t'}$ is the belief at time t' and $R(b_{t'}, \pi(t', b_{t'})) = \sum_{s \in S} R(s, \pi(t', b_{t'}))b_{t'}(s)$. For an optimal policy π^* it holds that it always achieves the highest possible expected reward during execution. Formally, it holds that $V^{\pi^*}(1,b) \geq V^{\pi}(1,b)$ for each belief b and for each possible policy π. The optimal value function $V^{\pi^*}(t,b) = \max_{\pi} V^{\pi}(t,b)$ is defined by the following recurrence:

$$V^{\pi^*}(t,b) = \begin{cases} \max_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) + \sum_{o \in O} P(o|b,a) V^{\pi^*}(t+1, b_a^o) \right] & t \leq h \\ 0 & t > h \end{cases}. \tag{9}$$

The optimal policy π^* corresponding to the optimal value function is defined as:

$$\pi^*(t,b) = \arg\max_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) + \sum_{o \in O} P(o|b,a) V^{\pi^*}(t+1, b_a^o) \right], \tag{10}$$

for $1 \leq t \leq h$. It returns the value-maximizing action given a time step and belief.

2.4 Vector-Based Value Functions, Backups and Value Iteration

The value functions in the previous sections have been defined over the continuous belief space. When computing value functions this can be inconvenient, because it requires function representations as well as function manipulations defined over a continuous space. Fortunately, it has been shown that POMDP value functions have a particular shape which allows for more efficient representations. In this section we provide an introduction to value functions for infinite-horizon problems. The connection with finite-horizon problems is made in Section 4. The notation in this section has been partially derived from Spaan (2012).

It turns out that value functions are piecewise linear and convex (Sondik, 1971). This means that the value function can be represented using a finite set of $|S|$-dimensional vectors. This also applies to infinite-horizon problems, because the discount factor γ implicitly defines an upper bound on the number of time steps that should be considered. A value function V can be represented as a set of vectors $\alpha \in V$, such that

$$V(b) = \max_{\alpha \in V} b \cdot \alpha, \tag{11}$$

where $\cdot$ denotes the inner product. In this representation V refers to a set of vectors, and $V(b)$ denotes the function value computed using b and the set of vectors.

Value iteration algorithms can be used to compute a value function $V^{\pi^*}(b)$ that characterizes an optimal POMDP solution, as defined in Equation 5. Value iteration executes a series of dynamic programming stages based on Equation 5 until the value function converges. If the agent executes only one action, then the initial value function $V_0(b)$ is defined as:

$$V_0(b) = \max_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) \right] = \max_{\{\alpha_0^a\}_{a \in A}} \alpha_0^a \cdot b, \tag{12}$$

where $\alpha_0^a(s) = R(s,a)$ denotes a vector containing the immediate rewards. Hence, we can define this value function in terms of vectors as $V_0 = \{\alpha_0^a \mid a \in A\}$.

Given a value function V_n, value iteration algorithms compute the value function V_{n+1} using the Bellman equation. We can abbreviate this as $V_{n+1} = HV_n$, in which H denotes the Bellman backup operator. For convenience we let $\alpha_n^b = \arg\max_{\alpha \in V_n} b \cdot \alpha$ denote the value-maximizing vector from the set V_n in belief b. Computing all vectors belonging to V_{n+1} seems computationally difficult, but given V_n and a belief b we can easily compute the vector α_{n+1}^b such that $\alpha_{n+1}^b = \arg\max_{\alpha \in V_{n+1}} b \cdot \alpha$, where V_{n+1} is the unknown set of vectors representing HV_n. We refer to this operation as executing a backup on belief b:

$$\alpha_{n+1}^b = \texttt{backup}(b), \tag{13}$$

such that $V_{n+1}(b) = b \cdot \texttt{backup}(b)$. It is important to observe that this vector represents the gradient of the value function V_{n+1} in belief b.

We can derive the computation of $\texttt{backup}(b)$ directly from the Bellman optimality equation. For convenience we first define

$$g_{ao}^{\alpha_n}(s) = \sum_{s' \in S} P(o|a,s')P(s'|s,a)\alpha_n(s') \tag{14}$$

as the backprojection of a vector $\alpha_n \in V_n$ based on action a and observation o. The derivation for the infinite-horizon case with discounting now proceeds as follows:

$$V_{n+1}(b) = \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} P(o|b,a)V_n(b_a^o)\right] \tag{15}$$

$$= \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} P(o|b,a) \max_{\alpha_n \in V_n}\left(\sum_{s' \in S} b_a^o(s')\alpha_n(s')\right)\right] \tag{16}$$

$$= \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} \max_{\alpha_n \in V_n} \sum_{s' \in S} P(o|a,s') \sum_{s \in S} P(s'|s,a)b(s)\alpha_n(s')\right] \tag{17}$$

$$= \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} \max_{\alpha_n \in V_n} \sum_{s \in S} b(s) \sum_{s' \in S} P(o|a,s')P(s'|s,a)\alpha_n(s')\right] \tag{18}$$

$$= \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} \max_{\{g_{ao}^{\alpha_n}\}_{\alpha_n \in V_n}} b \cdot g_{ao}^{\alpha_n}\right] \tag{19}$$

$$= \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} b \cdot \arg\max_{\{g_{ao}^{\alpha_n}\}_{\alpha_n \in V_n}} b \cdot g_{ao}^{\alpha_n}\right] \tag{20}$$

$$= \max_{a \in A}\left[b \cdot \alpha_0^a + \gamma b \cdot \sum_{o \in O} \arg\max_{\{g_{ao}^{\alpha_n}\}_{\alpha_n \in V_n}} b \cdot g_{ao}^{\alpha_n}\right] \tag{21}$$

$$= \max_{a \in A}\left[b \cdot \left(\alpha_0^a + \gamma \sum_{o \in O} \arg\max_{\{g_{ao}^{\alpha_n}\}_{\alpha_n \in V_n}} b \cdot g_{ao}^{\alpha_n}\right)\right] \tag{22}$$

$$= \max_{a \in A}\left[b \cdot g_a^b\right] \tag{23}$$

$$= b \cdot \underset{\{g_a^b\}_{a \in A}}{\arg\max} \left[b \cdot g_a^b \right] \tag{24}$$

with

$$g_a^b = \alpha_0^a + \gamma \sum_{o \in O} \underset{\{g_{ao}^{\alpha_n}\}_{\alpha_n \in V_n}}{\arg\max} \; b \cdot g_{ao}^{\alpha_n}. \tag{25}$$

Note that we have applied the definition of the belief update, the identity $b \cdot x + b \cdot y = b \cdot (x+y)$ and the identity $\max_\alpha b \cdot \alpha = b \cdot \arg\max_\alpha b \cdot \alpha$ in the derivation. Now we can define the backup operator as follows:

$$\texttt{backup}(b) = \underset{\{g_a^b\}_{a \in A}}{\arg\max} \left[b \cdot g_a^b \right]. \tag{26}$$

The operator is easy to implement and it provides the value-maximizing vector $\alpha_{n+1}^b \in V_{n+1}$ in belief b based on the value function V_n and b itself. It is also common to associate the maximizing action a with a vector α, which is denoted by $a(\alpha)$.

Value iteration for POMDPs repeatedly computes a value function V_{n+1} using the vectors representing the value function V_n from the previous stage. Based on the backup operator we can define this computation as

$$\bigcup_{b \in \Delta(S)} \texttt{backup}(b), \tag{27}$$

in which $\Delta(S)$ represents the continuous space of $|S|$-dimensional beliefs. However, the computation requires knowledge about the beliefs b which are needed to compute all vectors belonging to this set. Since there is an infinite number of beliefs, enumeration of beliefs is clearly not possible.

Most exact value iteration algorithms address the aforementioned problem by enumerating all possible vectors that can be generated by the backup operator, rather than enumerating all possible beliefs (Monahan, 1982). The current state of the art is the incremental pruning algorithm (Cassandra et al., 1997; Walraven & Spaan, 2017), which uses a vector enumeration procedure interleaved with a procedure that discards vectors that do not contribute to the value function. However, solving POMDPs to optimality is PSPACE-complete (Papadimitriou & Tsitsiklis, 1987) and this approach can only be used for relatively small POMDP models.

Point-based value iteration methods (Pineau et al., 2003) have emerged as a popular approach to address the tractability of solving POMDPs. These methods optimize based on a finite number of beliefs in a set B, rather than optimizing over the entire continuous belief simplex $\Delta(S)$:

$$\bigcup_{b \in B} \texttt{backup}(b). \tag{28}$$

The quality of solutions computed by point-based value iteration algorithms is highly dependent on the choice of B. Several different strategies have been proposed to initialize and update this set. For example, Perseus (Spaan & Vlassis, 2005) explores the POMDP environment randomly and stores the belief points it finds. More recent algorithms such as HSVI (Smith & Simmons, 2005), SARSOP (Kurniawati et al., 2008) and GapMin (Poupart et al., 2011) incrementally expand the set B based on heuristic search. This search aims to find belief points that are reachable during the execution of an (initially unknown) optimal policy. The relationship between the quality of the heuristic search and the complexity of solving POMDPs has been studied by Zhang, Hsu, and Lee (2014).

The aforementioned point-based algorithms provide an attractive approach to solve infinite-horizon POMDPs while providing guarantees on convergence to optimality. However, these algorithms have not been designed for solving finite-horizon problems, which raises the question how problems with a finite horizon can be solved. We further discuss this in the next section.

3. Limitations of POMDP Algorithms in Finite-Horizon Settings

The state of the art in solving POMDPs supports problems which include discounting of reward. Unfortunately, it turns out that these algorithms do not easily generalize to finite-horizon problems without discounting. In this section we provide an overview of approaches that may be used for solving such finite-horizon problems, and we argue why existing algorithms for infinite horizons cannot be applied with discount factor $\gamma = 1$. Throughout this section finite-horizon planning refers to planning with a finite time horizon without discounting of reward.

3.1 Solution Strategies for Finite-Horizon Problems

This section gives an overview of approaches that can be used to solve finite-horizon POMDPs without discounting. For each approach we explain how existing techniques for MDPs and infinite-horizon POMDPs can be potentially applied, and we argue why these techniques have limitations when solving finite-horizon problems without a discount factor.

The first approach we discuss treats the POMDP as a fully-observable MDP. Since the number of reachable beliefs in a finite-horizon POMDP is finite, it is possible to enumerate these beliefs prior to planning. Recall from Section 2.1 that POMDP belief states provide a Markovian signal for a POMDP planning task. Therefore, after belief enumeration it is possible to solve a regular MDP defined in terms of belief states rather than actual states. This approach is well-defined and it provides an optimal finite-horizon POMDP policy, but unfortunately it is often intractable due to the large number of beliefs that needs to be enumerated, which is at most $(|A||O|)^h$.

A second approach based on infinite-horizon algorithms would simply compute an infinite-horizon policy to take decisions in a finite-horizon problem. This is straightforward, because one can assume a discount factor, after which an infinite-horizon algorithm is invoked to obtain a policy. There are two disadvantages associated with this approach. First, invoking an infinite-horizon algorithm leads to undesirable effects if the algorithm thinks that reward can be collected late in time, whereas execution ends early due to the finite time horizon. For example, if a policy has been optimized under the assumption that high reward can be collected after 20 steps, then the policy is unlikely to be optimal if execution ends after 5 steps. The second disadvantage of the approach are the undesirable effects due to the discount factor that is assumed, which we illustrate using an example. We consider a POMDP with fully-observable states and deterministic state transitions, as shown in Figure 2. In the initial state, the agent chooses either action a_1 or a_2, leading to either the top or bottom

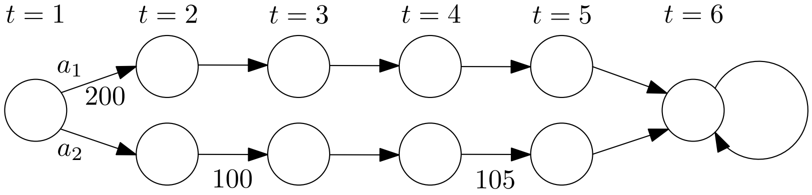

Figure 2: POMDP in which discounting causes suboptimality

trajectory. The numbers below the transitions correspond to reward, and transitions without a number have zero reward. When casting the problem to an infinite-horizon problem with $\gamma = 0.95$, then a_1 is optimal since it gives expected reward $0.95^0 \cdot 200 = 200$, while a_2 gives expected reward $0.95 \cdot (100 + 0.95^2 \cdot 105) = 185.02$. However, a_2 is optimal for a finite-horizon problem where all rewards are equally important, because the bottom trajectory gives reward 205 while the top trajectory gives 200. This shows that casting a finite-horizon problem to an infinite-horizon problem with discounting can lead to suboptimal policies.

A third strategy augments the POMDP with a time state variable as part of the state description. This means that states become time-indexed, and a trap state is entered at the end of the horizon, resulting in a model with $|S| \times h + 1$ states. More efficient encodings are possible if not all states are reachable during all steps, but in general we can conclude that this strategy does not scale well if a large number of time steps needs to be considered. Although the increase of the model size is linear in the number of time steps, the augmented POMDP model and solution representations (e.g., alpha vectors) quickly become too large, which significantly increases the running time and memory requirements of POMDP algorithms. Augmenting states with a time state variable is not sufficient to obtain a finite-horizon policy. In addition, it is required to assume $\gamma = 1$, but this assumption leads to implementation issues and undesirable effects in several state-of-the-art algorithms. This is further discussed in the next section.

A fourth approach would interpret the aforementioned augmented POMDP with a trap state as a stochastic shortest path problem for Goal POMDPs. The Goal POMDP formulation assumes that the POMDP has a fully-observable goal state that cannot be left, which is the case when defining the POMDP with a trap state at the end of the time horizon. Real-Time Dynamic Programming (RTDP) can be used to find solutions to such problems and it has been generalized to POMDPs as well (Bonet & Geffner, 2009). The resulting RTDP-Bel algorithm does not include discounting and it can potentially be adapted to support time-dependent value functions. However, due to discretization of belief states it does not provide performance guarantees and it does not keep track of an upper bound on the optimal value function. Existing RTDP extensions for MDPs do account for upper bounds (Smith & Simmons, 2006), but to the best of our knowledge these upper bounds have not been applied in RTDP for problems with partial observability.

A fifth approach for solving finite-horizon problems consists of an adaptation of the algorithm α-min (Dujardin, Dietterich, & Chadès, 2015). This algorithm keeps track of separate value functions for each time step, and it imposes the additional restriction that there should be a maximum of N vectors for each time step. The algorithm may be applied without this restriction and with a low gap tolerance, but in that case it starts to invoke a large number of mixed-integer linear programs in order to expand the belief sets, which are expensive to solve and this leads to scalability problems. Other adaptations of α-min provide more scalability but they do not provide any performance guarantees (Dujardin, Dietterich, & Chadès, 2017).

Based on our discussion we can conclude that there are several straightforward approaches for solving finite-horizon POMDPs without discounting, but all these approaches are affected by either scalability problems or undesirable effects. In the next section we describe why state-of-the-art POMDP algorithms cannot be used for finite-horizon models with a discount factor that is equal to 1.

3.2 Discarding the Discount Factor in Infinite-Horizon Algorithms

As noted in the previous section, the application of infinite-horizon algorithms to finite-horizon formulations with time-indexed states requires a discount factor γ that is equal to 1. Unfortunately, many state-of-the-art algorithms for infinite-horizon problems do not support such a discount factor, and they cannot be modified easily without changing the characteristics. Next, we discuss for each algorithm why it cannot be used for finite-horizon planning with the discount factor $\gamma = 1$. We also discuss whether the algorithms converge to optimality, and whether they compute an upper bound on an optimal solution. In our discussions in this section optimality refers to optimal with respect to an initial belief. This means that the computed solutions are not necessarily optimal for any initial belief.

GapMin (Poupart et al., 2011) is a point-based value iteration algorithm which computes both lower bounds and upper bounds on the optimal value function, and it converges in the limit to an optimal POMDP solution. The algorithm contains several subroutines which require $\gamma < 1$, and the algorithm is not well-defined in case we set $\gamma = 1$. Assuming a discount factor $\gamma < 1$ that is arbitrarily close to 1 leads to a situation in which many subroutines have slow convergence, which is undesirable. Without significant adaptations GapMin cannot be used with $\gamma = 1$.

The point-based value iteration algorithms SARSOP (Kurniawati et al., 2008) and HSVI (Smith & Simmons, 2005) follow a similar approach as GapMin, in the sense that they also incrementally expand a set of belief points based on heuristic search starting from the initial belief. They also produce an upper bound on the optimal value function, and the algorithms converge to optimality in the limit. The backups and upper bound updates performed by the algorithms are well-defined for $\gamma = 1$. However, the initialization of lower bounds and upper bounds require $\gamma < 1$ and therefore it is necessary to initialize them differently. Similar to GapMin, without adaptations both algorithms cannot be used with $\gamma = 1$.

Perseus (Spaan & Vlassis, 2005) is a randomized point-based value iteration algorithm which iteratively performs backups on a set of randomly-sampled belief points. The initialization of the lower bound requires $\gamma < 1$, and therefore this also requires modification. The algorithm does not keep track of an upper bound on the optimal value function, and it provides no guarantees on performance, which means that it is not guaranteed to converge to optimality.

The original PBVI algorithm (Pineau et al., 2003) executes backups on a belief set that is expanded incrementally. The algorithm can be interpreted as an anytime algorithm,

Algorithm	Upper bound	Convergence to optimality	Supports $\gamma = 1$
GapMin	✓	✓	
SARSOP	✓	✓	
HSVI	✓	✓	
Perseus			
PBVI		✓	✓
Exact VI		✓	✓
RTDP-Bel			✓
FiVI	✓	✓	✓

Table 1: Comparison of infinite-horizon algorithms and FiVI

and for reaching an optimal solution this boils down to full enumeration of the reachable belief space. The bounds on the worst-case error assume a discount factor $\gamma < 1$, but the algorithm itself can be used without discounting. In general this is still not desirable because the number of belief points is potentially large, and it has been shown empirically that the algorithms GapMin, SARSOP, HSVI and Perseus typically outperform the original PBVI algorithm.

Exact value iteration supports the discount factor $\gamma = 1$ and it always computes an optimal policy by definition. However, due to its limited scalability it is not desirable to use the algorithm for problems with large state spaces, which would be the case if we use a formulation with time-indexed states and a trap state. In contrast to the approximate methods that we discussed, exact value iteration computes solutions that are optimal for any initial belief.

An overview of the algorithm characteristics is presented in Table 1, which compares the algorithms in term of their ability to compute an upper bound, convergence to optimality, and immediate support for discount factors $\gamma = 1$. RTDP-Bel has also been included in the table, which we briefly discussed in Section 3.1. As can be seen, there is no existing algorithm which has all three properties simultaneously. In contrast, the algorithm FiVI presented in the next section does have all these properties, as shown in the table. The algorithm unifies the desirable characteristics of GapMin, SARSOP and HSVI in such a way that we obtain a finite-horizon point-based value iteration algorithm for problems without discounting, which converges to optimality and it also computes both lower bounds and upper bounds.

4. FiVI: Finite-Horizon Point-Based Value Iteration

In this section we describe FiVI, a point-based value iteration algorithm for solving finite-horizon POMDPs. The algorithm unifies techniques and concepts from existing state-of-the-art point-based value iteration algorithms and it provides attractive convergence characteristics and optimality guarantees. This section describes the solution representations used by FiVI, the actual algorithm, its theoretical properties and relations to existing algorithms.

We start with an overview of the high-level structure of the solution computed by the FiVI algorithm in Section 4.1, based on time-dependent value functions and time-dependent backups. In Section 4.2 we explain how time-dependent value upper bounds can be obtained in a finite-horizon setting using the sawtooth approximation. A full description of the FiVI algorithm is provided in Section 4.3, which includes the aforementioned value functions and upper bounds. The convergence and optimality characteristics of the FiVI algorithm depend on the belief points used for computing the value functions and upper bounds. This is the topic of Section 4.4, in which we provide a heuristic search procedure for finding beliefs, as well as a motivation which explains that FiVI converges to an optimal solution.

4.1 Time-Dependent Value Functions and Backups

Point-based value iteration algorithms compute value functions represented by a finite set of vectors, as introduced in Section 2.4. For infinite-horizon problems it suffices to keep track of one individual value function V, as defined in Equation 11, which represents the stationary policy that can be used to choose actions. In the finite-horizon case the policy is non-stationary, and in general it is no longer possible to encode the policy using just one value function. In our FiVI algorithm we use time-dependent value functions $\mathcal{V}_t$, in which t refers to a time step ranging from 1 to h. Note that we use $\mathcal{V}_t$ rather than V_t to avoid notation conflicts with infinite-horizon value iteration. The value function $\mathcal{V}_t$ is represented by a finite set of vectors Γ_t and it can be defined as follows:

$$\mathcal{V}_t(b) = \max_{\alpha \in \Gamma_t} b \cdot \alpha, \tag{29}$$

such that $\mathcal{V}_t(b)$ corresponds to the expected reward collected when executing the policy induced by the value functions $\mathcal{V}_t, \ldots, \mathcal{V}_h$ starting from belief b.

The vectors that constitute a value function can be computed using a point-based backup operator, as defined by Equation 26. In the infinite-horizon case the backups are executed on beliefs in a set B, as shown in Equation 28. Similar to the value functions, the belief sets can be made time dependent for the finite-horizon case, such that Γ_t is computed using the beliefs in the set B_t. In our algorithm we need to keep track of upper bounds $\bar{v}$ associated with beliefs b, and therefore the elements of the set B_t consist of pairs $(b, \bar{v}) \in B_t$. The role of the upper bounds will be further described in the next section. The vector set Γ_t can be obtained as follows:

$$\Gamma_t = \bigcup_{(b,\bar{v}) \in B_t} \texttt{backup}(b, t), \tag{30}$$

where $\texttt{backup}(b, t)$ denotes a time-dependent backup operator that uses the vectors in Γ_{t+1} to compute a vector belonging to Γ_t. The time-dependent backup operator corresponds to the original backup operator for infinite-horizon POMDPs, but it has been formulated based on multiple time-dependent vector sets rather than one individual vector set. The time-dependent backup operator $\texttt{backup}(b, t)$ is defined as follows:

$$\texttt{backup}(b, t) = \operatorname*{arg\,max}_{\{z_{b,a,t}\}_{a \in A}} b \cdot z_{b,a,t}, \tag{31}$$

where

$$z_{b,a,t} = \begin{cases} r_a + \sum_{o \in O} \arg\max_{\{z_{a,o}^{k,t+1}\}_k} b \cdot z_{a,o}^{k,t+1} & t < h \\ r_a & t = h \end{cases}, \tag{32}$$

and $z_{a,o}^{k,t}$ denotes the backprojection of vector $\alpha^{k,t} \in \Gamma_t$:

$$z_{a,o}^{k,t}(s) = \sum_{s' \in S} P(o|a, s')P(s'|s, a)\alpha^{k,t}(s') \quad \forall s. \tag{33}$$

The vector r_a contains the immediate reward for action a. In the remainder of the paper we assume that the backup operator has access to all vector sets and the reward vectors r_a, such that additional arguments can be discarded from the equations and pseudocode.

Our finite-horizon point-based value iteration algorithm FiVI computes multiple time-dependent value functions $\mathcal{V}_t$ represented by vector sets Γ_t using the time-dependent backup operator that we introduced. The actual integration in the algorithm will be explained in Section 4.3, which discusses the algorithm in more detail.

4.2 Time-Dependent Value Upper Bounds and Bound Updates

Point-based value iteration algorithms typically keep track of upper bounds on the optimal expected value, which enables assessment of the quality of the computed solution. Our FiVI algorithm also includes such computations of upper bounds, for which we provide the required notation and algorithms in this section. The algorithms closely follow the upper bound computations for infinite-horizon POMDPs, but in order to improve understandability we provide a full description in this section.

We consider a time step t and the corresponding belief set B_t. Recall from the previous section that the pairs $(b, \bar{v}) \in B_t$ also contain a value upper bounds $\bar{v}$ corresponding to belief b. These upper bounds $\bar{v}$ can be used to obtain an upper bound for another belief b' that is not represented in the set B_t, based on an upper bound interpolation using the existing beliefs in B_t (Hauskrecht, 2000). The interpolation can be obtained using the following linear program:

$$\begin{aligned} \min \quad & \sum_{(b,\bar{v}) \in B_t} c_b \cdot \bar{v} & \\ \text{s.t.} \quad & \sum_{(b,\bar{v}) \in B_t} c_b \cdot b(s) = b'(s) & \forall s \\ & c_b \geq 0 & \forall (b, \bar{v}) \in B_t. \end{aligned} \tag{34}$$

which assigns weights to the pairs in B_t and returns a linear combination of the upper bounds $\bar{v}$ represented by B_t.

Solving a linear program for every upper bound interpolation can be computationally expensive, and therefore it is more common to use a so-called sawtooth approximation (Hauskrecht, 2000). This approximation is based on the idea that the optimization problem can be simplified by imposing the constraint that weights c_b are assigned to corners of the belief simplex, and at most one belief that is not a corner of the belief simplex. A corner of the belief simplex is a belief in which the belief associated with one state equals 1, and we also refer to such a belief as a corner belief. Under the additional assumptions that we made the upper bound interpolation can be computed using a simple procedure that we call `UB`, as shown in Algorithm 1, rather than solving a linear program. The algorithm takes an arbitrary belief set B and a belief b' as input, and it returns an upper bound

Algorithm 1: Sawtooth approximation (`UB`)

```
input  : belief $b'$, set $B$ containing belief-bound pairs
output: upper bound corresponding to belief $b'$

for $(b, \bar{v}) \in B \setminus \{(e_s, \cdot) \mid s \in S\}$ do
    $f(b) \leftarrow \bar{v} - \sum_{s \in S} b(s) B(e_s)$
    $c(b) \leftarrow \min_{s \in S} b'(s) \ / \ b(s)$
end
$b^* \leftarrow \arg\min_{\{b \mid (b,\bar{v}) \in B \setminus \{e_s \ \mid \ s \in S\}\}} c(b) f(b)$
return $c(b^*) f(b^*) + \sum_{s \in S} b'(s) B(e_s)$
```

interpolation for b' based on the belief-bound pairs in B. In the algorithm e_s denotes the corner belief corresponding to state s, and the for loop iterates over all pairs $(b, \bar{v}) \in B$ for which b is not a corner of the belief simplex. Furthermore, $B(e_s)$ denotes the upper bound that is currently associated with e_s in the set B. Our notation closely follows the notation used by Poupart et al. (2011), and a justification of the procedure has been described by Smith (2007). An additional description of upper bound computations has been provided by Shani, Pineau, and Kaplow (2013).

In the finite-horizon setting the upper bounds associated with beliefs can be updated in a point-based fashion, similar to executing regular backups on beliefs. We consider a time step $t < h$ and a belief b that belongs to B_t. The upper bound $\bar{v}$ in $(b, \bar{v}) \in B_t$ can be updated as follows:

$$\max_{a \in A} \sum_{s \in S} R(s,a) b(s) + \sum_{o \in O} P(o|b,a) \cdot \texttt{UB}(b_a^o, B_{t+1}), \tag{35}$$

in which the upper bound interpolation is based on the set B_{t+1} corresponding to the next time step. For the final time step $t = h$ it suffices to consider the immediate rewards, and the upper bound is defined by $\max_{a \in A} r_a \cdot b$. In the next section we combine the upper bound update scheme and the time-dependent value functions to create our FiVI algorithm.

4.3 Algorithm Description of FiVI

The FiVI algorithm takes a POMDP model as input and computes a solution by executing a series of iterations. Within an iteration three phases can be distinguished. First the algorithm executes a procedure to find new belief points. After that, the algorithm computes a new vector set Γ_t for each time step t. Finally, the algorithm updates the upper bounds represented by the belief sets B_t. The full description of the algorithm is provided in Algorithm 2, which we discuss below in more detail.

On lines 1-5 the algorithm starts with initializing vector sets Γ_t, belief sets B_t and the immediate reward vectors r_a. Furthermore, the auxiliary variable τ' is used to keep track of the elapsed time, and δ represents an iteration counter. The latter is used in one of our heuristics in Section 5.

An iteration of FiVI starts with a call to a procedure `expand`, which is used to find additional beliefs on line 8. The quality of the solution returned by FiVI and the convergence of the algorithm completely depends on these beliefs, because these beliefs are used for

Algorithm 2: Finite-horizon point-based Value Iteration (FiVI)

```
input  : POMDP $M$, precision $\rho$, time limit $\tau$
output: sets $\Gamma_t$ for each time step $t$, upper bound $v_u$
$\Gamma_t \leftarrow \emptyset \quad \forall t$
$B_t \leftarrow \emptyset \quad \forall t$
$r_a \leftarrow (R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)) \quad \forall a$
add corner beliefs to $B_t$ with upper bound $\infty$  $(\forall t)$
$\tau' \leftarrow 0, \quad \delta \leftarrow 0$
do
    $\delta \leftarrow \delta + 1$
    expand$(M, \{\Gamma_1, \ldots, \Gamma_h\}, \{B_1, \ldots, B_h\}, r)$
    for $t = h, h-1, \ldots, 1$ do
        $\Gamma_t \leftarrow \emptyset$
        for $(b, \bar{v}) \in B_t$ do
            $\alpha \leftarrow$ backup$(b, t)$
            $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha\}$
        end
        for $(b, \bar{v}) \in B_t$ do
            $\bar{v} \leftarrow -\infty$
            for $a \in A$ do
                $v \leftarrow r_a \cdot b$
                if $t < h$ then
                    for $o \in O$ do
                        if $P(o|b, a) > 0$ then
                            $v \leftarrow \; v + P(o|b, a) \cdot \texttt{UB}(b_a^o, B_{t+1})$
                        end
                    end
                end
                $\bar{v} \leftarrow \max(\bar{v}, v)$
            end
        end
    end
    $v_l \leftarrow \max_{\alpha \in \Gamma_1} \alpha \cdot b_1$
    $v_u \leftarrow$ upper bound $\bar{v}$ associated with $(b_1, \bar{v}) \in B_1$
    $g_a \leftarrow 10^{\lceil \log_{10}(\max(|v_l|, |v_u|)) \rceil - \rho}$
    $\tau' \leftarrow$ elapsed time after the start of the algorithm
while $\tau' < \tau \;\wedge\; v_u - v_l > g_a$;
return $(\{\Gamma_1, \ldots, \Gamma_h\}, v_u)$
```

computing the value functions. A more detailed description of the procedure is deferred to the next section, which provides a detailed motivation and algorithmic description.

On lines 9-29 the algorithm computes alpha vectors and value upper bounds by iterating backwards over all time steps. The algorithm starts at the end of the horizon h, and it proceeds with the time steps $h-1, h-2, \ldots$ until the initial step is reached. On lines 11-14 the algorithm computes a new vector set Γ_t by executing backups based on a belief b and based on the value function Γ_{t+1} computed in the previous iteration. In this part of the algorithm we use the value functions and backup operator that we have introduced in Section 4.1. After computing the new vectors for Γ_t, the algorithm updates all upper bounds defined by B_t on lines 15-28. For this purpose it uses Equation 35 and the sawtooth approximation from Section 4.2.

An iteration of FiVI ends with the computation of the current value lower bound v_l and upper bound v_u for the initial belief b_1. The difference between these two bounds defines the current gap. Value iteration stops in case a time limit τ has been exceeded, or in case the gap is at most one unit at the ρ-th significant digit. The latter can be checked by computing the maximum allowed gap g_a under this criterion, as shown on line 32, and the algorithm terminates if the current gap is smaller than g_a. This condition is also used by GapMin, and it is more generic than imposing an absolute threshold on the gap.

The solution returned by FiVI consists of alpha vectors in sets $\Gamma_1, \ldots, \Gamma_h$, representing the lower bound. In addition, the algorithm returns the upper bound v_u that corresponds to the initial belief b_1. The gap defined by the lower bound and upper bound implicitly defines a guarantee on the quality of the computed solution.

4.4 Belief Points and Convergence of the Algorithm

The computation of vectors and upper bounds assumes that we have a set of beliefs B_t for each time step. However, the performance of the algorithm and the quality of computed solution are highly dependent on the actual belief points for which backups are executed. Computing high-quality policies requires coverage of the region of the belief space that is reachable under the execution of an optimal policy. Unfortunately, the optimal policy and the corresponding reachable belief region are initially unknown, which means that these reachable belief points need to be found while computing a policy.

The algorithm FiVI incrementally expands the belief sets using heuristic search, which is guided by the current gap between the value lower bound and upper bound. Our heuristic search procedure is similar to the procedures found in HSVI, SARSOP and GapMin. Below we describe why the action and observation selection strategies in our belief search steer the algorithm in the direction of an optimal solution.

The gap between the lower and upper bound of belief b_1 at time 1 implicitly defines the amount of uncertainty regarding the optimality of the solution. It is important to note that regret of the returned solution is bounded by the gap of the initial belief b_1. This means that the heuristic search procedure should choose actions and observations in such a way that backups and upper bound updates effectively reduce the overall gap.

In order to decide which action needs to be chosen, we first look at the effect of backups and upper bound updates on the gap associated with a belief. For the lower bound the backup is defined by Equation 9, and it maximizes over actions a. We define $V(t, b, a)$ as the new expected value when choosing action a in belief b at time t:

$$V(t,b,a) = \sum_{s\in S} R(s,a)b(s) + \sum_{o\in O} P(o|b,a)V(t+1,b_a^o). \tag{36}$$

In a similar way we can define the potential upper bound $U(t,b,a)$ that is considered for action a and belief b at time t in the update defined in Equation 35:

$$U(t,b,a) = \sum_{s\in S} R(s,a)b(s) + \sum_{o\in O} P(o|b,a)\cdot \texttt{UB}(b_a^o, B_{t+1}), \tag{37}$$

Since both Equation 9 and Equation 35 maximize over actions, the new gap associated with belief b at time t is defined by:

$$\max_{a\in A}\ U(t,b,a) - \max_{a\in A}\ V(t,b,a). \tag{38}$$

It can be seen that the new gap is determined by the actions a that maximize $U(t,b,a)$ and $V(t,b,a)$. This suggests that the heuristic search procedure should choose one of these two maximizing actions in order to affect the gap associated with b. An action a should be chosen that maximizes $U(t,b,a)$, because if a is suboptimal then its upper bound will eventually be lower than the upper bound associated with another action, which will change the action choice later. This behavior cannot be achieved using the action a that maximizes $V(t,b,a)$ because the lower bound can only increase and therefore it is not possible to detect the potential suboptimality of this action choice. The action selection strategy that we use is also known as the IE-MAX heuristic (Kaelbling, 1993) and it ensures the convergence of the algorithm. A theoretical analysis of the action selection rule has been provided by Ross, Pineau, and Chaib-Draa (2008) for general online heuristic search algorithms for POMDPs. The action selection rule ensures that the computed policy defines an ε-optimal action within finite time, which implies that the algorithm converges to optimality in the limit[1]. The action selection strategy that we use is identical to the strategy used in HSVI, SARSOP and GapMin for infinite-horizon problems.

After selecting an action the search procedure chooses a branch in the search tree that corresponds to an observation. It is important to note that the lower bounds and upper bounds associated with all reachable beliefs in time steps $t > 1$ contribute to the gap associated with the initial belief b_1. The reason is that both the lower bound computation and the upper bound update follow the structure of the Bellman equation, as shown in Equations 9 and 35. If one of the bounds associated with a reachable belief is not tight, it also contributes to the gap associated with the initial belief b_1, and therefore it is important to execute backups and updates on such reachable beliefs. Our algorithm chooses an observation leading to a belief with maximum gap in the next time step $t+1$:

$$\operatorname*{arg\,max}_{\{o\in O \ \mid\ P(o|b,a)>0\}} \left\{\ \texttt{UB}(b_a^o, B_{t+1}) - \max_{\alpha\in\Gamma_{t+1}} \alpha\cdot b_a^o\ \right\}. \tag{39}$$

A similar criterion, weighted appropriately by the discount factor, is also used by infinite-horizon algorithms.

[1] From a theoretical perspective it is required to use the exact upper bound computation in order to ensure that convergence results are unaffected. If the sawtooth approximation is used, then it is important that exact bounds are computed periodically, rather than using the approximation in every iteration.

Algorithm 3: Belief expansion (`expand`)

```
input  : $M, \{\Gamma_1, \dots, \Gamma_h\}, \{B_1, \dots, B_h\}, r$
$b \leftarrow b_1$
for $t = 1, \dots, h-1$ do
    $a \leftarrow \arg\max_{a \in A} \{ \ r_a \cdot b + \sum_{\{o \in O \ | \ P(o|b,a) > 0\}} P(o|b,a) \cdot \texttt{UB}(b_a^o, B_{t+1}) \ \}$
    $o \leftarrow \arg\max_{\{o \in O \ | \ P(o|b,a) > 0\}} \{ \ \texttt{UB}(b_a^o, B_{t+1}) - \max_{\alpha \in \Gamma_{t+1}} \alpha \cdot b_a^o \ \}$
    $B_{t+1} \leftarrow B_{t+1} \cup \{(b_a^o, \infty)\}$
    $b \leftarrow b_a^o$
end
```

The full description of the search procedure `expand` is shown in Algorithm 3. The algorithm performs a forward search starting from the initial belief, based on the action and observation selection rules that we described in this section. The belief points that are found during the search are added to the belief sets used by FiVI. The belief-bound pairs are added to the set B_{t+1} rather than B_t because the beliefs always correspond to the next time step. It is not required to consider the final time step $t = h$ in the for loop. The search procedure is invoked in each iteration of FiVI in order to add new beliefs, which ensures that the FiVI algorithm iteratively reduces the gap of the solution.

Based on the construction of the procedure `expand`, we can analyze the number of iterations performed by FiVI. The expand procedure finds at most $(|A||O|)^h$ new beliefs, and there are no iterations in which it does not find a new belief before convergence. This means that the total number of iterations of FiVI is $O((|A||O|)^h)$. The same bound applies to the space requirements of the algorithm, because the algorithm stores the beliefs and the corresponding vectors in memory. In practice it can be expected that the number of iterations is much lower than this worst case bound, since the `expand` procedure steers the search in the direction of beliefs reachable under the execution of an optimal policy. However, without making assumptions about the domain the bound cannot be tightened.

5. Backup and Update Heuristics

The point-based algorithm FiVI executes backups to compute new vector sets Γ_t in each iteration. This approach is clean and simple, but it can be relatively inefficient. For example, the algorithm constructs the new value functions from scratch by executing backups on all beliefs. Furthermore, the size of the B_t sets grows during the execution of the algorithm, and therefore an increasing amount of time is required to execute all backups.

The upper bound updates executed by the algorithm (lines 15-28) can also be considered inefficient, because in each iteration the algorithm computes a new upper bound for each belief. The upper bound interpolation function `UB` computes the upper bounds based on all beliefs, while only a few of these beliefs eventually affect the returned upper bound.

In the remainder of this section we address the aforementioned issues by discussing a strategy to enhance the efficiency of backups, and we identify a dependency structure which allows for more efficient upper bound updates.

Algorithm 4: Perseus Belief Selection (PBS)

```
input  : vector set $\Gamma_t$, belief set $B_t$
output: new vector set $\Gamma_t$ after executing backups
$\Gamma \leftarrow \Gamma_t,\ \Gamma_t \leftarrow \emptyset,\ B \leftarrow B_t$
while $B \neq \emptyset$ do
    $(b, \bar{v}) \leftarrow$ randomly selected pair from $B$
    $\alpha \leftarrow \texttt{Backup}(b, t)$
    $\alpha' \leftarrow \arg\max_{\alpha' \in \Gamma} \alpha' \cdot b$
    if $\alpha \cdot b \geq \alpha' \cdot b$ then
        $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha\}$
    else
        $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha'\}$
    end
    $B \leftarrow \{b \in B \mid \max_{\alpha \in \Gamma_t} \alpha \cdot b < \max_{\alpha \in \Gamma} \alpha \cdot b\}$
end
return $\Gamma_t$
```

5.1 Perseus Belief Selection (PBS)

In this section we present a strategy to improve the efficiency of backups, which employs a randomized belief selection method similar to the randomized backup stage found in Perseus (Spaan & Vlassis, 2005). The improve-only principle of this backup stage allows us to perform backups on randomly-selected points only, while ensuring that the newly computed Γ_t set is at least as good as in the previous iteration. A description is shown in Algorithm 4, which replaces lines 10-14 of our algorithm. The algorithm keeps track of a set B containing non-improved beliefs. The key improvement follows from the fact that one backup may improve the value for multiple beliefs. As a result, the set B shrinks quickly and it may not be required to execute backups for all beliefs.

5.2 Dependency-Based Bound Updates (DBBU)

In this section we improve the efficiency of upper bound updates performed during the execution of FiVI. Our preliminary observation is that we do not need to compute new upper bounds for beliefs with zero gap, because for such beliefs the upper bound is already the tightest possible bound. This means that we can mark beliefs with zero gap, and such beliefs will not be taken into account in remaining iterations of FiVI while computing new upper bounds. Unless stated otherwise, the implementations of our algorithms will always ignore beliefs with zero gap when computing new upper bounds.

Our main observation is that Algorithm 2 executes many upper bound interpolations when updating the bounds on lines 15-28. For several beliefs b_a^o there is a call to the function `UB`, which computes an upper bound interpolation based on the corner beliefs and just one additional belief $(b^*, \bar{v}) \in B_{t+1}$ (see line 5 of Algorithm 1). This structure is visually depicted in Figure 3. It shows a belief pair $(b, \bar{v}) \in B_t$ for which an upper bound is updated in the loop starting on line 15 of Algorithm 2. In order to compute the new upper bound,

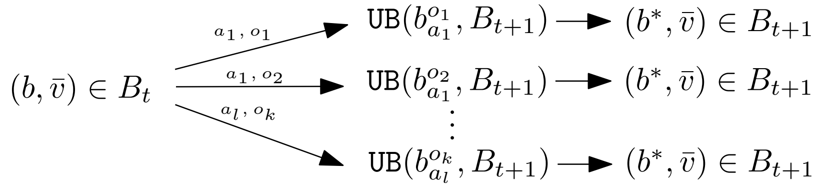
$(b, \bar{v}) \in B_t$
a_1, o_1
a_1, o_2
a_l, o_k
$\texttt{UB}(b_{a_1}^{o_1}, B_{t+1})$
$\texttt{UB}(b_{a_1}^{o_2}, B_{t+1})$
$\texttt{UB}(b_{a_l}^{o_k}, B_{t+1})$
$(b^*, \bar{v}) \in B_{t+1}$
$(b^*, \bar{v}) \in B_{t+1}$
$(b^*, \bar{v}) \in B_{t+1}$

Figure 3: Dependencies between a belief $(b, \bar{v}) \in B_t$ and beliefs $(b^*, \bar{v}) \in B_{t+1}$ used for upper bound interpolation

the algorithm needs several upper bound interpolations for different successor beliefs b_a^o. For clarity we denote these beliefs by $b_{a_1}^{o_1}, b_{a_1}^{o_2}, \ldots, b_{a_l}^{o_k}$ in the figure, indicating that the successor beliefs used for interpolation are different.

Each interpolation computed by the function UB in Algorithm 1 is based on one belief $(b^*, \bar{v}) \in B_{t+1}$. The arrows induce a dependency graph between the beliefs of subsequent steps, and this graph implicity indicates how upper bounds have been propagated from $t = h$ back to $t = 1$. It turns out that the dependency graph remains relatively constant during the execution[2]. In other words: when computing a new upper bound for a belief $(b, \bar{v}) \in B_t$, it is often selecting the same beliefs $(b^*, \bar{v}) \in B_{t+1}$ in the calls to the function UB. We propose to exploit this dependency structure to reduce the number of beliefs considered by UB.

An overview of our Dependency-Based Bound Update (DBBU) method is shown in Algorithm 5. Once in θ iterations of point-based value iteration we keep track of the dependencies between beliefs, as visualized in Figure 3. When updating the upper bound $\bar{v}$ for a pair $(b, \bar{v}) \in B_t$, these dependencies can be determined by looking at the beliefs b^* used in the calls to UB on line 22 of Algorithm 2. We store all beliefs b^* that were used in the set B_b. We periodically determine the dependencies, because in general we cannot assume that the dependency graph remains constant. The reason is that the algorithm iteratively adds new beliefs, and such beliefs may be used for interpolation as well.

In all other iterations we still compute a new bound for each belief pair $(b, \bar{v}) \in B_t$, but we replace the calls to $\texttt{UB}(b_a^o, B_{t+1})$. Rather than computing the interpolation based on all beliefs in B_{t+1}, we use a subset $B_{t+1}^* \subseteq B_{t+1}$, where B_{t+1}^* contains all beliefs b^* defined by the dependency graph (e.g., when updating for $(b, \bar{v}) \in B_t$ this would be the set B_b). Typically this set is much smaller than B_{t+1}, and if the dependency graph is constant then it also contains the beliefs that would be used by an interpolation based on B_{t+1}. As a result, the function UB iterates over much fewer beliefs.

Beliefs that were found after the last construction of the dependency graph are always included in B_{t+1}^*. For this purpose the algorithm defines the auxiliary variable B'_{t+1} when constructing the graph, which contains all beliefs that were part of B_{t+1} when constructing the graph. This variable is used in the definition of B_{t+1}^* on line 9, such that it includes new beliefs that are part of B_{t+1} which were not part of B'_{t+1} yet.

[2] A similar observation has been made for the calls to the exact upper bound interpolation in GapMin (Poupart et al., 2011), in which the convex combination remains fairly constant. The presented alternative uses a so-called augmented POMDP, but it is important to note that it still requires $|S||A||O|+1$ calls to UB, and it requires the fast-informed bound, which does not directly apply to finite-horizon settings. GapMin does not focus on the dependency structure of the upper bound update.

```
Algorithm 5: Dependency-Based Bound Updates (DBBU)
input  : belief sets $B_t$ and $B_{t+1}$, iteration $\delta$, interval $\theta$
if $\delta$ mod $\theta = 0$ then
    for $(b, \bar{v}) \in B_t$ do
        Lines 16-27 from Algorithm 2
        $B_b \leftarrow$ set containing beliefs $b^*$ used in UB calls
    end
    $B'_{t+1} \leftarrow B_{t+1}$
else
    for $(b, \bar{v}) \in B_t$ do
        $B^*_{t+1} \leftarrow \{(b, \bar{v}) \in B_{t+1} \mid b \in B_b \vee ((b, \bar{v}) \in B_{t+1} \wedge (b, \bar{v}) \notin B'_{t+1})\}$
        Lines 16-27 from Algorithm 2, where UB uses $B^*_{t+1}$ rather than $B_{t+1}$
    end
end
```

It is important to note that DBBU can be easily combined with PBS, because DBBU affects the upper bound updates of FiVI, while PBS only changes the procedure to execute backups on beliefs. The influence of the interval parameter θ on the performance of DBBU will be studied in the next section.

6. Experiments

In this section we present our experimental evaluation. We start with a comparison of multiple variants of FiVI, in which we test the influence of our strategies PBS and DBBU on runtime, convergence and solution quality. After that, we provide a more in-depth study of the behavior of PBS and DBBU, and we provide a comparison with 3 alternative approaches which may be used for finite-horizon problems.

6.1 Performance of FiVI with PBS and DBBU

In the first set of experiments we compare standard FiVI, FiVI augmented with PBS, and FiVI augmented with PBS and DBBU. For these variants of the algorithm we use the names VI, PBS and DBBU, respectively. We let the algorithms run for at most 15 minutes, after which execution is terminated. Furthermore, we stop algorithm execution if the gap between the lower bound and upper bound drops below 0.01. Since FiVI is an anytime algorithm, we can assess which variant of the algorithm provides the best solution given the fixed amount of computation time that is available.

We test our algorithms with multiple planning horizons h, which means that we discard the default discount factors defined by the domains. We use multiple domains from `pomdp.org`, which we solve with horizons $h = 5, 10, 15, 20$. The domains have been chosen such that the algorithm is able to reduce the gap to a value close to 0 within the time limit of 900 seconds. This is important for testing whether the dependency graph during algorithm execution becomes constant, and it enables us to test the effects of our heuristics until convergence of the algorithm. An overview of the domain properties is provided in

	4x5x2	AircraftID	Hallway	Network
$\|S\|$	39	12	60	7
$\|A\|$	4	6	5	4
$\|O\|$	4	5	21	2

Table 2: Properties of the domains involved in the experiments

Table 2. For DBBU we consider the parameters $\theta = 10, 20, 30, 40$, which we append to the names of the algorithms. We compare the algorithms by measuring the total runtime, the lower bound on the expected reward of the computed policy, as well as the gap associated with the computed policy. Each algorithm is executed 10 times, such that we can report the mean and standard deviation for these measures. Prior to running the algorithms, we intuitively expect that PBS improves the performance of VI since it is likely that it executes fewer backups. Furthermore, we expect that DBBU improves the performance even more, because in that case it iterates over fewer beliefs when computing upper bounds.

The results of our experiment are shown in Tables 3 and 4, in which each entry represents the mean based on 10 runs of the algorithm, and the small entries denote the standard deviation. Based on the runs in which the time out of 900 seconds was not reached, we can conclude that PBS consistently improves the performance of plain FiVI, meaning that it needs less time to reach a solution with a gap below 0.01. The variants of the algorithm which include DBBU become even faster, and they typically need even less time. This is especially noticable when increasing the horizon to, e.g., $h = 15$ and $h = 20$. In these cases the running time of DBBU becomes significantly lower than the running time of PBS, which confirms our initial expectations. The variants of the algorithm with PBS and DBBU include randomization, but the low standard deviations of the lower bounds and gaps indicate that it has very limited influence on the quality of the solution returned.

As can be seen in the table, the choice of the interval θ influences the performance of DBBU, but the results do not allow us to identify a generic choice for this parameter which provides the best performance throughout all domains. It should be noted, however, that DBBU becomes faster than FiVI with only PBS, regardless of the choice of θ. As a general rule we can say that setting θ too high (e.g., much higher than 40) is unlikely to give good performance because then potential changes in upper bounds are not taken into account quickly during the execution of the algorithm. In the Hallway domain with horizon $h = 5$ we can also see that our heuristics improve the performance of plain FiVI. For the horizons $h = 10, 15, 20$ we observe that the lower bounds and gaps of PBS and DBBU are slightly better when the algorithm reaches the timeout. However, it should be noted that the domain is difficult to solve, which means that it takes a long time to reach a solution with a gap that is lower than the tolerance. This makes it hard to derive conclusions regarding algorithm performance based on those results.

In Figure 4 we visualize how the gap decreases over time during one execution of the algorithm. From these graphs we can derive two conclusions about the performance of our two strategies PBS and DBBU. First, in the variants of FiVI which include either PBS or DBBU the gap tends to decrease faster, meaning that it approaches an optimal solution faster. Second, our bound update strategy DBBU almost always improves the performance of FiVI with PBS. The tables and graphs together confirm our initial expectation that

		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
4x5x2	Time (s)	0.915 0.049	0.307 0.011	0.202 0.025	0.197 0.006	0.196 0.003	0.203 0.007
$h = 5$	LB	0.429 0.000	0.429 0.000	0.429 0.000	0.429 0.000	0.429 0.000	0.429 0.000
	Gap	0.003 0.000	0.003 0.000	0.003 0.000	0.003 0.000	0.003 0.000	0.003 0.000
4x5x2	Time (s)	13.541 0.253	5.234 0.171	2.741 0.152	2.808 0.146	2.818 0.066	2.905 0.065
$h = 10$	LB	1.119 0.000	1.119 0.000	1.119 0.000	1.119 0.000	1.119 0.000	1.119 0.000
	Gap	0.010 0.000	0.010 0.001	0.010 0.000	0.010 0.000	0.010 0.000	0.009 0.000
4x5x2	Time (s)	112.401 0.495	39.147 1.021	17.756 0.576	17.64 0.452	17.726 0.401	18.746 0.679
$h = 15$	LB	1.619 0.000	1.619 0.000	1.619 0.000	1.619 0.000	1.619 0.000	1.619 0.000
	Gap	0.007 0.000	0.009 0.001	0.009 0.001	0.009 0.001	0.009 0.001	0.009 0.001
4x5x2	Time (s)	346.417 2.245	141.535 7.571	70.575 3.046	68.119 3.730	67.922 4.138	70.798 3.732
$h = 20$	LB	2.256 0.000	2.256 0.000	2.256 0.000	2.256 0.000	2.256 0.000	2.256 0.000
	Gap	0.009 0.000	0.009 0.001	0.009 0.000	0.009 0.01	0.009 0.001	0.009 0.001
		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
AircraftID	Time (s)	0.031 0.029	0.014 0.004	0.019 0.014	0.013 0.001	0.013 0.002	0.013 0.002
$h = 5$	LB	-45.393 0.000	-45.393 0.000	-45.393 0.000	-45.393 0.000	-45.393 0.000	-45.393 0.000
	Gap	0.002 0.000	0.002 0.001	0.004 0.001	0.003 0.001	0.004 0.001	0.003 0.001
AircraftID	Time (s)	1.068 0.059	0.450 0.047	0.456 0.060	0.432 0.024	0.460 0.052	0.450 0.034
$h = 10$	LB	-95.240 0.000	-95.241 0.000	-95.240 0.000	-95.240 0.000	-95.240 0.000	-95.241 0.000
	Gap	0.010 0.000	0.010 0.001	0.009 0.000	0.009 0.001	0.009 0.000	0.009 0.001
AircraftID	Time (s)	18.837 0.168	8.582 1.033	4.958 0.413	5.141 0.236	5.436 0.451	5.877 0.623
$h = 15$	LB	-149.467 0.000	-149.467 0.000	-149.467 0.000	-149.467 0.000	-149.467 0.000	-149.467 0.000
	Gap	0.010 0.000	0.010 0.000	0.010 0.001	0.010 0.000	0.010 0.000	0.010 0.000
AircraftID	Time (s)	163.189 2.356	78.369 4.647	29.956 1.658	29.895 2.247	31.219 1.746	33.127 2.192
$h = 20$	LB	-208.013 0.000	-208.013 0.000	-208.013 0.000	-208.013 0.000	-208.013 0.000	-208.013 0.000
	Gap	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000

Table 3: Algorithm comparison for domains 4x5x2 and AircraftID

PBS improves the performance of plain FiVI, and our expectation that DBBU improves the performance even more. Furthermore, our experiment shows that FiVI is an effective method to compute finite-horizon solutions while providing guarantees on the quality of the resulting solution.

		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
Hallway	Time (s)	9.843 0.466	5.045 0.219	6.358 0.276	6.370 0.213	6.523 0.286	6.498 0.296
$h = 5$	LB	0.098 0.000	0.098 0.001	0.098 0.001	0.098 0.000	0.098 0.000	0.098 0.000
	Gap	0.009 0.000	0.009 0.000	0.009 0.00	0.009 0.000	0.009 0.000	0.009 0.000
Hallway	Time (s)	909.567 4.671	903.979 3.390	904.373 3.380	904.200 2.492	903.358 2.502	902.969 2.185
$h = 10$	LB	0.327 0.000	0.334 0.000	0.335 0.000	0.335 0.000	0.334 0.000	0.334 0.000
	Gap	0.104 0.000	0.087 0.003	0.083 0.002	0.082 0.002	0.083 0.002	0.083 0.002
Hallway	Time (s)	911.973 5.962	908.061 3.659	904.740 3.136	904.405 2.191	904.415 2.638	905.049 2.830
$h = 15$	LB	0.628 0.001	0.632 0.001	0.635 0.002	0.635 0.002	0.634 0.001	0.635 0.001
	Gap	0.272 0.001	0.260 0.003	0.255 0.002	0.256 0.003	0.257 0.002	0.256 0.003
Hallway	Time (s)	916.264 5.809	906.485 4.881	905.811 2.717	905.738 2.744	908.350 5.132	906.481 3.907
$h = 20$	LB	0.902 0.000	0.918 0.003	0.921 0.002	0.920 0.003	0.920 0.003	0.919 0.002
	Gap	0.430 0.000	0.403 0.004	0.398 0.003	0.399 0.004	0.399 0.004	0.400 0.003
		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
Network	Time (s)	0.014 0.012	0.004 0.001	0.006 0.002	0.004 0.002	0.003 0.001	0.003 0.001
$h = 5$	LB	81.137 0.000	81.137 0.000	81.137 0.000	81.137 0.000	81.137 0.000	81.137 0.000
	Gap	0.000 0.000	0.000 0.000	0.000 0.000	0.000 0.000	0.000 0.000	0.000 0.000
Network	Time (s)	0.341 0.066	0.149 0.006	0.095 0.004	0.097 0.006	0.107 0.004	0.110 0.004
$h = 10$	LB	151.18 0.000	151.18 0.000	151.18 0.000	151.18 0.000	151.18 0.000	151.18 0.000
	Gap	0.010 0.000	0.009 0.001	0.009 0.001	0.009 0.001	0.008 0.001	0.009 0.001
Network	Time (s)	32.000 0.194	22.141 1.790	5.612 0.275	5.041 0.381	5.375 0.362	5.863 0.337
$h = 15$	LB	224.616 0.000	224.616 0.000	224.616 0.000	224.616 0.000	224.616 0.000	224.616 0.000
	Gap	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000
Network	Time (s)	901.721 0.682	901.124 0.544	267.056 62.270	204.782 66.242	114.482 12.793	142.049 23.324
$h = 20$	LB	298.149 0.000	298.149 0.000	298.149 0.000	298.149 0.000	298.149 0.000	298.149 0.000
	Gap	0.018 0.000	0.014 0.001	0.010 0.000	0.010 0.000	0.010 0.000	0.010 0.000

Table 4: Algorithm comparison for domains Hallway and Network

6.2 Number of Backups Executed by PBS

In our second experiment we study the hypothesis that PBS executes fewer backups due to the potential to skip beliefs for which the value function has improved. We measure the reduction of the number of backups due to PBS as follows. We let #beliefs_total denote the total number of beliefs that has been added so far, counted across all time steps involved. Furthermore, we let #num_backups denote the total number of backups executed by PBS.

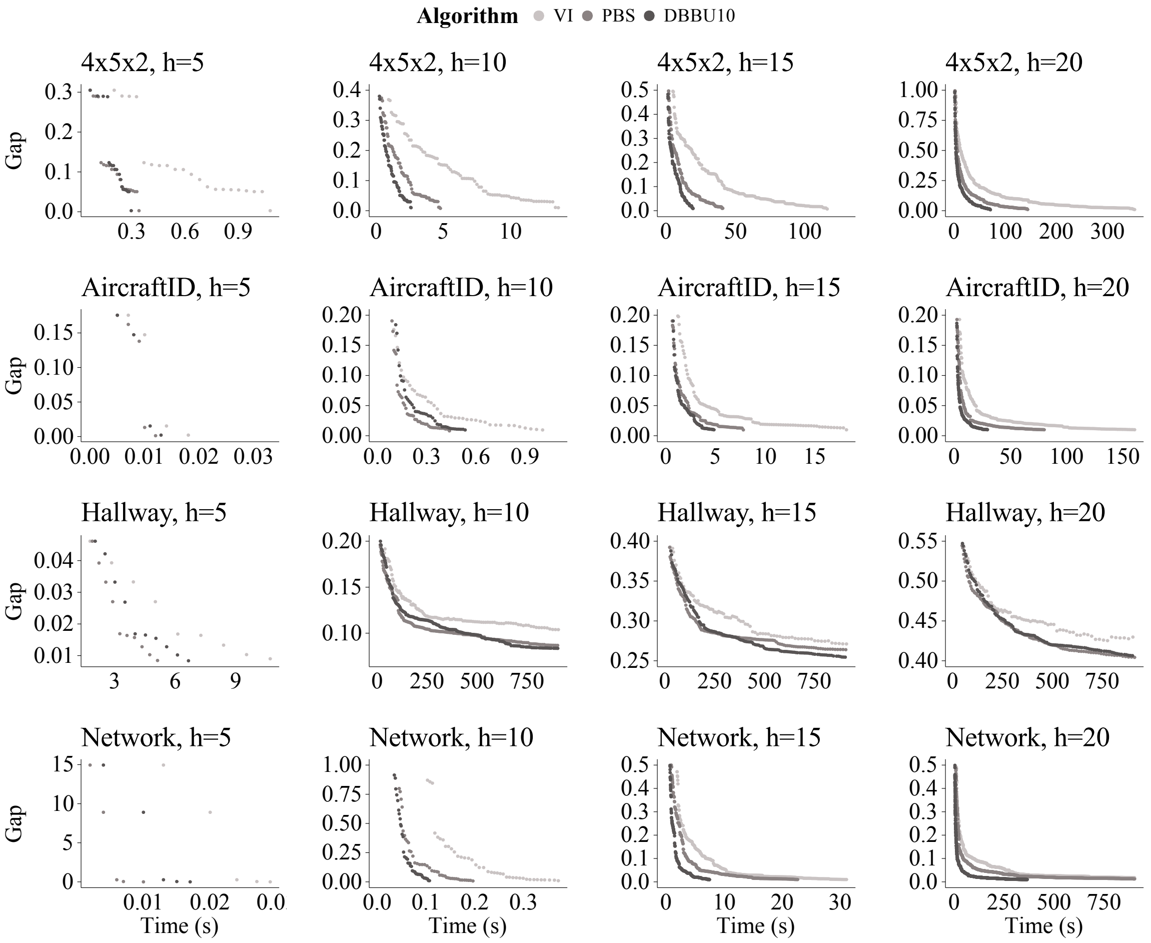

Figure 4: Gap during the execution of FiVI

Now the reduction of the number of backups can be expressed as follows:

$$-100 \times \left(\frac{\#\text{num\_backups} - \#\text{beliefs\_total}}{\#\text{beliefs\_total}} \right). \tag{40}$$

In Figure 5 we use this metric to visualize the reduction of the number of backups for each iteration of FiVI. These results confirm that PBS executes much fewer backups than plain FiVI, which also explains why PBS can run faster than plain FiVI.

Although PBS accelerates the iterations of FiVI, it is important to note that FiVI with PBS may need more iterations to reach a solution of the same quality. The explanation for this is that executing backups on all beliefs may result in a better value function than the value function obtained when executing backups on only a few beliefs. As we have concluded from Figure 5, an iteration with PBS generally runs faster, but as a consequence of the behavior of PBS the total number of iterations may increase. This can be seen in Table 5, which shows the number of iterations of FiVI performed until termination. However, based on the initial results in Table 3 and Table 4 we can conclude that the

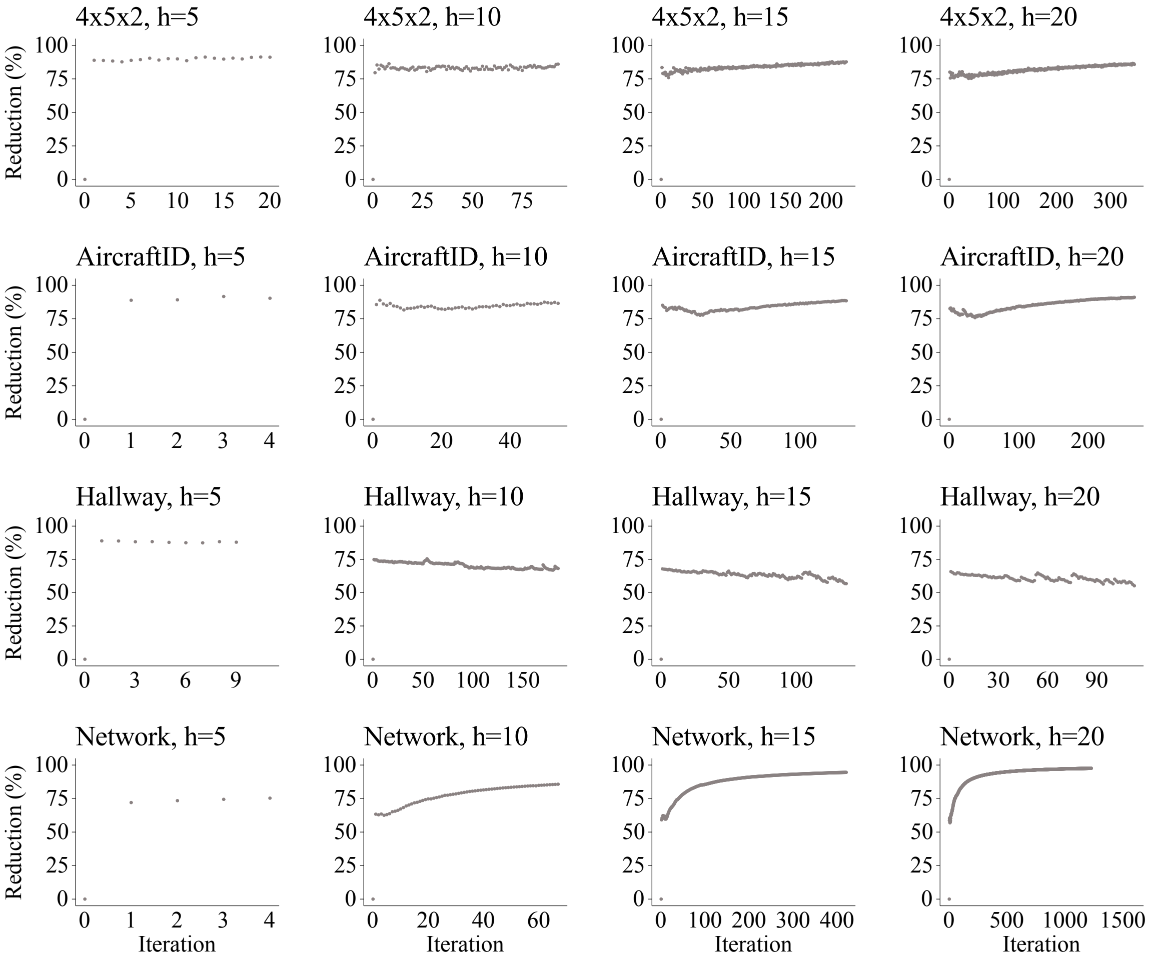

Figure 5: Reduction of the number of backups

gain introduced by PBS dominates the additional runtime introduced by running a few additional iterations.

6.3 Dependency Graph Construction in DBBU

Our bound update strategy DBBU is based on the intuition that the dependency structure of the beliefs does not change much during the execution of value iteration. In order to test whether this is indeed the case in our domains, we execute an experiment in which we keep track of the number of times that all the dependencies remain constant when constructing a new dependency graph. To be more specific, we let DBBU compute new dependency graphs in each iteration and we measure the number of graph constructions in which the dependencies remain the same compared to the previous iteration. We let #graph denote the total number of graph constructions in an iteration and #graph_unchanged denotes the number of times that the dependency graph associated with a belief does not change. The

Domain	h	Iterations VI	Iterations PBS
4x5x2	5	21	21
	10	92	94
	15	225	225
	20	332	345
AircraftID	5	4	5
	10	50	55
	15	138	134
	20	267	267
Hallway	5	9	10
	10	124	186
	15	96	138
	20	81	114
Network	5	5	5
	10	70	68
	15	381	421
	20	997	1229

Table 5: Number of iterations of FiVI until termination

total number of unchanged dependency graphs within an iteration now corresponds to:

$$100 \times \left(\frac{\#\text{graph\_unchanged}}{\#\text{graph}} \right). \tag{41}$$

Figure 6 visualizes this percentage as a function of the iterations of FiVI, which confirms that the dependency graphs remain almost always constant during the execution of FiVI. This result implies that the upper bounds returned by `UB` based on B^*_{t+1} are almost always the same as the upper bounds returned by `UB` based on B_{t+1} (see Algorithm 5), while iterating over only a small subset of beliefs during the computation of upper bound interpolations in `UB`. From this experiment we can conclude that the upper bound updates in FiVI take less time due to DBBU, and in many cases our strategy does not affect the computed upper bounds.

6.4 Comparison with Alternative Methods

In our final experiment we present a comparison with three alternative methods which may be used to compute solutions to finite-horizon problems. These methods do not provide performance guarantees and they have several limitations. From a practical point of view it can be suitable to use them (e.g., if strict performance guarantees are not required). Therefore, we want to show how such methods perform when solving the instances used in our previous experiments. We want to emphasize that the methods have not been designed for finite-horizon problems, and they have not been presented as such in the literature.

The first method we consider starts with sampling reachable beliefs in the belief space by exploring the environment randomly during a fixed number of episodes. After that, it performs only one iteration of FiVI to compute value functions and upper bounds. It never

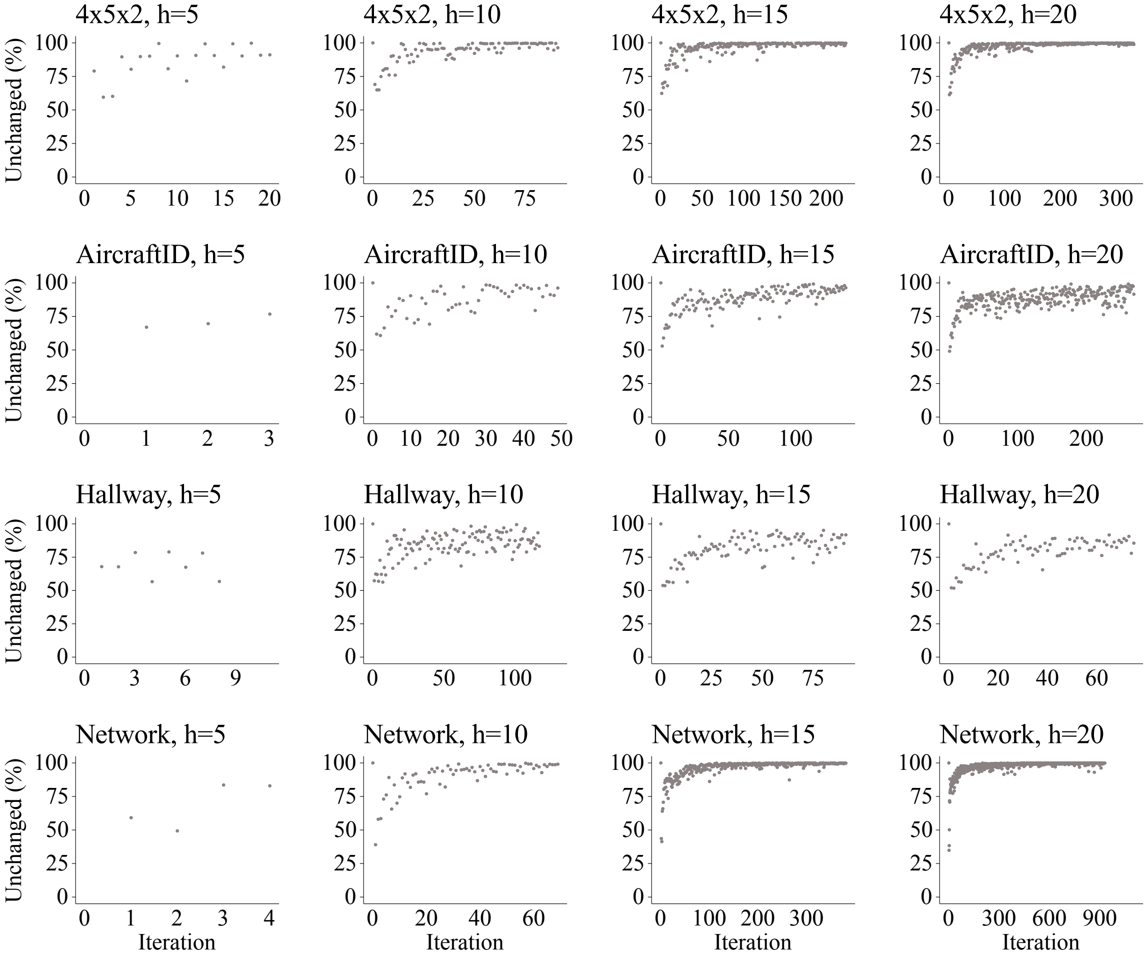

Figure 6: Unchanged dependency graphs as a function of the number of iterations

executes heuristic search to find additional beliefs. This method is simple to execute, but it does not provide any performance guarantees because exploring the environment randomly does not necessarily provide the belief points that are needed to compute a potentially optimal value function. We let the algorithm sample beliefs during 1000 episodes, and the algorithm is denoted by S1000.

The second method we consider is a finite-horizon version of RTDP-Bel (Bonet & Geffner, 2009). The only difference with the infinite-horizon version is that we use separate value functions for each time step. RTDP-Bel discretizes beliefs in order to represent value functions in memory, and due to this discretization it does not necessarily converge to optimality. During our experiments we use discretization parameter $D = 10$ because this parameter setting gives us good results, and we execute the algorithm for 900 seconds, similar to the previous experiments. For more details about RTDP-Bel we refer to our discussion in Section 3.2.

The third method we consider is the infinite-horizon algorithm GapMin which computes an infinite-horizon policy with $\gamma = 0.99$, which we subsequently evaluate during simulation

		VI	PBS	DBBU	S1000	RTDP-Bel	GapMin
4x5x2	LB	2.256	2.256	2.256	2.256	2.256	2.118
$h = 20$	Gap	0.009	0.009	0.009	0.237	N/A	N/A
AircraftID	LB	-208.013	-208.013	-208.013	-208.013	-208.765	-208.708
$h = 20$	Gap	0.010	0.010	0.010	1.096	N/A	N/A
Hallway	LB	0.902	0.918	0.921	0.972	0.078	0.974
$h = 20$	Gap	0.430	0.403	0.398	0.293	N/A	N/A
Network	LB	298.149	298.149	298.149	298.128	292.934	291.305
$h = 20$	Gap	0.018	0.014	0.010	13.679	N/A	N/A

Table 6: Comparison with random belief sampling, RTDP and infinite-horizon GapMin

runs. In Section 5 we already concluded that this approach is not suitable for finite-horizon problems, and in this section we validate empirically whether this is indeed the case. Similar to RTDP-Bel and the previous experiments, we use a time limit of 900 seconds.

We present the results of our experiment in Table 6, which shows the lower bound and the associated gap. The results for VI, PBS and DBBU have been copied from the previous experiments. For DBBU we selected the variant with the highest lower bound. For RTDP-Bel and GapMin we do not obtain a lower bound, and therefore the reported number is the mean reward measured during 1 million simulation runs. Below we discuss our most important observations and conclusions for each alternative method.

The mean reward of the solutions computed by RTDP-Bel is close to the lower bounds computed by FiVI, which shows that RTDP-Bel can be an effective and simple method to compute finite-horizon solutions. However, in general the algorithm does not keep track of upper bounds, which means that it is impossible to assess whether a solution computed by RTDL-Bel is close to optimal or not.

The mean reward of the solutions computed by GapMin is slightly lower than the lower bounds computed by FiVI in the domains 4x5x2, AircraftID and Network. This confirms that a policy computed for an infinite-horizon problem does not always perform well if the actual problem has a finite time horizon. In the Hallway domain the infinite-horizon policy performs very well. This seems surprising, but there is a very intuitive explanation which shows why this is the case. Upon reaching the goal the agent restarts from the initial belief, and an infinite-horizon policy tries to reach the goal as many times as possible in order to maximize its expected reward. In practice this means that the infinite-horizon policy tries to reach the goal as fast as possible, which is also the best strategy in case there is only a finite number of steps available. Similar to RTDP-Bel, it is important to note that the algorithm does not compute an upper bound for the finite-horizon case, which prevents assessment of policy quality in general.

For the belief sampling approach S1000 we observe that the lower bounds are very close or identical to the lower bounds found by the variants of FiVI in the domains 4x5x2, AircraftID and Network. However, the associated gap is significantly larger than the gaps returned by FiVI, which means that S1000 did not find beliefs that are needed to reduce the gap effectively. In order to provide a better comparison we execute another experiment in which we give VI, PBS and DBBU a time limit that equals the total runtime of S1000. This

		S1000	VI	PBS	DBBU10
4x5x2	Time (s)	26.661	26.661	26.661	26.661
$h = 20$	LB	2.256	2.249	2.256	2.256
	Gap	0.237	0.358	0.181	0.114
AircraftID	Time (s)	25.520	25.520	25.520	25.520
$h = 20$	LB	-208.013	-208.013	-208.013	-208.013
	Gap	1.096	0.035	0.020	0.011
Hallway	Time (s)	830.610	830.610	830.610	830.610
$h = 20$	LB	0.972	0.903	0.916	0.912
	Gap	0.293	0.429	0.407	0.408
Network	Time (s)	5.412	5.412	5.412	5.412
$h = 20$	LB	298.128	298.148	298.149	298.149
	Gap	13.679	0.702	0.596	0.247

Table 7: Solution quality for VI, PBS and DBBU10 with time limit equal to S1000 runtime

allows us to investigate whether the FiVI variants compute a better solution than S1000 within the same amount of time. The results of this comparison are shown in Table 7. In previous experiments we found that the standard deviation of the lower bound and gap is negligible, and therefore we report the results for 1 run. In the Hallway domain S1000 provides a better solution than the FiVI variants without belief sampling, which means that it has found reachable beliefs leading to a better solution, which were not found yet by the FiVI algorithm within its time limit. In the domains 4x5x2, AircraftID and Network our algorithm computes solutions with a smaller gap than the solution returned by S1000, which means that our algorithm performs better than the belief sampling approach. For S1000 in general we can conclude that it may work well in some domains, but it samples an excessively large number of reachable beliefs, which becomes intractable if the POMDP model is large.

To conclude, our experiment has shown that the belief sampling approach, RTDP-Bel and GapMin may be suitable for planning in finite-horizon settings. However, the algorithms do not provide performance guarantees and sampling a large number of beliefs can be computationally expensive (both the sampling itself and executing backups on those beliefs). FiVI, on the other hand, provides performance guarantees and typically it uses much fewer beliefs to compute the solution. Finally, the experiment confirms our observation that the algorithms are not suitable for solving finite-horizon problems in general, as we discussed in Section 3.

7. Conclusions

Finite-horizon POMDPs naturally arise in application domains in which policies need to be executed during a finite number of time steps. For example, in condition-based maintenance it can be important to optimize maintenance during the finite lifespan of a machine while the actual condition of the machine is not fully observable (Besnard & Bertling, 2010; Byon & Ding, 2010). Unfortunately, computing finite-horizon POMDP solutions turns out to be more complicated than intuitively expected. In the literature several approximate

POMDP algorithms have been presented which form the current state of the art, such as the class of point-based value iteration algorithms. These algorithms have been designed based on the assumption that an infinite horizon with discounting is considered, but due to this assumption the algorithms do not easily generalize to finite-horizon settings without discounting of reward. In other words, state-of-the-art approximate POMDP algorithms cannot be used to solve finite-horizon problems efficiently.

In this paper we presented FiVI, which is a generic point-based value iteration algorithm for finite-horizon problems. FiVI unifies several insights from the existing point-based algorithms SARSOP (Kurniawati et al., 2008), HSVI (Smith & Simmons, 2005) and GapMin (Poupart et al., 2011). FiVI is a point-based value iteration algorithm which computes time-dependent value functions, and it leverages a heuristic search procedure to find new belief points incrementally. It is an anytime algorithm which converges to an optimal finite-horizon POMDP solution. In addition to the algorithm itself we also presented two strategies which further improve the algorithm performance. First, we observed that we can employ randomized backup stages similar to the infinite-horizon algorithm Perseus (Spaan & Vlassis, 2005). Second, we made the updates of value upper bounds more efficient by exploiting the insight that these updates typically depend on only a few beliefs that remain constant during execution. Both strategies are complementary, in the sense that they focus on two different steps in FiVI that are computationally difficult, and therefore the strategies can be used simultaneously. In a series of experiments we tested the performance and characteristics of FiVI, which shows that the algorithm is an effective method for solving finite-horizon problems without discounting of reward.

Multiple directions of future work can be pursued to improve our algorithm. The main computational bottleneck of FiVI arises if the POMDP models have a large number of states, and in case there is a large number of beliefs required to reach a solution within the solution quality tolerance. The first problem can be addressed by implementing the algorithm based on sparse representations for alpha vectors. The second problem is more difficult since the convergence of the algorithm is dependent on the ability to sample belief points that are required to reach an optimal solution. It may be possible to derive a more efficient belief sampling scheme which replaces Algorithm 3. For example, domain-specific knowledge may be used to bias the heuristic search, or multiple promising outcomes can be selected during the forward search, similar to work by Zhang, Hsu, Lee, Lim, and Bai (2015). In infinite-horizon algorithms such as GapMin this is addressed by implementing a breadth-first search which prioritizes beliefs that are encountered in an early stage of policy execution. In the finite-horizon setting without discounting it can be expected that this is less effective because all rewards collected during policy execution are weighted equally. This leaves a gap for additional research. Another interesting avenue of future research is the application to POMDPs with a factored problem representation, which may provide additional computational advantages.

Acknowledgments

The research in this paper is funded by the Netherlands Organisation for Scientific Research (NWO), as part of the Uncertainty Reduction in Smart Energy Systems (URSES) program.

References

Besnard, F., & Bertling, L. (2010). An approach for condition-based maintenance optimization applied to wind turbine blades. *IEEE Transactions on Sustainable Energy*, *1*(2), 77–83.

Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., & Mihailidis, A. (2005). A Decision-Theoretic Approach to Task Assistance for Persons with Dementia. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1293–1299.

Bonet, B., & Geffner, H. (2009). Solving POMDPs: RTDP-Bel vs. Point-based Algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1641–1646.

Byon, E., & Ding, Y. (2010). Season-dependent condition-based maintenance for a wind turbine using a partially observed markov decision process. *IEEE Transactions on Power Systems*, *25*(4), 1823–1834.

Cassandra, A. R., Littman, M. L., & Zhang, N. L. (1997). Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Crites, R. H. (1996). *Large-scale dynamic optimization using teams of reinforcement learning agents*. Ph.D. thesis, University of Massachusetts Amherst.

Dujardin, Y., Dietterich, T., & Chadès, I. (2015). α-min: A Compact Approximate Solver For Finite-Horizon POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2582–2588.

Dujardin, Y., Dietterich, T., & Chadès, I. (2017). Three New Algorithms to Solve N-POMDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4495–4501.

Hansen, E. A. (2007). Indefinite-Horizon POMDPs with Action-Based Termination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 291–296.

Hauskrecht, M. (2000). Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, *13*, 33–94.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT press.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99–134.

Kurniawati, H., Hsu, D., & Lee, W. S. (2008). SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Proceedings of Robotics: Science and Systems IV*.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science*, *28*(1).

Morales-España, G., Latorre, J. M., & Ramos, A. (2013). Tight and Compact MILP Formulation for the Thermal Unit Commitment Problem. *IEEE Transactions on Power Systems*, *28*(4), 4897–4908.

Oliehoek, F. A., & Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1025–1030.

Poupart, P., Kim, K., & Kim, D. (2011). Closing the Gap: Improved Bounds on Optimal POMDP Solutions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 194–201.

Qi, W., Xu, Z., Shen, Z. M., Hu, Z., & Song, Y. (2014). Hierarchical coordinated control of plug-in electric vehicles charging in multifamily dwellings. *IEEE Transactions on Smart Grid*, *5*(3), 1465–1474.

Ross, S., Pineau, J., & Chaib-Draa, B. (2008). Theoretical Analysis of Heuristic Search Methods for Online POMDPs. In *Advances in Neural Information Processing Systems*, Vol. 20, pp. 1216–1225.

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based POMDP solvers. *Journal of Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51.

Smith, T. (2007). *Probabilistic Planning for Robotic Exploration*. Ph.D. thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Smith, T., & Simmons, R. (2005). Point-Based POMDP Algorithms: Improved Analysis and Implementation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 542–549.

Smith, T., & Simmons, R. (2006). Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1227–1232.

Sondik, E. J. (1971). *The optimal control of partially observable Markov processes*. Ph.D. thesis, Stanford University.

Spaan, M. T. J. (2012). Partially Observable Markov Decision Processes. In Wiering, M., & van Otterlo, M. (Eds.), *Reinforcement Learning – State-of-the-Art*, pp. 387–414. Springer.

Spaan, M. T. J., & Vlassis, N. (2005). Perseus: Randomized Point-based Value Iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

Walraven, E., & Spaan, M. T. J. (2017). Accelerated Vector Pruning for Optimal POMDP Solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3672–3678.

Walraven, E., & Spaan, M. T. J. (2018). Column Generation Algorithms for Constrained POMDPs. *Journal of Artificial Intelligence Research*, *62*, 489–533.

Williams, J. D., & Young, S. (2007). Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech & Language*, *21*(2), 393–422.

Zhang, Z., Hsu, D., Lee, W., Lim, Z., & Bai, A. (2015). PLEASE: Palm Leaf Search for POMDPs with Large Observation Spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 249–257.

Zhang, Z., Hsu, D., & Lee, W. S. (2014). Covering Number for Efficient Heuristic-Based POMDP Planning. In *Proceedings of the International Conference on Machine Learning*, pp. 28–36.