Offline Policy-search in Bayesian Reinforcement Learning

DOCTORAL THESIS

Author:
Michael CASTRONOVO

Advisor:
Damien ERNST

A thesis presented for the degree of
Doctor of Computer Science

in the

Faculty of Applied Sciences
Department of Electrical Engineering and Computer Science

2016

UNIVERSITY OF LIEGE

Abstract

Faculty of Applied Sciences
Department of Electrical Engineering and Computer Science

Doctor of Computer Science

Offline Policy-search in Bayesian Reinforcement Learning

by Michael CASTRONOVO

This thesis presents research contributions in the study field of Bayesian Reinforcement Learning — a subfield of Reinforcement Learning where, even though the dynamics of the system are unknown, the existence of some prior knowledge is assumed in the form of a distribution over Markov decision processes.

In this thesis, two algorithms are presented: OPPS (Offline Prior-based Policy Search) and ANN-BRL (Artificial Neural Networks for Bayesian Reinforcement Learning), whose philosophy consists to analyse and exploit the knowledge available beforehand prior to interacting with the system(s), and which differ by the nature of the model they make use of. The former makes use of formula-based agents introduced by Maes et al. in (Maes, Wehenkel, and Ernst, 2012), while the latter relies on Artificial Neural Networks built via SAMME (Stagewise Additive Modelling using a Multi-class Exponential loss function) — an AdaBoost algorithm developed by Zhu et al. in (Zhu et al., 2009).

Moreover, we also describe a comprehensive benchmark which has been created to compare Bayesian Reinforcement Learning algorithms. In real life applications, the choice of the best agent to fulfil a given task depends not only on their performances, but also on the computation times required to deploy them. This benchmark has been designed to identify the best algorithms by taking both criteria into account, and resulted in the development of an open-source library: BBRL (Benchmarking tools for Bayesian Reinforcement Learning)[1].

[1] https://github.com/mcastron/BBRL/wiki

UNIVERSITE DE LIEGE

Résumé

Faculté des Sciences Appliquées
Département d'Electricité, Electronique et Informatique

Docteur en Sciences Informatiques

Recherche directe de politique hors-ligne en apprentissage par renforcement Bayésien

par Michaël CASTRONOVO

Cette dissertation présente diverses contributions scientifiques dans le domaine de l'apprentissage par renforcement Bayésien, dans lequel les dynamiques du système sont inconnues et pour lequelles nous disposons de connaissances a priori, existant sous la forme d'une distribution sur un ensemble de processus décisionnels Markoviens.

Nous présentons tout d'abord deux algorithmes, OPPS (Offline Prior-based Policy Search — recherche directe de politique hors-ligne) et ANN-BRL (Artificial Neural Networks for Bayesian Reinforcement Learning — réseaux de neurones artificiels pour l'apprentissage par renforcement Bayésien), dont la philosophie repose sur l'analyse et l'exploitation de ces connaissances a priori avant de commencer à intéragir avec le(s) système(s). Ces méthodes diffèrent par la nature de leur modèle. La première utilise des agents à base de formule introduits par Maes et al. dans (Maes, Wehenkel, and Ernst, 2012), tandis que la seconde repose sur l'utilisation de réseaux de neurones artificiels construits grâce à SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function — modélisation additive par cycle basée sur une fonction de perte exponentielle multi-classe), un algorithme d'adaboosting développé par Zhu et al. dans (Zhu et al., 2009),

Nous décrivons également un protocole expérimental que nous avons conçu afin de comparer les algorithmes d'apprentissage par renforcement Bayésien entre eux. Dans le cadre d'applications réelles, le choix du meilleur agent pour traiter une tâche spécifique dépend non seulement des ses performances, mais également des temps de calculs nécessaires pour le déployer. Ce protocole expérimental permet de déterminer quel est le meilleur algorithme pour résoudre une tâche donnée en tenant compte de ces deux critères. Ce dernier a été mis à la disposition de la communauté scientifique sous la forme d'une bibliothèque logicielle libre : BBRL (Benchmarking tools for Bayesian Reinforcement Learning — outils de comparaison pour l'apprentissage par renforcement Bayésien)[2].

[2] https://github.com/mcastron/BBRL/wiki

Contents

Chapter 1

Introduction

Sequential decision-making is at the very heart of many applications such as medicine, finance and robotics (Murphy, 2005; Nevmyvaka, Feng, and Kearns, 2006; Kober et al., 2012). Essentially, a good automated system requires observing an environment, collecting data, as well as reacting appropriately. In 1988, R. S. Sutton introduced the very concept of Reinforcement Learning (RL) to tackle this problem (Sutton, 1988; Buşoniu et al., 2010): an agent observes a reward signal as a result of interactions with an initially unknown environment. Before each decision is made, the agent is allowed to perform a number of computations to determine the next action to take. Actions that yield the highest performance according to the current knowledge of the environment and those that maximise the gathering of valuable information may not be the same. This is the dilemma known as Exploration/Exploitation (E/E). The lack of information at the beginning is a major limiting factor. As new approaches were developed, RL researchers started to train their algorithms with more and more data obtained from the environment, and only evaluate the performance of the agent obtained afterwards. By focusing more on pure performance than learning efficiency, eventually the quality of the policy obtained started to become more important than the E/E dilemma. This no longer matches with the problem that RL was originally designed for.

Bayesian Reinforcement Learning (BRL) is a subfield of RL specially adapted for this task. Assuming some prior knowledge is available at the beginning, researchers in BRL have been able to formalise a new paradigm while keeping the E/E dilemma in mind. However, another problem arises when confronting these algorithms: they were intractable in practice (Duff, 2002). This was due to the greater dimensionality of the problem in its BRL form. In recent years, some researchers in BRL finally succeeded in developing tractable approximations (Guez, Silver, and Dayan, 2012; Asmuth and Littman, 2011; Castro and Precup, 2010). Unfortunately, most of them also decided to use benchmarks which do not take into account the computation time constraints, inherent to any real life application.

It is also necessary to use an appropriate benchmark when choosing a method to address a specific problem. Our main contribution consists of a framework allowing BRL algorithms to be compared to

each other. We believe the quality of a policy in BRL has to be put in perspective with both computation time and the real application. Given a specific decision-making problem, we can identify two major constraints:

1. Offline computation time: the time available prior to interacting with the process;

2. Online computation time: the time available before each decision-making process.

In practice, no algorithm is optimal in every aspect. It is up to us to determine which one is the most suitable for addressing a specific problem. To the extent of our knowledge, this aspect is often forgotten within the BRL community. Our benchmarking tools have been packaged under an open-source library: BBRL[1]. This library is available on GitHub[2] where all the material used in this thesis can be found. A wiki has been opened to help other researchers to use and modify BBRL according to their needs. BBRL presents a set of benchmarks along with several well-known BRL algorithms. It compares each algorithm on every benchmark and gives a general picture of their performance in regard to computation time.

Additionally, two algorithms have been developed taking advantage of the offline phase formalised with our benchmark: OPPS[3] and ANN-BRL[4]. OPPS is an extension of F. Maes' work on multi-armed bandit problems (Maes, Wehenkel, and Ernst, 2012) to MDPs, while ANN-BRL formalises the BRL setting as a Supervised Learning problem, and train an Artificial Neural Network (ANN) acting as a BRL agent.

The general Bayesian reinforcement learning setting is introduced in Section 1.1, and a brief description of each contribution is provided in Section 1.2.

1.1 Bayesian Reinforcement Learning

Let $M = (X, U, f(\cdot), \rho_M, \rho_{M,0}(\cdot), \gamma)$ be a Markov Decision Process (MDP), where $X = \{x^{(1)}, \ldots, x^{(n_X)}\}$ is its finite state space, $U = \{u^{(1)}, \ldots, u^{(n_U)}\}$ is its finite action space, $f : X \times U \times X$ is its transition matrix, ρ_M is its reward function, $\rho_{M,0}(\cdot)$ is its initial state distribution, and γ is the discount factor. In our setting, the transition matrix f is the only unknown element of MDP M.

At each time-step t, an agent interacts with MDP M by performing an action $u_t \in U$, and moves to state x_{t+1} with a probability

[1] BBRL stands for **B**enchmarking tools for **B**ayesian **R**einforcement **L**earning.

[2] http://www.github.com/mcastron/BBRL/wiki/

[3] OPPS stands for **O**ffline **P**rior-based **P**olicy **S**earch.

[4] ANN-BRL stands for **A**rtifical **N**eural **N**etwork for **B**ayesian **R**einforcement **L**earning.

$P(x_{t+1}|x_t, u_t) = f(x_t, u_t, x_{t+1})$. In return, the environment sends back a reward signal $r_t = \rho_M(x_t, u_t, x_{t+1})$. The action selected by the agent depends on history $h_t = < x_0, u_0, r_0, x_1 >, \ldots, < x_{t-1}, u_{t-1}, r_{t-1}, x_t >$, the set of all transitions observed at time-step t.

In Bayesian Reinforcement Learning, the E/E dilemma is formalised through a Bayes-adaptive MDP (BAMDP), which can be defined on top of any MDP M. Formally, let $M^+ = (X^+, U, f^+(\cdot), \rho^+_M, \rho_{M,0}(\cdot), \gamma)$ be the BAMDP corresponding to MDP M. The belief state space $X^+ = X \times H$, with H being the set of possible histories, is obtained by considering the current history as part of the current state, and the dynamics of the system are described as follows:

$$f^+(< x, h >, u, < x', h' >) = \mathbb{1}\left[h' = hux'\right] \int_f f(x, u, x')\ P(f|h)\ df, \quad (1.1)$$

$$\rho^+_M(< x, h >, u) = \rho_M(x, u). \quad (1.2)$$

A Bayes-optimal policy $\pi^*_{M^+}$ on M^+ addresses the E/E dilemma optimally on MDP M. In theory, since the dynamics are known, this policy can be inferred from its optimal value function as:

$$Q^*(< x_t, h_t >, u) = \sum_{s'} P(x'|x, h, u)\ [\ \rho^+_M(< x, h >, u) + \gamma\ \max_{u'} Q^*(< x, hus' >, u')\ ], \quad (1.3)$$

$$\pi^*_{M^+}(< x_t, h_t >, u) = \max_{u'} Q^*(< x_t, h_t >, u'). \quad (1.4)$$

However, in practice, computing $\pi^*_{M^+}$ is an intractable problem (Guez, Silver, and Dayan, 2013). First, the BAMDP state space is infinite. It is, therefore, not possible to fully explore it. Second, the optimal value function cannot be computed without sampling transition probabilities based on past observations. This process relies on the calculation of $P(f|h_t) \propto P(h_t|f)P(f)$, which is intractable (Duff, 2002; Kaelbling, Littman, and Cassandra, 1998; Kolter and Ng, 2009). Thus, the goal of a Bayesian Reinforcement Learning algorithm is to approximate this policy at best.

Most state-of-art BRL algorithms are able to improve the accuracy of their approximation by increasing their computation time. It is worth noting that any real-life application is subject to computation time constraints. In these circumstances, identifying the most suitable algorithm for a given problem depends on its efficiency in terms of both performance and computation time.

1.2 Main Contributions

The following chapters are slightly edited research publications. The next sections of this introduction will give you a brief summary of each one.

1.2.1 Chapter 2: Learning Exploration/Exploitation Strategies for Single Trajectory Reinforcement Learning

In Chapter 2, we adapt an algorithm of F. Maes et al. (Maes, Wehenkel, and Ernst, 2012), designed for addressing k-armed bandit problems, to MDPs with discrete state/action spaces[5]. They proposed to define an index-based agent whose decisions are determined by a formula which associates a score to each arm with respect to the current data available (e.g. the number of draws or the mean reward of each arm). By building a large set of various formulas, they were able to discover good formulas and build high-performance index-based agents.

In our algorithm, instead of evaluating arms, our formulas evaluate $<$ state, action $>$ pairs, and the agent plays the action which obtained the highest score among the pairs related to the current state. The features on which the formulas are based have also been adapted so as to address the MDP case. This algorithm has been referred to as OPPS in the studies that followed.

1.2.2 Chapter 3: Bayes Adaptive Reinforcement Learning versus On-line Prior-based Policy Search

In Chapter 3, OPPS is adapted to the Bayesian Reinforcement Learning setting and compared to BAMCP (Bayes-Adaptive Monte-Carlo Planning), which is a state-of-the-art Bayesian Reinforcement Learning algorithm. While OPPS allocates most of its computational resources prior to the first interaction, BAMCP performs its calculations before making any decision with adjustable computation times. As a consequence, the quality of the policy depends on the computation time allowed for BAMCP to make its calculations. In order to provide fair comparison between the two algorithms, a benchmark which takes into account both the performances and the computation times has been designed.

[5] It is also noted that the work of Maes on E/E has not been limited to bandit problems. It has been applied to tree search ones as well, see (Jung, Ernst, and Maes, 2013; Jung et al., 2014; Perrick et al., 2012).

1.2.3 Chapter 4: Benchmarking for Bayesian Reinforcement Learning

In Chapter 4, the benchmark introduced in Chapter 3 is improved and a detailed comparison between most Bayesian Reinforcement Learning algorithms is provided. Additionally, we present BBRL (Benchmarking for Bayesian Reinforcement Learning), which is an open-source library developed to help other researchers to make use of our benchmark.

1.2.4 Chapter 5: Artificial Neural Networks for Bayesian Reinforcement Learning

In Chapter 5, an innovative algorithm which combines the philosophy of OPPS with Artificial Neural Networks is described. ANN-BRL (Artificial Neural Networks for Bayesian Reinforcement Learning) formalises the Bayesian Reinforcement Learning setting into a Supervised Learning problem. The prior distribution is then used to generate a suitable dataset from several trajectories drawn during the offline phase. This dataset is eventually used to train ANNs (Artificial Neural Networks) in order to represent the model used during the online phase to make decisions.

Chapter 2

Learning Exploration/Exploitation Strategies for Single Trajectory Reinforcement Learning

We consider the problem of learning high-performance Exploration/Exploitation (E/E) strategies for finite Markov Decision Processes (MDPs) when the MDP to be controlled is supposed to be drawn from a known probability distribution $p_{\mathcal{M}}(\cdot)$. The performance criterion is the sum of discounted rewards collected by the E/E strategy over an infinite length trajectory. We propose an approach for solving this problem that works by considering a rich set of candidate E/E strategies and by looking for the one that gives the best average performances on MDPs drawn according to $p_{\mathcal{M}}(\cdot)$. As candidate E/E strategies, we consider index-based strategies parametrized by small formulas combining variables that include the estimated reward function, the number of times each transition has occurred and the optimal value functions $\hat{V}$ and $\hat{Q}$ of the estimated MDP (obtained through value iteration). The search for the best formula is formalized as a multi-armed bandit problem, each arm being associated with a formula. We experimentally compare the performances of the approach with R-MAX *as well as with ϵ-*GREEDY *strategies and the results are promising.*

The work presented in this chapter has been published in the *Proceedings of the European Workshop on Reinforcement Learning (EWRL 2012)* (Castronovo et al., 2012).

2.1 Introduction

Most Reinforcement Learning (RL) techniques focus on determining high-performance policies maximizing the expected discounted sum of rewards to come using several episodes. The quality of such a learning process is often evaluated through the performances of the final policy regardless of rewards that have been gathered during learning. Some approaches have been proposed to take these rewards into account by minimizing the undiscounted regret (Kearns and Singh, 2002; Brafman and Tennenholtz, 2002; Auer and Ortner, 2007; Jaksch, Ortner, and Auer, 2010), but RL algorithms have troubles solving the *original RL problem* of maximizing the expected discounted return over a single trajectory. This problem is almost intractable in the general case because the discounted nature of the regret makes early mistakes - often due to hazardous exploration - almost impossible to recover from. Roughly speaking, the agent needs to learn very fast in one pass. One of the best solutions to face this Exploration/Exploitation (E/E) dilemma is the R-MAX algorithm (Brafman and Tennenholtz, 2002) which combines model learning and dynamic programming with the "optimism in the face of uncertainty" principle. However, except in the case where the underlying Markov Decision Problem (MDP) comes with a small number of states and a discount factor very close to 1 (which corresponds to giving more chance to recover from bad initial decisions), the performance of R-MAX is still very far from the optimal (more details in Section 2.5).

In this paper, we assume some prior knowledge about the targeted class of MDPs, expressed in the form of a probability distribution over a set of MDPs. We propose a scheme for learning E/E strategies that makes use of this probability distribution to sample training MDPs. Note that this assumption is quite realistic, since before truly interacting with the MDP, it is often possible to have some prior knowledge concerning the number of states and actions of the MDP and/or the way rewards and transitions are distributed.

To instantiate our learning approach, we consider a rich set of candidate E/E strategies built around parametrized index-functions. Given the current state, such index-functions rely on all transitions observed so far to compute E/E scores associated to each possible action. The corresponding E/E strategies work by selecting actions that maximize these scores. Since most previous RL algorithms make use of small formulas to solve the E/E dilemma, we focus on the class of index-functions that can be described by a large set of such small formulas. We construct our E/E formulas with variables including the estimated reward function of the MDP (obtained from observations), the number of times each transition has occurred and the estimated optimal value functions $\hat{V}$ and $\hat{Q}$ (computed through off-line value iteration) associated with the estimated MDP. We then formalize the search for an optimal formula within that space as a multi-armed

bandit problem, each formula being associated to an arm.

Since it assumes some prior knowledge given in the form of a probability distribution over possible underlying MDPs, our approach is related to Bayesian RL (BRL) approaches (Poupart et al., 2006; Asmuth et al., 2009) that address the E/E trade-off by (i) assuming a prior over possible MDP models and (ii) maintaining - from observations - a posterior probability distribution (i.e., "refining the prior"). In other words, the prior is used to reduce the number of samples required to construct a good estimate of the underlying MDP and the E/E strategy itself is chosen a priori following Bayesian principles and does not depend on the targeted class of MDPs. Our approach is specific in the sense that the prior is not used for better estimating the underlying MDP but rather for identifying the best E/E strategy for a given class of targeted MDPs, among a large class of diverse strategies. We therefore follow the work of (Maes, Wehenkel, and Ernst, 2012), which already proposed to learn E/E strategies in the context of multi-armed bandit problems, which can be seen as state-less MDPs.

This paper is organized as follows. Section 2.2 formalizes the E/E strategy learning problem. Section 2.3 describes the space of formula-based E/E strategies that we consider in this paper. Section 2.4 details our algorithm for efficiently learning formula-based E/E strategies. Our approach is illustrated and empirically compared with R-MAX as well as with ϵ-GREEDY strategies in Section 2.5. Finally, Section 2.6 concludes.

2.2 Background

Let $M = (\mathcal{S}, \mathcal{A}, p_{M,f}(\cdot), \rho_M, p_{M,0}(\cdot), \gamma)$ be a MDP. $\mathcal{S} = \left\{s^{(1)}, \ldots, s^{(n_\mathcal{S})}\right\}$ is its state space and $\mathcal{A} = \left\{a^{(1)}, \ldots, a^{(n_\mathcal{A})}\right\}$ its action space. When the MDP is in state s_t at time t and action a_t is selected, the MDP moves to a next state s_{t+1} drawn according to the probability distribution $p_{M,f}(\cdot|s_t, a_t)$. A deterministic instantaneous scalar reward $r_t = \rho_M(s_t, a_t, s_{t+1})$ is associated with the stochastic transition (s_t, a_t, s_{t+1}).

$H_t = [s_0, a_0, r_0, \ldots, s_t, a_t, r_t]$ is a vector that gathers the history over the first t steps and we denote by $\mathcal{H}$ the set of all possible histories of any length. An exploration / exploitation (E/E) strategy is a stochastic algorithm π that, given the current state s_t, processes at time t the vector H_{t-1} to select an action $a_t \in \mathcal{A}$: $a_t \sim \pi(H_{t-1}, s_t)$. Given the probability distribution over initial states $p_{M,0}(\cdot)$, the performance/return of a given E/E strategy π with respect to the MDP M can be defined as: $J^\pi_M = \mathop{\mathbb{E}}_{p_{M,0}(\cdot), p_{M,f}(\cdot)}[\mathcal{R}^\pi_M(s_0)]$ where $\mathcal{R}^\pi_M(s_0)$ is the stochastic discounted return of the E/E strategy π when starting from the state s_0. This return is defined as:

$$\mathcal{R}^\pi_M(s_0) = \sum_{t=0}^{\infty} \gamma^t r_t \ ,$$

where $r_t = \rho_M(s_t, \pi(H_{t-1}, s_t), s_{t+1})$ and $s_{t+1} \sim p_{M,f}(.|s_t, \pi(H_{t-1}, s_t))$ $\forall t \in \mathbb{N}$ and where the discount factor γ belongs to $[0, 1)$. Let $p_{\mathcal{M}}(\cdot)$ be a probability distribution over MDPs, from which we assume that the actual underlying MDP M is drawn. Our goal is to learn a high performance finite E/E strategy π given the prior $p_{\mathcal{M}}(\cdot)$, i.e. an E/E strategy that maximizes the following criterion:

$$J^\pi = \mathop{\mathbb{E}}_{M' \sim p_{\mathcal{M}}(\cdot)} [J^\pi_{M'}] \ . \tag{2.1}$$

2.3 Formula-based E/E Strategies

In this section, we describe the set of E/E strategies that are considered in this paper.

2.3.1 Index-based E/E Strategies

Index-based E/E strategies are implicitly defined by maximizing history-dependent state-action index functions. Formally, we call a history-dependent state-action index function any mapping $I : \mathcal{H} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Given such an index function I, a decision can be taken at time t in the state $s_t \in \mathcal{S}$ by drawing an optimal action according to I: $\pi(H_{t-1}, s_t) \in \arg\max_{a \in \mathcal{A}} I(H_{t-1}, s_t, a)$[1]. Such a procedure has already been vastly used in the particular case where the index function is an estimate of the action-value function, eventually randomized using $\epsilon-$greedy or Boltzmann exploration, as in Q-LEARNING (Watkins and Dayan, 1992).

2.3.2 Formula-based E/E Strategies

We consider in this paper index functions that are given in the form of small, closed-form formulas. This leads to a very rich set of candidate E/E strategies that have the advantage of being easily interpretable by humans. Formally, a formula $F \in \mathbb{F}$ is:

• either a binary expression $F = B(F', F'')$, where B belongs to a set of binary operators $\mathbb{B}$ and F' and F'' are also formulas from $\mathbb{F}$,

• or a unary expression $F = U(F')$ where U belongs to a set of unary operators $\mathbb{U}$ and $F' \in \mathbb{F}$,

• or an atomic variable $F = V$, where V belongs to a set of variables $\mathbb{V}$ depending on the history H_{t-1}, the state s_t and the action a,

• or a constant $F = C$, where C belongs to a set of constants $\mathbb{C}$.

Since it is high dimensional data of variable length, the history H_{t-1} is non-trivial to use directly inside E/E index-functions. We proceed as follows to transform the information contained in H_{t-1}

[1] Ties are broken randomly in our experiments.

into a small set of relevant variables. We first compute an estimated model of the MDP $\hat{M}$ that differs from the original M due to the fact that the transition probabilities and the reward function are not known and need to be learned from the history H_{t-1}. Let $\hat{P}(s,a,s')$ and $\hat{\rho}(s,a)$ be the transition probabilities and the reward function of this estimated model. $\hat{P}(s,a,s')$ is learned by computing the empirical frequency of jumping to state s' when taking action a in state s and $\hat{\rho}(s,a)$ is learned by computing the empirical mean reward associated to all transitions originating from (s,a)[2]. Given the estimated MDP, we run a value iteration algorithm to compute the estimated optimal value functions $\hat{V}(\cdot)$ and $\hat{Q}(\cdot,\cdot)$. Our set of variables is then defined as: $\mathbb{V} = \left\{\hat{\rho}(s_t,a), N(s_t,a), \hat{Q}(s_t,a), \hat{V}(s_t), t, \gamma^t\right\}$ where $N(s,a)$ is the number of times a transition starting from (s,a) has been observed in H_{t-1}.

We consider a set of operators and constants that provides a good compromise between high expressiveness and low cardinality of $\mathbb{F}$. The set of binary operators $\mathbb{B}$ includes the four elementary mathematical operations and the $\min$ and $\max$ operators: $\mathbb{B} = \{+,-,\times,\div,\min,\max\}$. The set of unary operators $\mathbb{U}$ contains the square root, the logarithm and the absolute value: $\mathbb{U} = \left\{\sqrt{\cdot}, \ln(\cdot), |\cdot|\right\}$. The set of constants is: $\mathbb{C} = \{1,2,3,5,7\}$.

In the following, we denote by π^F the E/E strategy induced by formula F:

$$\pi^F(H_{t-1}, s_t) \in \arg\max_{a\in\mathcal{A}} F\left(\hat{\rho}(s_t,a), N(s_t,a), \hat{Q}(s_t,a), \hat{V}(s_t), t, \gamma^t\right)$$

We denote by $|F|$ the description length of the formula F, i.e. the total number of operators, constants and variables occurring in F. Let K be a maximal formula length. We denote by $\mathbb{F}^K$ the set of formulas whose length is not greater than K. This defines our so-called set of small formulas.

2.4 Finding a High-performance Formula-based E/E Strategy for a given Class of MDPs

We look for a formula F^* whose corresponding E/E strategy is specifically efficient for the subclass of MDPs implicitly defined by the probability distribution $p_{\mathcal{M}}(\cdot)$. We first describe a procedure for accelerating the search in the space $\mathbb{F}^K$ by eliminating equivalent formulas in Section 2.4.1. We then describe our optimization scheme for finding a high-performance E/E strategy in Section 2.4.2.

[2]If a pair (s,a) has not been visited, we consider the following default values: $\hat{\rho}(s,a) = 0$, $\hat{P}(s,a,s) = 1$ and $\hat{P}(s,a,s') = 0, \forall s' \neq s$.

2.4.1 Reducing $\mathbb{F}^K$

Notice first that several formulas $\mathbb{F}^K$ can lead to the same policy. All formulas that rank all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ in the same order define the same policy. We partition the set $\mathbb{F}^K$ into equivalence classes, two formulas being equivalent if and only if they lead to the same policy. For each equivalence class, we then consider one member of minimal length, and we gather all those minimal members into a set $\bar{\mathbb{F}}^K$.

Computing the set $\bar{\mathbb{F}}^K$ is not trivial: given a formula, equivalent formulas can be obtained through commutativity, associativity, operator-specific rules and through any increasing transformation. We thus propose to approximately discriminate between formulas by comparing how they rank (in terms of values returned by the formula) a set of d random samples of the variables $\hat{\rho}(\cdot,\cdot), N(\cdot,\cdot), \hat{Q}(\cdot,\cdot), \hat{V}(\cdot), t, \gamma^t$. More formally, the procedure is the following:

• we first build $\mathbb{F}^K$, the space of all formulas such that $|F| \leq K$;

• for $i = 1 \ldots d$, we uniformly draw (within their respective domains) some random realizations of the variables $\hat{\rho}(\cdot,\cdot)$, $N(\cdot,\cdot)$, $\hat{Q}(\cdot,\cdot)$, $\hat{V}(\cdot)$, t, γ^t that we concatenate into a vector Θ_i;

• we cluster all formulas from $\mathbb{F}^K$ according to the following rule: two formulas F and F' belong to the same cluster if and only if they rank all the Θ_i points in the same order, i.e.: $\forall i, j \in \{1, \ldots, d\}, i \neq j, F(\Theta_i) \geq F(\Theta_j) \iff F'(\Theta_i) \geq F'(\Theta_j)$. Formulas leading to invalid index functions (caused for instance by division by zero or logarithm of negative values) are discarded;

• among each cluster, we select one formula of minimal length;

• we gather all the selected minimal length formulas into an approximated reduced set of formulas $\tilde{\mathbb{F}}^K$.

In the following, we denote by N the cardinality of the approximate set of formulas $\tilde{\mathbb{F}}^K = \{F_1, \ldots, F_N\}$.

2.4.2 Finding a High-performance Formula

A naive approach for determining a high-performance formula $F^* \in \tilde{\mathbb{F}}^K$ would be to perform Monte-Carlo simulations for all candidate formulas in $\tilde{\mathbb{F}}^K$. Such an approach could reveal itself to be time-inefficient in case of spaces $\tilde{\mathbb{F}}^K$ of large cardinality.

We propose instead to formalize the problem of finding a high-performance formula-based E/E strategy in $\tilde{\mathbb{F}}^K$ as a $N-$armed bandit problem. To each formula $F_n \in \tilde{\mathbb{F}}^K$ ($n \in \{1, \ldots, N\}$), we associate an arm. Pulling the arm n consists first in randomly drawing a MDP M according to $p_{\mathcal{M}}(\cdot)$ and an initial state s_0 for this MDP according to $p_{M,0}(\cdot)$. Afterwards, an episode starting from this initial state is generated with the E/E strategy π^{F_n} until a truncated time horizon T. This leads to a reward associated to arm n whose value is the discounted return $\mathcal{R}^{\pi}_{M}(s_0)$ observed during the episode. The purpose

of multi-armed bandit algorithms is here to process the sequence of such observed rewards to select in a smart way the next arm to be played so that when the budget of pulls has been exhausted, one (or several) high-quality formula(s) can be identified.

Multi-armed bandit problems have been vastly studied, and several algorithms have been proposed, such as for instance all UCB-type algorithms (Auer, Cesa-Bianchi, and Fischer, 2002; Audibert, Munos, and Szepesvári, 2007). New approaches have also recently been proposed for identifying automatically empirically efficient algorithms for playing multi-armed bandit problems (Maes, Wehenkel, and Ernst, 2011).

2.5 Experimental Results

In this section, we empirically analyze our approach on a specific class of random MDPs defined hereafter.

Random MDPs. MDPs generated by our prior $p_{\mathcal{M}}(\cdot)$ have $n_{\mathcal{S}} = 20$ states and $n_{\mathcal{A}} = 5$ actions. When drawing a MDP according to this prior, the following procedure is called for generating $p_{M,f}(\cdot)$ and $\rho_M(\cdot,\cdot,\cdot)$. For every state-action pair (s,a) : (i) it randomly selects 10% of the states to form a set of successor states $Succ(s,a) \subset \mathcal{S}$ (ii) it sets $p_{M,f}(s'|s,a) = 0$ for each $s' \in \mathcal{S} \setminus Succ(s,a)$ (iii) for each $s' \in Succ(s,a)$, it draws a number $N(s')$ at random in $[0,1]$ and sets $p_{M,f}(s'|s,a) = \frac{N(s')}{\sum_{s'' \in Succ(s,a)} N(s'')}$ (iv) for each $s' \in Succ(s,a)$, it sets $\rho_M(s,a,s')$ equal to a number chosen at random in $[0,1]$ with a 0.1 probability and to zero otherwise. The distribution $p_{M,0}(\cdot)$ of initial states is chosen uniform over $\mathcal{S}$. The value of γ is equal to 0.995.

Learning protocol. In our experiments, we consider a maximal formula length of $K = 5$ and use $d = 1000$ samples to discriminate between formulas, which leads to a total number of candidate E/E strategies $N = 3834$. For solving the multi-armed bandit problem described in Section 2.4.2, we use an Upper Confidence Bound (UCB) algorithm (Auer, Cesa-Bianchi, and Fischer, 2002). The total budget allocated to the search of a high-performance policy is set to $T_b = 10^6$. We use a truncated optimization horizon $T = \log_\gamma((1-\gamma)\delta)$ for estimating the stochastic discounted return of an E/E strategy where $\delta = 0.001$ is the chosen precision (which is also used as stopping condition in the off-line value iteration algorithm for computing $\hat{Q}$ and $\hat{V}$). At the end of the T_b plays, the five E/E strategies that have the highest empirical return mean are returned.

Baselines. Our first baseline, the OPTIMAL strategy, consists in using for each test MDP, a corresponding optimal policy. The next

Baselines		Learned strategies	
Name	J^π	Formula	J^π
OPTIMAL	65.3	$(N(s,a) \times \hat{Q}(s,a)) - N(s,a)$	30.3
RANDOM	10.1	$\max(1, (N(s,a) \times \hat{Q}(s,a)))$	22.6
GREEDY	20.0	$\hat{Q}(s,a)$ (= GREEDY)	20.0
ϵ-GREEDY($\epsilon = 0$)	20.0	$\min(\gamma^t, (\hat{Q}(s,a) - \hat{V}(s)))$	19.4
R-MAX ($m = 1$)	27.7	$\min(\hat{\rho}(s,a), (\hat{Q}(s,a) - \hat{V}(s)))$	19.4

TABLE 2.1: Performance of the top-5 learned strategies with respect to baseline strategies.

baselines, the RANDOM and GREEDY strategies perform pure exploration and pure exploitation, respectively. The GREEDY strategy is equivalent to an index-based E/E strategy with formula $\hat{Q}(s,a)$. The last two baselines are classical E/E strategies whose parameters have been tuned so as to give the best performances on MDPs drawn from $p_{\mathcal{M}}(\cdot)$: ϵ-GREEDY and R-MAX. For ϵ-GREEDY, the best value we found was $\epsilon = 0$ in which case it behaves as the GREEDY strategy. This confirms that hazardous exploration is particularly harmful in the context of single trajectory RL with discounted return. Consistently with this result, we observed that R-MAX works at its best when it performs the least exploration ($m = 1$).

Results. Table 2.1 reports the mean empirical returns obtained by the E/E strategies on a set of 2000 test MDPs drawn from $p_{\mathcal{M}}(\cdot)$. Note that these MDPs are different from those used during learning and tuning. As we can see, the best E/E strategy that has been learned performs better than all baselines (except the OPTIMAL), including the state-of-the-art approach R-MAX.

We may wonder to what extent the E/E strategies found by our learning procedure would perform well on MDPs which are not generated by $p_{\mathcal{M}}(\cdot)$. As a preliminary step to answer this question, we have evaluated the mean return of our policies on sets of 2000 MDPs drawn from slightly different distributions as the one used for learning: we changed the number of states $n_{\mathcal{S}}$ to different values in $\{10, 20, \ldots, 50\}$. The results are reported in Figure 2.1. We observe that, except in the case $n_{\mathcal{S}} = 10$, our best E/E strategy still performs better than the R-MAX and the ϵ-GREEDY strategies tuned on the original distribution $p_{\mathcal{M}}(\cdot)$ that generates MDPs with 20 states. We also observe that for larger values of $n_{\mathcal{S}}$, the performances of R-MAX become very close to those of GREEDY, whereas the performances of our best E/E strategy remain clearly above. Investigating why this formula performs well is left for future work, but we notice that it is analog to the formula $t_k(r_k - C)$ that was automatically discovered as being well-performing in the context of multi-armed bandit problems (Maes, Wehenkel, and Ernst, 2011).

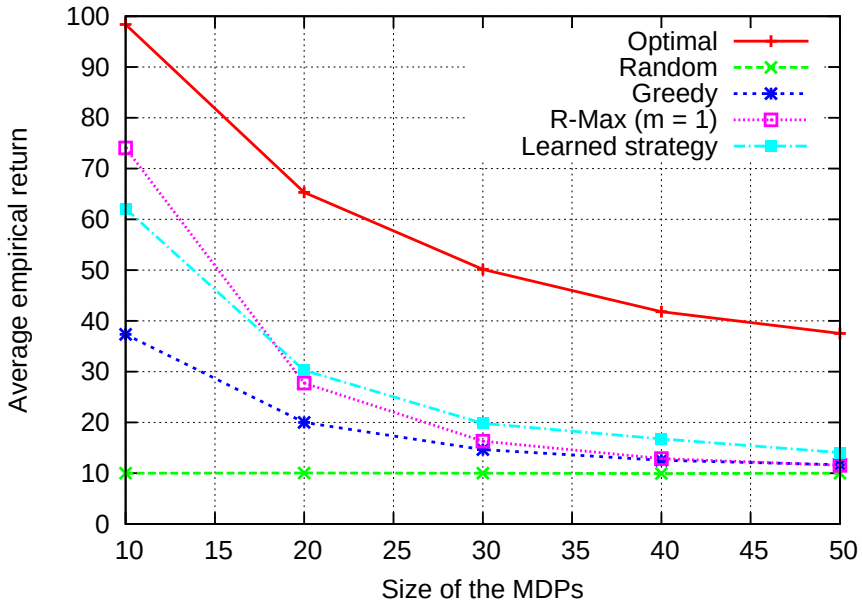


FIGURE 2.1: Performances of the learned and the baseline strategies for different distributions of MDPs that differ by the size of the MDPs belonging to their support.

2.6 Conclusions

In this paper, we have proposed an approach for learning E/E strategies for MDPs when the MDP to be controlled is supposed to be drawn from a known probability distribution $p_{\mathcal{M}}(\cdot)$. The strategies are learned from a set of training MDPs (drawn from $p_{\mathcal{M}}(\cdot)$) whose size depends on the computational budget allocated to the learning phase. Our results show that the learned strategies perform very well on test problems generated from the same distribution. In particular, they outperform on these problems R-MAX and ϵ-GREEDY policies. Interestingly, the strategies also generalize well to MDPs that do not belong to the support of $p_{\mathcal{M}}(\cdot)$. This is demonstrated by the good results obtained on MDPs having a larger number of states than those belonging to $p_{\mathcal{M}}(\cdot)$'s support.

These encouraging results suggest several future research direction. First, it would be interesting to better study the generalization performances of our approach either theoretically or empirically. Second, we believe that our approach could still be improved by considering richer sets of formulas w.r.t. the length of the formulas and the number of variables extracted from the history. Finally, it would be worth investigating ways to improve the optimization procedure upon which our learning approach is based so as to be able to deal with spaces of candidate E/E strategies that are so large that even running once every strategy on a single training problem would be impossible.

Chapter 3

Bayes Adaptive Reinforcement Learning versus On-line Prior-based Policy Search

This paper addresses the problem of decision making in unknown finite Markov Decision Processes (MDPs). The uncertainty about the MDPs is modelled, using a prior distribution over a set of candidate MDPs. The performance criterion is the expected sum of discounted rewards collected over an infinite length trajectory. Time constraints are defined as follows: (i) an off-line phase with a given time budget, which can be used to exploit the prior distribution and (ii) at each time step of the on-line phase, decisions have to be computed within a given time budget. In this setting, two decision-making strategies are compared. Firstly, OPPS, which is a recently proposed meta-learning scheme that mainly exploits the off-line phase to perform the policy search, as well as BAMCP—that is a state-of-the-art model-based Bayesian reinforcement learning algorithm, which mainly exploits the on-line time budget. These approaches are empirically compared in a real Bayesian setting, with their performances computed over a large set of problems. As far as this particular area of study is concerned, it is the first time that this is done in the Reinforcement Learning literature. Several settings are considered by varying the prior distribution and the distribution from which test problems are drawn. The main finding of these experiments is that there may be a significant benefit of having an off-line prior-based optimization phase, in the case of informative and accurate priors, especially when on-line time constraints are tight.

The work presented in this chapter has been published in the *Proceedings of the Belgian-Dutch Conference on Machine Learning (BENELEARN 2014)*. It also has been translated in french under the name "Apprentissage par renforcement bayésien versus recherche directe de politique hors-ligne en utilisant une distribution a priori: comparaison empirique", and published in the *Proceedings des Journée Francophones de Planification, Décision et Apprentissage (JFPDA 2014)* (Castronovo, Fonteneau, and Ernst, 2014).

3.1 Introduction

Optimally interacting with an unknown Markov Decision Process (MDP) remains a challenging Reinforcement Learning (RL) problem (Buşoniu et al., 2010). At the heart of this challenge lies the so-called Exploration/Exploitation (E/E) dilemma: on one hand, the agent needs to collect relevant data by exploring the environment, at the cost of taking bad decisions in the short term, while exploiting its current knowledge, facing the risk to take sub-optimal actions in the long term.

In the last fifteen years, Bayesian RL (Dearden, Friedman, and Andre, 1999; Strens, 2000) developed an interesting method to deal with the fact that the actual MDP is unknown. It assumes a prior distribution over a set of candidate MDPs from which the actual MDP is likely to be drawn. When interacting with the actual MDP, a posterior distribution is maintained, given the prior and the transitions observed so far. Such a posterior is used at each time-step to compute near-optimal Bayesian decisions, as a strategy to deal with the E/E dilemma. Model-based Bayesian RL methods maintain a posterior distribution over transition models (Ross and Pineau, 2008; Poupart, 2008; Asmuth et al., 2009; Hennig, Stern, and Graepel, 2009; Milani Fard and Pineau, 2010; Ross et al., 2011). On the other hand, the model-free Bayesian RL methods do not explicitly maintain a posterior over transition models, but rather value functions from which a decision can be extracted (see e.g. (Dearden, Friedman, and Russell, 1998; Engel, Mannor, and Meir, 2003; Engel, Mannor, and Meir, 2005; Engel, Szabo, and Volkinshtein, 2005; Ghavamzadeh and Engel, 2006; Ghavamzadeh and Engel, 2007).

Recently, Guez et al. (Guez, Silver, and Dayan, 2012) have introduced the BAMCP algorithm (for Bayes-adaptive Monte Carlo planning), a model-based Bayesian RL approach, which combines the principle of the UCT—Upper Confidence Trees—with sparse sampling methods, and obtained state-of-the-art performances. At the same time, Castronovo et al. (Castronovo et al., 2012) proposed an algorithm that exploits a prior distribution, in an off-line phase, by solving a policy search problem in a wide space of candidate, index-based E/E strategies, and by applying the obtained strategy on the actual MDP afterwards. The purpose of this paper is to empirically compare those two approaches in a “real Bayesian setting". In order to achieve this, several MDP distributions are considered, which can either be used as a prior distribution or as a test distribution, from which test problems are drawn. Several possible configurations in terms of prior/test distribution association are also considered, in order to observe the effect of the “flatness" of the prior distributions or their “accuracy" on the performances of the algorithms. Moreover, in order to be objective, comparisons will take into account the minimal

computation time required to run each of these algorithms. Our experiments mainly show that exploiting a prior distribution in an off-line phase makes sense in the context of informative and accurate priors, especially for problems where on-line time constraints are tight.

The paper is organized in the following manner: Section 3.2 formalizes the problem addressed in this paper. Section 3.3 presents the experimental protocol and the empirical results. Section 3.4 discusses the obtained results, and finally Section 3.5 concludes the paper.

3.2 Problem Statement

The goal of this paper is to compare two Reinforcement Learning (RL) strategies, in the presence of a prior distribution. First we describe the RL setting in Section 3.2.1. Then the prior distribution assumption is formalized in Section 3.2.2, and the basics of the BAMCP and OPPS approaches are briefly described. Section 3.2.3 formalizes the computational time constraints that these algorithms must satisfy, and Section 3.2.4 explains the specificities of our empirical evaluation.

3.2.1 Reinforcement Learning

Let $M = (X, U, f(\cdot), \rho_M, \rho_{M,0}(\cdot), \gamma)$ be a given unknown MDP, where $X = \{x^{(1)}, \ldots, x^{(n_X)}\}$ denotes its finite state space and $U = \{u^{(1)}, \ldots, u^{(n_U)}\}$ its finite action space. When the MDP is in state x_t at time t and action u_t is selected, the agent moves instantaneously to a next state x_{t+1} with a probability of $P(x_{t+1}|x_t, u_t) = f(x_t, u_t, x_{t+1})$. An instantaneous deterministic, bounded reward $r_t = \rho(x_t, u_t, x_{t+1}) \in [R_{\min}, R_{\max}]$ is observed simultaneously. In this paper, the reward function ρ is assumed to be fully known, which is often true in practice.

Let $H_t = (x_0, u_0, r_0, x_1, \cdots, x_{t-1}, u_{t-1}, r_{t-1}, x_t)$ denote the history observed until time t. An E/E strategy is a stochastic policy h that, given the current state x_t, returns an action $u_t \sim h(H_t)$. Given a probability distribution over initial states $\rho_{M,0}(\cdot)$, the expected return of a given E/E strategy h with respect to the MDP M can be defined as follows:

$$J_M^h = \mathop{\mathbb{E}}_{x_0 \sim \rho_{M,0}(\cdot)}[\mathcal{R}_M^h(x_0)],$$

where $\mathcal{R}_M^h(x_0)$ is the stochastic discounted sum of rewards received when applying the E/E strategy h, starting from an initial state x_0, defined as indicated below:

$$\mathcal{R}_M^h(x_0) = \sum_{t=0}^{+\infty} \gamma^t r_t,$$

where the discount factor γ belongs to $[0, 1)$. Within this setting, reinforcement learning amounts in finding a policy h^* which leads to

the maximization of J_M^h:

$$h^* \in \arg\max_h \quad J_M^h \ .$$

3.2.2 Prior Distribution over a Set of Candidate Models

In the context where the actual MDP is initially unknown, the Bayesian RL techniques propose to model the uncertainty about the actual model, using a probability distribution. This amounts to assuming that the actual MDP to be played is drawn from a distribution $p^0_{\mathcal{M}}(\cdot)$. In the so-called model-based Bayesian RL setting, this prior distribution is assumed to be known. In this paper, it is assumed that there is access to a prior distribution $p^0_{\mathcal{M}}(\cdot)$ over a set of MDPs $\mathcal{M}$. It is further assumed that:

- One can easily draw MDPs models from $p^0_{\mathcal{M}}(\cdot)$;
- One can easily compute the posterior distribution from $p^0_{\mathcal{M}}(\cdot)$ given the observation of an history H_t and the prior distribution.

Using these assumptions, the goal is to determine an E/E strategy h which leads to the maximization of the expected return over the set of transition models $\mathcal{M}$:

$$h^* \in \arg\max_h \mathop{\mathbb{E}}_{M \sim p^0_{\mathcal{M}}(\cdot)} \left[J_M^h \right] \ .$$

In this paper, two algorithms that can take advantage of such a prior are compared; these are the BAMCP and OPPS algorithms.

The BAMCP Algorithm

The BAMCP (Bayes-adaptive Monte Carlo planning) algorithm is a state-of-the-art performance Bayesian RL algorithm, originally proposed in (Guez, Silver, and Dayan, 2012). The principle of this algorithm is to adapt the UCT (Upper Confidence bounds applied to Trees, see (Kocsis and Szepesvári, 2006) principle for planning in a Bayes-adaptive MDP, also called the belief-augmented MDP, which is a MDP obtained when considering augmented states made of the concatenation of the actual state and the posterior. The BAMCP algorithm is made computationally tractable by using a sparse sampling strategy, which avoids sampling a model from the posterior distribution at every node of the planification tree. In practice, given a prior $p^0_{\mathcal{M}}(.)$ and a history H_t, the BAMCP algorithm computes a policy h_K^{BAMCP} based on the building of a planification tree with exactly K nodes, from which a decision is outputted:

$$u_t \sim h_K^{BAMCP}\left(H_t, p^0_{\mathcal{M}}(.)\right) \ .$$

Note that, as the number of node expansions K increases to infinity, the decision computed by the BAMCP algorithm converges towards Bayesian optimality.

The OPPS Algorithm

The Off-line, Prior-based Policy Search (OPPS) algorithm was originally introduced in (Castronovo et al., 2012). The OPPS approach works as follows: (i) a set of candidate E/E strategies $\mathcal{S}$ is built, and (ii) a policy search scheme is run over the set of strategies. The strategy space is obtained by considering index-based strategies, where the index is generated using small formulas, combining the standard mathematical operators with standard RL features (i.e., value functions). The search of an optimal E/E strategy is formalized as a multi-armed bandit problem, with a number of arms being equal to the number of candidate E/E strategies. Pulling an arm amounts to draw a MDP from the prior, and to proceed with one single run of the candidate E/E corresponding to that arm. Formally, the OPPS algorithm computes — during the off-line phase — a policy $h_{\mathcal{S}}^{OPPS}$ from which decisions are extracted on-line, given the prior $p_{\mathcal{M}}^0(.)$ and the history H_t:

$$u_t \sim h_{\mathcal{S}}^{OPPS}(H_t, p_{\mathcal{M}}^0(.))$$

where

$$h_{\mathcal{S}}^{OPPS} \in \arg\max_{s \in \mathcal{S}} \mathop{\mathbb{E}}_{M \sim p_{\mathcal{M}}^0(\cdot)} [J_M^s] \ .$$

In this paper, the set of variables from which formulas are built is slightly different than the one used in (Castronovo et al., 2012). Such a set is fully described in Appendix 3.5.1.

3.2.3 Time Constraints

Bayesian RL has acquired the reputation of being computationally intensive, mainly because of the incorporation of the posterior updates in the planification phase. In this paper, we propose to explicitly formalize the computational time budget allocated at every phase of the use of the algorithms. Thus, two types of time constraints are considered:

- an “off-line" time period B_{-1}, corresponding to a phase when the prior distribution is available to the agent, but the actual MDP is not yet available for interaction;
- a sequence of “on-line" time periods is considered $B_0, B_1 \ldots$, where, for all $t \in \mathbb{N}$, B_t corresponds to the time period available to compute a decision $u_t \in U$ given the prior $p_{\mathcal{M}}^0(.)$ and the history H_t observed so far.

3.2.4 Bayesian Empirical Evaluation

In this paper, we propose a real Bayesian empirical evaluation, in the sense that we compare the algorithms on a large set of problems drawn according to a test probability distribution. Such a distribution can be similar ("accurate") or different ("inaccurate") from the prior. Formally, for each experiment, a prior distribution is considered $p^0_{\mathcal{M}}(\cdot)$, which is given to the algorithms as an input, and a test distribution $p_{\mathcal{M}}(\cdot)$, which is used to draw test problems, on which each algorithm is evaluated. As far as this area of study is concerned, this is the first time that the Bayesian RL algorithms are compared on average over a large set of problems, rather than on standard benchmarks.

3.3 Experiments

Each experiment is characterized by the following:

- A prior distribution $p^0_{\mathcal{M}}(\cdot)$,
- A test distribution $p_{\mathcal{M}}(\cdot)$,
- An off-line time budget B_{-1},
- On-line time budgets $B_0, B_1, \cdots$ for computing decisions applied on the actual MDP.

The goal of those experiments is to identify the influence of the above mentioned elements on the performance of the algorithms, and, consequently, to identify the domain of excellence of each algorithm.

Subsection 3.3.1 describes the experimental protocol used to compare the algorithms described in Section 3.2.2. Subsection 3.3.2 defines accurately the MDP distributions considered in the experiments presented in Subsection 3.3.3.

3.3.1 The Experimental Protocol

For each algorithm:

- a pool of $10,000$ MDPs is drawn from $p_{\mathcal{M}}(.)$;
- one single run of the algorithm is simulated on each MDP of the pool;
- its empirical expected average of discounted returns is computed.

Trajectories are truncated after T steps, where T is defined as follows:

$$T = \left\lceil \frac{\frac{\epsilon \times (1-\gamma)}{R_{max}}}{\log \gamma} \right\rceil \text{ with } \epsilon = 0.001.$$

The mean μ is measured and the standard deviation σ of the set of observed returns. This data allows us to compute the 95% confidence interval of $J_{p_{\mathcal{M}}(\cdot)}^{h(p_{\mathcal{M}}^0(\cdot))}$:

$$J_{p_{\mathcal{M}}(\cdot)}^{h(p_{\mathcal{M}}^0(\cdot))} \in \left[\mu - \frac{2\sigma}{\sqrt{10,000}}; \mu + \frac{2\sigma}{\sqrt{10,000}}\right] \text{ with probability at least } 95\%.$$

Since each MDP distribution described below can be used, either as a prior distribution $p_{\mathcal{M}}^0(\cdot)$ or as a test distribution $p_{\mathcal{M}}(\cdot)$, the process is repeated for each possible combination.

3.3.2 MDP Distributions

The MDP distributions introduced in this paper are inspired from the well-known five-state chain MDP (Strens, 2000). For all the MDP distributions considered in this paper, the set of candidate MDPs shares the same state space X, action space U, reward function ρ_M, initial state distribution $\rho_{M,0}(\cdot)$ and discount factor γ. In our experiments, $X = \{1, 2, 3, 4, 5\}$, $U = \{1, 2, 3\}$, $\gamma = 0.95$, $x_0 = 1$ with probability 1 and the reward function ρ_M is defined as follows:

$$\begin{aligned} \forall (x,u) \in X \times U, \rho_M(x,u,1) &= 2.0 \\ \forall (x,u) \in X \times U, \rho_M(x,u,5) &= 10.0 \\ \forall (x,u) \in X \times U, y \in \{2,3,4\}, \rho_M(x,u,y) &= 0.0. \end{aligned}$$

In this context, a MDP is entirely specified by its transition matrix. Therefore, the probability distribution over sets of candidate transition matrices is defined, using the Flat Dirichlet Multinomial (FDM) distributions, which are widely used in the Bayesian RL, mostly because their Bayes update is straightforward. One independent Dirichlet distribution per state-action pair (x, u) is assumed, which leads to a density d_{FDM}:

$$d_{FDM}(\mu; \boldsymbol{\theta}) = \prod_{x,u} D(\mu_{x,u}; \theta_{x,u})$$

where $D(\cdot;\cdot)$ are independent Dirichlet distributions. The parameter $\boldsymbol{\theta}$ gathers all the counters of observed transitions $\theta_{x,u}^t$ until time t, including $\theta_{x,u}^0$ which represents a priori observations.

The density of $p_{\mathcal{M}}(\cdot)$ is therefore defined as:

$$d_{p_{\mathcal{M}}(\cdot)}(\mu, \boldsymbol{\theta}) = d_{FDM}(\mu; \boldsymbol{\theta})$$

Consequently, a MDP distribution is parameterised by $\boldsymbol{\theta}$, and will be denoted by $p^{\boldsymbol{\theta}}(\cdot)$. In the following section, we introduce four MDP distributions, the “Generalized Chain" distribution, the “Optimistic

Generalized Chain" distribution, the “Pessimistic Generalized Chain" distribution and the “Uniform" distribution.

Generalized Chain Distribution

This MDP distribution is a generalisation of the well-known Chain MDP. For each action, two different outcomes are possible:

- The agent moves from state x to state $x+1$ (or remains in state x when $x=5$) or;
- The agent “slips" and goes back to the initial state.

The probabilities associated with those outcomes are drawn uniformly. Formally, the $\boldsymbol{\theta}^{GC}$ parameter characterising the corresponding $p^{\boldsymbol{\theta}^{GC}}(\cdot)$ distribution is defined as follows:

$$\begin{aligned}
\forall u \in U : \theta^{GC}_{1,u} &= [1,1,0,0,0] \\
\forall u \in U : \theta^{GC}_{2,u} &= [1,0,1,0,0] \\
\forall u \in U : \theta^{GC}_{3,u} &= [1,0,0,1,0] \\
\forall u \in U : \theta^{GC}_{4,u} &= [1,0,0,0,1] \\
\forall u \in U : \theta^{GC}_{5,u} &= [1,0,0,0,1]
\end{aligned}$$

Optimistic Generalized Chain Distribution

This distribution is an alternative to the Generalized Chain MDPs, where higher weights are put on transitions, allowing the agent to move forward in the chain. Formally, the $\boldsymbol{\theta}^{OGC}$ parameter characterising the corresponding $p^{\boldsymbol{\theta}^{OGC}}(\cdot)$ distribution is defined as follows:

$$\begin{aligned}
\forall u \in U : \theta^{OGC}_{1,u} &= [1,5,0,0,0] \\
\forall u \in U : \theta^{OGC}_{2,u} &= [1,0,5,0,0] \\
\forall u \in U : \theta^{OGC}_{3,u} &= [1,0,0,5,0] \\
\forall u \in U : \theta^{OGC}_{4,u} &= [1,0,0,0,5] \\
\forall u \in U : \theta^{OGC}_{5,u} &= [1,0,0,0,5]
\end{aligned}$$

Pessimistic Generalized Chain Distribution

This distribution is an alternative to the Generalized Chain MDPs, where higher weights are put on transitions, moving the agent to the initial state. Formally, the $\boldsymbol{\theta}^{PGC}$ parameter characterising the

corresponding $p^{\boldsymbol{\theta}^{PGC}}(\cdot)$ distribution is defined as follows:

$$\begin{aligned}
\forall u \in U : \theta^{PGC}_{1,u} &= [5, 1, 0, 0, 0] \\
\forall u \in U : \theta^{PGC}_{2,u} &= [5, 0, 1, 0, 0] \\
\forall u \in U : \theta^{PGC}_{3,u} &= [5, 0, 0, 1, 0] \\
\forall u \in U : \theta^{PGC}_{4,u} &= [5, 0, 0, 0, 1] \\
\forall u \in U : \theta^{PGC}_{5,u} &= [5, 0, 0, 0, 1]
\end{aligned}$$

Uniform Distribution

All transition probabilities are drawn uniformly. Formally, the $\boldsymbol{\theta}^U$ parameter characterising the corresponding $p^{\boldsymbol{\theta}^U}(\cdot)$ distribution is defined as follows:

$$\forall x \in X, u \in U : \theta^{U}_{x,u} = [1, 1, 1, 1, 1]$$

Finally, note that unlike the original chain MDP, in which action 1 is optimal in any given state, the optimal behaviour in any MDP drawn according to one of these distributions is not defined a priori, as it changes from one MDP to another.

3.3.3 The Results of the Experiments

Several experiments are presented, where different prior distribution / test distribution combinations are considered.

Concerning OPPS, four different strategy spaces are considered. The set of variables, operators and constants has been fixed once and for all. The four strategy spaces differ only by the maximal length of the small formulas, which can be built from them. Those spaces were named $\mathbb{F}_n$, where n is the maximal length of the formulas of the corresponding strategy space. The implementation of OPPS used in these experiments differs from the one of (Castronovo et al., 2012) by the chosen set of variables. These variables are described in the Appendix 3.5.1.

Concerning BAMCP, the default parameters provided by Guez et al. in (Guez, Silver, and Dayan, 2012) were used. Several instances of BAMCP are built by varying the number of nodes, which are created at each time-step. This parameter has been denoted by K.

Our experiments are organized in four different parts, one for each possible test distribution, i.e. the distribution from which test problems are drawn. In each part, we present a table of experimental results, obtained when the prior and test distributions are identical, comparing the algorithms, in term of performances and minimal required off-line / on-line time budgets. In addition, a figure is joined, comparing the performances of the approaches for different prior distributions.

"Generalized Chain" Test Distribution

Agent	Offline time	Online time	Mean score
OPPS ($\mathbb{F}_3$)	$\sim$ 6h	$\sim$ 40ms	42.29 ± 0.45
OPPS ($\mathbb{F}_4$)	$\sim$ 6h	$\sim$ 42ms	41.89 ± 0.41
OPPS ($\mathbb{F}_5$)	$\sim$ 6h	$\sim$ 42ms	41.89 ± 0.41
BAMCP ($K = 1$)	$\sim$ 1ms	$\sim$ 7ms	31.71 ± 0.23
BAMCP ($K = 10$)	$\sim$ 1ms	$\sim$ 54ms	33.23 ± 0.26
BAMCP ($K = 25$)	$\sim$ 1ms	$\sim$ 136ms	33.26 ± 0.26
BAMCP ($K = 50$)	$\sim$ 1ms	$\sim$ 273ms	33.73 ± 0.26
BAMCP ($K = 100$)	$\sim$ 1ms	$\sim$ 549ms	33.99 ± 0.27
BAMCP ($K = 250$)	$\sim$ 1ms	$\sim$ 2s	34.02 ± 0.26
BAMCP ($K = 500$)	$\sim$ 1ms	$\sim$ 3s	34.27 ± 0.26

TABLE 3.1: Comparison with prior "Generalized Chain" on "Generalized Chain"

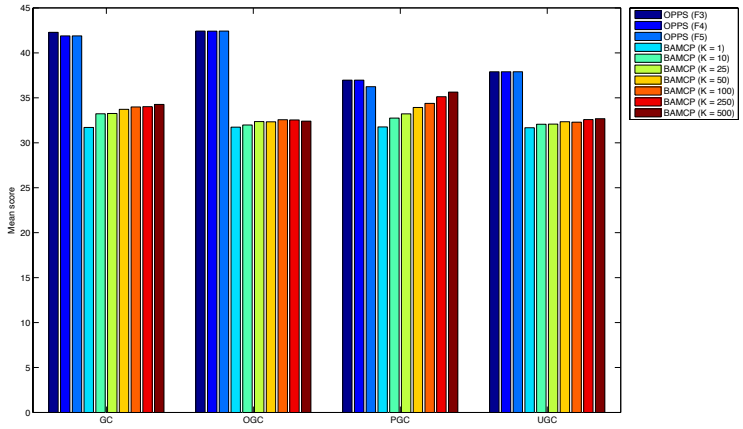


FIGURE 3.1: Comparison on "Generalized Chain" distribution

Table 3.1 shows that OPPS outperforms BAMCP in every single case, even for higher on-line time budgets. The choice of the prior has a significant impact on the performances of OPPS, as shown by Figure 3.1. The "Generalized Chain" and "Optimistic Generalized Chain" priors show similar performances for OPPS, while the "Uniform Generalized Chain" prior degrades them. On its side, BAMCP is steady except for the "Pessimistic Generalized Chain" prior, which has a positive effect on its performances, contrary to OPPS.

"Optimistic Generalized Chain" Test Distribution

Agent	Offline time	Online time	Mean score
OPPS ($\mathbb{F}_3$)	$\sim$ 6h	$\sim$ 44ms	110.48 ± 0.61
OPPS ($\mathbb{F}_4$)	$\sim$ 6h	$\sim$ 44ms	110.51 ± 0.61
OPPS ($\mathbb{F}_5$)	$\sim$ 6h	$\sim$ 45ms	110.48 ± 0.61
BAMCP ($K = 1$)	$\sim$ 1ms	$\sim$ 7ms	92.71 ± 0.58
BAMCP ($K = 10$)	$\sim$ 1ms	$\sim$ 56ms	93.97 ± 0.57
BAMCP ($K = 25$)	$\sim$ 1ms	$\sim$ 138ms	94.24 ± 0.58
BAMCP ($K = 50$)	$\sim$ 1ms	$\sim$ 284ms	94.31 ± 0.57
BAMCP ($K = 100$)	$\sim$ 1ms	$\sim$ 555ms	94.59 ± 0.57
BAMCP ($K = 250$)	$\sim$ 1ms	$\sim$ 2s	95.06 ± 0.57
BAMCP ($K = 500$)	$\sim$ 1ms	$\sim$ 3s	95.27 ± 0.58

TABLE 3.2: Comparison with prior "Optimistic Generalized Chain" on "Optimistic Generalized Chain"

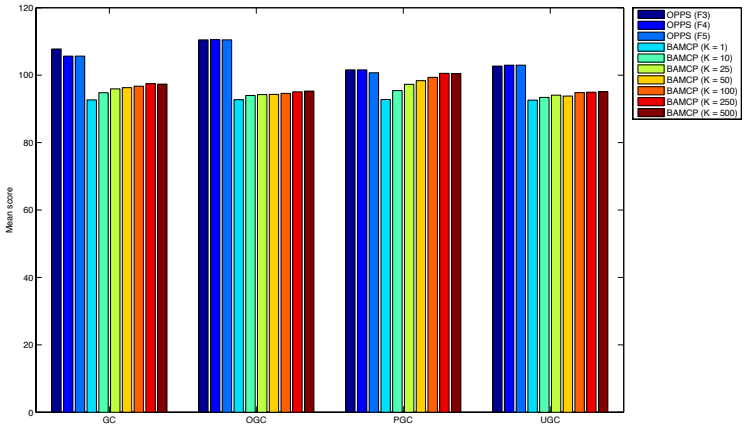


FIGURE 3.2: Comparison on "Optimistic Generalized Chain" distribution

Table 3.2 shows that OPPS clearly outperforms BAMCP, even for pretty high time budgets. However, in Figure 3.2, we can see that BAMCP becomes more competitive when using the "Pessimistic Generalized Chain" prior distribution. In this case, BAMCP is near OPPS performances.

“Pessimistic Generalized Chain" Test Distribution

Agent	Offline time	Online time	Mean score
OPPS ($\mathbb{F}_3$)	$\sim$ 5h	$\sim$ 37ms	35.89 ± 0.06
OPPS ($\mathbb{F}_4$)	$\sim$ 5h	$\sim$ 39ms	35.89 ± 0.06
OPPS ($\mathbb{F}_5$)	$\sim$ 5h	$\sim$ 38ms	35.83 ± 0.06
BAMCP ($K = 1$)	$\sim$ 1ms	$\sim$ 6ms	33.77 ± 0.07
BAMCP ($K = 10$)	$\sim$ 1ms	$\sim$ 54ms	33.97 ± 0.06
BAMCP ($K = 25$)	$\sim$ 1ms	$\sim$ 133ms	34.1 ± 0.06
BAMCP ($K = 50$)	$\sim$ 1ms	$\sim$ 265ms	34.21 ± 0.06
BAMCP ($K = 100$)	$\sim$ 1ms	$\sim$ 536ms	34.37 ± 0.06
BAMCP ($K = 250$)	$\sim$ 1ms	$\sim$ 2s	34.62 ± 0.06
BAMCP ($K = 500$)	$\sim$ 1ms	$\sim$ 3s	34.9 ± 0.06

TABLE 3.3: Comparison with prior “Pessimistic Generalized Chain" on “Pessimistic Generalized Chain"

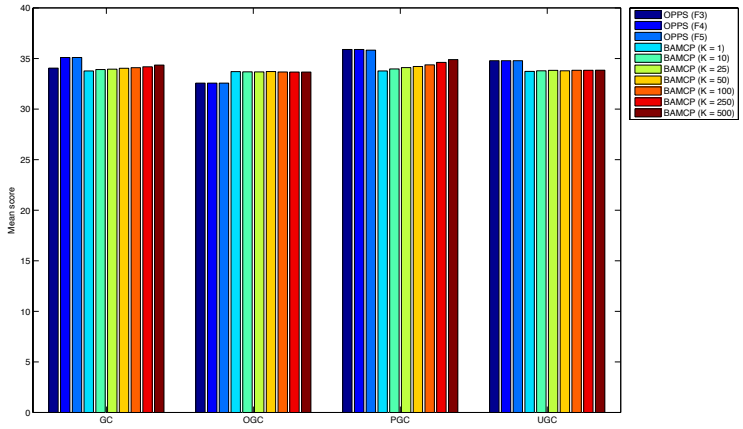


FIGURE 3.3: Comparison on “Pessimistic Generalized Chain" distribution

Table 3.3 shows that OPPS and BAMCP share similar performances, even if BAMCP stays behind. Nevertheless, BAMCP requires on-line time budgets that are eighty times higher than the one required by OPPS, in order to get a slightly lower score. As shown in Figure 3.3, this difference remains in all cases except for the “Optimistic Generalized Chain" case where BAMCP clearly outperforms OPPS.

"Uniform Generalized Chain" Test Distribution

Agent	Offline time	Online time	Mean score
OPPS ($\mathbb{F}_3$)	$\sim$ 8h	$\sim$ 52ms	57.37 ± 0.38
OPPS ($\mathbb{F}_4$)	$\sim$ 8h	$\sim$ 53ms	57.37 ± 0.38
OPPS ($\mathbb{F}_5$), UGC)	$\sim$ 8h	$\sim$ 51ms	57.37 ± 0.38
BAMCP ($K = 1$)	$\sim$ 1ms	$\sim$ 6ms	47.92 ± 0.29
BAMCP ($K = 10$)	$\sim$ 1ms	$\sim$ 52ms	48.81 ± 0.3
BAMCP ($K = 25$)	$\sim$ 1ms	$\sim$ 132ms	48.95 ± 0.3
BAMCP ($K = 50$)	$\sim$ 1ms	$\sim$ 256ms	49.3 ± 0.3
BAMCP ($K = 100$)	$\sim$ 1ms	$\sim$ 521ms	49.39 ± 0.31
BAMCP ($K = 250$)	$\sim$ 1ms	$\sim$ 2s	50.08 ± 0.31
BAMCP ($K = 500$)	$\sim$ 1ms	$\sim$ 3s	50.06 ± 0.31

TABLE 3.4: Comparison with prior "Uniform" on "Uniform"

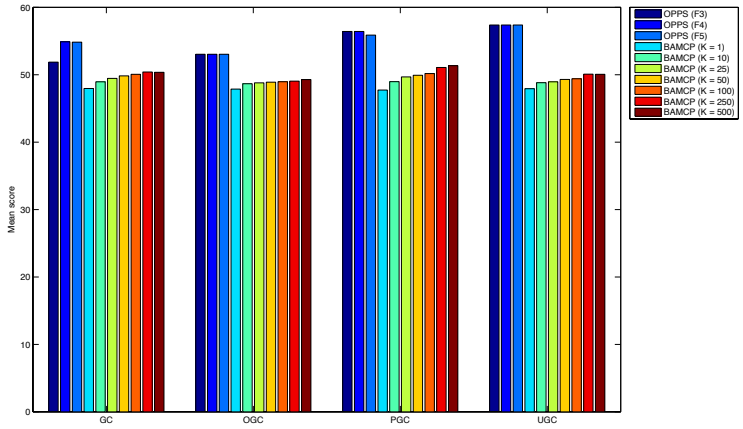


FIGURE 3.4: Comparison on "Uniform Generalized Chain" distribution

Both Table 3.4 and Figure 3.4 show a clear victory for OPPS for any prior distribution, even with pretty high on-line time budgets. We can also notice that OPPS is more efficient when using the correct prior distribution.

3.4 Discussion

As a general remark, it is observed that OPPS performs better than BAMCP, even for high on-line time budgets, at the cost of several hours of offline computation time. However, we can notice that BAMCP was a decent challenger in the case of “Pessimistic Generalized Chain" distribution.

Regarding the accuracy of the prior, it appears that using a prior distribution, which differs from the test problem distribution impacts the performances of OPPS in a negative manner, which is expected, since OPPS performs policy search, using the prior. This impact is strengthened in the case of a tight test distribution (“Generalized Chain", “Optimistic Generalized Chain" and “Pessimistic Generalized Chain"). Thanks to the posterior update, the performance of BAMCP seems less affected by a prior inaccuracy.

3.5 Conclusion

An extensive experimental comparison between two different Bayesian approaches was presented, exploiting either off-line or on-line time budgets, in order to interact efficiently with an unknown MDP. Our experiments suggest that: (i) exploiting a prior distribution in an off-line phase is never a bad idea, even for problems where on-line time constraints are loose, whereas (ii) when on-line time budget are less constrained, maintaining a posterior distribution definitely decreases the impact of an inaccurate prior on the performances of the agent.

Appendices

3.5.1 OPPS Settings

The OPPS implementation used in this paper differs from the one introduced in (Castronovo et al., 2012) by the set of variables used to build formulas. The set of variables considered in this paper is composed of three variables. Those three variables correspond to three Q-functions computed through value iteration, using three different models. Formally, given a set of transitions observed so-far, h, $N_h(x,u)$ the number of times a transition starting from the state-action pair (x,u) occurs in h, and $N_h(x,u,y)$ the number of times the transition (x,u,y) occurs in h, three transition functions are defined $f_{mean}, f_{uniform}, f_{self}$ as follows:

1. f_{mean} corresponds to the expectation of a Dirichlet posterior distribution computed from the current history and the chosen prior distribution. If θ_0 denotes the counters of the observed transitions of prior $p^0_{\mathcal{M}}(\Delta)$, f_{mean} is defined as follows:

$$\forall x,u,y : \theta^h_{x,u}(y) = \theta^0_{x,u}(y) + N_h(x,u,y)$$

   Formally, the mean transition model is defined as follows:

$$\forall x,u,y : f_{mean}(x,u,y) = \frac{\theta^h_{x,u}(y)}{\sum_{y'} \theta^h_{x,u}(y')}$$

2. $f_{uniform}$ corresponds to the expectation of a Dirichlet posterior distribution computed from the current history and a uniform Dirichlet prior distribution. Formally, the uniform transition model is defined as follows:

$$\forall x,u,y : f_{uniform}(x,u,y) = \frac{1 + N_h(x,u,y)}{|U| + N_h(x,u)}$$

3. f_{self} corresponds to the expectation of a Dirichlet posterior distribution computed from the current history and a counter initialization corresponding to a Dirac centred over a deterministic MDP where each state can only be reached from itself (for all actions). Formally, the self transition model is defined as follows:

$$\forall x,u : f_{self}(x,u,x) = \frac{1 + N_h(x,u,x)}{1 + N_h(x,u)}$$
$$\forall x,u,y \neq x : f_{self}(x,u,x) = \frac{N_h(x,u,x)}{1 + N_h(x,u)}.$$

Chapter 4

Benchmarking for Bayesian Reinforcement Learning

In the Bayesian Reinforcement Learning (BRL) setting, agents try to maximise the collected rewards while interacting with their environment while using some prior knowledge that is accessed beforehand. Many BRL algorithms have already been proposed, but even though a few toy examples exist in the literature, there are still no extensive or rigorous benchmarks to compare them. The paper addresses this problem, and provides a new BRL comparison methodology along with the corresponding open source library. In this methodology, a comparison criterion that measures the performance of algorithms on large sets of Markov Decision Processes (MDPs) drawn from some probability distributions is defined. In order to enable the comparison of non-anytime algorithms, our methodology also includes a detailed analysis of the computation time requirement of each algorithm. Our library is released with all source code and documentation: it includes three test problems, each of which has two different prior distributions, and seven state-of-the-art RL algorithms. Finally, our library is illustrated by comparing all the available algorithms and the results are discussed.

The work presented in this chapter has been published in the *PLoS ONE Journal (PLoS ONE 2016)* (Castronovo et al., 2016).

4.1 Introduction

Reinforcement Learning (RL) agents aim to maximise collected rewards by interacting over a certain period of time in initially unknown environments. Actions that yield the highest performance according to the current knowledge of the environment and those that maximise the gathering of new knowledge on the environment may not be the same. This is the dilemma known as Exploration/Exploitation (E/E). In such a context, using prior knowledge of the environment is extremely valuable, since it can help guide the decision-making process in order to reduce the time spent on exploration. Model-based Bayesian Reinforcement Learning (BRL) (Dearden, Friedman, and Andre, 1999; Strens, 2000) specifically targets RL problems for which such a prior knowledge is encoded in the form of a probability distribution (the "prior") over possible models of the environment. As the agent interacts with the actual model, this probability distribution is updated according to the Bayes rule into what is known as "posterior distribution". The BRL process may be divided into two learning phases: the offline learning phase refers to the phase when the prior knowledge is used to warm-up the agent for its future interactions with the real model. The online learning phase, on the other hand, refers to the actual interactions between the agent and the model. In many applications, interacting with the actual environment may be very costly (e.g. medical experiments). In such cases, the experiments made during the online learning phase are likely to be much more expensive than those performed during the offline learning phase.

In this paper, we investigate how the way BRL algorithms use the offline learning phase may impact online performances. To properly compare Bayesian algorithms, the first comprehensive BRL benchmarking protocol is designed, following the foundations of (Castronovo, Fonteneau, and Ernst, 2014). "Comprehensive BRL benchmark" refers to a tool which assesses the performance of BRL algorithms over a large set of problems that are actually drawn according to a prior distribution. In previous papers addressing BRL, authors usually validate their algorithm by testing it on a few test problems, defined by a small set of predefined MDPs. For instance, BAMCP (Guez, Silver, and Dayan, 2012), SBOSS (Castro and Precup, 2010), and BFS3 (Asmuth and Littman, 2011) are all validated on a fixed number of MDPs. In their validation process, the authors select a few BRL tasks, for which they choose one arbitrary transition function, which defines the corresponding MDP. Then, they define one prior distribution compliant with the transition function. This type of benchmarking is problematic in the sense that the authors actually know the hidden transition function of each test case. It also creates an implicit incentive to over-fit their approach to a few specific transition functions, which should be completely unknown before interacting with the model. In this paper, we compare BRL algorithms

in several different tasks. In each task, the real transition function is defined using a random distribution, instead of being arbitrarily fixed. Each algorithm is thus tested on an infinitely large number of MDPs, for *each* test case. To perform our experiments, we developed the *BBRL* library, whose objective is to also provide other researchers with our benchmarking tool.

This paper is organised as follows: Section 5.3 presents the problem statement. Section 4.3 formally defines the experimental protocol designed for this paper. Section 4.4 briefly presents the library. Section 4.5 shows a detailed application of our protocol, comparing several well-know BRL algorithms on three different benchmarks. Section 5.6 concludes the study.

4.2 Problem Statement

This section is dedicated to the formalisation of the different tools and concepts discussed in this paper.

4.2.1 Reinforcement Learning

Let $M = (X, U, f(\cdot), \rho_M, p_{M,0}(\cdot), \gamma)$ be a given unknown MDP, where $X = \{x^{(1)}, \ldots, x^{(n_X)}\}$ denotes its finite state space and $U = \{u^{(1)}, \ldots, u^{(n_U)}\}$ refers to its finite action space. When the MDP is in state x_t at time t and action u_t is selected, the agent moves instantaneously to a next state x_{t+1} with a probability of $P(x_{t+1}|x_t, u_t) = f(x_t, u_t, x_{t+1})$. An instantaneous deterministic, bounded reward $r_t = \rho_M(x_t, u_t, x_{t+1}) \in [R_{\min}, R_{\max}]$ is observed.

Let $h_t = (x_0, u_0, r_0, x_1, \cdots, x_{t-1}, u_{t-1}, r_{t-1}, x_t) \in H$ denote the history observed until time t. An E/E strategy is a stochastic policy π which, given the current state x_t, returns an action $u_t \sim \pi(h_t)$. Given a probability distribution over initial states $p_{M,0}(\cdot)$, the expected return of a given E/E strategy π with respect to the MDP M can be defined as follows:

$$J_M^\pi = \mathop{\mathbb{E}}_{x_0 \sim p_{M,0}(\cdot)}[\mathcal{R}_M^\pi(x_0)],$$

where $\mathcal{R}_M^\pi(x_0)$ is the stochastic sum of discounted rewards received when applying the policy π, starting from an initial state x_0:

$$\mathcal{R}_M^\pi(x_0) = \sum_{t=0}^{+\infty} \gamma^t \, r_t.$$

RL aims to learn the behaviour that maximises J_M^π, i.e. learning a policy π^* defined as follows:

$$\pi^* \in \arg\max_\pi \quad J_M^\pi.$$

4.2.2 Prior Knowledge

In this paper, the actual MDP is assumed to be initially unknown. Model-based Bayesian Reinforcement Learning (BRL) proposes to the model the uncertainty, using a probability distribution $p^0_{\mathcal{M}}(\cdot)$ over a set of candidate MDPs $\mathcal{M}$. Such a probability distribution is called a prior distribution and can be used to encode specific prior knowledge available before interaction. Given a prior distribution $p^0_{\mathcal{M}}(\cdot)$, the expected return of a given E/E strategy π is defined as:

$$\mathfrak{J}^{\pi}_{p^0_{\mathcal{M}}(\cdot)} = \mathop{\mathbb{E}}_{M \sim p^0_{\mathcal{M}}(\cdot)} \left[J^{\pi}_{M} \right],$$

In the BRL framework, the goal is to maximise $\mathfrak{J}^{\pi}_{p^0_{\mathcal{M}}(\cdot)}$, by finding π^*, which is called "Bayesian optimal policy" and defined as follows:

$$\pi^* \in \arg\max_{\pi} \quad \mathfrak{J}^{\pi}_{p^0_{\mathcal{M}}(\cdot)}.$$

4.2.3 Computation Time Characterisation

Most BRL algorithms rely on some properties which, given sufficient computation time, ensure that their agents will converge to an optimal behaviour. However, it is not clear to know beforehand whether an algorithm will satisfy fixed computation time constraints while providing good performances.

The parameterisation of the algorithms makes the selection even more complex. Most BRL algorithms depend on parameters (number of transitions simulated at each iteration, etc.) which, in some way, can affect the computation time. In addition, for one given algorithm and fixed parameters, the computation time often varies from one simulation to another. These features make it nearly impossible to compare BRL algorithms under strict computation time constraints. In this paper, to address this problem, algorithms are run with multiple choices of parameters, and we analyse their time performance a posteriori.

Furthermore, a distinction between the offline and online computation time is made. Offline computation time corresponds to the moment when the agent is able to exploit its prior knowledge, but cannot interact with the MDP yet. One can see it as the time given to take the first decision. In most algorithms concerned in this paper, this phase is generally used to initialise some data structure. On the other hand, online computation time corresponds to the time consumed by an algorithm for taking each decision.

There are many ways to characterise algorithms based on their computation time. One can compare them based on the average time needed per step or on the offline computation time alone. To remain flexible, for each run of each algorithm, we store its computation times $(B_i)_{-1 \leq i}$, with i indexing the time step, and B_{-1} the offline learning

time. Then a feature function $\phi((B_i)_{-1\leq i})$ is extracted from this data. This function is used as a metric to characterise and discriminate algorithms based on their time requirements.

In our protocol, which is detailed in the next section, two types of characterisation are used. For a set of experiments, algorithms are classified based on their offline computation time only, i.e. we use $\phi((B_i)_{-1\leq i}) = B_{-1}$. Afterwards, the constraint is defined as $\phi((B_i)_{-1\leq i}) \leq K$, $K > 0$ in case it is required to only compare the algorithms that have an offline computation time lower than K.

For another set of experiments, algorithms are separated according to their empirical average online computation time. In this case, $\phi((B_i)_{-1\leq i}) = \frac{1}{n}\sum_{0\leq i<n} B_i$. Algorithms can then be classified based on whether or not they respect the constraint $\phi((B_i)_{-1\leq i}) \leq K$, $K > 0$.

This formalisation could be used for any other computation time characterisation. For example, one could want to analyse algorithms based on the longest computation time of a trajectory, and define $\phi((B_i)_{-1\leq i}) = \max_{-1\leq i} B_i$.

4.3 A New Bayesian Reinforcement Learning Benchmark Protocol

4.3.1 A Comparison Criterion for BRL

In this paper, a real Bayesian evaluation is proposed, in the sense that the different algorithms are compared on a large set of problems drawn according to a test probability distribution. This is in contrast with the Bayesian literature (Guez, Silver, and Dayan, 2012; Castro and Precup, 2010; Asmuth and Littman, 2011), where authors pick a fixed number of MDPs on which they evaluate their algorithm.

Our criterion to compare algorithms is to measure their average rewards against a given random distribution of MDPs, using another distribution of MDPs as a prior knowledge. In our experimental protocol, an experiment is defined by a prior distribution $p^0_{\mathcal{M}}(\cdot)$ and a test distribution $p_{\mathcal{M}}(\cdot)$. Both are random distributions over the set of possible MDPs, not stochastic transition functions. To illustrate the difference, let us take an example. Let (x, u, x') be a transition. Given a transition function $f : X \times U \times X \to [0; 1]$, $f(x, u, x')$ is the probability of observing x' if we chose u in x. In this paper, this function f is assumed to be the only unknown part of the MDP that the agent faces. Given a certain test case, f corresponds to a unique MDP $M \in \mathcal{M}$. A Bayesian learning problem is then defined by a probability distribution over a set $\mathcal{M}$ of possible MDPs. We call it a test distribution, and denote it $p_{\mathcal{M}}(\cdot)$. Prior knowledge can then be encoded as another distribution over $\mathcal{M}$, and denoted $p^0_{\mathcal{M}}(\cdot)$. We call "accurate" a prior which is identical to the test distribution

($p^0_{\mathcal{M}}(\cdot) = p_{\mathcal{M}}(\cdot)$), and we call "inaccurate" a prior which is different ($p^0_{\mathcal{M}}(\cdot) \neq p_{\mathcal{M}}(\cdot)$).

In previous Bayesian literature, authors select a fixed number of MDPs $M_1, ..., M_n$, train and test their algorithm on them. Doing so does not guarantee any generalisation capabilities. To solve this problem, a protocol that allows rigorous comparison of BRL algorithms is designed. Training and test data are separated, and can even be generated from different distributions (in what we call the inaccurate case).

More precisely, our protocol can be described as follows: Each algorithm is first trained on the prior distribution. Then, their performances are evaluated by estimating the expectation of the discounted sum of rewards, when they are facing MDPs drawn from the test distribution. Let $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}}$ be this value:

$$\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}} = \mathop{\mathbb{E}}_{M \sim p_{\mathcal{M}}} \left[J^{\pi(p^0_{\mathcal{M}})}_M \right],$$

where $\pi(p^0_{\mathcal{M}})$ is the algorithm π trained offline on $p^0_{\mathcal{M}}$. In our Bayesian RL setting, we want to find the algorithm π^* which maximises $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}}$ for the $\langle p^0_{\mathcal{M}}, p_{\mathcal{M}} \rangle$ experiment:

$$\pi^* \in \arg\max_{\pi} \quad \mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}}.$$

In addition to the performance criterion, we also measure the empirical computation time. In practice, all problems are subject to time constraints. Hence, it is important to take this parameter into account when comparing different algorithms.

4.3.2 The Experimental Protocol

In practice, we can only sample a finite number of trajectories, and must rely on estimators to compare algorithms. In this section our experimental protocol is described, which is based on our comparison criterion for BRL and provides a detailed computation time analysis.

An experiment is defined by (i) a prior distribution $p^0_{\mathcal{M}}$ and (ii) a test distribution $p_{\mathcal{M}}$. Given these, an agent is evaluated π as follows:

1. Train π offline on $p^0_{\mathcal{M}}$.
2. Sample N MDPs from the test distribution $p_{\mathcal{M}}$.
3. For each sampled MDP M, compute estimate $\bar{J}^{\pi(p^0_{\mathcal{M}})}_M$ of $J^{\pi(p^0_{\mathcal{M}})}_M$.
4. Use these values to compute an estimate $\bar{\mathfrak{J}}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}}$.

To estimate $J^{\pi(p^0_{\mathcal{M}})}_M$, the expected return of agent π trained offline on $p^0_{\mathcal{M}}$, one trajectory is sampled on the MDP M, and the cumulated return is computed $\bar{J}^{\pi(p^0_{\mathcal{M}})}_{M_i} = \mathcal{R}^{\pi(p^0_{\mathcal{M}})}_M(x_0)$.

To estimate this return, each trajectory is truncated after T steps. Therefore, given an MDP M and its initial state x_0, we observe $\bar{\mathcal{R}}_M^{\pi(p^0_{\mathcal{M}})}(x_0)$, an approximation of $\mathcal{R}_M^{\pi(p^0_{\mathcal{M}})}(x_0)$:

$$\bar{\mathcal{R}}_M^{\pi(p^0_{\mathcal{M}})}(x_0) = \sum_{t=0}^{T} \gamma^t r_t.$$

If R_{max} denotes the maximal instantaneous reward an agent can receive when interacting with an MDP drawn from $p_{\mathcal{M}}$, then choosing T as guarantees the approximation error is bounded by $\epsilon > 0$:

$$T = \left\lfloor \frac{\log(\epsilon \times \frac{(1-\gamma)}{R_{max}})}{\log \gamma} \right\rfloor.$$

$\epsilon = 0.01$ is set for all experiments, as a compromise between measurement accuracy and computation time.

Finally, to estimate our comparison criterion $\mathfrak{J}_{p_{\mathcal{M}}}^{\pi(p^0_{\mathcal{M}})}$, the empirical average of the algorithm performance is computed over N different MDPs, sampled from $p_{\mathcal{M}}$:

$$\bar{\mathfrak{J}}_{p_{\mathcal{M}}}^{\pi(p^0_{\mathcal{M}})} = \frac{1}{N} \sum_{0 \leq i < N} \bar{J}_{M_i}^{\pi(p^0_{\mathcal{M}})} = \frac{1}{N} \sum_{0 \leq i < N} \bar{\mathcal{R}}_{M_i}^{\pi(p^0_{\mathcal{M}})}(x_0) \tag{4.1}$$

For each agent π, we retrieve $\mu_\pi = \bar{J}_M^\pi$ and σ_π, the empirical mean and standard deviation of the results observed respectively. This gives us the following statistical confidence interval at 95% for J_M^π:

$$J_M^\pi \in \left[\bar{J}_M^\pi - \frac{2\sigma_\pi}{\sqrt{N}};\ \bar{J}_M^\pi + \frac{2\sigma_\pi}{\sqrt{N}} \right].$$

The values reported in the following figures and tables are estimations of the interval within which J_M^π is, with probability 0.95.

As introduced in Section 4.2.3, in our methodology, a function ϕ of computation times is used to classify algorithms based on their time performance. The choice of ϕ depends on the type of time constraints that are the most important to the user. In this paper, we reflect this by showing three different ways to choose ϕ. These three choices lead to three different ways to look at the results and compare algorithms. The first one is to classify algorithms based on their offline computation time, the second one is to classify them based on the algorithms average online computation time. The third is a combination of the first two choices of ϕ, that we denote $\phi_{off}((B_i)_{-1 \leq i}) = B_{-1}$ and $\phi_{on}((B_i)_{-1 \leq i}) = \frac{1}{n} \sum_{0 \leq i < n} B_i$. The objective is that for each pair of constraints $\phi_{off}((B_i)_{-1 \leq i}) < K_1$ and $\phi_{on}((B_i)_{-1 \leq i}) < K_2$, $K_1, K_2 > 0$, we want to identify the best algorithms that respect these constraints.

In order to achieve this: (i) All agents that do not satisfy the constraints are discarded; (ii) for each algorithm, the agent leading to the best performance in average is selected; (iii) we build the list of agents whose performances are not significantly different[1].

The results will help us to identify, for each experiment, the most suitable algorithm(s) depending on the constraints the agents must satisfy. This protocol is an extension of the one presented in (Castronovo, Fonteneau, and Ernst, 2014).

4.4 BBRL Library

BBRL[2] is a C++ open-source library for Bayesian Reinforcement Learning (discrete state/action spaces). This library provides high-level features, while remaining as flexible and documented as possible to address the needs of any researcher of this field. To this end, we developed a complete command-line interface, along with a comprehensive website:

https://github.com/mcastron/BBRL

BBRL focuses on the core operations required to apply the comparison benchmark presented in this paper. To do a complete experiment with the BBRL library, follow these five steps:

1. We create a test and a prior distribution. Those distributions are represented by Flat Dirichlet Multinomial distributions (FDM), parameterised by a state space X, an action space U, a vector of parameters $\boldsymbol{\theta}$, and reward function ρ. For more information about the FDM distributions, check Section 4.5.2.

```
./BBRL-DDS --mdp_distrib_generation \
           --name <name> \
           --short_name <short name> \
           --n_states <n_X> --n_actions <n_U> \
           --ini_state <x_0> \
           --transition_weights \
             <θ(1)> ··· <θ(n_X n_U n_X)> \
           --reward_type "RT_CONSTANT" \
           --reward_means \
             <ρ(x^(1), u^(1), x^(1))> \
             ...
             <ρ(x^(n_X), u^(n_U), x^(n_X))> \
           --output <output file>
```

[1] A paired sampled Z-test with a confidence level of 95% has been used to determine when two agents are statistically equivalent (more details in Appendix 4.6.3).

[2] BBRL stands for **B**enchmaring tools for **B**ayesian **R**einforcement **L**earning.

A distribution file is created.

2. We create an experiment. An experiment is defined by a set of N MDPs, drawn from a test distribution defined in a *distribution file*, a discount factor γ and a horizon limit T.

```
./BBRL-DDS --new_experiment \
             --name <name> \
             --mdp_distribution \
               "DirMultiDistribution" \
               --mdp_distribution_file \
                 <distribution file> \
             --n_mdps <N> \
             --n_simulations_per_mdp 1 \
             --discount_factor <γ> \
             --horizon_limit <T> \
             --compress_output \
             --output <output file>
```

An experiment file is created and can be used to conduct the same experiment for several agents.

3. We create an agent. An agent is defined by an algorithm alg, a set of parameters ψ, and a prior distribution defined in a *distribution file*, on which the created agent will be trained.

```
./BBRL-DDS --offline_learning \
             --agent <alg> [<parameters ψ>] \
             --mdp_distribution \
               "DirMultiDistribution" \
               --mdp_distribution_file \
                 <distribution file> \
             --output <output file>
```

An agent file is created. The file also stores the computation time observed during the offline training phase.

4. We run the experiment. We need to provide an *experiment file*, an algorithm alg and an *agent file*.

```
./BBRL-DDS --run_experiment \
             --experiment \
               --experiment_file \
                 <experiment file> \
             --agent <alg> \
               --agent_file <agent file> \
             --n_threads 1 \
             --compress_output \
             --safe_simulations \
```

```
                --refresh_frequency 60 \
                --backup_frequency 900 \
                --output <output file>
```

A *result file* is created. This file contains a set of all transitions encountered during each trajectory. Additionally, the computation times we observed are also stored in this file. It is often impossible to measure precisely the computation time of a single decision. This is why only the computation time of each trajectory is reported in this file.

5. Our results are exported. After each experiment has been performed, a set of K *result files* is obtained. We need to provide all *agent files* and *result files* to export the data.

```
./BBRL-export --agent <alg^(1)> \
                    --agent_file <agent file #1> \
                  --experiment \
                    --experiment_file \
                      <result file #1> \
                  ...
                  --agent <alg^(K)> \
                    --agent_file <agent file #K> \
                  --experiment \
                    --experiment_file \
                      <result file #K>
```

BBRL will sort the data automatically and produce several files for each experiment.

- A graph comparing offline computation cost w.r.t. performance;
- A graph comparing online computation cost w.r.t. performance;
- A graph where the X-axis represents the offline time bound, while the Y-axis represents the online time bound. A point of the space corresponds to set of bounds. An algorithm is associated to a point of the space if its best agent, satisfying the constraints, is among the best ones when compared to the others;
- A table reporting the results of each agent.

BBRL will also produce a report file in LaTeX gathering the 3 graphs and the table for each experiment.

More than 2.000 commands have to be entered in order to reproduce the results of this paper. We decided to provide several Lua

script in order to simplify the process. By completing some configuration files, the user can define the agents, the possible values of their parameters and the experiments to conduct.

```
local agents =
{
   --   e-Greedy
   {
      name = "EGreedyAgent",
      params =
      {
         {
            opt = "--epsilon",
            values =
            {
               0.0, 0.1, 0.2, 0.3, 0.4, 0.5,
               0.6, 0.7, 0.8, 0.9, 1.0
            }
         }
      },
      olOptions = { "--compress_output" },
      memory = { ol = "1000M", re = "1000M" },
      duration = { ol = "01:00:00", re = "01:00:00" }
   },
   ...
}
```

FIGURE 4.1: Example of a configuration file for the agents.

```
local experiments =
{
   {
      prior = "GC",   priorFile = "GC-distrib.dat",
      exp   = "GC",   testFile  = "GC-distrib.dat",
      N = 500, gamma = 0.95, T = 250
   },
   ...
}
```

FIGURE 4.2: Example of a configuration file for the experiments.

Those configuration files are then used by a script included within the library, called `make_scripts.sh`, whose purpose is to generate four other scripts:

- `0-init.sh`
  Create the experiment files, and create the formulas sets required by OPPS agents.

- `1-ol.sh`
  Create the agents and train them on the prior distribution(s).

- `2-re.sh`
  Run all the experiments.

- `3-export.sh`
  Generate the LaTeX reports.

Due to the high computation power required, we made those scripts compatible with workload managers such as SLURM. In this case, each cluster should provide the same amount of CPU power in order to get consistent time measurements. To sum up, when the configuration files are completed correctly, one can start the whole process by executing the four scripts, and retrieve the results in nice LaTeX reports.

It is worth noting that there is no computation budget given to the agents. This is due to the diversity of the algorithms implemented. No algorithm is "anytime" natively, in the sense that we cannot stop the computation at any time and receive an answer from the agent instantly. Strictly speaking, it is possible to develop an anytime version of some of the algorithms considered in *BBRL*. However, we made the choice to stay as close as possible to the original algorithms proposed in their respective papers for reasons of fairness. In consequence, although computation time is a central parameter in our problem statement, it is never explicitly given to the agents. We instead let each agent run as long as necessary and analyse the time elapsed afterwards.

Another point which needs to be discussed is the impact of the implementation of an algorithm on the comparison results. For each algorithm, many implementations are possible, some being better than others. Even though we did our best to provide the best possible implementations, *BBRL* does not compare algorithms but rather the implementations of each algorithms. Note that this issue mainly concerns small problems, since the complexity of the algorithms is preserved.

4.5 Illustration

This section presents an illustration of the protocol presented in Section 4.3. We first describe the algorithms considered for the comparison in Section 4.5.1, followed by a description of the benchmarks in Section 4.5.2. Section 4.5.3 shows and analyses the results obtained.

4.5.1 Compared Algorithms

In this section, we present the list of the algorithms considered in this study. The pseudo-code of each algorithm can be found in Appendix 4.6.1. For each algorithm, a list of "reasonable" values is provided to test each of their parameters. When an algorithm has more than one parameter, all possible parameter combinations are tested.

Random

At each time-step t, the action u_t is drawn uniformly from U.

ϵ-Greedy

The ϵ-Greedy agent maintains an approximation of the current MDP and computes, at each time-step, its associated Q-function. The selected action is either selected randomly (with a probability of ϵ ($1 \geq \epsilon \geq 0$), or greedily (with a probability of $1 - \epsilon$) with respect to the approximated model.

Tested values:

- $\epsilon \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

Soft-max

The Soft-max agent maintains an approximation of the current MDP and computes, at each time-step, its associated Q-function. The selected action is selected randomly, where the probability to draw an action u is proportional to $Q(x_t, u)$. The temperature parameter τ allows to control the impact of the Q-function on these probabilities ($\tau \to 0^+$: greedy selection; $\tau \to +\infty$: random selection).

Tested values:

- $\tau \in \{0.05, 0.10, 0.20, 0.33, 0.50, 1.0, 2.0, 3.0, 5.0, 25.0\}$.

OPPS

Given a prior distribution $p^0_{\mathcal{M}}(.)$ and an E/E strategy space $\mathcal{S}$ (either discrete or continuous), the Offline, Prior-based Policy Search algorithm (OPPS) identifies a strategy $\pi^* \in \mathcal{S}$ which maximises the expected discounted sum of returns over MDPs drawn from the prior.

The OPPS for Discrete Strategy spaces algorithm (OPPS-DS) (Castronovo et al., 2012; Castronovo, Fonteneau, and Ernst, 2014) formalises the strategy selection problem as a k-armed bandit problem,

where $k = |\mathcal{S}|$. Pulling an arm amounts to draw an MDP from $p^0_{\mathcal{M}}(.)$, and play the E/E strategy associated to this arm on it for one single trajectory. The discounted sum of returns observed is the return of this arm. This multi-armed bandit problem has been solved by using the UCB1 algorithm (Auer, Cesa-Bianchi, and Fischer, 2002; Audibert, Munos, and Szepesvári, 2007). The time budget is defined by a variable β, corresponding to the total number of draws performed by the UCB1.

The E/E strategies considered by Castronovo et. al are index-based strategies, where the index is generated by evaluating a small formula. A formula is a mathematical expression, combining specific features (Q-functions of different models) by using standard mathematical operators (addition, subtraction, logarithm, etc.). The discrete E/E strategy space is the set of all formulas which can be built by combining at most n features/operators (such a set is denoted by $\mathbb{F}_n$).

OPPS-DS does not come with any guarantee. However, the UCB1 bandit algorithm used to identify the best E/E strategy within the set of strategies provides statistical guarantees that the best E/E strategies are identified with high probability after a certain budget of experiments. However, it is not clear that the best strategy of the E/E strategy space considered yields any high-performance strategy regardless the problem.

Tested values:

- $\mathcal{S} \in \{\mathbb{F}_2, \mathbb{F}_3, \mathbb{F}_4, \mathbb{F}_5, \mathbb{F}_6\}$[3],
- $\beta \in \{50, 500, 1250, 2500, 5000, 10000, 100000, 1000000\}$.

BAMCP

Bayes-adaptive Monte Carlo Planning (BAMCP) (Guez, Silver, and Dayan, 2012) is an evolution of the Upper Confidence Tree (UCT) algorithm (Kocsis and Szepesvári, 2006), where each transition is sampled according to the history of observed transitions. The principle of this algorithm is to adapt the UCT principle for planning in a Bayes-adaptive MDP, also called the belief-augmented MDP, which is an MDP obtained when considering augmented states made of the concatenation of the actual state and the posterior. The BAMCP algorithm is made computationally tractable by using a sparse sampling strategy, which avoids sampling a model from the posterior distribution at every node of the planification tree. Note that the BAMCP also comes with theoretical guarantees of convergence towards Bayesian optimality.

[3]The number of arms k is always equal to the number of strategies in the given set. For your information: $|\mathbb{F}_2| = 12, |\mathbb{F}_3| = 43, |\mathbb{F}_4| = 226, |\mathbb{F}_5| = 1210, |\mathbb{F}_6| = 7407$

In practice, the BAMCP relies on two parameters: (i) Parameter K which defines the number of nodes created at each time-step, and (ii) Parameter $depth$ which defines the depth of the tree from the root.

Tested values:

- $K \in \{1, 500, 1250, 2500, 5000, 10000, 25000\}$,
- $depth \in \{15, 25, 50\}$.

BFS3

The Bayesian Forward Search Sparse Sampling (BFS3) (Asmuth and Littman, 2011) is a Bayesian RL algorithm whose principle is to apply the principle of the FSSS (Forward Search Sparse Sampling, see (Kearns and Singh, 2002) algorithm to belief-augmented MDPs. It first samples one model from the posterior, which is then used to sample transitions. The algorithm then relies on lower and upper bounds on the value of each augmented state to prune the search space. The authors also show that BFS3 converges towards Bayes-optimality as the number of samples increases.

In practice, the parameters of BFS3 are used to control how much computational power is allowed. The parameter K defines the number of nodes to develop at each time-step, C defines the branching factor of the tree and $depth$ controls its maximal depth.

Tested values:

- $K \in \{1, 500, 1250, 2500, 5000, 10000\}$,
- $C \in \{2, 5, 10, 15\}$,
- $depth \in \{15, 25, 50\}$.

SBOSS

The Smarter Best of Sampled Set (SBOSS) (Castro and Precup, 2010) is a Bayesian RL algorithm which relies on the assumption that the model is sampled from a Dirichlet distribution. From this assumption, it derives uncertainty bounds on the value of state action pairs. It then uses those bounds to decide how many models to sample from the posterior, and how often the posterior should be updated in order to reduce the computational cost of Bayesian updates. The sampling technique is then used to build a merged MDP, as in (Asmuth et al., 2009), and to derive the corresponding optimal action with respect to that MDP. In practice, the number of sampled models is determined dynamically with a parameter ϵ. The re-sampling frequency depends

on a parameter δ.

Tested values:

- $\epsilon \in \{1.0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6\}$,
- $\delta \in \{9, 7, 5, 3, 1, 1e-1, 1e-2, 1e-3n1e-4, 1e-5, 1e-6\}$.

BEB

The Bayesian Exploration Bonus (BEB) (Kolter and Ng, 2009) is a Bayesian RL algorithm which builds, at each time-step t, the expected MDP given the current posterior. Before solving this MDP, it computes a new reward function $\rho^{(t)}_{BEB}(x,u,y) = \rho_M(x,u,y) + \frac{\beta}{c^{(t)}_{<x,u,y>}}$, where $c^{(t)}_{<x,u,y>}$ denotes the number of times transition $< x,u,y >$ has been observed at time-step t. This algorithm solves the mean MDP of the current posterior, in which we replaced $\rho_M(\cdot,\cdot,\cdot)$ by $\rho^{(t)}_{BEB}(\cdot,\cdot,\cdot)$, and applies its optimal policy on the current MDP for one step. The bonus β is a parameter controlling the E/E balance. BEB comes with theoretical guarantees of convergence towards Bayesian optimality.

Tested values:

- $\beta \in \{0.25, 0.5, 1, 1.5, 2, 2.5, 3, 4, 8, 16\}$.

Computation times variance

Each algorithm has one or more parameters that can affect the number of sampled transitions from a given state, or the length of each simulation. This, in turn, impacts the computation time requirement at each step. Hence, for some algorithms, no choice of parameters can bring the computation time below or over certain values. In other words, each algorithm has its own range of computation time. Note that, for some methods, the computation time is influenced concurrently by several parameters. We present a qualitative description of how computation time varies as a function of parameters in Table 4.1.

[4] If a random decision is chosen, the model is not solved.

[5] K defines the number of nodes to develop at each step, and $depth$ defines the maximal depth of the tree.

[6] K defines the number of nodes to develop at each step, C the branching factor of the tree and $depth$ its maximal depth.

[7] The number of models sampled is inversely proportional to ϵ, while the frequency at which the models are sampled is inversely proportional to δ. When an MDP has been sufficiently explored, the number of models to sample and the frequency of the sampling will decrease.

	Offline phase duration	Online phase duration
Random	Almost instantaneous.	Almost instantaneous.
ϵ-Greedy[4]	Almost instantaneous.	Varies in inverse proportion to ϵ. Can vary a lot from one step to another.
OPPS-DS	Varies proportionally to β.	Varies proportionally to the number of features implied in the selected E/E strategy.
BAMCP[5]	Almost instantaneous.	Varies proportionally to K and $depth$.
BFS3[6]	Almost instantaneous.	Varies proportionally to K, C and $depth$.
SBOSS[7]	Almost instantaneous.	Varies in inverse proportion to ϵ and δ. Can vary a lot from one step to another, with a general decreasing tendency.
BEB	Almost instantaneous.	Constant.

TABLE 4.1: Influence of the algorithm and their parameters on the offline and online phases duration.

4.5.2 Benchmarks

In our setting, the transition matrix is the only element which differs between two MDPs drawn from the same distribution. For each $<$ state, action $>$ pair $< x, u >$, we define a Dirichlet distribution, which represents the uncertainty about the transitions occurring from $< x, u >$. A Dirichlet distribution is parameterised by a set of concentration parameters $\alpha^{(1)}_{<x,u>}, \cdots, \alpha^{(n_X)}_{<x,u>}$.

We gathered all concentration parameters in a single vector $\boldsymbol{\theta}$. Consequently, our MDP distributions are parameterised by ρ_M (the reward function) and several Dirichlet distributions, parameterised by $\boldsymbol{\theta}$. Such a distribution is denoted by $p^{\rho_M,\boldsymbol{\theta}}(\cdot)$. In the Bayesian Reinforcement Learning community, these distributions are referred to as Flat Dirichlet Multinomial distributions (FDMs).

We chose to study two different cases:

- Accurate case: the test distribution is fully known ($p^0_{\mathcal{M}}(.) = p_{\mathcal{M}}(.)$),
- Inaccurate case: the test distribution is unknown ($p^0_{\mathcal{M}}(.) \neq p_{\mathcal{M}}(.)$).

In the inaccurate case, we have no assumption on the transition matrix. We represented this lack of knowledge by a uniform FDM distribution, where each transition has been observed one single time

($\boldsymbol{\theta} = [1, \cdots, 1]$).

Sections 4.5.2, 4.5.2 and 4.5.2 describes the three distributions considered for this study.

Generalised Chain Distribution

The Generalised Chain (GC) distribution ($p^{\rho^{GC}, \boldsymbol{\theta}^{GC}}(\cdot)$) is inspired from the five-state chain problem (5 states, 3 actions) Dearden, Friedman, and Russell, 1998. The agent starts at State 1, and has to go through State 2, 3 and 4 in order to reach the last state (State 5), where the best rewards are. The agent has at its disposal 3 actions. An action can either let the agent move from State $x^{(n)}$ to State $x^{(n+1)}$ or force it to go back to State $x^{(1)}$. The transition matrix is drawn from a FDM parameterised by $\boldsymbol{\theta}^{GC}$, and the reward function is denoted by ρ^{GC}. More details can be found in Appendix 4.6.2.

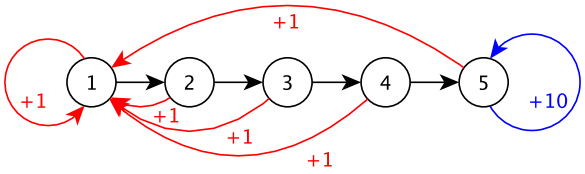


FIGURE 4.3: Illustration of the GC distribution.

Generalised Double-Loop Distribution

The Generalised Double-Loop (GDL) distribution ($p^{\rho^{GDL}, \boldsymbol{\theta}^{GDL}}(\cdot)$) is inspired from the double-loop problem (9 states, 2 actions) (Dearden, Friedman, and Russell, 1998). Two loops of 5 states are crossing at State 1, where the agent starts. One loop is a trap: if the agent enters it, it has no choice to exit but crossing over all the states composing it. Exiting this loop provides a small reward. The other loop is yielding a good reward. However, each action of this loop can either let the agent move to the next state of the loop or force it to return to State 1 with no reward. The transition matrix is drawn from an FDM parameterised by $\boldsymbol{\theta}^{GDL}$, and the reward function is denoted by ρ^{GDL}. More details can be found in Appendix 4.6.2.

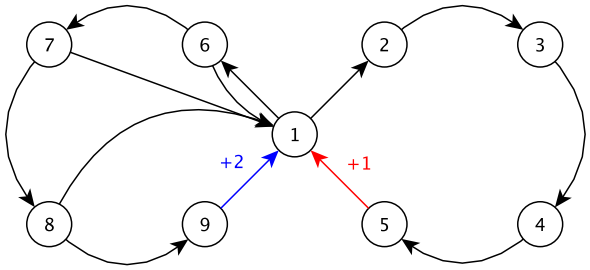


FIGURE 4.4: Illustration of the GDL distribution.

Grid Distribution

The Grid distribution ($p^{\rho^{Grid},\boldsymbol{\theta}^{Grid}}(\cdot)$) is inspired from the Dearden's maze problem (25 states, 4 actions) (Dearden, Friedman, and Russell, 1998). The agent is placed at a corner of a 5x5 grid (the **S** cell), and has to reach the opposite corner (the **G** cell). When it succeeds, it returns to its initial state and receives a reward. The agent can perform 4 different actions, corresponding to the 4 directions (up, down, left, right). However, depending on the cell on which the agent is, each action has a certain probability to fail, and can prevent the agent to move in the selected direction. The transition matrix is drawn from an FDM parameterised by $\boldsymbol{\theta}^{Grid}$, and the reward function is denoted by ρ^{Grid}. More details can be found in Appendix 4.6.2.

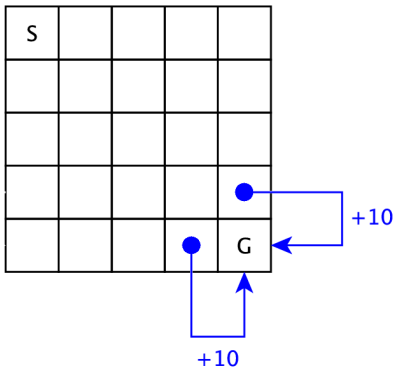


FIGURE 4.5: Illustration of the Grid distribution.

4.5.3 Discussion of the Results

Accurate case

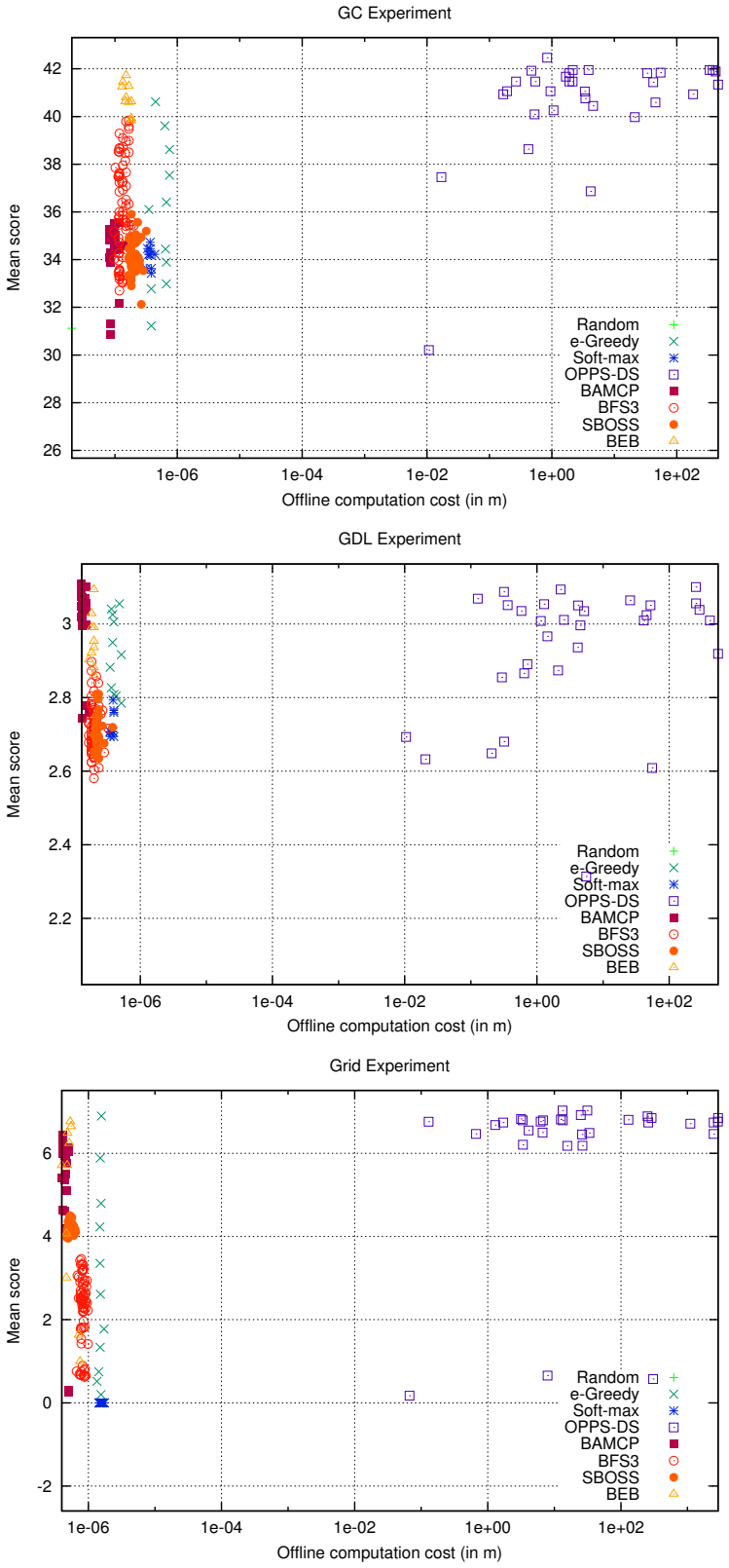


FIGURE 4.6: Offline computation cost Vs. Performance

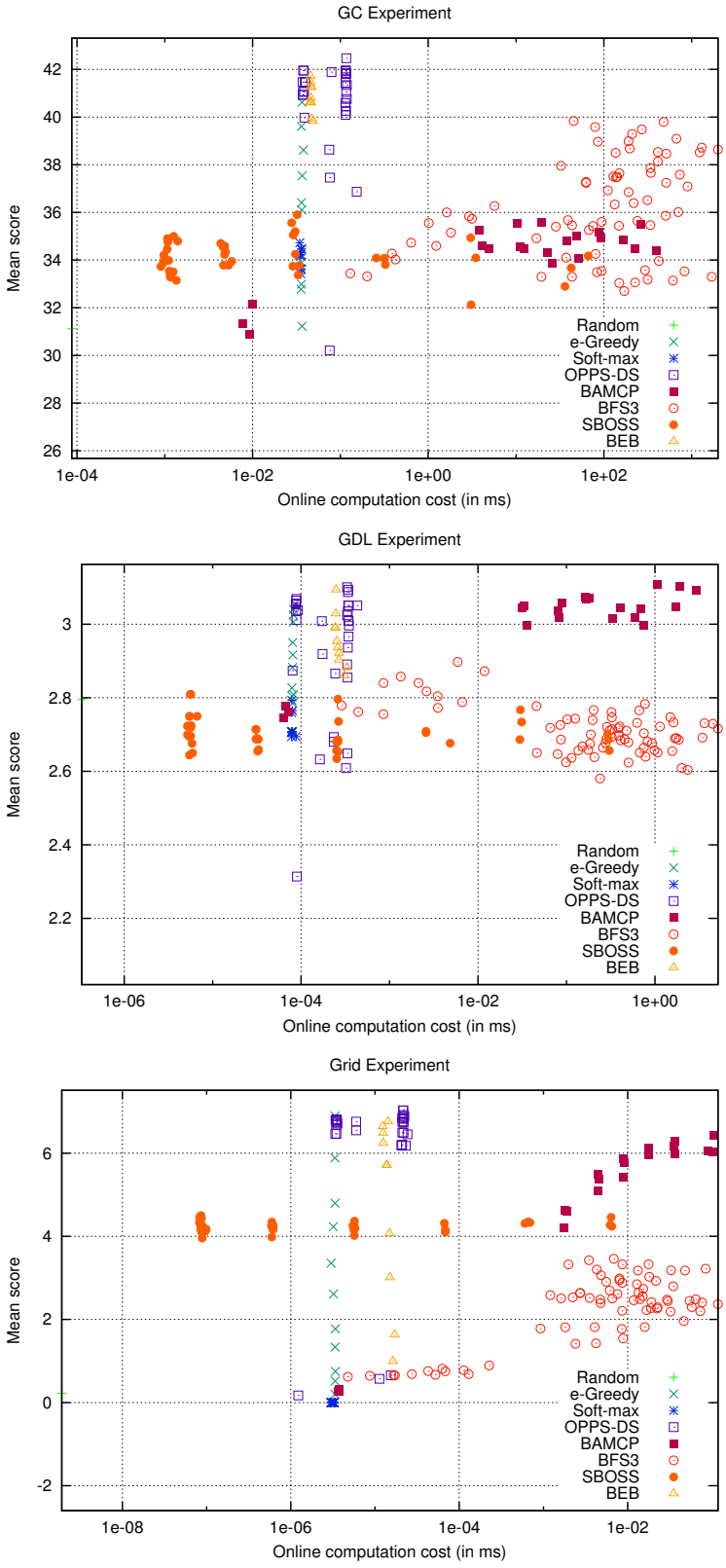


FIGURE 4.7: Online computation cost Vs. Performance

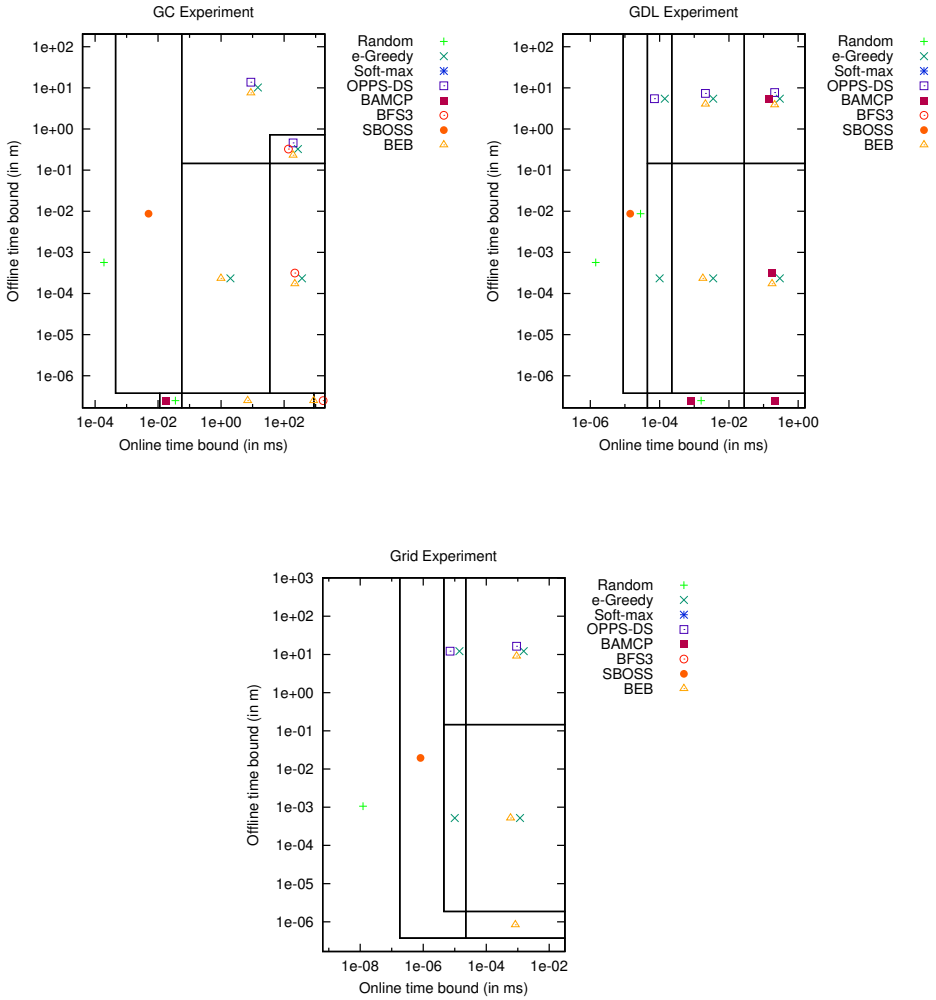


FIGURE 4.8: Best algorithms w.r.t offline/online time periods

GC Experiment

Agent	**Score**
Random	31.12 ± 0.9
e-Greedy (ϵ = 0)	40.62 ± 1.55
Soft-Max (τ = 0.1)	34.73 ± 1.74
OPPS-DS ($Q_2(x,u)/Q_0(x,u)$)	$\mathbf{42.47 \pm 1.91}$
BAMCP (K = 2500, $depth$ = 15)	35.56 ± 1.27
BFS3 (K = 500, C = 15, $depth$ = 15)	39.84 ± 1.74
SBOSS (ϵ = 0.001, δ = 7)	35.9 ± 1.89
BEB (β = 2.5)	41.72 ± 1.63

GDL Experiment

Agent	**Score**
Random	2.79 ± 0.07
e-Greedy (ϵ = 0.1)	3.05 ± 0.07
Soft-Max (τ = 0.1)	2.79 ± 0.1
OPPS-DS ($\max(Q_0(x,u), \lvert Q_2(x,u)\rvert)$)	3.1 ± 0.07
BAMCP (K = 10000, $depth$ = 15)	$\mathbf{3.11 \pm 0.07}$
BFS3 (K = 1, C = 15, $depth$ = 25)	2.9 ± 0.07
SBOSS (ϵ = 1, δ = 1)	2.81 ± 0.1
BEB (β = 0.5)	3.09 ± 0.07

Grid Experiment

Agent	**Score**
Random	0.22 ± 0.06
e-Greedy (ϵ = 0)	6.9 ± 0.31
Soft-Max (τ = 0.05)	0 ± 0
OPPS-DS ($Q_0(x,u) + Q_2(x,u)$)	$\mathbf{7.03 \pm 0.3}$
BAMCP (K = 25000, $depth$ = 15)	6.43 ± 0.3
BFS3 (K = 500, C = 15, $depth$ = 50)	3.46 ± 0.23
SBOSS (ϵ = 0.1, δ = 7)	4.5 ± 0.33
BEB (β = 0.5)	6.76 ± 0.3

FIGURE 4.9: Best algorithms w.r.t Performance

As it can be seen in Figure 4.6, OPPS is the only algorithm whose offline time cost varies. In the three different settings, OPPS can be launched after a few seconds, but behaves very poorly. However, its performances increased very quickly when given at least one minute of computation time. Algorithms that do not use offline computation time have a wide range of different scores. This variance represents the different possible configurations for these algorithms, which only lead to different online computation time.

On Figure 4.7, BAMCP, BFS3 and SBOSS have variable online time costs. BAMCP behaved poorly on the first experiment, but obtained the best score on the second one and was pretty efficient on the last one. BFS3 was good only on the second experiment. SBOSS was never able to get a good score in any cases. Note that OPPS online time cost varies slightly depending on the formula's complexity.

If we take a look at the top-right point in Figure 4.8, which defines the less restrictive bounds, we notice that OPPS-DS and BEB were always the best algorithms in every experiment. ϵ-Greedy was a good candidate in the two first experiments. BAMCP was also a very good choice except for the first experiment. On the contrary, BFS3 and SBOSS were only good choices in the first experiment.

If we look closely, we can notice that OPPS-DS was always one of the best algorithm since we have met its minimal offline computation time requirements.

Moreover, when we place our offline-time bound right under OPPS-DS minimal offline time cost, we can see how the top is affected from left to right:

GC: (Random), (SBOSS), (BEB, ϵ-Greedy), (BEB, BFS3, ϵ-Greedy),

GDL: (Random), (Random, SBOSS), (ϵ-Greedy), (BEB, ϵ-Greedy), (BAMCP, BEB, ϵ-Greedy),

Grid: (Random), (SBOSS), (ϵ-Greedy), (BEB, ϵ-Greedy).

We can clearly see that SBOSS was the first algorithm to appear on the top, with a very small online computation cost, followed by ϵ-Greedy and BEB. Beyond a certain online time bound, BFS3 emerged in the first experiment while BAMCP emerged in the second experiment. Neither of them was able to compete with BEB or ϵ-Greedy in the last experiment.

Soft-max was never able to reach the top regardless the configuration.

Figure 4.9 reports the best score observed for each algorithm, disassociated from any time measure. Note that the variance is very similar for all algorithms in GDL and Grid experiments. On the contrary, the variance oscillates between 1.0 and 2.0. However, OPPS seems to be the less stable algorithm in the three cases.

Inaccurate case

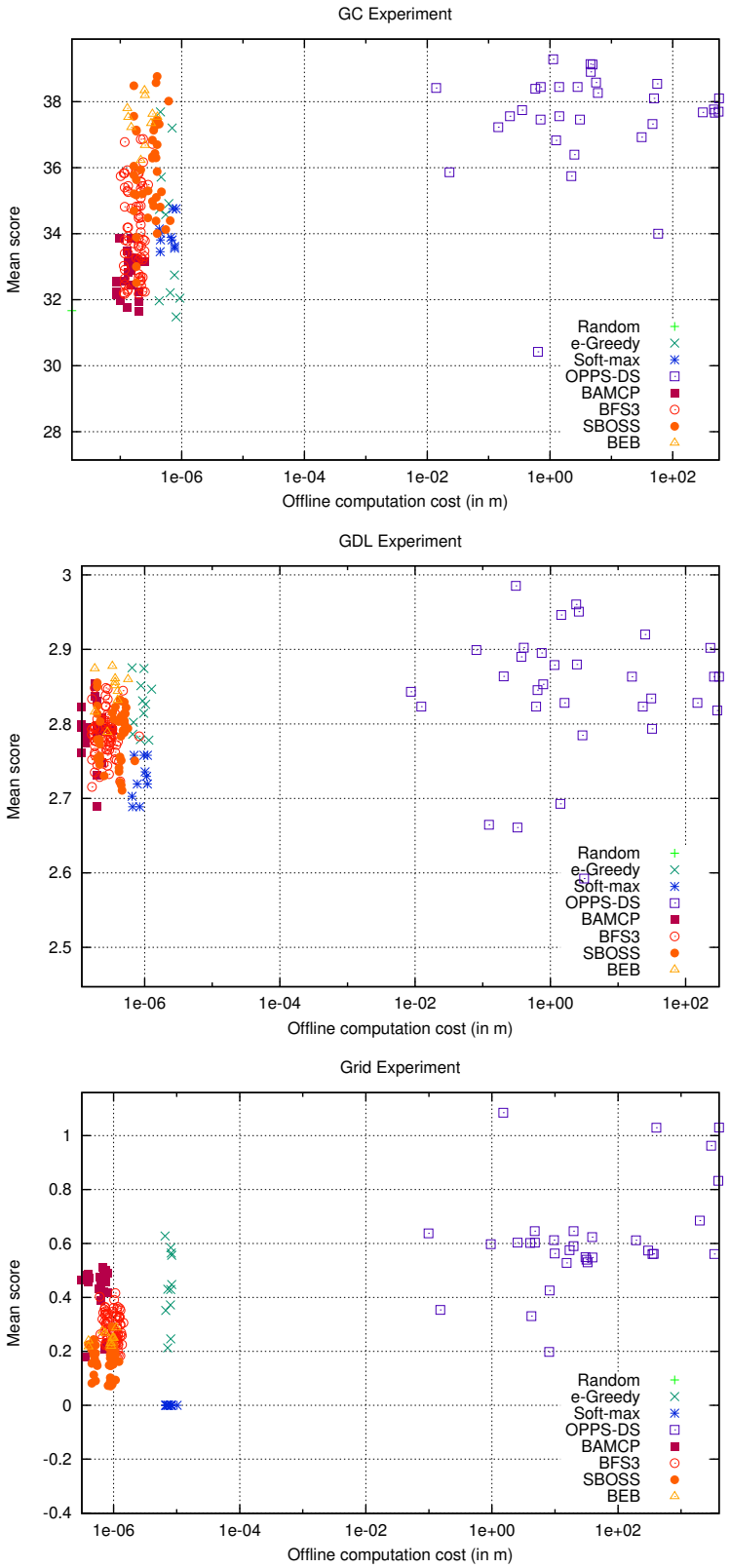


FIGURE 4.10: Offline computation cost Vs. Performance

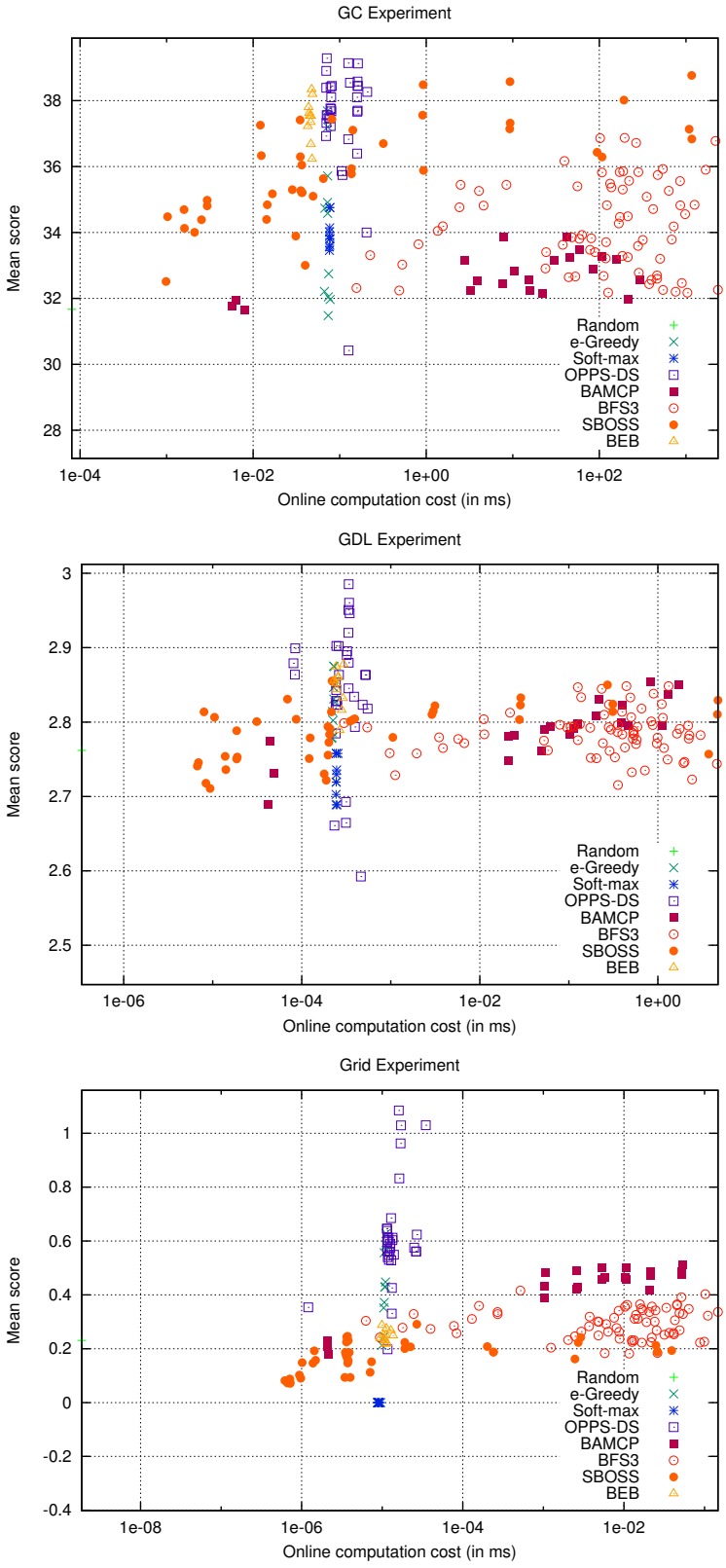


FIGURE 4.11: Online computation cost Vs. Performance

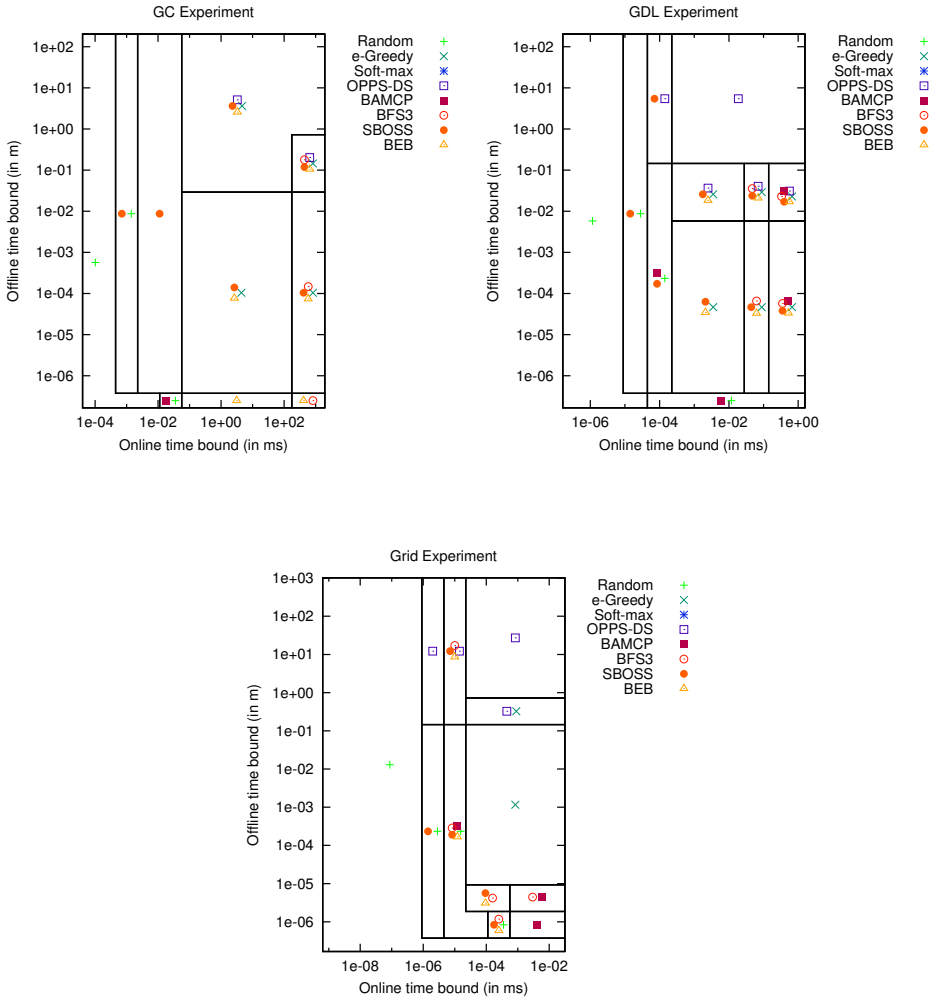


FIGURE 4.12: Best algorithms w.r.t offline/online time periods

GC Experiment

Agent	**Score**
Random	31.67 ± 1.05
e-Greedy (ϵ = 0)	37.69 ± 1.75
Soft-Max (τ = 0.33)	34.75 ± 1.64
OPPS-DS ($\boldsymbol{Q_0(x,u)}$)	$\mathbf{39.29 \pm 1.71}$
BAMCP (K = 1250, $depth$ = 25)	33.87 ± 1.26
BFS3 (K = 1250, C = 15, $depth$ = 25)	36.87 ± 1.82
SBOSS (ϵ = 1e-06, δ = 0.0001)	38.77 ± 1.89
BEB (β = 16)	38.34 ± 1.62

GDL Experiment

Agent	**Score**
Random	2.76 ± 0.08
e-Greedy (ϵ = 0.3)	2.88 ± 0.07
Soft-Max (τ = 0.05)	2.76 ± 0.1
OPPS-DS ($\max(\boldsymbol{Q_0(x,u)}, \boldsymbol{Q_1(x,u)}$)	$\mathbf{2.99 \pm 0.08}$
BAMCP (K= 10000, $depth$ = 50)	2.85 ± 0.07
BFS3 (K = 1250, C = 15, $depth$ = 50)	2.85 ± 0.07
SBOSS (ϵ = 0.1, δ = 0.001)	2.86 ± 0.07
BEB (β = 2.5)	2.88 ± 0.07

Grid Experiment

Agent	**Score**
Random	0.23 ± 0.06
e-Greedy (ϵ = 0.2)	0.63 ± 0.09
Soft-Max (τ = 0.05)	0 ± 0
OPPS-DS ($\boldsymbol{Q_1(x,u) + Q_2(x,u)}$))	$\mathbf{1.09 \pm 0.17}$
BAMCP (K = 25000, $depth$ = 25)	0.51 ± 0.09
BFS3 (K = 1, C = 15, $depth$ = 50)	0.42 ± 0.09
SBOSS (ϵ = 0.001, δ = 0.1)	0.29 ± 0.07
BEB (β = 0.25)	0.29 ± 0.05

FIGURE 4.13: Best algorithms w.r.t Performance

As seen in the accurate case, Figure 4.10 also shows impressive performances for OPPS-DS, which has beaten all other algorithms in every experiment. We can also notice that, as observed in the accurate case, in the Grid experiment, the OPPS-DS agents scores are very close. However, only a few were able to significantly surpass the others, contrary to the accurate case where most OPPS-DS agents were very good candidates.

Surprisingly, SBOSS was a very good alternative to BAMCP and BFS3 in the two first experiments as shown in Figure 4.11. It was able to surpass both algorithms on the first one while being very close to BAMCP performances in the second. Relative performances of BAMCP and BFS3 remained the same in the inaccurate case, even if the BAMCP advantage is less visible in the second experiment. BEB was no longer able to compete with OPPS-DS and was even beaten by BAMCP and BFS3 in the last experiment. ϵ-Greedy was still a decent choice except in the first experiment. As observed in the accurate case, Soft-max was very bad in every case.

In Figure 4.12, if we take a look at the top-right point, we can see OPPS-DS is the best choice in the second and third experiment. BEB, SBOSS and ϵ-Greedy share the first place with OPPS-DS in the first one.

If we place our offline-time bound right under OPPS-DS minimal offline time cost, we can see how the top is affected from left to right:

GC: (Random), (Random, SBOSS), (SBOSS), (BEB, SBOSS, ϵ-Greedy), (BEB, BFS3, SBOSS, ϵ-Greedy),

GDL: (Random), (Random, SBOSS), (BAMCP, Random, SBOSS), (BEB, SBOSS, ϵ-Greedy), (BEB, BFS3, SBOSS, ϵ-Greedy), (BAMCP, BEB, BFS3, SBOSS, ϵ-Greedy),

Grid: (Random), (Random, SBOSS), (BAMCP, BEB, BFS3, Random, SBOSS), (ϵ-Greedy).

SBOSS is again the first algorithm to appear in the rankings. ϵ-Greedy is the only one which could reach the top in every case, even when facing BAMCP and BFS3 fed with high online computation cost. BEB no longer appears to be undeniably better than the others. Besides, the two first experiments show that most algorithms obtained similar results, except for BAMCP which does not appear on the top in the first experiment. In the last experiment, ϵ-Greedy succeeded to beat all other algorithms.

Figure 4.13 does not bring us more information than those we observed in the accurate case.

Summary

In the accurate case, OPPS-DS was always among the best algorithms, at the cost of some offline computation time. When the offline time

budget was too constrained for OPPS-DS, different algorithms were suitable depending on the online time budget:

- **Low online time budget:** SBOSS was the fastest algorithm to make better decisions than a random policy.
- **Medium online time budget**[8]**:** BEB reached performances similar to OPPS-DS on each experiment.
- **High online time budget**[9]**:** In the first experiment, BFS3 managed to catch up BEB and OPPS-DS when given sufficient time. In the second experiment, it was BAMCP which has achieved this result. Neither BFS3 nor BAMCP was able to compete with BEB and OPPS-DS in the last experiment.

The results obtained in the inaccurate case were very interesting. BEB was not as good as it seemed to be in the accurate case, while SBOSS improved significantly compared to the others. For its part, OPPS-DS obtained the best overall results in the inaccurate case by outperforming all the other algorithms in two out of three experiments while remaining among the best ones in the last experiment.

4.6 Conclusion

We have proposed a new extensive BRL comparison methodology which takes into account both performance and time requirements for each algorithm. In particular, our benchmarking protocol shows that no single algorithm dominates all other algorithms on all scenarios. The protocol we introduced can compare any time algorithm to non-anytime algorithms while measuring the impact of inaccurate offline training. By comparing algorithms on large sets of problems, we avoid over fitting to a single problem. Our methodology is associated with an open-source library, BBRL, and we hope that it will help other researchers to design algorithms whose performances are put into perspective with computation times, that may be critical in many applications. This library is specifically designed to handle new algorithms easily, and is provided with a complete and comprehensive documentation website.

[8] $\pm$ 100 times more than the low online time budget
[9] $\pm$ 100 times more than the medium online time budget

Appendices

4.6.1 Pseudo-code of the Algorithms

Algorithm 1 ϵ-Greedy

procedure OFFLINE-LEARNING($p^0_{\mathcal{M}}(.)$)
    $\hat{M} \leftarrow$ "Build an initial model based on $p^0_{\mathcal{M}}(.)$"
end procedure

function SEARCH(x, h)
    {Draw a random value in $[0; 1]$}
    $r \leftarrow \mathcal{U}(0, 1)$

    if $r < \epsilon$ **then** {Random case}
        return "An action selected randomly"

    else {Greedy case}
        $\pi^*_{\hat{M}} \leftarrow$VALUE-ITERATION($\hat{M}$)
        return $\pi^*_{\hat{M}}(x)$
    end if
end function

procedure ONLINE-LEARNING(x, u, y, r)
    "Update model $\hat{M}$ w.r.t. transition $< x, u, y, r >$"
end procedure

Algorithm 2 Soft-max

procedure OFFLINE-LEARNING($p^0_{\mathcal{M}}(.)$)
    $\hat{M} \leftarrow$ "Build an initial model based on $p^0_{\mathcal{M}}(.)$"
end procedure

function SEARCH(x, h)
    {Draw a random value in $[0; 1]$}
    $r \leftarrow \mathcal{U}(0, 1)$

    {Select an action randomly, with a probability proportional to $Q^*_{\hat{M}}(x, u)$}
    $Q^*_{\hat{M}} \leftarrow$ "Compute the optimal Q-function of $\hat{M}$"
    for $1 \leq i \leq |U|$ **do**
        if $r < \sum_{j \leq i} \frac{\exp\left(Q^*_{\hat{M}}(x,u^{(j)})/\tau\right)}{\sum_{u'} \exp\left(Q^*_{\hat{M}}(x,u')/\tau\right)}$ **then**
            return $u^{(i)}$
        end if
    end for
end function

procedure ONLINE-LEARNING(x, u, y, r)
    "Update model $\hat{M}$ w.r.t. transition $< x, u, y, r >$"
end procedure

Algorithm 3 OPPS-DS

procedure OFFLINE-LEARNING($p^0_{\mathcal{M}}(.)$)
  {Initialise the k arms of UCB1}
  for $1 \leq i \leq k$ **do**
    $M \sim p^0_{\mathcal{M}}(.)$
    $R^{\pi_i}_M \leftarrow$ "Simulate strategy π_i on MDP M over a single trajectory"
    $\mu(i) \leftarrow R^{\pi_i}_M$
    $\theta(i) \leftarrow 1$
  end for

  {Run UCB1 with a budget of β}
  for $k+1 \leq b \leq \beta$ **do**
    $a \leftarrow arg\,max_{a'} \mu(a') + \sqrt{\frac{2\log(b)}{\theta(a')}}$
    $M \sim p^0_{\mathcal{M}}(.)$
    $R^{\pi_a}_M \leftarrow$ "Simulate strategy π_a on MDP M over a single trajectory"
    $\mu(a) \leftarrow \frac{\theta(a)\mu(a)+R^{\pi_a}_M}{\theta(a)+1}$
    $\theta(a) \leftarrow \theta(a) + 1$
  end for

  {Select the E/E strategy associated to the most drawn arm}
  $a^* \leftarrow arg\,max_{a'} \theta(a')$
  $\pi_{OPPS} \leftarrow \pi_{a^*}$
end procedure

function SEARCH(x, h)
  return $u \sim \pi_{OPPS}(x, h)$
end function

procedure ONLINE-LEARNING(x, u, y, r)
  "Update strategy π_{OPPS} w.r.t. transition $< x, u, y, r >$"
end procedure

Algorithm 4 BAMCP (1/2)

function SEARCH(x, h)
  {Develop a MCTS and compute $Q(.,.)$}
  for $1 \leq k \leq K$ **do**
    $M \sim p^h_{\mathcal{M}}$
    SIMULATE($\langle x, h \rangle, M, 0$)
  end for

  {Return the best action w.r.t. $Q(.,.)$}
  return $arg\, max_u Q(\langle x, h \rangle, u)$
end function

function SIMULATE($\langle x, h \rangle, M, d$)
  if $N(\langle x, h \rangle) = 0$ **then** {New node reached}
    "Initialise $N(\langle x, h \rangle, u)$, $Q(\langle x, h \rangle, u)$"
    $u \sim \pi_0(\langle x, h \rangle)$
    "Sample x', r from model M"

    {Estimate the score of this node by using the rollout policy}
    $R \leftarrow r + \gamma$ ROLLOUT($\langle x', hux' \rangle, P, d$)

    "Update $N(\langle x, h \rangle)$, $N(\langle x, h \rangle, u)$, $Q(\langle x, h \rangle, u)$"
    return R
  end if

  {Select the next branch to explore}
  $u \leftarrow arg\, max_{u'} Q(\langle x, h \rangle, u) + c\sqrt{(\frac{log(N(\langle x,h \rangle))}{N(\langle x,h \rangle, u')})}$
  "Sample x', r from model M"

  {Follow the branch and evaluate it}
  $R \leftarrow r + \gamma$ SIMULATE($\langle x', hux' \rangle, M, d + 1$)

  "Update $N(\langle x, h \rangle)$, $N(\langle x, h \rangle, u)$, $Q(\langle x, h \rangle, u)$"
  return R
end function

Algorithm 5 BAMCP (2/2)

procedure ROLLOUT($\langle x, h \rangle, M, d$)
    {Truncate the trajectory if precision ϵ has been reached}
    if $\gamma^d R_{max} < \epsilon$ **then**
        return 0
    end if

    {Use the rollout policy to choose the action to perform}
    $u \sim \pi_0(x, h)$

    {Simulate a single transition from M and continue the rollout process}
    $y \sim P_M$
    $r \leftarrow \rho_M(x, u, y)$
    return $r + \gamma$ ROLLOUT($\langle y, huy \rangle, M, d + 1$)
end procedure

procedure ONLINE-LEARNING(x, u, y, r)
    "Update the posterior w.r.t. transition $< x, u, y, r >$"
end procedure

Algorithm 6 BFS3

function SEARCH(x, h)
  {Update the current Q-function}
  $M_{mean} \leftarrow$ "Compute the mean MDP of $p^t_{\mathcal{M}}(.)$."
  for all $u \in$ U **do**
    for $1 \leq i \leq C$ **do**
      {Draw y and r from the mean MDP of the posterior}
      $y \sim P_{M_{mean}}$
      $r \leftarrow \rho_M(x, u, y)$

      {Update the Q-value in (x, u) by using $FSSS$}
      $Q(x, u) \leftarrow Q(x, u) + \frac{1}{C}\left[r + \gamma \text{ FSSS}(y, d, t)\right]$
    end for
  end for

  {Return the action u with the maximal Q-value in x}
  return $arg\, max_u Q(x, u)$
end function

Algorithm 7 FSSS (1/2)

function FSSS(x, d, t)
  {Develop a MCTS and compute bounds on $V(x)$}
  for $1 \leq i \leq t$ **do**
    ROLLOUT($s, d, 0$)
  end for

  {Make an optimistic estimation of $V(x)$}
  $\hat{V}(x) \leftarrow \max_u U_d(x, u)$
  return $\hat{V}(x)$
end function

Algorithm 8 FSSS (2/2)

procedure ROLLOUT(x, d, l)
  if $d = l$ **then** {Stop when reaching the maximal depth}
    return
  end if

  if $\neg Visited_d(x)$ **then** {New node reached}
    {Initialise this node}
    for all $u \in U$ **do**
      "Initialise $N_d(x, u, x')$,$R_d(x, u)$"
      for $1 \leq i \leq C$ **do**
        "Sample x', r from M"
        "Update $N_d(x, u, x')$,$R_d(x, u)$"

        if $\neg Visited_d(x')$ **then**
          $U_{d+1}(x'), L_{d+1}(x') = V_{max}, V_{min}$
        end if
      end for
    end for

    {Back-propagate this node's information}
    BELLMAN-BACKUP(x, d)

    $Visited_d(x) \leftarrow$true
  end if

  {Select an action and simulate a transition optimistically}
  $u \leftarrow arg\,max_u U_d(x, u)$
  $x' \leftarrow arg\,max_{x'} \big(U_{d+1}(x') - L_{d+1}(x')\big) N_d(x, u, x')$

  {Continue the rollout process and back-propagate the result}
  ROLLOUT($x', d, l + 1$)
  BELLMAN-BACKUP(x, d)
  return
end procedure

Algorithm 9 SBOSS (1/2)

function SEARCH(x, h)
  {Compute the transition matrix of the mean MDP of the posterior}
  $M_{mean} \leftarrow$ "Compute the mean MDP of $p^t_{\mathcal{M}}(.)$."
  $P_t \leftarrow P_{M_{mean}}$

  {Update the policy to follow if necessary}
  $\forall (x,u) : \Delta(x,u) = \sum_{y \in X} \frac{|P_t(x,u,y) - P_{lastUpdate}(x,u,y)|}{\sigma(x,u,y)}$
  if $t = 1$ or $\exists (x',u') : \Delta(x',u') > \delta$ **then**
    {Sample some transition vectors for each state-action pair}
    $S \leftarrow \{\}$
    for all $(x,u) \in X \times U$ **do**
      {Compute the number of transition vectors to sample for (x,u)}
      $K_t(x,u) \leftarrow \max_y \left\lceil \frac{\sigma^2(x,u,y)}{\epsilon} \right\rceil$

      {Sample $K_t(x,u)$ transition vectors from $< x,u >$, sampled from the posterior}
      for $1 \leq k \leq K_t(x,u)$ **do**
        $S \leftarrow S \cup$ "A transition vector from $< x,u >$, sampled from the posterior"
      end for
    end for

    $M^{\#} \leftarrow$ "Build a new MDP by merging all transitions from S"
    $\pi^*_{M^{\#}} \leftarrow$VALUE-ITERATION($M^{\#}$)
    $\pi_{SBOSS} \leftarrow$FIT-ACTION-SPACE($\pi^*_{M^{\#}}$)
    $P_{lastUpdate} \leftarrow P_t$
  end if

  {Return the optimal action in x w.r.t. π_{SBOSS}}
  return $u \sim \pi_{SBOSS}(x)$
end function

Algorithm 10 SBOSS (2/2)

function FIT-ACTION-SPACE($\pi^*_{M\#}$)
  for all $x \in X$ **do**
    $\pi(x) \leftarrow \pi^*_{M\#}(x) \mod |U|$
  end for

  return π
end function

procedure ONLINE-LEARNING(x, u, y, r)
  "Update the posterior w.r.t. transition $< x, u, y, r >$"
end procedure

Algorithm 11 BEB

procedure SEARCH(x, h)
  $M \leftarrow$ "Compute the mean MDP of $p^t_{\mathcal{M}}(.)$."

  {Add a bonus reward to all transitions}
  for $< x, u, y > \in \mathcal{X} \times \mathcal{U} \times \mathcal{X}$ **do**
    $\rho_M(x, u, y) \leftarrow \rho_M(x, u, y) + \frac{\beta}{c^{(t)}_{<x,u,y>}}$
  end for

  {Compute the optimal policy of the modified MDP}
  $\pi^*_M \leftarrow$VALUE-ITERATION(M)

  {Return the optimal action in x w.r.t. π^*_M}
  return $u \sim \pi^*_M(x)$
end procedure

procedure ONLINE-LEARNING(x, u, y, r)
  "Update the posterior w.r.t. transition $< x, u, y, r >$"
end procedure

4.6.2 MDP Distributions in Detail

In this section, we describe the MDPs drawn from the considered distributions in more detail. In addition, we also provide a formal description of the corresponding $\boldsymbol{\theta}$ (parameterising the FDM used to draw the transition matrix) and ρ_M (the reward function).

Generalised Chain Distribution

On those MDPs, we can identify two possibly optimal behaviours:

- The agent tries to move along the chain, reaches the last state, and collect as many rewards as possible before returning to State 1;
- The agent gives up to reach State 5 and tries to return to State 1 as often as possible.

Formal description $X = \{1, 2, 3, 4, 5\},\ U = \{1, 2, 3\}$

$$\forall u \in U :$$

$$\begin{aligned}
\theta^{GC}_{1,u} &= [1, 1, 0, 0, 0] \\
\theta^{GC}_{2,u} &= [1, 0, 1, 0, 0] \\
\theta^{GC}_{3,u} &= [1, 0, 0, 1, 0] \\
\theta^{GC}_{4,u} &= [1, 0, 0, 0, 1] \\
\theta^{GC}_{5,u} &= [1, 1, 0, 0, 1]
\end{aligned}$$

$$\forall x, u \in X \times U :$$

$$\begin{aligned}
\rho^{GC}(x, u, 1) &= 2.0 \\
\rho^{GC}(x, u, 5) &= 10.0 \\
\rho^{GC}(x, u, y) &= 0.0,\ \forall y \in X \setminus \{1, 5\}
\end{aligned}$$

Generalised Double-Loop Distribution

Similarly to the GC distribution, we can also identify two possibly optimal behaviours:

- The agent enters the "good" loop and tries to stay in it until the end;
- The agent gives up and chooses to enter the "bad" loop as frequently as possible.

Formal description $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\},\ U = \{1, 2\}$

$$\forall u \in U :$$

$$\begin{aligned}
\theta^{GDL}_{1,u} &= [0,1,0,0,0,1,0,0,0] \\
\theta^{GDL}_{2,u} &= [0,0,1,0,0,0,0,0,0] \\
\theta^{GDL}_{3,u} &= [0,0,0,1,0,0,0,0,0] \\
\theta^{GDL}_{4,u} &= [0,0,0,0,1,0,0,0,0] \\
\theta^{GDL}_{5,u} &= [1,0,0,0,0,0,0,0,0] \\
\theta^{GDL}_{6,u} &= [1,0,0,0,0,0,1,0,0] \\
\theta^{GDL}_{7,u} &= [1,0,0,0,0,0,0,1,0] \\
\theta^{GDL}_{8,u} &= [1,0,0,0,0,0,0,0,1] \\
\theta^{GDL}_{9,u} &= [1,0,0,0,0,0,0,0,0]
\end{aligned}$$

$$\forall u \in U :$$

$$\begin{aligned}
&\rho^{GDL}(5,u,1) = 1.0 \\
&\rho^{GDL}(9,u,1) = 2.0 \\
&\rho^{GDL}(x,u,y) = 0.0,\ \forall x \in X,\ \forall y \in X : y \neq 1
\end{aligned}$$

Grid Distribution

MDPs drawn from the Grid distribution are 2-dimensional grids. Since the agents considered do not manage multi-dimensional state spaces, the following bijection was defined:

$$\{1,2,3,4,5\} \times \{1,2,3,4,5\} \rightarrow X = \{1,2,\cdots,25\} : n(i,j) = 5(i-1)+j$$

where i and j are the row and column indexes of the cell on which the agent is.

When the agent reaches the **G** cell (in $(5,5)$), it is directly moved to $(1,1)$, and will perceive its reward of 10. In consequence, State $(5,5)$ is not reachable.

To move inside the Grid, the agent can perform four actions: $U = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. Those actions only move the agent to one adjacent cell. However, each action has a certain probability to fail (depending on the cell on which the agent is). In case of failure, the agent does not move at all. Besides, if the agent tries to move out of the grid, it will not move either. Discovering a reliable (and short) path to reach the **G** cell will determine the success of the agent.

Formal description $X = \{1,2,\cdots,25\}, U = \{\uparrow,\ \downarrow,\ \leftarrow,\ \rightarrow\}$

$$\begin{aligned}
&\forall (i,j) \in \{1,2,3,4,5\} \times \{1,2,3,4,5\} \\
&\forall (k,l) \in \{1,2,3,4,5\} \times \{1,2,3,4,5\} :
\end{aligned}$$

$$
\begin{aligned}
&\theta^{Grid}_{n(i,j),u}\ (n(i,j)) &&= 1,\ \forall u \in U\\
&\theta^{Grid}_{n(i,j),\uparrow}\ (n(i-1,j)) &&= 1, (i-1) \geq 1\\
&\theta^{Grid}_{n(i,j),\downarrow}\ (n(i+1,j)) &&= 1, (i+1) \leq 5, (i,j) \neq (4,5)\\
&\theta^{Grid}_{n(i,j),\leftarrow}\ (n(i,j-1)) &&= 1, (j-1) \geq 1\\
&\theta^{Grid}_{n(i,j),\rightarrow}\ (n(i,j+1)) &&= 1, (j+1) \leq 5, (i,j) \neq (5,4)\\
&\theta^{Grid}_{n(4,5),\downarrow}\ (n(1,1)) &&= 1\\
&\theta^{Grid}_{n(5,4),\rightarrow}(n(1,1)) &&= 1\\
&\theta^{Grid}_{n(i,j),u}\ (n(k,l)) &&= 0, else
\end{aligned}
$$

$$
\begin{aligned}
&\rho^{Grid}((4,5),\downarrow,\ (1,1)) = 10.0\\
&\rho^{Grid}((5,4),\rightarrow,(1,1)) = 10.0\\
&\rho^{Grid}((i,j),u,\ \ (k,l)) = 0.0,\ \forall u \in U
\end{aligned}
$$

4.6.3 Paired Sampled Z-test

Let π_A and π_B be the two agents we want to compare. We played the two agents on the same N MDPs, denoted by $M_1, \cdots, M_N$. Let $R^{\pi_A}_{M_i}$ and $R^{\pi_B}_{M_i}$ be the scores we observed for the two agents on M_i.

Step 1 - Hypothesis

We compute the mean and the standard deviation of the differences between the two sample sets, denoted by $\bar{x}_d$ and $\bar{s}_d$, respectively.

$$
\begin{aligned}
\bar{x}_d &= \frac{1}{N}\sum_{i=1}^{N} R^{\pi_A}_{M_i} - R^{\pi_B}_{M_i}\\
\bar{s}_d &= \frac{1}{N}\sum_{i=1}^{N} (\bar{x}_d - (R^{\pi_A}_{M_i} - R^{\pi_B}_{M_i}))^2
\end{aligned}
$$

If $N \geq 30$, $\bar{s}_d$ is a good estimation of σ_d, the standard deviation of the differences between the two populations ($\bar{s}_d \approx \sigma_d$). In order words, σ_d is the standard deviation we should observe when testing the two algorithms on a number of MDPs tending towards infinity. This was always the case in our experiments.

We now set Hypothesis H_0 and Hypothesis H_α:

$$
\begin{aligned}
H_0 &: \mu_d = 0\\
H_\alpha &: \mu_d > 0
\end{aligned}
$$

Our goal is to determine if μ_d, the mean of the differences between the two populations, is equal or greater than 0. More expressly, we

want to know if the differences between the two agents' performances is significant (H_α is correct) or not (H_0 correct). Only one of those hypotheses can be true.

Step 2 - Test statistic

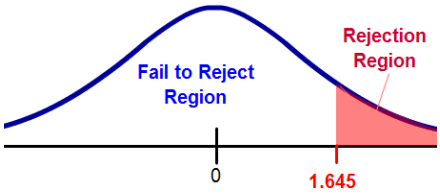
The test statistic consists to compute a certain value Z:

$$Z = \frac{\bar{x_d}}{\sigma_d / \sqrt{N}}$$

This value will help us to determine if we should accept (or reject) hypothesis H_α.

Step 3 - Rejection region

Assuming we want our decision to be correct with a probability of failure of α, we will have to compare Z with Z_α, a value of a Gaussian curve. If $Z > Z_\alpha$, it means we are in the rejection region (R.R.) with a probability equal to $1 - \alpha$. For a confidence of 95%, Z_α should be equal to 1.645.



Being in the R.R. means we have to reject Hypothesis H_0 (and accept Hypothesis H_α). In the order case, we have to accept Hypothesis H_0 (and reject Hypothesis H_α).

Step 4 - Decision

At this point, we have either accepted Hypothesis H_0 or Hypothesis H_α.

- **Accepting Hypothesis H_0 ($Z < Z_\alpha$)**: The two algorithms π_A and π_B are not significantly different.
- **Accepting Hypothesis H_α ($Z \geq Z_\alpha$)**: The two algorithms π_A and π_B are significantly different. Therefore, the algorithm with the greatest mean is definitely better with 95% confidence.

Chapter 5

Approximate Bayes Optimal Policy Search using Neural Networks

Bayesian Reinforcement Learning (BRL) agents aim to maximise the expected collected rewards obtained when interacting with an unknown Markov Decision Process (MDP) while using some prior knowledge. State-of-the-art BRL agents rely on frequent updates of the belief on the MDP, as new observations of the environment are made. This offers theoretical guarantees to converge to an optimum, but is computationally intractable, even on small-scale problems. In this paper, we present a method that circumvents this issue by training a parametric policy able to recommend an action directly from raw observations. Artificial Neural Networks (ANNs) are used to represent this policy, and are trained on the trajectories sampled from the prior. The trained model is then used online, and is able to act on the real MDP at a very low computational cost. Our new algorithm shows strong empirical performance, on a wide range of test problems, and is robust to inaccuracies of the prior distribution.

The work presented in this chapter has been published in the *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2017)* (Castronovo et al., 2017).

5.1 Introduction

Bayes-Adaptive Markov Decision Processes (BAMDP) (Silver, 1963; Martin, 1967) form a natural framework to deal with sequential decision-making problems when some of the information is hidden. In these problems, an agent navigates in an initially unknown environment and receives a numerical reward according to its actions. However, actions that yield the highest instant reward and actions that maximise the gathering of knowledge about the environment are often different. The BAMDP framework leads to a rigorous definition of an optimal solution to this learning problem, which is based on finding a policy that reaches an optimal balance between exploration and exploitation.

In this research, the case where prior knowledge is available about the environment is studied. More specifically, this knowledge is represented as a random distribution over possible environments, and can be updated as the agent makes new observations. In practice, this happens for example when training a drone to fly in a safe environment before sending it on the operation field (Zhang et al., 2015). This is called offline training and can be beneficial to the online performance in the real environment, even if prior knowledge is inaccurate (Castronovo, Fonteneau, and Ernst, 2014).

State-of-the-art Bayesian algorithms generally do not use offline training. Instead, they rely on Bayes updates and sampling techniques during the interaction, which may be too computationally expensive, even on very small MDPs (Castronovo et al., 2016). In order to reduce significantly this cost, we propose a new practical algorithm to solve BAMDPs: Artificial Neural Networks for Bayesian Reinforcement Learning (ANN-BRL). Our algorithm aims at finding an optimal policy, i.e. a mapping from observations to actions, which maximises the rewards in a certain environment. This policy is trained to act optimally on some MDPs sampled from the prior distribution, and then it is used in the test environment. By design, our approach does not use any Bayes update, and is thus computationally inexpensive during online interactions. Our policy is modelled as an ensemble of ANNs, combined by using SAMME (Zhu et al., 2009), a boosting algorithm.

Artificial Neural Networks offer many advantages for the needed purpose. First, they are able to learn complex functions and are, thus, capable of encoding almost any policy. Second, ANNs can be trained very efficiently, using the backpropagation method, even on a large dataset. Lastly, ANNs' forward pass is fast, which makes them ideal to perform predictions during the online phase, when the computation time constraints are tight.

In our experiments, we used a benchmark recently introduced in (Castronovo et al., 2016). It compares all the major state-of-the-art BRL algorithms on a wide array of test problems, and provides a detailed

computation time analysis. Since most state-of-the-art agents found in the literature are not any time algorithms, this last feature is very useful to compare solvers that have different time constraints.

This paper is organised as follows: Section 5.2 gives an overview of the state-of-the-art in Bayesian Reinforcement Learning. Section 5.3 presents the problem statement. Section 5.4 describes the algorithm. Section 5.5 shows a comparison between our algorithm and state-of-the-art algorithms of the domain. Section 5.6 offers a conclusion and discusses future work.

5.2 State-of-the-art

Bayesian Reinforcement Learning (BRL) algorithms rely on Bayesian updates of the prior knowledge on the environment as new observations are made.

Model-based approaches maintain explicitly a posterior distribution, given the prior and the transitions observed so far. Bayes-adaptive Monte Carlo Planning (BAMCP) (Guez, Silver, and Dayan, 2012) and Bayesian Forward Search Sparse Sampling (BFS3) (Asmuth and Littman, 2011) rely on the exploration of the belief state space with a belief-lookahead (BL) approach. In this case, the posterior is used to explore efficiently the look-ahead tree and estimate the Q-values of the current belief-state. The accuracy is depending on the number of nodes those algorithms are able to visit, which is limited by an on-line computation time budget. Despite theoretical guarantees to reach Bayesian optimality offered by BL approaches[1], they may not be applicable when the time budget that can be allocated for on-line decision making is short (Castronovo et al., 2016). Another method, Smarter Best of Sampled Set (SBOSS) (Castro and Precup, 2010), samples several MDPs from the posterior distribution, builds a merged MDP, and computes its Q-function. The number of MDPs to sample and the frequency at which a merged MDP has to be built is determined by uncertainty bounds on the Q-values. As a consequence, the online computation time of SBOSS may vary at each time-step. However, the number of samples and the frequency are depending on two parameters, which are used to fix the online computation time *on average*. More computation time improves the accuracy of the computed Q-values. However, on the downside, this approach remains computationally expensive (Castronovo et al., 2016).

On the other hand, model-free approaches only maintain a list of the transitions observed, and compute value functions. In this case, the prior distribution is used to initialise this list (e.g.: a uniform distribution consisting to assume each transition has been observed once). Bayesian Exploration Bonus (BEB) (Kolter and Ng, 2009) builds the expected MDP given the current history at each time-step. The

[1] e.g. BAMCP (Guez, Silver, and Dayan, 2012).

reward function of this MDP is slightly modified to give an exploration bonus to transitions which have been observed less frequently. The optimal Q-function of this MDP is then used to determine which action to perform. BEB is a simple, but efficient algorithm that remains computationally inexpensive for accurate prior distributions. Nevertheless, BEB's performance drops significantly for inaccurate prior distributions (Castronovo et al., 2016).

Another approach was proposed a few years ago with Offline Prior-based Policy Search (OPPS) (Castronovo et al., 2012; Castronovo, Fonteneau, and Ernst, 2014). During an offline phase, OPPS builds a discrete set of E/E strategies, and identifies which strategy of the set is the most efficient on average, to address any MDP drawn from the prior distribution. Instead of evaluating the performance of each strategy with the same accuracy, OPPS uses a multi-armed bandit strategy to discard gradually the worst strategies. This idea allows OPPS to consider a strategy space large enough to contain good candidates for many problems. Besides, the E/E strategies considered are computationally inexpensive for on-line decision making, but the approach lacks theoretical guarantees (Castronovo et al., 2016).

A more detailed description of each algorithm is available in the Appendix 5.6.1.

5.3 Preliminaries

5.3.1 Bayes Adaptive Markov Decision Process

We, hereafter, describe the formulation of optimal decision-making in a BAMDP. Let $M = (X, U, f(\cdot), \rho_M, \gamma)$ be a given unknown MDP, where

- $X = \{x^{(1)}, \dots, x^{(n_X)}\}$ denotes its finite state space
- $U = \{u^{(1)}, \dots, u^{(n_U)}\}$ denotes its finite action space
- $r_t = \rho_M(x_t, u_t, x_{t+1}) \in [R_{\min}, R_{\max}]$ denotes an instantaneous deterministic, bounded reward
- $\gamma > 0$ its discount factor

When the MDP is in state x_t at time t and action u_t is selected, the agent moves instantaneously to a next state x_{t+1} with a probability $P(x_{t+1}|x_t, u_t) = f(x_t, u_t, x_{t+1})$. In the BAMDP setting, the dynamics are unknown, and we assume that f is drawn according to a known distribution $P(f)$. Such a probability distribution is called a prior distribution; it represents what the MDP is believed to be before interacting with it. Let $h_t = (x_0, u_0, r_0, x_1, \cdots, x_{t-1}, u_{t-1}, r_{t-1}, x_t)$ denote the history observed until time t. Given the current history h_t, a policy π returns an action $u_t = \pi(h_t)$. Given an MDP M and a policy

π, we define the cost $\mathfrak{J}_M^\pi = \mathbb{E}_M^\pi \left[\sum_t \gamma^t r_t\right]$ as the expected cumulated discounted reward on M, when applying policy π. Given a prior distribution $p_{\mathcal{M}}^0(\cdot)$, the goal is to find a policy π^*, called *Bayes optimal* that maximises the expected cost with respect to the prior distribution:

$$\pi^* = \arg\max_\pi \mathop{\mathbb{E}}_{M \sim p_{\mathcal{M}}^0(\cdot)} \mathfrak{J}_M^\pi \tag{5.1}$$

It is important to note that although this policy is good on average, with respect to the prior, it does not necessarily perform efficiently on each MDP sampled from the prior. Conversely, given a fixed and fully known MDP M, a policy that is optimal on M is likely to be very different from π^*.

5.3.2 Solving BAMDP

Though solving a BAMDP exactly is theoretically well defined, it is intractable in practice (Guez, Silver, and Dayan, 2013) for two reasons. First, sampling possible transition probabilities, based on past observations, relies on the computation of $P(f|h_t) \propto P(h_t|f)P(f)$, which is intractable for most probabilistic models (Duff, 2002; Kaelbling, Littman, and Cassandra, 1998; Kolter and Ng, 2009). Second, the BAMDP state space is actually made of all possible histories and is infinite. Therefore, all known tractable algorithms rely on some form of approximation. They can be divided in two main classes: online methods, and offline methods. The former group (Fonteneau, Busoniu, and Munos, 2013; Asmuth and Littman, 2011; Walsh, Goschin, and Littman, 2010; Kolter and Ng, 2009) relies on sparse sampling of possible models based on the current observations, to reduce the number of transition probabilities computations. The latter group (Wang et al., 2012) uses the prior knowledge to train an agent able to act on all possible sequences of observations. Our approach belongs to this group, and is described in Section 5.4.

5.4 Algorithm Description

A Bayes optimal policy π^*, as defined by Eq. 5.1, maps histories to Bayes actions. Although π^* is unknown, an approximation may be computed. Let π_θ be a parametric policy whose model parameters are θ. The model is fed up with the current history h_t, and computes an output vector, associating a confidence score to each action in return. The agent simply selects the action with the highest score.

Our model is composed of several ANNs, where the model parameters, denoted by θ, are the weights of all the networks. All ANNs are fed up with the same inputs, and build several output vectors which are merged by using a weighted linear combination.

The training of this model requires a training dataset, whose generation is described in Section 5.4.1. It consists in performing simulations on MDPs drawn from the prior distribution to generate a training set. To each history observed during these simulations, we recommend an optimal action. Each $<$ history, recommended action $>$ pair is a sample of the training dataset.

A history is a series of transitions whose size is unbounded, but ANNs can only be fed up with input vectors of a fixed size. To address this issue, histories are processed into fixed-size input vectors prior to training our model. This procedure is described in Section 5.4.2.

More specifically, the ANNs are built iteratively by using SAMME — an Adaboosting algorithm. It consists in modifying the training dataset in order to increase the weights of the samples misclassified by the ANNs built previously. Section 5.4.3 details the SAMME algorithm and the necessary changes to fit the BRL setting.

Moreover, we also add pseudo-code descriptions in both offline and online phases (Algorithm 12 and Algorithm 13 respectively) along with UML diagrams (Figure 5.1 and Figure 5.2 respectively).

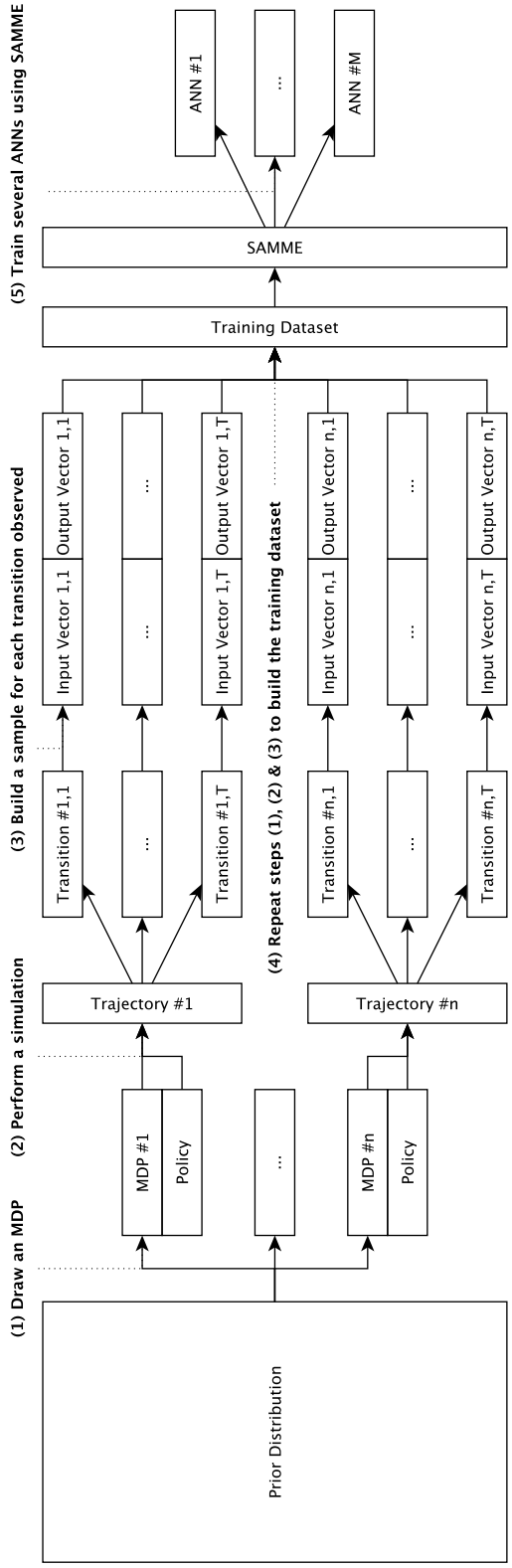


FIGURE 5.1: ANN-BRL - Offline phase

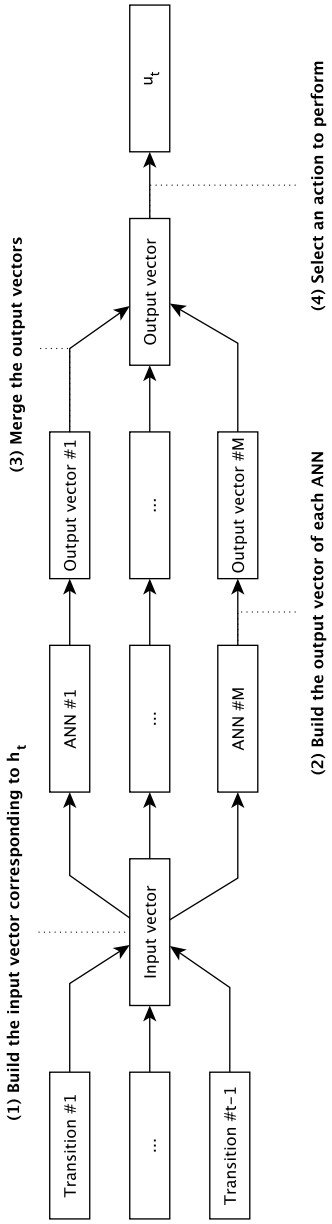


FIGURE 5.2: ANN-BRL - Online phase

5.4.1 Generation of the Training Dataset

During the offline phase, we use the prior knowledge to generate samples which will compose the training dataset. For a given series of observations h_t, we consider the optimal action w.r.t. the MDP from which h_t has been generated. In other words, we give a label of 1 to actions that are optimal when the transition function $f(.)$ is known, and -1 to the others.

Our dataset is, thus, filled with suboptimal recommendations, from the Bayes optimal perspective. However, our samples are generated from multiple MDPs which are themselves sampled from the

Algorithm 12 ANN-BRL - Offline phase

Input: Time horizon T, prior distribution $p^0_{\mathcal{M}}(.)$
Output: A classifier $\mathcal{C}(.)$

{Generate transitions}
for $i = 1$ **to** n **do**
  $M^{(i)} \sim p^0_{\mathcal{M}}(.)$
  $H^{(i)} \leftarrow$ "Simulate 1 trajectory of length T on $M^{(i)}$"
end for

{Compute input/output vectors for each transition}
for $i = 1$ **to** n **do**
  $h_T \leftarrow H^{(i)}$
  for $j = 1$ **to** T **do** {Compute the input vector of sample (i, j)}
    $h_j \leftarrow \left(h_T^{(1)}, \ldots, h_T^{(j)}\right)$
    $\varphi_{i,j} \leftarrow$ Reprocess h_j

    {Compute the output vector of sample (i, j)}
    $Q^*_{i,j} \leftarrow$ Q-Iteration$(M^{(i)}, T)$
    for $k = 1$ **to** n_U **do**
      if k maximises $Q^*_{i,j}(x, u^{(\cdot)})$ **then**
        $output^{(k)}_{i,j} = 1$
      else
        $output^{(k)}_{i,j} = -1$
      end if
    end for

    $DataSet^{(i,j)} \leftarrow \{\varphi_{i,j}, output_{i,j}\}$
  end for
end for

{Train a model and compute a policy}
$\mathcal{C}(.) \leftarrow$ Run *SAMME* on $DataSet$

Algorithm 13 ANN-BRL - Online phase

Input: Prior distribution $p^0_{\mathcal{M}}(.)$, current history $h_t = (x_0, u_0, r_0, x_1, \cdots, x_{t-1}, u_{t-1}, r_{t-1}, x_t)$, classifier $\mathcal{C}(.)$
Output: u_t, the action to perform at time-step t

{Compute the input vector}
$\varphi_t \leftarrow$ Reprocess h_t
$input \leftarrow \varphi_t$

{Compute the output vector}
$output \leftarrow \mathcal{C}(input)$

{Choose action u_t w.r.t. the output vector}
$k \leftarrow k$ maximising $output^{(\cdot)}$
$u_t \leftarrow u^{(k)}$

prior distribution. As a consequence, a history h can appear multiple times in our dataset but with different output vectors, because it has been generated from different MDPs for which the labels were different. The average output vector for a history h approximates the probability of each action u to be the optimal response to h when $f_M(.)$ is known, where $M \sim p^0_{\mathcal{M}}(\cdot)$. To a certain extent, it is similar to what is done by other BRL algorithms, such as BAMCP (Guez, Silver, and Dayan, 2012) when it explores a specific part of the belief-states space using Tree-Search techniques.

During the data generation phase, it is necessary to choose which parts of the state space to explore. Generating samples by following what is believed to be an optimal policy is likely to provide examples in rewarding areas of the state space, but only for the current MDP. Since it is not possible to know in advance which MDPs our agent will encounter during the online phase, we choose to induce some random exploration in the data generation process. More precisely, we define an ϵ-Optimal agent, which makes optimal decisions[2] w.r.t. to the MDP with a probability $1 - \epsilon$, and random decisions otherwise. By varying the value of $0 < \epsilon < 1$ from one simulation to another, we are able to cover the belief-states space more efficiently than using a random agent.

5.4.2 Reprocess of a History

The raw input fed to our model is h_t, an ordered series of observations up to time t. In order to simplify the problem and reduce training time, a data preprocessing step is applied to reduce h_t to a fixed number of features $\varphi_{h_t} = [\ \varphi^{(1)}_{h_t}, \ldots, \varphi^{(N)}_{h_t}\ ],\ N \in \mathbb{N}$. There are two types of

[2]By optimal we mean the agent knows the transition matrix of the MDP, and solve it in advance.

features that are considered in this paper: Q-values and transition counters.

Q-values are obtained by building an approximation of the current MDP from h_t and computing its Q-function, thanks to the well-known Q-Iteration algorithm (Sutton and Barto, 1998). Each Q-value defines a different feature:

$$\varphi_{h_t} = [\ Q_{h_t}(x^{(1)}, u^{(1)}),\ \ldots,\ Q_{h_t}(x^{(n_X)}, u^{(n_U)})\ ]$$

A transition counter represents the number of occurrences of specific transition in h_t. Let $C_{h_t}(< x, u, x' >)$ be the transition counter of transition $< x, u, x' >$. The number of occurrences of all transitions defines the following features:

$$\varphi_{h_t} = [\ C_{h_t}(< x^{(1)}, u^{(1)}, x^{(1)} >),\ \ldots,\ C_{h_t}(< x^{(n_X)}, u^{(n_U)}, x^{(n_X)} >)\ ] \quad (5.2)$$

At this stage, we computed a set of features which do not take into account the order of appearance of each transition. We consider that this order is not necessary as long as the current state x_t is known. In this paper, two different cases have been studied:

1. **Q-values:** We consider the set of all Q-values defined above. However, in order to take x_t into account, those which are not related to x_t are discarded.

$$\varphi_{h_t} = [\ Q_{h_t}(x_t, u^{(1)}),\ \ldots,\ Q_{h_t}(x_t, u^{(n_U)})\ ]$$

2. **Transition counters:** We consider the set of all transition counters defined above to which we add x_t as an extra feature.

$$\varphi_{h_t} = [\ C_{h_t}(< x^{(1)}, u^{(1)}, x^{(1)} >),\ \ldots,\ C_{h_t}(< x^{(n_X)}, u^{(n_U)}, x^{(n_X)} >),\ x_t\ ] \quad (5.3)$$

5.4.3 Model Definition and Training

The policy is now built from the training dataset by supervised learning on the multi-class classification problem where the classes c are the actions, and the vectors v are the histories. SAMME has been chosen to address this problem. It is a boosting algorithm which directly extends Adaboost from the two-class classifcation problem to the multi-class case. As a reminder, a full description of SAMME is provided in Appendix 5.6.2.

SAMME builds iteratively a set of weak classifiers in order to build a strong one. In this paper, the weak classifiers are neural networks in the form of multilayer perceptrons (MLPs). SAMME

Algorithm 14 Resampling algorithm

Input: The original training dataset $DataSet$ (size = N), a set of weights $w_1, \ldots, w_N$
Output: A new training dataset $DataSet'$ (size = p)

{Normalise w_k such that $0 \leq \bar{w}_k \leq 1$ and $\sum_k \bar{w}_k = 1$}
for $i = 1$ **to** N **do**
  $\bar{w}_k \leftarrow \frac{w_k}{\sum_{k'} w_{k'}}$
end for

{Resample $DataSet$}
for $i = 1$ **to** p **do**
  $DataSet'(i) \leftarrow$ "Draw a sample s from $DataSet$"
  {$P(s = DataSet(k))$ is equal to $\bar{w}_k, \forall k$)}
end for

algorithm aims to allow the training of a weak classifier to focus on the samples misclassified by the previous weak classifiers. This results in associating weights to the samples which reflect how bad the previous weak classifiers are for this sample.

MLPs are trained by backpropagation[3], which does not support weighted samples. Schwenk et al. presented different resampling approaches to address this issue with neural networks in (Schwenk and Bengio, 2000). The approach we have chosen samples from the dataset by interpreting the (normalised) weights as probabilities. Algorithm 14 describes it formally.

One of the specificities of the BRL formalisation lies in the definition of the classification error δ of a specific sample. This value is critical for SAMME in the evaluation of the performances of an MLP and the tuning of the sample weights. Our MLPs do not recommend specific actions, but rather give a confidence score to each one. As a consequence, different actions can receive the same level of confidence by our MLP(s), in which case the agent will break the tie by selecting one of those actions randomly. Therefore, we define the classification error δ as the probability for an agent following a weak classifier $\mathcal{C}'(.)$ (= an MLP) to select the class c associated to a sample v ($< v, c >$ being an $<$ history, recommended action $>$ pair):

[3]In order to avoid overfitting, the dataset is divided into two sets: a learning set (LS) and a validation set (VS). The training is terminated once it begins to be less efficient on VS. The samples are distributed 2/3 for LS and 1/3 for VS.

$$u^* = u^{(c)},\ \hat{p} = \mathcal{C}'(v)$$
$$\hat{U} = \{u \in U \mid u = \arg\max_u\ \hat{p}_u\}$$
$$\delta = \frac{|\hat{U} \setminus \{u^*\}|}{|\hat{U}|}$$

5.5 Experiments

5.5.1 Experimental Protocol

In order to empirically evaluate our algorithm, it is necessary to measure its expected return on a test distribution $p_{\mathcal{M}}$, after an offline training on a prior distribution $p^0_{\mathcal{M}}$. Given a policy π, we denote this expected return $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}} = \mathop{\mathbb{E}}_{M\sim p_{\mathcal{M}}(\cdot)}\left[\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{M}\right]$. In practice, we can only approximate this value. The steps to evaluate an agent π are defined as follows:

1. Train π offline on $p^0_{\mathcal{M}}$
2. Sample N MDPs from the test distribution $p_{\mathcal{M}}$[4]
3. For each sampled MDP M, compute estimate of $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{M}$
4. Use these values to compute an empirical estimate of $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}}$

To estimate $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{M}$, the expected return of agent π trained offline on $p^0_{\mathcal{M}}$, we sample one trajectory on the MDP M, and compute the truncated cumulated return up to time T. The constant T is chosen so that the approximation error is bounded by $\epsilon = 0.01$.

Finally, to estimate our comparison criterion $\mathfrak{J}^{\pi(p^0_{\mathcal{M}})}_{p_{\mathcal{M}}}$, we compute the empirical average of the algorithm performance over N different MDPs, sampled from $p_{\mathcal{M}}$. For all our experiments, we report the measured values along with the corresponding 0.95 confidence interval.

The results will allow us to identify, for each experiment, the most suitable algorithm(s) depending on the constraints the agents must satisfy. Note that this protocol has been first presented in more details in (Castronovo et al., 2016).

[4]In practice, we can only sample a finite number of trajectories, and must rely on estimators to compare algorithms.

5.5.2 Algorithms Comparison

In our experiment, the following algorithms have been tested, from the most elementary to the state-of-the-art BRL algorithms: Random, ϵ-Greedy, Soft-max, OPPS-DS (Castronovo et al., 2012; Castronovo, Fonteneau, and Ernst, 2014), BAMCP (Guez, Silver, and Dayan, 2012), BFS3 (Asmuth and Littman, 2011), SBOSS (Castro and Precup, 2010), and BEB (Kolter and Ng, 2009). For detailed information on an algorithm and its parameters, please refer to the Appendix 5.6.1.

Most of the above algorithms are not any-time methods, i.e. they cannot be interrupted at an arbitrary time and yield a sensible result. Given an arbitrary time constraint, some algorithms may just be unable to yield anything. And out of those that do yield a result, some might use longer time than others. To give a fair representation of the results, we simply report, for each algorithm and each test problem, the recorded score (along with confidence interval), and the computation time needed. We can then say, for a given time constraint, what the best algorithms to solve any problem from the benchmark are.

5.5.3 Benchmarks

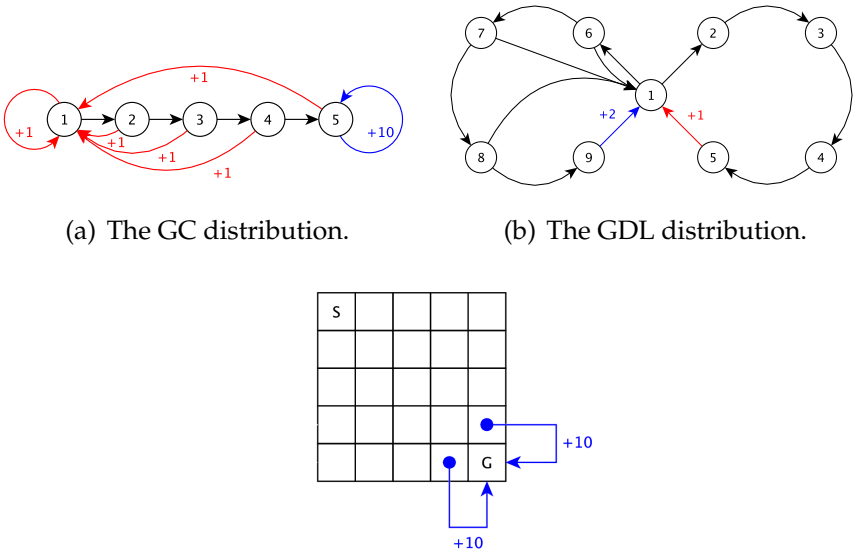
In our setting, the transition matrix is the only element which differs between two MDPs drawn from the same distribution. Generating a random MDP is, therefore, equivalent to generating a random transition matrix. In the BRL community, a common distribution used to generate such matrices is the Flat Dirichlet Multinomial distribution (FDM). It is chosen for the ease of its Bayesian updates. A FDM is defined by a parameter vector that we call θ.

We study two different cases: when the prior knowledge is accurate, and when it is not. In the former, the prior distribution over MDPs, called $p^{\theta_0}_{\mathcal{M}}(.)$, is exactly equal to the test distribution that is used during online training, $p^{\theta}_{\mathcal{M}}(.)$. In the latter, the inaccuracy of the prior means that $p^{\theta_0}_{\mathcal{M}}(.) \neq p^{\theta}_{\mathcal{M}}(.)$.

Sections 5.5.3, 5.5.3 and 5.5.3 describes the three distributions considered for this study.

Generalised Chain Distribution

The Generalised Chain (GC) distribution is inspired from the 5-states chain problem (5 states, 3 actions) (Dearden, Friedman, and Russell, 1998). The agent starts at state 1, and has to go through state 2, 3 and 4 in order to reach the last state, state 5, where the best rewards are. This cycle is illustrated in Figure 5.3(a).



(a) The GC distribution.

(b) The GDL distribution.

(c) The Grid distribution.

FIGURE 5.3: Studied distributions for benchmarking.

Generalised Double-Loop Distribution

The Generalised Double-Loop (GDL) distribution is inspired from the double-loop problem (9 states, 2 actions) (Dearden, Friedman, and Russell, 1998). Two loops of 5 states are crossing at state 1 (where the agent starts) and one loop yields more rewards than the other. This problem is represented in Figure 5.3(b).

Grid Distribution

The Grid distribution is inspired from the Dearden's maze problem (25 states, 4 actions) (Dearden, Friedman, and Russell, 1998). The agent is placed at a corner of a 5x5 grid (the **S** cell), and has to reach the goal corner (the **G** cell). The agent can perform 4 different actions, corresponding to the 4 directions (up, down, left, right), but the actual transition probabilities are conditioned by the underlying transition matrix. This benchmark is illustrated in Figure 5.3(c).

5.5.4 Results

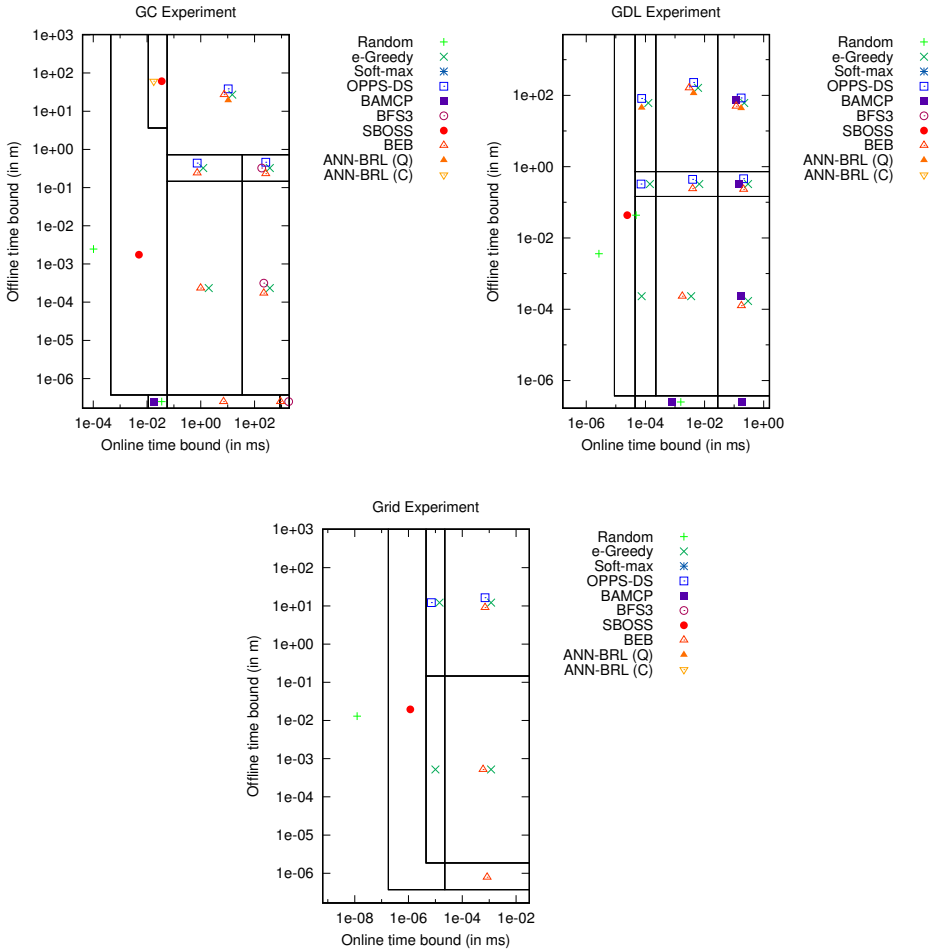


FIGURE 5.4: Best algorithms w.r.t offline/online periods (accurate case)

Agent	Score on GC	Score on GDL	Score on Grid
Random	31.12 ± 0.90	2.79 ± 0.07	0.22 ± 0.06
e-Greedy	40.62 ± 1.55	3.05 ± 0.07	6.90 ± 0.31
Soft-Max	34.73 ± 1.74	2.79 ± 0.10	0.00 ± 0.00
OPPS-DS	42.47 ± 1.91	3.10 ± 0.07	7.03 ± 0.30
BAMCP	35.56 ± 1.27	3.11 ± 0.07	6.43 ± 0.30
BFS3	39.84 ± 1.74	2.90 ± 0.07	3.46 ± 0.23
SBOSS	35.90 ± 1.89	2.81 ± 0.10	4.50 ± 0.33
BEB	41.72 ± 1.63	3.09 ± 0.07	6.76 ± 0.30
ANN-BRL (Q)	42.01 ± 1.80	3.11 ± 0.08	6.15 ± 0.31
ANN-BRL (C)	35.95 ± 1.90	2.81 ± 0.09	4.09 ± 0.31

TABLE 5.1: Best algorithms w.r.t Performance (accurate case)

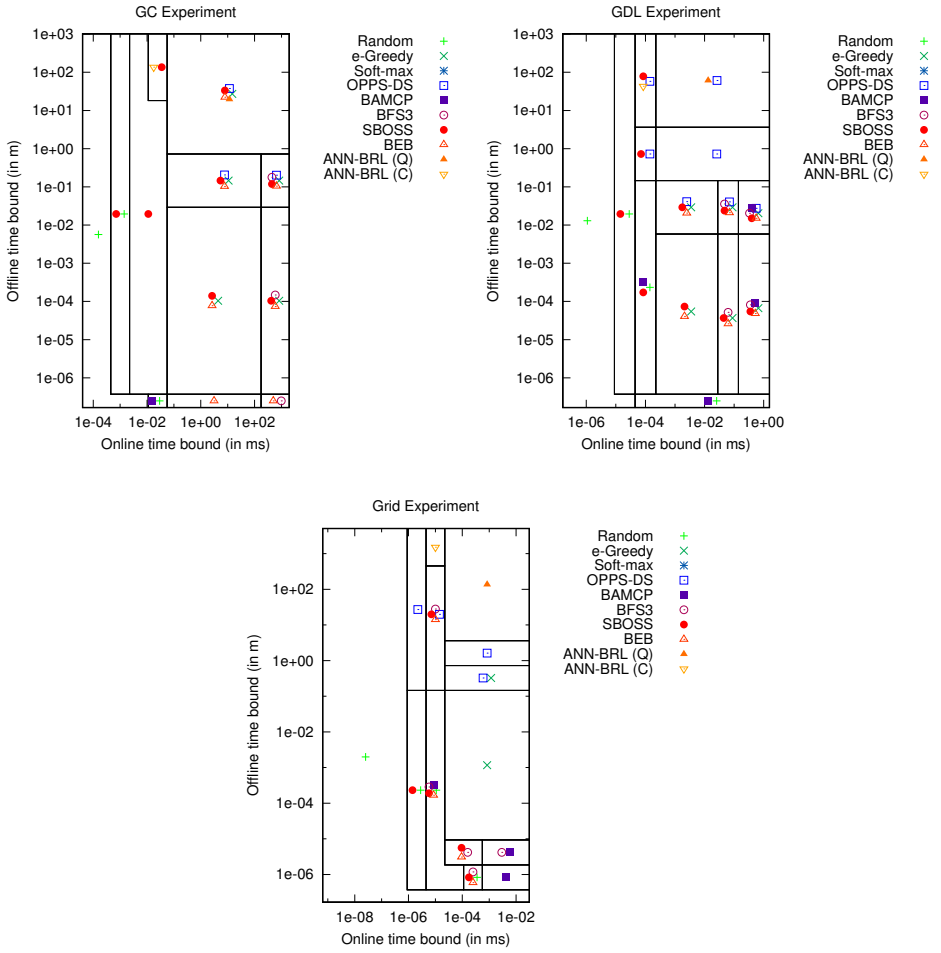


FIGURE 5.5: Best algorithms w.r.t offline/online time (inaccurate case)

Agent	**Score on GC**	**Score on GDL**	**Score on Grid**
Random	31.67 ± 1.05	2.76 ± 0.08	0.23 ± 0.06
e-Greedy	37.69 ± 1.75	2.88 ± 0.07	0.63 ± 0.09
Soft-Max	34.75 ± 1.64	2.76 ± 0.10	0.00 ± 0.00
OPPS-DS	39.29 ± 1.71	2.99 ± 0.08	1.09 ± 0.17
BAMCP	33.87 ± 1.26	2.85 ± 0.07	0.51 ± 0.09
BFS3	36.87 ± 1.82	2.85 ± 0.07	0.42 ± 0.09
SBOSS	38.77 ± 1.89	2.86 ± 0.07	0.29 ± 0.07
BEB	38.34 ± 1.62	2.88 ± 0.07	0.29 ± 0.05
ANN-BRL (Q)	38.76 ± 1.71	2.92 ± 0.07	4.29 ± 0.22
ANN-BRL (C)	36.30 ± 1.82	2.84 ± 0.08	0.91 ± 0.15

TABLE 5.2: Best algorithms w.r.t Performance (inaccurate case)

For each experiment, we tested each algorithm with several values for their parameter(s). The values considered in this paper are detailed in Appendix 5.6.1. Three pieces of information have been measured for each test: (i) an empirical score, obtained by testing the agent on 500 MDPs drawn from the test distribution[5]; (ii) a mean online computation time, corresponding to the mean time taken by the agent for performing an action; (iii) an offline computation time, corresponding to the time consumed by the agent while training on the prior distribution[6].

Each of the plots in Fig. 5.4 and Fig. 5.5 present a 2-D graph, where the X-axis represents a mean online computation time constraint, while the Y-axis represents an offline computation time constraint. For each point of the graph: (i) all agents that do not satisfy the constraints are discarded; (ii) for each algorithm, the agent leading to the best performance in average is selected; (iii) the list of agents whose performances are not significantly different is built. For this purpose, a paired sampled Z-test (with a confidence level of 95%) has been used to discard the agents which are significantly worst than the best one. Since several algorithms can be associated to a single point, several boxes have been drawn to gather the points which share the same set of algorithms.

Accurate Case

In Table 5.1, it is noted that ANN-BRL (Q)[7] gets extremely good scores on the two first benchmarks. When taking into account time constraints, ANN-BRL (Q) requires a slightly higher offline time bound to be on par with OPPS, and can even surpass it on the last benchmark as shown in Fig. 5.4.

ANN-BRL (C)[8] is significantly less efficient than ANN-BRL (Q) on the first and last benchmarks. The difference is less noticeable in the second one.

Inaccurate Case

Similar results have been observed for the inaccurate case and can be shown in Fig. 5.5 and Table 5.2 except for the last benchmark : ANN-BRL (Q) obtained a very high score, 4 times larger than the one measured for OPPS-DS. It is even more noteworthy that such a difference is observed on the most difficult benchmark. In terms of time constraints, ANN-BRL (Q) is still very close to OPPS-DS except for the last benchmark, where ANN-BRL (Q) is significantly better than the others above certain offline/online time periods.

[5]The same MDPs are used for comparing the agents. This choice has been made to reduce drastically the variance of the mean score.

[6]Notice that some agents do not require an offline training phase.

[7]Refers to ANN-BRL using Q-values as its features.

[8]Refers to ANN-BRL using transition counters as its features.

Another difference is that even though ANN-BRL (C) is still outperformed by ANN-BRL (Q), Fig. 5.5 reveals some cases where ANN-BRL (C) outperforms (or is on par with) all other algorithms considered. This occurs because ANN-BRL (C) is faster than ANN-BRL (Q) during the online phase, which allows it to comply with smaller online time bounds.

5.6 Conclusion and Future work

We developed ANN-BRL, an offline policy-search algorithm for addressing BAMDPs. As shown by our experiments, ANN-BRL obtained state-of-the-art performance on all benchmarks considered in this paper. In particular, on the most challenging benchmark[9], a score 4 times higher than the one measured for the second best algorithm has been observed. Moreover, ANN-BRL is able to make online decisions faster than most BRL algorithms.

Our idea is to define a parametric policy as an ANN, and train it using backpropagation algorithm. This requires a training set made of observations-action pairs and in order to generate this dataset, several simulations have been performed on MDPs drawn from prior distribution. In theory, we should label each example with a Bayes optimal action. However, those are too expensive to compute for the whole dataset. Instead, we chose to use optimal actions under full observability hypothesis. Due to the modularity of our approach, a better labelling technique could easily be integrated in ANN-BRL, and may bring stronger empirical results.

Moreover, two types of features have been considered for representing the current history: Q-values and transition counters. The use of Q-values allows to reach state-of-the-art performance on most benchmarks and outperfom all other algorithms on the most difficult one. On the contrary, computing a good policy from transition counters only is a difficult task to achieve, even for Artificial Neural Networks. Nevertheless, we found that the difference between this approach and state-of-the-art algorithms was much less noticeable when prior distribution differs from test distribution, which means that at least in some cases, it is possible to compute efficient policies without relying on online computationally expensive tools such as Q-values.

An important future contribution would be to provide theoretical error bounds in simple problems classes, and to evaluate the performance of ANN-BRL on larger domains that other BRL algorithms might not be able to address.

[9] *Grid* benchmark with a uniform prior.

Appendices

5.6.1 BRL Algorithms

Each algorithm considered in our experiments is detailed precisely. For each algorithm, a list of "reasonable" values is provided to test each of their parameters. When an algorithm has more than one parameter, all possible parameter combinations are tested.

Random

At each time-step t, the action u_t is drawn uniformly from U.

ϵ-Greedy

The ϵ-Greedy agent maintains an approximation of the current MDP and computes, at each time-step, its associated Q-function. The selected action is either selected randomly (with a probability of ϵ ($1 \geq \epsilon \geq 0$), or greedily (with a probability of $1 - \epsilon$) with respect to the approximated model.

Tested values:

$$\epsilon \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}.$$

Soft-max

The Soft-max agent maintains an approximation of the current MDP and computes, at each time-step, its associated Q-function. The selected action is selected randomly, where the probability to draw an action u is proportional to $Q(x_t, u)$. The temperature parameter τ allows to control the impact of the Q-function on these probabilities ($\tau \to 0^+$: greedy selection; $\tau \to +\infty$: random selection).

Tested values:

$$\tau \in \{0.05, 0.10, 0.20, 0.33, 0.50, 1.0, 2.0, 3.0, 5.0, 25.0\}.$$

OPPS

Given a prior distribution $p^0_{\mathcal{M}}(.)$ and an E/E strategy space $\mathcal{S}$, the Offline, Prior-based Policy Search algorithm (OPPS) identify a strategy $\pi^* \in \mathcal{S}$ which maximises the expected discounted sum of returns over MDPs drawn from the prior. The OPPS for Discrete Strategy spaces algorithm (OPPS-DS) (Castronovo et al., 2012; Castronovo, Fonteneau, and Ernst, 2014) formalises the strategy selection problem

for a discrete strategy space of index-based strategies. The E/E strategy spaces tested are the ones introduced in (Castronovo et al., 2016) and are denoted by $\mathbb{F}_2, \mathbb{F}_3, \mathbb{F}_4, \mathbb{F}_5, \mathbb{F}_6$. β is a parameter used during the strategy selection.

Tested values:

$\mathcal{S} \in \{\mathbb{F}_2, \mathbb{F}_3, \mathbb{F}_4, \mathbb{F}_5, \mathbb{F}_6\}$[10],

$\beta \in \{50, 500, 1250, 2500, 5000, 10^4, 10^5, 10^6\}$.

BAMCP

Bayes-adaptive Monte Carlo Planning (BAMCP) (Guez, Silver, and Dayan, 2012) is an evolution of the Upper Confidence Tree algorithm (UCT) (Kocsis and Szepesvári, 2006), where each transition is sampled according to the history of observed transitions. The principle of this algorithm is to adapt the UCT principle for planning in a Bayes-adaptive MDP, also called the belief-augmented MDP, which is an MDP obtained when considering augmented states made of the concatenation of the actual state and the posterior. BAMCP relies on two parameters: (i) K, which defines the number of nodes created at each time-step, and (ii) $depth$ defines the depth of the tree.

Tested values:

$K \in \{1, 500, 1250, 2500, 5000, 10000, 25000\}$,

$depth \in \{15, 25, 50\}$.

BFS3

The Bayesian Forward Search Sparse Sampling (BFS3) (Asmuth and Littman, 2011) is a BRL algorithm whose principle is to apply the principle of the FSSS (Forward Search Sparse Sampling, see (Kearns and Singh, 2002) algorithm to belief-augmented MDPs. It first samples one model from the posterior, which is then used to sample transitions. The algorithm then relies on lower and upper bounds on the value of each augmented state to prune the search space. K defines the number of nodes to develop at each time-step, C defines the branching factor of the tree, and finally $depth$ controls its maximal depth.

Tested values:

[10]The number of arms k is always equal to the number of strategies in the given set. For your information: $|\mathbb{F}_2| = 12, |\mathbb{F}_3| = 43, |\mathbb{F}_4| = 226, |\mathbb{F}_5| = 1210, |\mathbb{F}_6| = 7407$

$K \in \{1, 500, 1250, 2500, 5000, 10000\}$,

$C \in \{2, 5, 10, 15\}$, $depth \in \{15, 25, 50\}$.

SBOSS

The Smarter Best of Sampled Set (SBOSS) (Castro and Precup, 2010) is a BRL algorithm which relies on the assumption that the model is sampled from a Dirichlet distribution. Based on this assumption, it derives uncertainty bounds on the value of state action pairs. Following this step, it uses those bounds to decide the number of models to sample from the posterior, and the frequency with which the posterior should be updated in order to reduce the computational cost of Bayesian updates. The sampling technique is then used to build a merged MDP, as in (Asmuth et al., 2009), and to derive the corresponding optimal action with respect to that MDP. The number of sampled models is determined dynamically with a parameter ϵ, while the re-sampling frequency depends on a parameter δ.

Tested values:

$\epsilon \in \{1.0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6\}$,

$\delta \in \{9, 7, 5, 3, 1, 1\text{e}{-1}, 1\text{e}{-2}, 1\text{e}{-3}, 1\text{e}{-4}, 1\text{e}{-5}, 1\text{e}{-6}\}$.

BEB

The Bayesian Exploration Bonus (BEB) (Kolter and Ng, 2009) is a BRL algorithm that builds, at each time-step t, the expected MDP given the current posterior. Before solving this MDP, it computes a new reward function $\rho^{(t)}_{BEB}(x, u, y) = \rho_M(x, u, y) + \frac{\beta}{c^{(t)}_{<x,u,y>}}$, where $c^{(t)}_{<x,u,y>}$ denotes the number of times transition $< x, u, y >$ has been observed at time-step t. This algorithm solves the mean MDP of the current posterior, in which we replaced $\rho_M(\cdot, \cdot, \cdot)$ by $\rho^{(t)}_{BEB}(\cdot, \cdot, \cdot)$, and applies its optimal policy on the current MDP for one step. The bonus β is a parameter controlling the E/E balance.

Tested values:

$\beta \in \{0.25, 0.5, 1, 1.5, 2, 2.5, 3, 4, 8, 16\}$.

ANN-BRL

The Artificial Neural Network for Bayesian Reinforcement Learning algorithm (ANN-BRL) is fully described in Section 5.4. It samples n MDPs from prior distribution, and generates 1 trajectory for

each MDP drawn. The transitions are then used to build training data (one SL sample per transition), and several ANNs are trained on this dataset by SAMME and backpropagation[11]. The training is parametrised by n_h, the number of neurons on the hidden layer of the ANN[12], p, the number of samples resampled from the original training set at each epoch, ϵ, the learning rate used during the training, r, the maximal number of epoch steps during which the error on VS can increase before stopping the backpropagation training, and M, the maximal number of ANNs built by SAMME. When interacting with an MDP, the BRL agent uses the ANN trained during the offline phase to determine which action to perform.

Fixed parameters:

$n = 750$, $p = 5T$[13], $\epsilon = 1\text{e}{-3}$, $r = 1000$.

Tested values:

$n_h \in \{10, 30, 50\}$, $M \in \{1, 50, 100\}$,

$\varphi = \{[$ Q-values not related to x_t $],$
$[\text{Transition counters}, \text{current state}\ ]\}$.

[11] 2/3 for the learning set (LS) and 1/3 for the validation set (VS).

[12] In this paper, we only consider 3-layers ANNs in order to build weak classifiers for SAMME.

[13] The number of samples in LS is equal to $n \times T = 500T$. We resample 1% of LS at each epoch, which equals to $5T$.

5.6.2 SAMME Algorithm

A multi-class classification problem consists to find a rule $\mathcal{C}(.)$ which associates a class $c \in \{1, \ldots, K\}$ to any vector $v \in \mathbb{R}^n,\ n \in \mathbb{N}$. To achieve this task, we are given a set of training samples $< v^{(1)}, c^{(1)} > , \ldots, < v^{(N)}, c^{(N)} >$, from which a classification rule has to be inferred.

SAMME is a boosting algorithm whose goal is to build iteratively a set of weak classifiers $\mathcal{C}'^{(1)}(.), \ldots, \mathcal{C}'^{(M)} : \mathbb{R}^n \rightarrow \mathbb{R}^K$, and combine them linearly in order to build a strong classifier $\mathcal{C}(.)$. In our case, the weak classifiers are Multilayer Perceptrons (MLPs).

$$\mathcal{C}(h) = \frac{1}{M} \sum_{m=1}^{M} \alpha^{(m)}\ \mathcal{C}'^{(m)}(h),$$

where $\alpha^{(1)} \ldots \alpha^{(M)}$ are chosen to minimise the classification error.

Given a set of training samples $< v^{(1)}, c^{(1)} >, \ldots, < v^{(N)}, c^{(N)} >$, we associate a weight w_i to each sample. Let $err(\mathcal{C}'(.))$ be the weighted classification error of a classifier $\mathcal{C}'(.)$:

$$err(\mathcal{C}'(.)) = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{i=1}^{N} w_i\ \delta_i^{\mathcal{C}'},$$

where $\delta_i^{\mathcal{C}'}$ is the classification error of $\mathcal{C}'(.)$ for $< v^{(i)}, c^{(i)} >$.

At each iteration m, a weak classifier is trained to minimise the weighted classification error.

$$\mathcal{C}'^{(m)}(.) = \arg\min_{\mathcal{C}''(.)} err(\mathcal{C}''(.))$$
$$err^{(m)} = err(\mathcal{C}'(.))$$

If this classifier behaves better than a random classifier ($err^{(m)} < (n_U - 1)/n_U$), we compute its coefficient $\alpha^{(m)}$, update the weights of the samples, and build another classifier. Otherwise, we quit.

$$\alpha^{(m)} = \log\left(\frac{1 - err^{(m)}}{err^{(m)}}\right) + \log\left(n_U - 1\right)$$
$$w_i = w_i\ \exp\left(\alpha^{(m)}\ \delta_i^{\mathcal{C}'}\right)$$

In other words, each new classifier will focus on training samples misclassified by the previous classifiers. Algorithm 15 presents the pseudo-code description for SAMME.

Algorithm 15 SAMME

Input: A training dataset $DataSet$
Output: A classifier $\mathcal{C}(.)$

{Initialise the weight of each sample}
$N \leftarrow |DataSet|$
$w_i^{(1)} \leftarrow \frac{1}{N}, \forall i \in \{1, \ldots, N\}$

{Train weak classifiers}
$m \leftarrow 1$
repeat
  {Train a weak classifier}
  $\mathcal{C}'^{(m)} \leftarrow$ "Train a classifier on $DataSet$ w.r.t. $w^{(m)}$"

  {Compute its weighted error and its coefficient}
  $err^{(m)} \leftarrow \frac{1}{\sum_i w_i^{(m)}} \sum_i w_i^{(m)} \delta_i^{\mathcal{C}'}$
  $\alpha^{(m)} \leftarrow \log\left(\frac{1-err^{(m)}}{err^{(m)}}\right) + \log(n_U - 1)$

  {Adjust the weights for the next iteration}
  $w_i^{(m+1)} \leftarrow w_i^{(m)} \exp(\alpha^{(m)} \delta_i^{\mathcal{C}'}), \forall i$
  "Normalise the weights $w^{(m+1)}$"

  $m \leftarrow m + 1$
until $err^{(m)} \geq \frac{n_U - 1}{n_U}$ {Stop if $\mathcal{C}'^{(m)}$ is random}

$\mathcal{C}(.) \leftarrow \{ < \mathcal{C}'^{(1)}, \alpha^{(1)} >, \ldots, < \mathcal{C}'^{(m)}, \alpha^{(m)} > \}$

Chapter 6

Conclusion

This dissertation investigates the Exploration/Exploitation dilemma in Reinforcement Learning for which some prior knowledge is assumed to be known in advance. This chapter presents the main contributions as well as potential research directions.

6.1 Contributions

In Chapter 2, OPPS, which is an algorithm inspired by the work of F. Maes et al. is presented on multi-armed bandit problems (Maes, Wehenkel, and Ernst, 2012). By assuming the existence of some prior knowledge on an MDP, OPPS builds an efficient Exploration/Exploitation policy. The meta-learning scheme on which OPPS is based consists to build a large set of agents and identify the best one based on the information provided by the prior knowledge. With OPPS, we chose to build formula-based agents[1]. By combining in different ways a set of features chosen in advance, OPPS is able to generate a large set of formulas and, as such, a large set of agents. Identifying the best agent of this set was made tractable thanks to two major optimisations: (i) reducing the number of agents to consider by discarding duplicate or similar agents via a heuristic and (ii) optimising the distribution of the computational resources in the identification of the best agent by formalising this problem as a multi-armed bandit problem.

In Chapter 3, OPPS has been adapted to the Bayesian Reinforcement Learning (BRL) setting which integrates the very concept of prior knowledge. While OPPS handles any type of MDP distribution, BRL algorithms only consider prior knowledge encoded in the form of a Flat-Dirichlet Multinomial distribution. The performance of OPPS in BRL has been evaluated by comparing it with BAMCP— a state-of-the-art BRL algorithm. Contrary to OPPS, BAMCP performs most of its calculations prior to each decision. A look-ahead tree is explored until a time limit fixed in advance has been reached. The more computation time is provided to BAMCP, the better this exploration is. The differences between both algorithms highlighted the

[1] A formula-based agent relies on a small formula to evaluate the quality of each action and make its decisions.

necessity for computation times to be part of the comparison criteria. The existing benchmarks were not satisfying due to two main reasons: (i) the computation time was not taken into account; only the performance was, and (ii) the experiments were performed on a few MDPs rather than MDP distributions. For these reasons, our own benchmark has been designed. The results showed that the best algorithm was greatly dependant on the time constraints of the task to address.

In Chapter 4, the work presented in Chapter 3 has been extended and elaborated on. Our benchmark has been improved by adding more experiments and plotting new graphs providing another point of view on the results. Additionally, a comparison of most BRL algorithms of the field has been presented. The material used to perform those experiments have been compiled into a single open-source library, BBRL[2], to allow other researchers of the field to conduct their own experiments on this benchmark. It has also been found that OPPS was a good candidate on problems where (i) online computation time is limited and (ii) the prior distribution is not accurate.

In Chapter 5, ANN-BRL, which an algorithm based on artificial neural networks (ANNs), has been presented. This work can be described in three steps: (i) the original BRL problem has been formalised as a classification problem, (ii) a sampling algorithm has been defined to generate a relevant training dataset from the prior distribution, and (iii) ANNs have been combined with a boosting algorithm to build a BRL policy. As it was the case with OPPS, the policies built by ANN-BRL are time-efficient in regard to the updates (w.r.t. to the response of the system) and the decision-making process, contrary to those built by most BRL algorithms. The results obtained also suggest that ANN-BRL handles inaccuracies on the prior knowledge more efficiently than OPPS.

6.2 Future Work

6.2.1 Offline Prior-based Policy-search

OPPS defines a general approach which consists of three simple steps: (i) defining a rich set of formulas, (ii) building a rich set of BRL agents by building a BRL agent for each formula and (iii) search for the best BRL agent within this set. In order to avoid an explosion of the number of BRL agents to evaluate, we had to limit the number of features on which the formulas are based on. Instead of an arbitrary choice, an algorithm similar to PCA could be developed in order to build a compact and informative set of features. Another interesting extension to OPPS would be to replace the multi-armed bandit formalisation by

[2] https://github.com/mcastron/BBRL/wiki

its continuous equivalent, allowing the usage of new types of agents (e.g. formula-based agents with parametric formula).

6.2.2 Artificial Neural Networks for Bayesian Reinforcement Learning

ANN-BRL showed that neural networks are able to infer a good policy from the data directly sent by the system with no additional processing in some cases, relying only on the number of times each transition has been observed. These results may encourage further experiments on high dimensional systems for which the usage of computationally expensive tools like Q-values is limited or impossible. Besides, ANN-BRL relies on a classification formalisation which could also be exploited by other machine learning algorithms such as decision trees or SVMs.

6.2.3 Bayesian Reinforcement Learning Benchmark

Our benchmark links the performances of an algorithm with its computation time on three MDP distributions. However, it gives no clue behind the success or the failure of an algorithm on a given distribution. An interesting extension would be to use the distributions for which Bayes-optimal policies are known. Comparing the computed policies with the expected ones would certainly bring a new perspective on the results.

6.2.4 Flexible Bayesian Reinforcement Learning Algorithm

To the extent of our knowledge, there is no BRL algorithm which completely consumes the time available during both offline and online phases. If OPPS and ANN-BRL can handle various offline time constraints, there is no control in the resulting online computation time, while most BRL algorithms have little to no calculations prior to interacting with the system. As shown by the present work, there is valuable knowledge to retrieve during both phases. An algorithm which provides some control on both offline and online computation times would not only be more flexible, but could also surpass existing algorithms.

Bibliography

Asmuth, J. and M. Littman (2011). "Approaching Bayes-optimalilty using Monte-Carlo tree search". In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling*.

Asmuth, J. et al. (2009). "A Bayesian sampling approach to exploration in reinforcement learning". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, pp. 19–26.

Audibert, J. Y., R. Munos, and C. Szepesvári (2007). "Tuning bandit algorithms in stochastic environments". In: *Algorithmic Learning Theory*. Springer, pp. 150–165.

Auer, P., N. Cesa-Bianchi, and P. Fischer (2002). "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2, pp. 235–256.

Auer, P. and R. Ortner (2007). "Logarithmic online regret bounds for undiscounted reinforcement learning". In: *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. Vol. 19. The MIT Press, p. 49.

Brafman, R. I. and M. Tennenholtz (2002). "R-max – a general polynomial time algorithm for near-optimal reinforcement learning". In: *The Journal of Machine Learning Research* 3, pp. 213–231.

Buşoniu, L. et al. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Automation and Control Engineering. CRC Press.

Castro, P. S. and D. Precup (2010). "Smarter sampling in model-based Bayesian reinforcement learning". In: *Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 200–214.

Castronovo, M., R. Fonteneau, and D. Ernst (2014). "Bayes Adaptive Reinforcement Learning versus Off-line Prior-based Policy Search: an Empirical Comparison". In: *Proceedings of the Belgian-Dutch Conference on Machine Learning (BENELEARN 2014)*, pp. 1–9.

Castronovo, M. et al. (2012). "Learning exploration/exploitation strategies for single trajectory reinforcement learning". In: *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, pp. 1–9.

Castronovo, M. et al. (2016). "Benchmarking for Bayesian Reinforcement Learning". In: *PLoS ONE* 11, pp. 1–25.

Castronovo, M. et al. (2017). "Approximate Bayes Optimal Policy Search using Neural Networks". In: *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, pp. 1–12.

Dearden, R., N. Friedman, and D. Andre (1999). "Model based Bayesian exploration". In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann, pp. 150–159.

Dearden, R., N. Friedman, and S. Russell (1998). "Bayesian Q-learning". In: *Proceedings of Fifteenth National Conference on Artificial Intelligence (AAAI)*. AAAI Press, pp. 761–768.

Duff, M. O. G. (2002). "Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes". PhD thesis. University of Massachusetts Amherst.

Engel, Y., S. Mannor, and R. Meir (2003). "Bayes meets Bellman: the Gaussian process approach to temporal difference learning". In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pp. 154–161.

— (2005). "Reinforcement learning with Gaussian processes". In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pp. 201–208.

Engel, Y., P. Szabo, and D. Volkinshtein (2005). "Learning to control an octopus arm with Gaussian process temporal difference methods". In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. MIT Press, pp. 347–354.

Fonteneau, R., L. Busoniu, and R. Munos (2013). "Optimistic planning for belief-augmented Markov decision processes". In: *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*. IEEE, pp. 77–84.

Ghavamzadeh, M. and Y. Engel (2006). "Bayesian policy gradient algorithms". In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

— (2007). "Bayesian actor-critic algorithms". In: *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)*.

Guez, A., D. Silver, and P. Dayan (2012). "Efficient Bayes-adaptive reinforcement learning using sample-based search". In: *Neural Information Processing Systems (NIPS)*.

— (2013). "Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-Carlo tree search". In: *Journal of Artificial Intelligence Research*, pp. 841–883.

Hennig, P., D. H. Stern, and T. Graepel (2009). "Bayesian quadratic reinforcement learning". In: *Neural Information Processing Systems (NIPS)*.

Jaksch, T., R. Ortner, and P. Auer (2010). "Near-optimal regret bounds for reinforcement learning". In: *The Journal of Machine Learning Research* 11, pp. 1563–1600.

Jung, T., D. Ernst, and F. Maes (2013). "Optimized Look-Ahead Trees: Extensions to Large and Continuous Action Spaces". In: *Proceedings of IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*.

Jung, T. et al. (2014). "Optimized look-ahead tree policies: a bridge between look-ahead tree policies and direct policy search". In:

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1–2, pp. 99–134.

Kearns, M. and S. Singh (2002). "Near-optimal reinforcement learning in polynomial time". In: *Machine Learning* 49, pp. 209–232.

Kober, J. et al. (2012). "Reinforcement learning to adjust parametrized motor primitives to new situations". In: *Autonomous Robots, Springer*.

Kocsis, L. and C. Szepesvári (2006). "Bandit based Monte-Carlo planning". In: *European Conference on Machine Learning (ECML)*, pp. 282–293.

Kolter, J. Z. and A. Y. Ng (2009). "Near-Bayesian exploration in polynomial time". In: *Proceedings of the 26th Annual International Conference on Machine Learning*.

Maes, F., L. Wehenkel, and D. Ernst (2011). "Automatic discovery of ranking formulas for playing with multi-armed bandits". In: *9th European workshop on reinforcement learning*. Athens, Greece.

— (2012). "Learning to play K-armed bandit problems". In: *International Conference on Agents and Artificial Intelligence*. Vilamoura, Algarve, Portugal.

Martin, J. J. (1967). *Bayesian decision problems and Markov chains*. English. Thesis. "Originally submitted as a Ph.D. thesis [Massachusetts Institute of Technology, 1965]".

Milani Fard, M. and J. Pineau (2010). "PAC-Bayesian model selection for reinforcement learning". In: *Neural Information Processing Systems (NIPS)*.

Murphy, S. A. (2005). "An Experimental Design for the Development of Adaptive Treatment Strategies". In: *Statistics in Medicine* 24, pp. 1455–1481.

Nevmyvaka, Y., Y. Feng, and M. Kearns (2006). "Reinforcement learning for optimized trade execution". In: *Proceedings of the 23rd IInternational Conference on Machine Learning*.

Perrick, P. et al. (2012). "Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron". In: *IEEE Conference on Computational and Intelligence in Games (CIG)*.

Poupart, P. (2008). "Model-based Bayesian reinforcement learning in partially observable domains". In: *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*.

Poupart, P. et al. (2006). "An analytic solution to discrete Bayesian reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 697–704.

Ross, S. and J. Pineau (2008). "Model-based Bayesian reinforcement learning in large structured domains". In: *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press.

Ross, S. et al. (2011). "A Bayesian approach for learning and planning in partially observable Markov decision processes". In: *Journal of Machine Learning Research (JMLR)* 12, pp. 1729–1770.

Schwenk, H. and Y. Bengio (2000). "Boosting Neural Networks". In: *Neural Comp.* 12.8, pp. 1869–1887.

Silver, E. A. (1963). *Markovian decision processes with uncertain transition probabilities or rewards*. Tech. rep. DTIC Document.

Strens, M. (2000). "A Bayesian framework for reinforcement learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*. ICML, pp. 943–950.

Sutton, R. S. (1988). "Learning to predict by the methods of temporal differences". In: *Machine Learning*. Kluwer Academic Publishers, pp. 9–44.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

Walsh, T. J., S. Goschin, and M. L. Littman (2010). "Integrating Sample-Based Planning and Model-Based Reinforcement Learning". In: *AAAI*.

Wang, Y. et al. (2012). "Monte carlo bayesian reinforcement learning". In: *arXiv preprint arXiv:1206.6449*.

Watkins, C. J. and P. Dayan (1992). "Q-Learning". In: *Machine Learning* 8.3-4, pp. 179–192.

Zhang, T. et al. (2015). "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search". In: *The International Conference on Robotics and Automation (ICRA)*.

Zhu, J. et al. (2009). "Multi-class adaboost". In: *Statistics and its Interface* 2.3, pp. 349–360.