The Value of Measuring in Q-learning for Markov Decision Processes

Merlijn Krale, Thiago D. Simão, and Nils Jansen

Radboud University, Nijmegen
Institute for Computing and Information Sciences
{merlijn.krale, thiago.simao, nils.jansen}@ru.nl

Abstract. We study Markov decision processes (MDPs) that exhibit explicit measuring actions with associated observation costs. In particular, we aim to balance these observation costs with the value of the information gained by measuring. For this goal, we firstly define *Fully Measurable MDPs* as an extension of MDPs to represent our problem. Then, we develop a Q-learning based agent that learns to optimise the total *scalarized reward*, defined as the standard reward minus the observation cost. The agent uses an optimistic exploration strategy based on a Q-table biased towards unexplored states-action pairs. The decision of whether to take a measure is based on two newly introduced metrics. These metrics approximate (1) the value of improving the accuracy of estimating the underlying MDP, and (2) the value of knowing the current state according to this estimated MDP. The empirical analysis shows that our method is able to dynamically switch between information gathering and exploitation in different environments without requiring environment-specific hyper-parameter tuning, while outperforming prior algorithms in the tested environments.

1 Introduction

In recent years, Partially Observable Markov Decision Processes (POMDPs) have become widespread for modelling and solving real-world problems[8, 16, 17]. Although POMDPs can accurately describe the noisiness of real-life observations, they do not generally take into account costs or constraints that might be associated with making observations in the first place. For example, railway companies have to make decisions about where and how often to inspect certain sections of rails, and hospitals with limited diagnostic resources might have to consider which patients can make use of them, and for who less accurate diagnostic methods suffice. In both cases, being able to make these decisions requires a model, but getting such a model also requires some initial observations. To model these situations more accurately, prior work proposed a framework in which both these kinds of observations have associated costs [3, 19]. However, it is still an open question how to take in account the uncertainty of the current state when deciding whether or not to measure.

In short, this paper has the following aim:

> Develop reinforcement learning algorithms that (1) can actively learn the dynamics of the underlying environment, while taking into account the costs for taking measurements, and that (2) eventually find policies that are able to trade off the cost of taking measures and the value of the information gained.

To achieve this goal, we propose to track the belief state and use it while computing the value-function and deciding whether or not to measure. Furthermore, we introduce two metrics for the value of measuring, used while exploring the environment and while exploiting it. We compare the performance of our new agent with an AMRL agent (AMRL-Q) [3], both in standard RL-environments and in a minimum environment that showcases the value of measuring.

Section 2 discusses related previous research. Section 3 formalizes the problem and reviews prior work. Section 4 introduces *Belief-based Active Measure QMDP* (BAM-QMDP) as a new agent for these environments. Sections 5 and 6 describe both the setup and results of our empirical analysis. Lastly, Section 7 evaluates the performance of BAM-QMDP and Section 8 summarizes our findings.

2 Related work

To our knowledge, there exist only two algorithms for solving active-measure problems. The first is the ARML-Q algorithm introduced by Bellinger et al. [3], which is the main inspiration for the current work. Section 3.2 presents a comparison between AMRL-Q and BAM-QMDP and Section 6 compares them.

The second is the ACNO-MDP framework introduced by Nam, Fleming, and Brunskill [19]. Their general strategy involves transforming the problems to POMDPs and solving these using existing RL algorithms. They distinguish two variants: one in which these two tasks are done simultaneously, and one where they are split. The latter performs slightly better given enough exploration time, but the former most closely resembles our approach.

Another closely related work is that of Doshi-Velez, Pineau, and Roy [10]. They introduce a framework in which agents explore a POMDP, but have the additional option to make 'action queries' to a oracle. The general method they use is comparable to ours, their concept of 'Bayesian Risk' resembles the concept of 'Measurement Regret' introduced in this paper. However, since their method relies on a different way of measuring, results cannot easily be compared.

We also note some other related papers, which explore active measure learning in different contexts. Yin et al. [23] propose a method for AMRL which relies on a pre-trained neural network to infer missing information with the measured data. Ghasemi and Topcu [14] propose a method to choose near-optimal measurements on a limited budget per step, which can be used to improve pre-computed 'standard' POMDP policies. Bernardino et al. [4] try solve the task of diagnosing patients using an MDP-approach, in which the action themselves correspond to taking measurements. Lastly, Araya-López et al. [1] study how to approximate an MDP without any reward function.

3 Background

3.1 Defining our framework: ACNO-MDPs and AMRL

Following work from Nam, Fleming, and Brunskill [19], and in agreement with definitions from Bellinger et al. [3], we define our problem as an *Action-contingent Noiselessly Observable MDP*, or ACNO-MDP for short. Intuitively, ACNO-MDPs are systems described by MDPs, in which the agent can only observe the current state by making an (active) measurement, with related cost.

An ACNO-MDP is defined by a tuple $(S, \tilde{A} = A \times M, P, R, C, \Omega, O, \gamma)$, where (S, A, P, R, γ) are the components of a standard MDP: S is the state space, A is the action space, $P(s'|s,a)$ is the transition function, $R(s,a)$ is the reward function and $\gamma \in [0,1]$ is the discount factor. However, in the ACNO-MDP framework $\tilde{A}$ consists of actions and measurements pairs. These are tuples of the form $\tilde{a} = (a, m) \in A \times M$, where $M = \{\text{not observe, observe}\} = \{0, 1\}$. Control actions a affect the environment, while $m \in M$ only affects what the agent observes. Following the typical notation from POMDPs, Ω is the observation space and O the observation function, so $O(o|s', (a,m))$ is the probability of receiving observation $o \in \Omega$ when taking measurement m and action a, after transitioning to the state s'. In ACNO-MDPs all measurements are complete and noiseless, so we can define $\Omega = S \cup \{\perp\}$, where $\perp$ indicates an empty observation. Then, the observation function is defined as $O(o|s', (a,1)) = 1 \iff o = s'$, and 0 otherwise. Similarly, $O(o|s', (a,0)) = 1 \iff o = \perp$, and 0 otherwise. Measuring has an associated cost $C(m) \geq 0$, which gets subtracted from our reward, giving us a *scalarized-reward* $\tilde{r}_t = R(s_t, a_t) - C(m_t)$.

In an ACNO-MDP, the environment starts in some initial state s_0. For each time-step t, the agent executes an action-pair (a_t, m_t) according to a policy π that maps past interactions with the environment to actions. In general, the policy is defined for a belief state b_t, a distribution over the states representing the probability of being in each state of the environment summarizing the past interactions with the environment. Then, the environment transitions to a new state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, and returns to the agent a reward $r_t = R(s_t, a_t)$, a cost $c_t = C(m_t)$ and observation $o_{t+1} \sim O(\cdot \mid s_{t+1}, (a_t, m_t))$. In this paper, we focus on reinforcement learning in ACNO-MDPs, meaning we assume the agent only has access to the total number of states and the variables returned at each step, but otherwise has no prior information about the environment. Therefore, the goal of the agent is to compute a policy π with the highest expected total discounted scalarized-reward $\mathbb{E}_\pi \left[\sum_t \gamma^t \tilde{r}_t\right]$.

3.2 The AMRL-Q agent

Bellinger et al. [3] proposed to solve the ACNO-MDP problem using an adaptation of the Q-learning algorithm [22]. To choose the best action pair, the agent estimates return with a Q-table Q of size $(S \times \tilde{A})$, and the transition probability function with a table $\hat{P}$ of size $(S \times A \times S)$. $\hat{P}$ is initialised uniformly. Q is initialised uniformly, except that for all action-pairs with $m = 1$ are given some initial bias to promote measuring in early episodes.

Beginning at the initial state, for every state s_t the agent executes an ϵ-greedy action-pair (a_t, m_t) according to Q. When $m_t = 1$, the successor state $s' = s_{t+1}$ is observed so the algorithm updates the transition probability $\hat{P}(\cdot \mid s_t, a_t)$. When $m_t = 0$, AMRL-Q does not update $\hat{P}$ and assumes the successor state is the *most likely next state* given $\hat{P}$:

$$s' = \arg\max_{s \in S} \hat{P}(s \mid s_t, a_t).$$

Using the reward r_t and the (potentially estimated) successor state s', AMRL-Q updates both $Q(s_t, (a_t, 0))$ and $Q(s_t, (a_t, 1))$, as follows:

$$\begin{aligned} Q(s_t, (a_t, m)) \leftarrow & \ (1 - \alpha)\ Q(s_t, (a_t, m)) \\ & + \alpha \left[(r_t - C(m)) + \gamma \max_{a', m'} Q(s', (a', m')) \right]. \end{aligned} \tag{1}$$

After this update, we reset the environment if a terminal state is reached, and begin picking an ϵ-greedy action-pair again.

3.3 Agent shortcomings

The AMRL-Q agent has two main drawbacks. Firstly, the current choice of measuring depends solely on the initialisation of the value estimator. Since Q gets updated for all possible m-values every actions, and $(r_{t+1} - C(m))$ is always lower for $m = 0$, Q will eventually favour not measuring after enough updates. This behaviour can be useful in contexts where the final policy may not require taking any measurements, but may be sub-optimal in stochastic environments, where it is important to identify the outcome of the action executed.

Secondly, the current algorithm does not make a distinction between belief states and measured states. Although the choice of m influences whether or not $\hat{P}$ gets updated in each step, the estimated state is assumed to be the actual state of environment after that step. This means $Q(s_t, a_t, m)$ always gets updated using the values for s_{t+1}, even if s_{t+1} is a wrong estimation of the next state. Similarly, for the next step $\hat{P}(s_{t+2} | s_{t+1}, a_{t+1})$ may be updated incorrectly too.

4 Solving ACNO-MDPs: The BAM-QMDP algorithm

For reinforcement learning in ACNO-MDP settings, we propose the Q-learning based algorithm *Belief-based Active Measure QMDP*, abbreviated as BAM-QMDP. Pseudocode for the complete algorithm can be found in Appendix A, while the next sections will go through its features in more detail.

4.1 Basic Structure

The basic structure of BAM-QMDP follows that of a regular Q-learning algorithm, but with two major changes. Firstly, we incorporate active measuring by making the choice of whether or not to take a measurement an explicit part of the algorithm. Secondly, we introduce *belief states* b_t to approximate the current state if it has not been measured. We use a Q-table Q (of size $S \times A$, initialised with 0's) to keep track of estimated returns, and a transition function $\hat{P}$ (of size $S \times A \times S$, initialised with probabilities $1/|S|$) to estimate the transition function. Starting in some initial (known) belief state b_0, our algorithm repeatedly goes through the following steps (as also given in Algorithm 2, see Appendix A):

- Greedily find an action a_t to take, according to Q^t;
- Compute b_{t+1}, our next belief state after taking this action (Section 4.2);
- Decide whether or not to measure (m_t), based on b_{t+1} (Section 4.3);
- Take action-pair (a_t, m_t), receiving scalarized reward $\tilde{r}_t$ and observation o;
- If $m_t = 1$, replace b_{t+1} by a belief state representing o;
- If $m_t = 1$, update P^{t+1} (Section 4.2);
- Update Q^{t+1} (Section 4.2).

This process repeats until our episode is over, after which the BAM-QMDP+ variant performs a global update of Q to speed up convergence (Section 4.4). Then we record our data and reset our belief state to b_0.

4.2 Implementing Belief States and Optimism

In BAM-QMDP, we implement basic belief states as discrete probability distribution over the states, with $b^t(s)$ denoting the (estimated) probability of being in state s at time t. After measuring, we have a deterministic belief

$$b_{t+1}(s) = \begin{cases} 1 & \text{if } s = s_{t+1} \\ 0 & \text{otherwise.} \end{cases}$$

After taking an action without measuring, we calculate belief state b_{t+1} given previous belief state b_t, action a_t and probability table P, we use a Monte-Carlo approach to sample our next belief state. Concretely, this has the following steps (as also described in Algorithm 3):

1. sample N states $s_{t,i}$ according to b_t;
2. sample N next states $s_{t+1,i}$ according $P(\cdot \mid s_{t,i}, a_t)$, one for each $s_{t,i}$; and
3. combine these into our new probability distribution b_{t+1}:

$$b_{t+1}(s) = \frac{\sum \mathbb{I}(s_{t+1,i} = s)}{N}.$$

To express our transition function, we implement a slightly altered version of the BAMDP-framework as introduced by Dearden, Friedman, and Andre [9]. In this framework, we represent our transition function $P(\cdot|s, a)$ by a *Dirichlet* *distribution* $Dir(s,a)$, parameterized by $\vec{\alpha}_{s,a} = \{\alpha_{s,a,s_0}, \alpha_{s,a,s_1}, ...\}$. In a MDP-settings, $\alpha_{s,a,s'}$ is given by some initial bias (in our case uniform), plus the number of times the transition $s \xrightarrow{a} s'$ has occurred. However, in our partially observable setting, we cannot be certain of the latter. Instead, we interpret this as the number of times we have already *measured* this transition, but weighted by our belief. Updating our transition function, than, can be done using the following formula (as also described in Algorithm 5

$$\alpha^t_{s,a,s'} = \begin{cases} \alpha^{t-1}_{s,a,s'} + b_{t-1}(s) & \text{if } a_{t-1} = a \wedge s_t = s' \wedge m_t = 1 \\ \alpha^{t-1}_{s,a,s'} & \text{otherwise.} \end{cases} \tag{2}$$

Using this, we get our transition probabilities using the expected value for Dirichlet distribution: $\mathbb{E}[P(s,a,s')] = \alpha_{s,a,s'} / \sum_{s'' \in S} \alpha_{s,a,s''}$.

Implementing Q-learning in an environment with partial observability is a non-trivial task. For POMDPs, the problem is sometimes solved by using a neural network to estimate Q-values [11, 24, 15], inspired by similar methods for regular MDPs such as Deep Q-Learning [18]. For our algorithm, we instead choose a simpler and more direct method, in which we keep our standard Q-table but update it according to our belief state. Inspired by Even-Dar and Mansour [12], we implement this by introducing a weighted linear asynchronous learning rate $\eta_t(s,a)$, defined as:

$$\eta_t(s,a) = \frac{b_t(s)}{N^t_Q(s,a)}, \tag{3}$$

where N^t_Q is a (weighted) counter for the number of times state-action pair (s,a) has already been visited:

$$N^t_Q(s,a) = \begin{cases} N^{t-1}_Q(s,a) + b_t(s) & \text{if } a_t = a \wedge s_t = s \\ N^{t-1}_Q(s,a) & \text{otherwise.} \end{cases} \tag{4}$$

We note that $\eta_t(s,a) = 0$ if $b_t(s) = 0$, as required, and otherwise scales such that $Q_t(s,a)$ is always the average estimated return of all visits.[1] Since we already have an estimate of the transition function, we use it to approximate the future return:

$$\Psi(s,a) = \sum_{s' \in S} P^t(s' \mid s,a) \max_{a'} Q^t(s',a'). \tag{5}$$

Using these, our Q-update function becomes:

$$Q^t(s,a) = (1 - \eta_t(s,a)) \cdot Q^{t-1}(s,a) + \eta(s,a) \Big[\tilde{r}_t + \gamma \Psi(s,a) \Big] \tag{6}$$

Lastly, instead of relying on an ϵ-greedy policy for exploration, we implement a simple form of *optimism* inspired by RMAX into our return estimation [6]. For

[1] The latter is relatively common in Monte-Carlo based RL-algorithms, such as Gelly and Silver [13]

this, we define a number of *optimistic tries* N_{opt}, representing for how long an action should be biased towards exploration. We then define the *optimistic return estimation* $Q^t_{opt}(s,a)$ as the maximum value $Q(s,a)$ could have after N_{opt} tries, given its current value:

$$Q^t_{opt}(s,a) = N^t_Q(s,a) \cdot Q(s,a) + (N_{opt} - N^t_Q(s,a)) \cdot R_{max}, \tag{7}$$

where R_{max} is a parameter representing the (estimated) maximum return an action could give. For any s and a where $N^t_Q(s,a) < N_{opt}$, our algorithm will use $Q^t_{opt}(s,a)$ instead of $Q^t(s,a)$[2]. For a complete overview of a Q-update, see Algorithm 6

4.3 Active Measuring: Transition Support & Measurement Regret

Using the algorithm of the previous section as a basis, we now need to add some functionality to decide when to take measurements. We notice that there are two distinct reasons for wanting to measure: either to improve the accuracy of Q and P (i.e. exploration), or to improve our immediate expected return. The next sections describe two basic metrics to express these, after which we will describe how they are used in BAM-QMDP.

Transition Support At any point in time after deciding to take some action a_t, the agent estimates its next belief state b_{t+1} according to $\hat{P}$. However, if our agent has not done a lot of exploration yet, this belief state could be inaccurate. Supposing the agent knew for certain its current state was s_t and took action a_t, an intuitive metric of accuracy would be $N^t_P(s_t,a_t)$, as defined in the previous section. However, since the previous state is generally not known, we define the *transition support* $\mathrm{Sup}(b_t,a_t)$ as the weighted sum of all these measurements from the current belief state, that is:

$$\mathrm{Sup}(b_t,a_t) = \sum_{s \in S} b_t(s) \cdot N^t_P(s_t,a_t). \tag{8}$$

Measurement Regret As mentioned in Section 3.2, one way to expand upon the AMRL-method is by making more explicit the value of taking measurements. As an example of this problem, let us assume that we are in the belief state visualised in Fig. 1. For a such simple belief state in a normal POMDP, the optimal action can be found by calculating which action on average would give the highest reward. We'll call this the *belief-optimal* action a_{bo}, and define it as

$$a_{bo}(b) = \max_{a \in A} \sum_{s \in S} b(s) Q(s,a). \tag{9}$$

[2] The main difference between RMAX and our approach is that our optimistic Q-values gradually converge to the measured values (like is common in UCB-based methods [2]), while in RMAX the change is sudden.

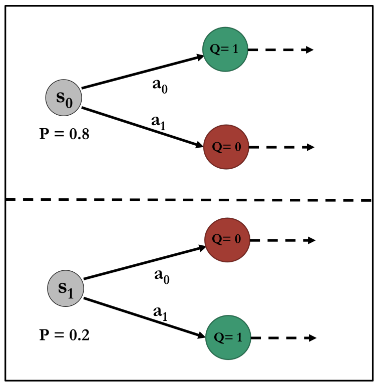


Fig. 1: An example of a simple belief state.

In Fig. 1, for example, a_0 is belief-optimal with an expected return of 0.8. When taking measuring into account, we can simply look at the difference in return between the belief-optimal action (as taken when not measuring) and the 'actual' *state-optimal* action (taken if the current state is known), at each state. We refer to this difference as the *Regret of not measuring* **R**. For a single state and a given a_{bo}, **R** can be calculated using

$$\mathbf{R}(a_{bo}, s) = Q(s, a_{bo}) - \max_{a \in A} Q(s, a), \tag{10}$$

which can be expanded over the complete belief state as

$$\begin{aligned}\mathbf{R}(a_{bo}, b) &= \sum_{s \in S} b(s)\mathbf{R}(a_{bo}, s). &(11)\\ &= \sum_{s \in S} b(s)\big[Q(s, a_{bo}) - \max_{a \in A} Q(s, a)\big] &(12)\end{aligned}$$

For our example, $\mathbf{R}(a_{bo}, s_0) = 0$ (since a_{bo} matches the optimal action in this state), and $\mathbf{R}(a_{bo}, s_1) = 1$, giving a total Loss $\mathbf{R}(a_{bo}, b) = 0.2$. In other words, having full information on the state we are in would on average yield us an added reward of 0.2. In our algorithm, before taking a measurement we compute both b and a_{bo} for the *next* state, and decide to measure the next state if $\mathbf{R}(a_{t+1,bo}, b_{t+1}) > C(m_t)$.

Combining the two conditions above, the decision of whether or not to measure is made in the following way:

$$m_t = \begin{cases} \text{observe} & \text{if } \mathrm{Sup}(b_t, a_t) \le N_O \vee \mathbf{R}(a_{t+1,bo}, b_{t+1}) \ge C(m_t) \\ \text{not observe} & \text{otherwise} \end{cases} \tag{13}$$

4.4 Speeding up Convergence: Global Q-updates & BAM-QMDP+

Although Q-learning is in general a model-free RL-algorithm, both AMRL and BAM-QMDP use an estimated transition function to model their environment.

To test if this information can be exploited further, we implement a *global Q-update function*, in which we alter all values of the Q-table based on the current model, without taking into account previous Q-values explicitly. We do this by repeatedly updating (randomly chosen) Q-values, using the following formula:

$$Q(s,a) = \hat{R}(s,a) + \gamma \Psi(s,a), \tag{14}$$

where $\hat{R}(s,a)$ is the average immediate reward recorded for (s,a), and $\Psi(s,a)$ is the estimated future return (Eq. (5)). In our algorithm, we keeps doing such updates until no states maximum Q-value is changed by more than a factor δ, as shown in Algorithm 7.

5 Experimental Setup

In this section, we describe the experimental setup to test BAM-QMDP and BAM-QMDP+, and compare both with AMRL-Q. We consider both standard RL-environments interpreted as ACNO-MDPs and a new environment designed to test the metric of measurement regret. The code to reproduce the experiments can be found on `https://github.com/MKrale/BAM-QMDP/tree/BNAIC`.

5.1 Algorithms & Hyper-parameters

We test the following algorithms:

- *AMRL-Q*: the agent as described by Bellinger et al. [3].
- *BAM-QMDP*: the agent described in Section 4 *without* the global Q-update.
- *BAM-QMDP+*: the same as BAM-QMDP, except with the global Q-update as described in Section 4.4 enabled (see Algorithm 1).

For BAM-QMDP and BAM-QMDP+, we note that there are only 4 hyper-parameters to tune: the number of optimistic tries N_{opt}, the minimal required support N_{sup}, the number of particles N used for the belief updates, and the accuracy parameter δ for the global Q-update. We set $N = 100$ and $\delta = 10^{-4}$. For N_{opt} and N_{sup}, optimal values depend on the environment used, so for these more fine-tuning is required. However, both parameters relate to very concrete conditions (particularly, the number of state-action visits required before switching from biased to unbiased behaviour), which makes it possible to reason about them intuitively. For all our testing, we will set $N_{opt} = N_{sup} = 10$, which gives good results on all environments.

For AMRL-Q, we set both ϵ and bias-factor β to 0.1, in accordance with the settings mentioned in the original paper. We keep these parameters equal for every environment tested, which is something BAM-QMDP was designed for, but AMRL-Q was not. For this reason, the performance of AMRL-Q in our testing is slightly worse than could be achieved with more hyper-parameter optimisation.

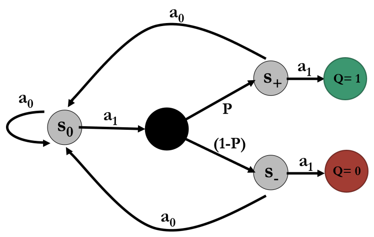


Fig. 2: The *Measurement Regret* environment used to test if an agent is able to determine the value of measuring.

5.2 Environments

This section describes the environments used for the experiments. All problems use a discount factor $\gamma = 0.95$.

Chain. This environment consists of n states $(s_0, s_1, \cdots, s_{n-1})$ connected as a chain, where s_0 is the initial and s_{n-1} the final state [21, 1]. Reaching the final state yields a reward $r = 1$, at every other step the agent gets a time penalty of $r = -0.01$. Measuring cost is set to $c = 0.05$. The agents choose from action space $A = \{backwards, forwards\}$: the first one brings the agent back to s_0, the second moves it from s_i to state s_{i+1}[3]. Although the environment is simple, by choosing a large value for n, we can use it to discover if an agent is able to discover simple but long policies.

Measurement Regret. As a simple environment to test *Measurement Regret*, we convert our example from Fig. 1 to a graph, as shown in Fig. 2. This environment consist of three state $S = \{s_0, s_+, s_-\}$, with s_0 as the initial state. Our agent can choose actions from action space $A = \{a_0, a_1\}$, where a_0 always returns the agent to the initial state. From state s_0, taking action a_1 results in a transition to s_+ with probability p, and a transition to s_- with probability $p - 1$. Taking action a_1 in the states s_+ and s_- ends the episode and returns rewards $r = 1$ and $r = 0$, respectively. For our specific testing, we choose $p = 0.8$ and $c = 0.1$, which means the optimal policy is to always measure in states s_+ and s_-.

Frozen Lake. As a more complex toy environment, we use the standard *openAI gym* Frozen Lake environment [7], which describes an $n \times n$ grid with a number of 'holes'. The goal of the agent is to walk from its initial state to some goal state without landing on any hole spaces. The agent receives reward $r = 1$ if it reaches the goal, and $r = 0$ otherwise. The episode ends once the agent reaches the goal state or a hole tile. For simplicity, we only use the pre-defined 8×8 ('large') map setting, adding a measuring cost $c = 0.01$.

[3] This is slightly different from the environment used in [3], in which the *back*-action returned the agent just one space.

Table 1: Average scalarized return and number of measurements for AMRL-Q, BAM-QMDP and BAM-QMDP+ after training. Results are gathered over 25 repetitions, and present the average over the last 500 of 5000 total episodes.

	Chain	Measurement Regret	Frozen Lake (deterministic)	Frozen Lake (slippery)	Frozen Lake (semi-slippery)
	Scalarized Return (higher is better)				
AMRL-Q	0.69	0.36	0.02	0.01	0.10
BAM-QMDP	0.79	0.85	1.00	0.02	0.56
BAM-QMDP+	0.80	0.82	1.00	0.01	0.43
	Number of Measures (lower is better)				
AMRL-Q	0.78	0.55	0.51	0.61	0.34
BAM-QMDP	0.00	0.83	0.00	0.35	5.04
BAM-QMDP+	0.00	0.25	0.00	2.79	6.32

The agent has action space $A = \{Left, Down, Right, Up\}$, but we consider three variations of the . Firstly, we use both the predefined deterministic and non-deterministic (or *slippery*) settings from the standard gym. In the deterministic case, the agent is always moved in the given direction, in the non-deterministic case it is moved in any direction except the direction opposite to the given one, with equal probability (so given $a = Left$, the agent can move $Up, Left$ or $Down$, all with probabilities $p = 1/3$). We also tested a more predictable *semi-slippery* setting, where the agent would always move in the given direction but has a 0.5 chance of moving two spaces instead.

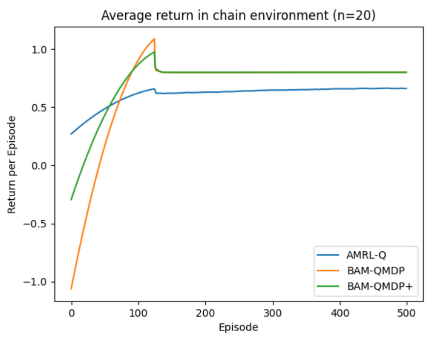
6 Results

The results of running AMRL-Q, BAM-QMDP and BAM-QMDP+ over 25 repetitions in the environments can be found in Table 1 and Figs. 3 and 4. Below, we give a brief summary of the most important results for each environment:

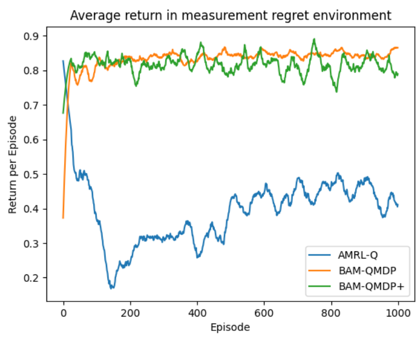
Chain. For our first test, we run the three algorithms on the chain environment with $n = 20$. As can be seen in Fig. 3a, all algorithms converge quickly. However, AMRL-Q converges to a policy yielding a scalarized return of 0.69 on average, while both BAM-QMDP agents yield approximately 0.8.

Measurement regret. Figure 3b shows the performance of all three agents in the custom measurement regret environment. As expected, both our algorithms perform significantly better than AMRL-Q, with average scalarized return of 0.41 for AMRL-Q, and 0.85 and 0.82 for BAM-QMDP and BAM-QMDP+.

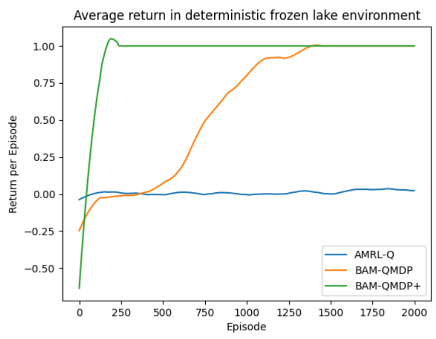
Deterministic lake environment. In Fig. 3c, we find that AMRL-Q is only able to achieve a low positive scalarized return of 0.02 in the deterministic lake environment. In contrast, both BAM-QMDP and BAM-QMDP+ are able to find an



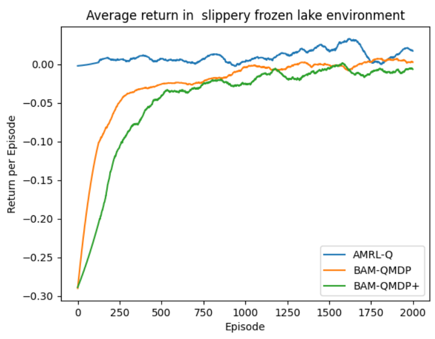
(a) Chain (n=20)



(b) Measurement regret



(c) Deterministic lake



(d) Slippery lake

Fig. 3: Plots of the average scalarized return per episode for AMRl-Q, BAM-QMDP and BAM-QMDP+, in 4 different environments.

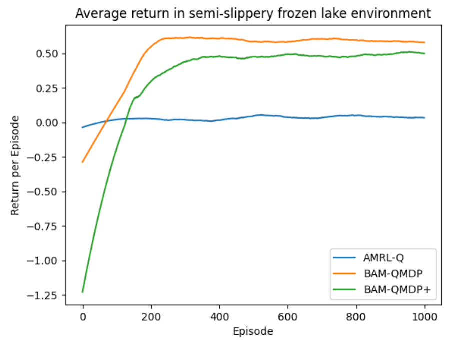
optimal strategy yielding a scalarized return of 1, although the latter converges to this policy quicker.

Slippery lake environment. As can be seen in Fig. 3d, we find that all agents converge to policies yielding similar scalarized return of approximately 0. Both BAM-QMDP and BAM-QMDP+ take more measurements in early episodes, giving them slightly lower average scalarized return in early episodes.

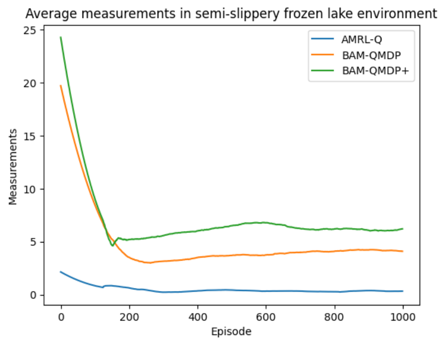
Semi-slippery lake environment. In Fig. 4a, we see that in the less random slippery lake environment, on the one hand BAM-QMDP and BAM-QMDP+ are able to achieve scalarized return of 0.56 and 0.42. On the other hand, AMRL-Q is still only able to find a policy with scalarized return of 0.01

7 Discussion

Based on the results in the previous sections, we make the following observations:



(a) Average scalarized return.



(b) Average number of measurements.

Fig. 4: The average return and number of measurements per episode for AMRL-Q, BAM-QMDP and BAM-QMDP+ in the *slightly slippery lake* environment.

Measurement regret works as a metric for valuing measurements. In Table 1, BAM-QMDP achieves returns higher than 0.8, the return of the optimal non-measuring policy. Since after initialisation measuring is only governed by measurement regret, we conclude it works as a metric for valuing measurements.

Measurement regret does not ensure policies with optimal measurements. To determine the optimal return for a policy which may include measuring, we note that in such policies we always measure in both s_+ and s_-, since we cannot distinguish these states. In s_+, this returns $(1-c)$, and in s_- this gives the average return of s_0 minus c. Putting this together, we find we can calculate the return of this policy with following infinite sum:

$$\mathbb{E}_\pi\left[\sum_t \gamma^t \tilde{r}_t\right] = \sum_{n=0}\Big(p\cdot\big(1-p\big)^n\big(1-c(n+1)\big)\Big), \tag{15}$$

where n is the number of measurements required. Using our values for p and c, we find an optimal return of 0.875. Although the return for BAM-QMDP comes close to this, we see it is still lower, showing that our algorithm is not able to determine the value of measuring perfectly.

Measurement regret yields non-measuring policies in deterministic environments. As seen in Table 1, both variants of BAM-QMDP make no measurements in the last 500 episodes in the deterministic environments (chain and deterministic frozen lake). On the other hand, for non-deterministic environments it generally converges to policies which do include measurements. After convergence, we see AMRL-Q takes measurements in both stochastic and non-stochastic environments alike due to it's ϵ-greedy policy, but otherwise never measures.

In simple environments, BAM-QMDP yields policies with higher scalarized return than AMRL-Q. Table 1 shows both versions of BAM-QMDP achieve higher scalarized return than AMRL-Q for all environments, except the Slippery Lake environment, where none of our algorithms were able to reach the goal consistently.

Although unoptimised now, BAM-QMDP has the potential to be scaled to larger environments. To be useful for large problems, an algorithm should scale well with $S \times A$ in terms of both time complexity and convergence speed. For the former, we note that sampling, finding an action and updating Q or P all take $O(A \cdot N_P)$ time. This means running time only scales with A, N_p and the length of the paths taken. Convergence rates of value iteration algorithms generally scale poorly with $S \times A$, but offline training methods such as used in BAM-QMDP+ can easily overcome this. Note, though, that for this to be useful, their time complexity should also be low, which is not the case for BAM-QMDP+.

8 Conclusion

In this paper, we studied how to solve MDPs with explicit measurement actions with associated costs. For this framework, we propose two versions of the Q-learning based algorithm BAM-QMDP. The empirical analysis shows that both versions outperform previous algorithm in terms of return of final policy, in both deterministic and non-deterministic environments. Future work include exploring partial measurements, for instance dealing with factored spaces [5]. Furthermore, we could extend our approach to setting with limited measuring budget [14], considering the measuring costs are known, we could try to learn a policy without violating the budget constraints [20]. Finally, we are also interested in extending this approach to larger environments [15].

References

[1] Mauricio Araya-López et al. “Active Learning of MDP Models”. In: *EWRL*. Vol. 7188. 2011, pp. 42–53.

[2] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. “Exploration–exploitation tradeoff using variance estimates in multi-armed bandits”. In: *Theoretical Computer Science* 410.19 (2009), pp. 1876–1902.

[3] Colin Bellinger et al. “Active Measure Reinforcement Learning for Observation Cost Minimization”. In: *Canadian Conference on AI*. 2021.

[4] Gabriel Bernardino et al. “Reinforcement Learning for Active Modality Selection During Diagnosis”. In: *MICCAI*. Vol. 13431. Springer, 2022, pp. 592–601.

[5] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. “Stochastic dynamic programming with factored representations”. In: *Artif. Intell.* 121.1-2 (2000), pp. 49–107.

[6] Ronen I Brafman and Moshe Tennenholtz. “R-max - a general polynomial time algorithm for near-optimal reinforcement learning”. In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231.

[7] Greg Brockman et al. *OpenAI Gym.* arXiv preprint arXiv:1606.01540. 2016.

[8] Antonio Coronato et al. “Reinforcement learning for intelligent healthcare applications: A survey”. In: *Artif. Intell. Medicine* 109 (2020), p. 101964.

[9] Richard Dearden, Nir Friedman, and David Andre. “Model based Bayesian Exploration”. In: *UAI.* Morgan Kaufmann, 1999, pp. 150–159.

[10] Finale Doshi-Velez, Joelle Pineau, and Nicholas Roy. “Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs”. In: *Artif. Intell.* 187 (2012), pp. 115–132.

[11] Maxim Egorov. *Deep Reinforcement Learning with POMDPs.* 2015.

[12] Eyal Even-Dar and Yishay Mansour. “Learning Rates for Q-learning”. In: *J. Mach. Learn. Res.* 5 (2003), pp. 1–25.

[13] Sylvain Gelly and David Silver. “Monte-Carlo tree search and rapid action value estimation in computer Go”. In: *Artif. Intell.* 175.11 (2011), pp. 1856–1875.

[14] Mahsa Ghasemi and Ufuk Topcu. “Online Active Perception for Partially Observable Markov Decision Processes with Limited Budget”. In: *CDC.* IEEE, 2019, pp. 6169–6174.

[15] Matthew J. Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: *AAAI Fall Symposia.* 2015, pp. 29–37.

[16] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. “Reinforcement Learning in Robotics: Applications and Real-World Challenges”. In: *Robotics* 2.3 (2013), pp. 122–148.

[17] Lei Lei et al. “Deep Reinforcement Learning for Autonomous Internet of Things: Model, Applications and Challenges”. In: *IEEE Commun. Surv. Tutorials* 22.3 (2020), pp. 1722–1760.

[18] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nat.* 518.7540 (2015), pp. 529–533.

[19] HyunJi Alex Nam, Scott Fleming, and Emma Brunskill. “Reinforcement Learning with State Observation Costs in Action-Contingent Noiselessly Observable Markov Decision Processes”. In: *NeurIPS.* Vol. 34. 2021.

[20] Thiago D. Simão, Nils Jansen, and Matthijs T. J. Spaan. “AlwaysSafe: Reinforcement Learning without Safety Constraint Violations during Training”. In: *AAMAS.* ACM, 2021, pp. 1226–1235.

[21] Malcolm J. A. Strens. “A Bayesian Framework for Reinforcement Learning”. In: *ICML.* 2000, pp. 943–950.

[22] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning.* 1992, pp. 279–292.

[23] Haiyan Yin et al. *Reinforcement Learning with Efficient Active Feature Acquisition.* arXiv preprint arXiv:2011.00825. 2020.

[24] Pengfei Zhu et al. *On Improving Deep Reinforcement Learning for POMDPs.* arXiv preprint arXiv:1704.07978. 2017.

A Algorithm Pseudocode

Algorithm 1 BAM-QMDP(episodes)

```
for $s, s' \in S$ and $a \in A$ do
    Set $\alpha_{s,a,s'} = 1/|S|, N_Q(s,a) = 0$
    Set $Q(s,a) = 0$, $Q_{opt}(s,a) = 1, R(s,a) = 0$
Set $r_{total} = 0$
for $i <$episodes do
    $r_{eps}, Q, Q_{opt}, \vec{\alpha}, R$
        $\leftarrow$ RunEpisode$(Q, Q_{opt}, \vec{\alpha}, R, b_0)$
    $r_{total} \leftarrow r_{total} + r_{eps}$
    if global update enabled then
        UpdateQGlobally$(Q, P)$
return$r_{total}$
```

Algorithm 2 RunEpisode $(Q, Q_{opt}, \vec{\alpha}, R, b_0)$

```
$b \leftarrow b_0$
$r_{episode} = 0$
while episode not done do
    $a \leftarrow$FindGreedyAction$(Q_{\text{opt}}, b)$
    $b_{\text{next}} \leftarrow$SampleNextBelief(P,b,a)
    $a_{bo} \leftarrow \max_{a \in A} \sum_{s \in S} b(s) Q_{\text{opt}}(s,a)$
    $m \leftarrow Sup(b,a) > N_{sup} \wedge \mathbf{R}(b_{\text{next}}) < c$
    take action $(a, m) \rightarrow (o, r)$
    $Q, Q_{\text{opt}}, R \leftarrow$ UpdateQ$(P, Q, b, a, o, r)$
    if $m = 1$ then
        $P, \alpha \leftarrow$UpdateP$(\vec{\alpha}, b, a, o)$
        $b \leftarrow o$
    else
        $b \leftarrow b_{\text{next}}$
    $r_{episode} \leftarrow r_{episode} + r$
return$r_{episode}, Q, Q_{opt}, \vec{\alpha}, R$
```

Algorithm 3 SampleNextBelief(P, b, a)

```
for $i < N$ do
    $s_{t,i} \sim b$
    $s_{t+1,i} \sim P(s_{t,i}, a, \cdot)$
for $s \in S$ do
    $b(s) \leftarrow \frac{\sum \mathbb{I}(s_{t+1,i}=s)}{N}$
return $b$
```

Algorithm 4 FindGreedyAction(Q, b)

```
initialise $Q_b(a) = 0, \forall a \in A$
for $(s, a) \in S \times A$ do
    $Q_b(a) \leftarrow Q_b(a) + b(s) Q(s,a)$
return $\text{argmax}_{a \in A} Q_b(a)$
```

Algorithm 5 UpdateP$(\vec{\alpha}, b, a, o)$

```
for $s \in S$ do
    $\alpha_{s,a,o} \leftarrow \alpha_{s,a,o} + b(s)$
    for $s' \in S$ do
        $P(s' \mid s, a) = \alpha_{s,a,s'} / \sum_{s'' \in S} \alpha_{s,a,s''}$
return$P, \vec{\alpha}$
```

Algorithm 6 UpdateQ(P, Q, R, b, a, o, r)

```
for $s \in S$ do
    $R(s,a) \leftarrow \frac{R(s,a) N_Q(s,a) + b(s) r}{N_Q(s,a) + b(s)}$
    $N_Q(s,a) \leftarrow N_Q(s,a) + b(s)$
    $\eta(s,a) \leftarrow \frac{b(s)}{N_Q(s,a)}$
    $\Psi = \sum_{s'' \in S} P(s'' \mid s', a) \max_{a''} Q(s'', a'')$
    $Q(s,a) \leftarrow [1 - \eta_{s,a}] Q(s,a) + \eta_{s,a}[r + \gamma \Psi]$
    if $N_Q(s,a) < N_{opt}$ then
        $Q_{\text{opt}}(s,a) \leftarrow \frac{Q(s,a) \cdot N_Q(s,a) + (N_O - N_Q(s,a))}{N_O}$
    else
        $Q_{\text{opt}}(s,a) \leftarrow Q(s,a)$
return$Q, Q_{\text{opt}}, R$
```

Algorithm 7 UpdateQGlobally(Q, P, R)

```
$L \leftarrow S$
while $L$ not empty do
    $Q_{prev} \leftarrow Q$
    $s \leftarrow \arg\max_s Q(s,a)$
    for $[s', a' | P(s, a', s') > \epsilon]$ do
        $\Psi = \sum_{s'' \in S} P(s'' \mid s', a) \max_{a''} Q(s'', a'')$
        $Q(s', a') = R + \gamma Psi$
        for $[s'' \in S | \exists a'' \in A,$
                $|Q_{prev}(s'', a'') - Q(s'', a'')| \geq \delta]$ do
            $L \leftarrow L \cup s''$
    $L \leftarrow L \setminus s$
```