POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Computer Engineering

Active Transfer of Samples in Reinforcement Learning

AI & R Lab
The Artificial Intelligence and Robotics Lab
of the Politecnico di Milano

Supervisor: Prof. Marcello Restelli
Co-Supervisor: Dr. Andrea Tirinzoni

Candidate:
Mahsa Shekari
Matr. 864207

Academic Year 2019-2020

To My Parents

Acknowledgement

I would like to express my sincere gratitude to my advisor Professor Marcello Restelli for giving me the opportunity to work on this project in a professional area (PoliMi AI & R Lab) and all his support and guidance in writing this thesis.

I would also like to thank my co-advisor, Dr. Andrea Tirinzoni, who was a source of motivation for this new experience in the research world and his constant, valuable and honest help and feedback.

My sincere gratitude goes to my family and my parents, who genuinely supported me in this process.

I am grateful to all my friends for their support and contributions.

Thanks to my advisors in my previous school, Dr. Morteza Saberi and Dr. Peyman Shahverdi, who encouraged me to pursue my study.

Thanks to all professors who taught me at Politecnico di Milano University.

Thanks to my dear and supportive friend Shirin Nobakhti.

I would like to thank my great friend, the person to whom words are not enough to express my gratitude, Dr. Marjan Hosseini.

At the end, thanks to my love Alireza Javadian Sabet, who has always been there for me in achieving my dreams.

Glossary

AWFQI	Active Importance Weighted Fitted Q-Iteration
DM	Data Mining
DP	Dynamic Programming
FQI	Fitted Q-Iteration
GPI	Generalised Policy Improvement
GPs	Gaussian Processes
HiP-MDP	Hidden Parameter Markov Decision Process
IS	Importance Sampling
IWFQI	Importance Weighted Fitted Q-Iteration
LP	Linear Programming
MAB	Multi-Armed Bandit
MABs	Multi-Armed Bandits
MDP	Markov Decision Process
MDPs	Markov Decision Processes
MI	Mutual Information
MIS	Multiple Importance Sampling
ML	Machine Learning
MRP	Markov Reward Process
MRPs	Markov Reward Processes

MSE	Mean Square Error
POMDP	Partially Observable Markov Decision Process
RBT	Relevance-based Transfer
RL	Reinforcement Learning
SARSA	State-action-reward-state-action
SDT	Shared-dynamics Transfer
SFs	Successor Features
TD	Temporal Difference
TL	Transfer Learning

Contents

List of Figures

Abstract

In this work, we address the shortcomings of previous sample transferring methods by handling the dissimilarity between tasks efficiently. The contribution of this work is designing an *active* algorithm, Active Weighted Fitted Q-Iteration (AWFQI), for transferring the samples from the source tasks to the target task, under the assumption that a generative model of the target task is available. Our algorithm actively demands samples that yield high information for solving the task itself. For this purpose, we estimate the conditional variance among the source tasks by applying Gaussian process regression to the collected samples. This variance captures the dissimilarities among tasks, and it is applicable in both discrete and continuous domains. In particular, we treat it as a score function indicating what state-action pairs are likely to be more informative for solving the target task and use it to actively query the generative model. We compare the performance of our method with other state-of-the-art algorithms on transferring the samples such as Relevance-based Transfer, Shared-dynamics Transfer, Importance Weighted Fitted Q-Iteration and a non-transfer algorithm (Fitted Q-Iteration). We show that our method outperforms these algorithms in many situations, specifically when the budget for transferring is limited.

Keywords: Reinforcement Learning; Transfer Learning; Importance Weighted Transfer; Active Transfer; Sample Reuse; Transfer of Samples; Batch Learning.

Sommario

In questo lavoro cerchiamo di risolvere alcune delle maggiori limitazioni di lavori esistenti sul trasferimento di campioni nel contesto di apprendimento per rinforzo. Il maggior contributo consiste nella progettazione e valutazione empirica di un nuovo algoritmo, Active Weighted Fitted Q-Iteration (AWFQI), per trasferire campioni da task sorgenti ad un task target gestendo in modo efficiente le dissimilarita' fra i task. L'algoritmo proposto richiede attivamente campioni (ovvero richiede l'esecuzione di certe azioni in stati arbitrari) al task target sotto l'assunzione che un modello generative di quest'ultimo sia disponibile. Tali campioni sono selezionati in modo da portare molta informazione per la risoluzione del task stesso. A questo fine proponiamo di stimare la varianza condizionata dei modelli di transizione e di reward fra i task sorgenti usando processi Gaussiani per effetttuare regressione sui campioni osservati. Questa varianza cattura le dissimilirita' fra i task e puo' essere calcolata sia in domini discreti che in quelli continui. Nel nostro caso viene trattata come una funzione indicatrice di quali coppie stato-azione sono piu' informative per la risoluzione del task target e, di consequenza, viene usata per richiedere attivamente campioni al modello generativo. Infine, valutiamo empiricamente ll nostro algoritmo in un dominio continuo, confrontandolo con approcci allo stato dell'arte come Relevance-based Transfer, Shared-dynamics Transfer, Importance Weighted Fitted Q-Iteration e un algoritmo che non effettua alcun trasferimento di conoscenza (Fitted Q-Iteration). Mostriamo che il nostro algoritmo ottiene spesso le performance migliori, in particolare in casi in cui il budget di campioni che e' possibile richiedere dal task target o che e' disponibile dai sorgenti e' limitato.

Chapter 1

Introduction

This chapter provides an overview of the work. In Section 1.1, we explain the context and motivations, which led us to conduct this research. In Sections 1.2, we specify the objectives and possible improvements over existing methods and our contribution. Then, Section 1.3 presents the structure of the work.

1.1 Context and Motivations

Many situations in the real-world involve the interaction between two entities, essentially an entity which we can call the agent that interacts with another entity *i.e.* environment which it deals with. In these types of problems, the real challenge is finding the optimal solution such that the agent can benefit the environment. These problems usually involve some agents' actions in specific states at a time and can be modeled as Markov Decision Processes (MDPs). The main objective in these problems is solving such MDPs *i.e.* finding an optimal policy which can be efficiently addressed employing Reinforcement Learning (RL) algorithms [70].

Many of these real-world problems are defined in the field of Machine Learning (ML) and Data Mining (DM), in which knowing about the benefits of past experiences to solve the new problems. In almost all of the ML approaches, we consider test data, as well as training data. The challenge is

understanding the pattern of the test data through the acquired knowledge from training samples. In fact, Transfer Learning (TL) is a generalization that knowledge can be transferred not only within tasks but also across different tasks [75]. Now, if the characterization and features in the training (previous) and test (new) data are similar, TL is helpful to discover the patterns in a new environment. However, if the characteristics/distribution of the past experiences and future situations do not quite match, the performance of our prediction/solution [62] deteriorates. Consequently, the learner needs to adopt more efficient ways to transfer knowledge. In the RL terminologies, we call these past experiences/training data as *source* tasks, and the future environment/test data as *target* tasks.

TL has been studied and used in various contexts. The range of its application spread from very seemingly irrelevant fields such as psychology [63, 89] and cognitive architectures [15, 37] to more relevant applications such as in ML [14, 77] and planning tasks [23, 30]. In addition, TL in RL is a worthy topic to address, since recently the contribution of RL in solving hard problems has been considerable compared to ML. Examples of this success can be TD-Gammon [76], job shop scheduling [90], elevator control [18], helicopter control [52], marble maze control [12], Robot Soccer Keepaway [66], quadruped locomotion [34, 38]. Moreover, many traditional ML approaches in classification, regression and *etc.* are sufficiently well-established to be employed in TL. Finally, TL algorithms have shown promising results in terms of effectiveness in solving problems and accelerating the solutions. With these motivations, TL in the domains of RL has been discussed deeply in [42, 45, 61, 75].

1.2 Contribution

In this work, we studied TL in RL. Previous methods in transferring samples such as TIMBREL [74] transfers all the samples as a bulk without accounting for the inherent difference between the source and target MDPs. The methods that considered these criteria like Relevance-based Transfer (RBT) [43], do not expect any specific condition for deciding what to transfer and makes an assumption in terms of similarity between the source and target tasks. Moreover, since RBT [43] computes the compliance and relevance metric such that they mutually account for both the reward and transition mod-

els, it ignores the samples in case one of the models is very different among the tasks. Shared-dynamics Transfer (SDT) [40] transfers the samples into Fitted Q-Iteration (FQI) with the assumption that tasks follow the same transition dynamics. This assumption potentially hinders its applicability as a transfer method in most of the real-world tasks. In this work, we aimed to design an algorithm so that it can efficiently handle the dissimilarity while considering the trade-off discussed in [25]. We base our work on the method proposed by Tirinzoni *et. al* [82] in 2018 (Importance Weighted Fitted Q-Iteration (IWFQI)), as they address many of the above mentioned shortcomings in previous methods. IWFQI uses standard exploration strategies (like $\epsilon-$Greedy) to get samples from the target task while transferring the source instances to augment the available samples. However, since some samples are already available from the source tasks, standard exploration might be overkill as it might collect samples where they are not really needed (*e.g.*, where everything is known from the sources). Therefore, it is important to design exploration strategies that are aware of the knowledge that is already available from the sources and the one that still needs to be obtained. Here, under the assumption that we have a generative model, we present a technique, Active Importance Weighted Fitted Q-Iteration (AWFQI), for obtaining samples from regions of the state-action space where little is known about the target task without further assumption about the characteristics of the source and target tasks.

1.3 Structure of the Work

The rest of the work is organized as follow. Chapter 2 explains some of the theoretical frameworks and concepts applied in this work. In Chapter 3, we discuss the background and related works in transfer learning. In Chapter 4, we explain our proposed solution for designing an active transfer learning algorithm. The empirical evaluation of the proposed methodology are presented in Chapter 5. Finally, Chapter 6 presents the conclusion and future works.

Chapter 2

Theoretical Background

This chapter provides the reader with the basic theoretical frameworks and concepts used in this work. First, we discuss fundamental elements of RL, then we provide discussions regarding Markov Decision Process (MDP) and how to learn policies in that setting. Then we compare some of the important concepts in RL and describe some methods in learning policy. After that, we explain the regression methods used for prediction.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a discipline in ML in which the agent learns by interacting with an environment, and this is the nature of learning. As an example of RL consider when a newborn waves its arms, it has no teacher, but it does have a direct connection to its environment. Therefore, a child, according to the consequences of actions and what to do for the sake of achieving goals, learns about its environment [71]. In RL, the agent selects some actions at a time; accordingly, that action leads to changes in the state the agent is in and obtain some reward returned by the environment. The agent's goal is to maximize rewards over a defined time horizon [51]. In the example mentioned above, although the baby does not have any explicit teacher to tell the rules, it can evaluate possible actions according to the rewards it receives from the environment. The agent is learning to choose actions that maximize rewards and minimize punishments or losses. RL is

much more focused on goal-directed learning from interaction in comparison with other ML approaches.

So, we need to maximize the numerical reward signal according to the way we map situations to actions, which is part of the goal of RL as it is depicted in Figure 2.1.

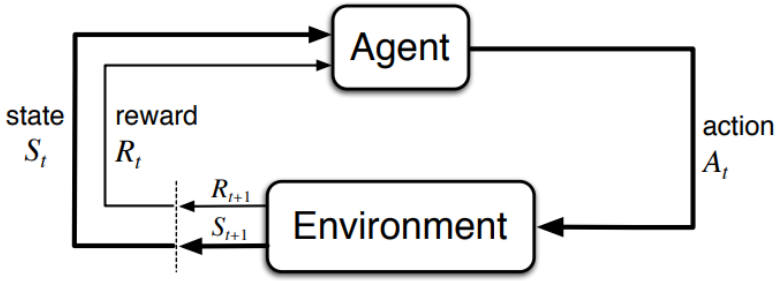

Figure 2.1: The Reinforcement Learning framework [71].

As it can be inferred from the figure, RL problems are closed-loop. In general, the following are the most important characteristics of RL problems [72].

1. The problem is a closed-loop, *i.e.* at each time step, the agent action is dependent on the feedback that it receives from the process in the form of a value called reward or the discounted reward [20].
2. There is no direct instructions. The agent is not told explicitly which actions they should take, as opposed to some other areas of ML, but instead must discover which actions yield the most reward by taking those actions.
3. Actions may affect not only the immediate reward but also the next situations and through that, all subsequent rewards. So the reward an agent receives at one timestamp is partly due to not only the previous action but also the consequences of all the previous actions it has taken.

Agent – Environment Interface: In RL, the learner and decision-maker are called the agent, and everything outside is considered as the environment in which the agent interacts with. According to the agent selection of the actions, the environment responds to those actions and presents new situations to the agent. The environment also generates rewards.

More in detail, the agent and environment interact at a sequential discrete time step, starting from 0. At each time t, the agent receives some signal which shows the state in which it is in which we call the environment's state or $S_t \in S$, where S is the whole states space, and accordingly selects a possible action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in state S_t. One time step later, the agent receives a numerical reward, $R_{t+1} \in R \subset \mathbb{R}$, because of its action,then the state position changes to the new state of S_{t+1}.

We use R_{t+1} instead of R_t to denote the reward due to A_t because it implies that the next reward and state, R_{t+1} and S_{t+1}, are determined together. At each time step, the agent considers a mapping from states to probabilities of choosing one action in the action space. This mapping is actually the agent's policy [71] and is denoted by π_t, where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

Trade-off between exploration and exploitation: One of the most critical challenges in the RL domain is finding a trade-off between exploration and exploitation. Exploration means the agent selects actions to collect information about the environment (e.g., randomly), while in exploitation, the agent selects actions to maximize rewards according to its knowledge about the environment greedily. To obtain more rewards, an agent must prefer actions that it has tried in the past and found effective in producing reward. However, to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward [71].

2.1.1 Elements of Reinforcement Learning

In the area of RL, we can consider four main elements as follow: *Policy*, *Reward*, *Value Function*, *Model*. For the sake of completeness, we explain them briefly; then, in the later sections, they will be explained in more detail.

Policy

The policy defines how the learning agents should behave at a given time step. In other words, a policy is a mapping from perceived states of the environment to the most appropriate action to be taken that maximizes the overall reward [83]. The optimal policy is usually denoted as π^* [48]. In some cases, the policy may be a simple function or lookup table, whereas it might involve extensive computation, such as a search process. Simply, the policy might be greedy. The greedy policy follows a greedy algorithm for solving the problem of finding the best action at a given step [11]. Greedy policies heuristically maximize the immediate reward and not the overall reward. As a result, they fail to find the global optimum, and they are often sub-optimal. On the other hand, $\epsilon-$ greedy policies [70] take a random action with a probability of ϵ. These policies will be explained more in detail in section 2.4.6. Another trivial policy is a random policy, which picks actions uniformly from the action state at a given time without considering the reward value either immediate or discounted [7, 31].

Reward

A reward signal determines the goal in a RL problem. A reward is a single number that the environment sends to the agent at a time step. In a typical RL problem, the agent's objective is maximizing the total reward. The reward signal thus defines what the good and bad events are for the agent. The agent has no access to the process that generates the reward function, however, it can have access and modify the signal that the process produces directly by its actions and indirectly by changing its environment's state since the reward signal depends on these, but the agent cannot alter the function that generates the signal. Since if the agent could change the reward signal, that would mean that it can modify the first problem it is facing into another problem. The reward signal is the primary component needed for updating the policy, in a sense that the policy tries to select the actions according to the reward the agent receives if that action is taken. Accordingly, if an action selected by the policy yields a low reward, the policy learns not to select that action in that particular situation in the future. Nevertheless, it is essential for an algorithm to sacrifice some immediate rewards if they lead the agent to gain higher rewards in the long term [72].

Value Function

As mentioned before, the reward signal indicates what is good at each state, in an immediate sense. However, the value function determines what is good in the long run. In fact, the value of a state is equal to the total amount of reward an agent can expect to collect in the future if it starts from that particular state [70, 72]. In other words, the reward signal determines the immediate, intrinsic desirability of environmental states. In contrast, values function (value of a state) indicate the long-term desirability of that particular state after considering the states that are likely to follow if a specific action is taken, and the rewards available in those states. For instance, we could have a state that gives a low immediate reward for taking different actions, but it still has a high value because it is regularly led to some other states that yield high rewards. A state can be exactly the opposite too.

Rewards are making the values, and without them, values cannot be defined, and the goal of estimating values is to achieve a higher overall reward. In this sense, we are more concerned with knowing the value rather than the reward when making and evaluating decisions. Correspondingly, we search among the actions for the one that brings the states of the highest value, not the highest reward, because, in the long run, we could make more reward by taking these actions. In fact, the most crucial component of reinforcement learning algorithms is to estimate values efficiently. It is typically harder to determine values rather than rewards [65].

Model

Another element of some of the RL problems is a model of the environment that simulates the behavior of the environment. Intuitively, the model makes inferences about how the environment will behave. For instance, the model can predict the next state and reward, given that the agent is in a particular state and takes an action. Models are used for planning, *i.e.* for deciding on a course of action by considering possible future situations in advance. In general, having a model is optional, but models can easily be used if they are available, or they can be learned [57].

2.2 Markov Decision Processes

MDPs are useful in many of the planning applications in the presence of uncertainty [59]. In this section, we explain about MDPs, their definition, and some of the background information that is relevant or related to a better understanding of the topic.

2.2.1 Definition

MDPs bring a mathematical framework for modeling sequential decision-making problems when results are randomly [2] and in controlling a goal-directed agent [83]. Here we consider MDPs are discrete-time stochastic control processes in which the agent decides which action to take in each state then observes a reward for taking that action in that particular state. Maximizing the long-term total reward is an agent's goal according to the sequence of actions [8].

Definition 1. *A discounted MDP is a tuple* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, q, p_0, \gamma \rangle$*, where:*

- $\mathcal{S}$ *is the set of states that the agent can have in the environment;*
- $\mathcal{A}$ *is the set of actions that the agent can apply in the environment;*
- $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ *are the transition probabilities;*
- $q : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ *is the reward distribution;*
- $p_0 \in \mathcal{P}(\mathcal{S})$ *is the initial state distribution, where* $p_0(s)$ *is the probability that is the state where the process starts;*
- $\gamma \in [0, 1]$ *is a discount factor.*

If the set of states $\mathcal{S}$ and the set of actions $\mathcal{A}$ are finite, an MDP can be finite or discrete. Otherwise, it is called infinite or continuous. As we know, stationary is one of the important properties of MDPs, which means the reward function q, and the transition function p does not change over time.

2.2.2 Partially Observable Markov Decision Process

Partially Observable Markov Decision Process (POMDP) is a generalization of the MDP by considering that the hidden states and exploring the observations and observation of state probability [53]. Intuitively consider the

following example. We usually go to our favorite restaurant and try anew dish that we had not tried before intending to find a better dish. If our choice for every restaurant never changes, the bandit is a very proper formulation RL. If our preference changes to Chinese food after having Japanese food, then modeling and exploring the state transition is more suitable MDP. Finally, if the restaurant's choice is decided by many hidden factors, *e.g.* hidden interest, mood, *etc.* it is preferable to consider POMDP [56].

Definition 2. *A POMDP is a tuple* $\langle \mathcal{S}, \mathcal{A}, \Omega, p, O, q, b_0, \gamma \rangle$ *,where:*

- $\mathcal{S}$ *is a finite set of states;*
- $\mathcal{A}$ *is a finite set of actions;*
- Ω *is a finite set of observation, partial information about the state the agent provides by each observation;*
- $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ *are the transition probabilities;*
- O *are the conditional observation probabilities, where* $O(o_{t+1}|s_{t+1}, a_t)$ *is the probability of observing* o_{t+1} *after taking action* a_t *and transitioning to next state* s_{t+1}*;*
- $q : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ *is the reward distribution;*
- b_0 *is the initial state probability distribution(initial belief state);*
- $\gamma \in [0, 1]$ *is the discounted factor and it is used to discount future rewards in infinite-horizon processes.*

At time $t = 1$, where the first state s_1 is drawn from b_0 process is started. As far as, the agent cannot observe such a state directly, it receives an observation o_1. Then, the agent selects an action a_1, transitions to the unobserved state s_2 according to $p(s_2|s_1, a_1)$, receives an observation o_2 according to $O(o_2|s_2, a_1)$, and at the end collects a reward $q(s_1, a_1)$. After that, the process is repeated (forever in case of infinite-horizon). The agent's goal is to select the sequence of actions maximizing the total (discounted) reward over time.

2.2.3 Value Functions

In this section, we briefly discuss state-value and action-value Functions.

State-Value Functions

As far as an agent needs to know some information about the utility of being in a state or select an action in a specific state, this function provides this information. A function of states that estimates how good it is for the agent to be in a given state, which means we need to evaluate future rewards that can be expected in terms of expected return. Of course, the rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined *w.r.t.* particular policies. Recall that a policy, π , is a mapping from each state, $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$, to the probability $\pi(a|s)$ of taking action a when in state s. Generally, the value of a state s under a policy π, denoted $V^\pi(s)$, is the expected return when starting in s and following π after that. Given a policy π, it is possible to define the utility of each state. For MDP we can define $V^\pi(s)$ formally as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k R(s_{t+k}, a_{t+k}) | s_t = s \right] \tag{2.1}$$

Action-Value Functions

For control purposes, it is easier to compute the action-value function, $Q^\pi(s, a)$ as it is more intuitive to know the value of a specific action in a given state, and then derive the optimal policy [71]. It is defined as the expected return starting from s, taking the action a, and after that following policy π. We call Q^π the action-value function for policy π.

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k R(s_{t+k}, a_{t+k}) | s_t = s, a_t = a \right] \tag{2.2}$$

2.2.4 Bellman Equations and Operators

If we decompose the definition of the value function, $V^\pi(s)$ and consider the value function as the sum of an immediate reward in state s with the expected discounted reward in the following state. This kind of recursive definition will show itself useful in solving MDPs.

Bellman Expectation Equation

The Bellman Expectation Equation [9] for state-value function is decomposed into immediate reward plus discounted value of next state:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s] \\ &= \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') \right) \end{aligned} \tag{2.3}$$

The action-value function can be decomposed in the same way as presented in equation 2.4:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1})|s_t = s, a_t = a] \\ &= R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') \\ &= R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a \in A} \pi(a'|s') Q^{\pi}(s',a') \end{aligned} \tag{2.4}$$

We can write the Bellman Expectation Equation in a concise matrix form of the equation 2.5, by using the Markov Reward Process (MRP) induced by policy π.

$$V^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi} \tag{2.5}$$

Where V^{π} is "vector" notation which holds for finite MDPs, while for continuous MDPs this is "operator" notation.

Besides, the solution presents in the equation 2.9 which has a high computational complexity for inverting the matrix $(I - \gamma P^{\pi})$:

$$V^{\pi} = (I - \gamma P^{\pi})^{-1} R^{\pi} \tag{2.6}$$

So, a way to solve the MDP while avoiding the complexity inversion is to use the Bellman Operator [9] defined as $\mathcal{T}^{\pi} : \mathbb{R}^{(|S|)} \to \mathbb{R}^{(|S|)}$:

$$(T^{\pi}V^{\pi})(s) = \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') \right), \quad \forall s \in S \quad (2.7)$$

It can be proved that the state value function v^{π} is the unique fixed point of the Bellman Operator, T^{π} , *i.e.*, it satisfies $T^{\pi}[V^{\pi}] = V^{\pi}$ [59].

For Q function also we can define the Bellman Expectation Operator as well, $T^{\pi} : R^{S \times A} \rightarrow R^{S \times A}$, defied as:

$$(T^{\pi}Q^{\pi})(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a \in A} \pi(a'|s') Q^{\pi}(s',a') \quad \forall s \in S \quad (2.8)$$

Similar to the value function, Q^{π} is the unique fixed point of T^{π} , *i.e.* $T^{\pi}[Q^{\pi}] = Q^{\pi}$. It worth mentioning that both operators are linear. As far as they satisfy the contraction property in $\mathcal{L}_{\infty}$ norm, *i.e.* $\| T^{\pi} f_1 - T^{\pi} f_2 \|_{\infty} \leq \gamma \| f_1 - f_2 \|_{\infty}$, so the repeated application of T^{π} makes any function converge to the value function which is very important property in RL. A partial order over policies are defined. Namely, for any two policies π, π':

$$\pi \geq \pi' \quad if \quad V^{\pi}(s) \geq V^{\pi'}(s), \quad \forall s \in S \quad (2.9)$$

2.2.5 Optimality Conditions

If we call Π the set of all possible Markovian policies then the optimal value function, V^* is:

Definition 3. *The optimal value function in any state $s \in S$ in a given MDP $\mathcal{M}$ is:*

$$V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s), \quad \forall s \in S \quad (2.10)$$

Solving an MDP means finding the optimal value function and the optimal value function specifies the best possible performance an agent can gain in any state of the MDP. In addition, we can describe the optimal action-value

function, $Q^*(s,a)$ as:

Definition 4. *The optimal action-value function in any state-action pair* $(s,a) \in S \times A$ *in a given MDP* $\mathcal{M}$ *is:*

$$Q^*(s,a) = \max_{\pi \in \Pi} Q^\pi(s,a), \quad \forall (s,a) \in S \times A \tag{2.11}$$

Bellman Optimality Equation

Likewise, the value function of a given policy, we can define the optimal value function of an MDP in a recursive way, [73]as:

$$V^*(s) = \max_{a \in A} Q^*(s,a) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right) \tag{2.12}$$

Definition 5. *The Bellman optimality operator,* $T^* : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ *is defined as:*

$$(T^*V)(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right), \quad \forall s \in S \tag{2.13}$$

This operator has the same properties as the Bellman Expectation Operator mentioned before, namely, its a contraction *w.r.t.* $\| \cdot \|_\infty$, and the optimal value function V^* is a unique fixed point of the operator.

2.2.6 Policies in Markov Decision Processes

A policy determines the behavior of an agent in a given state. A distribution over actions given the state is a policy π. In other words, a policy at any given point in time chooses which action the agent selects because MDPs have the Markovian property, the current state s_t is enough to determine the next action a_t. [71]. These kinds of policies are called Markovian Policies. On another side, the policy is called stationary if the actions mapped are not dependent on the time step t.

$$\pi(a|s) = \mathbb{P}[a|s] \tag{2.14}$$

Definition 6. *A stationary randomized (deterministic) control policy is a mapping $\pi : \mathcal{S} \to \mathcal{P}(A)$ ($\pi : \mathcal{S} \to \mathcal{A}$). The set of all stationary randomized (deterministic) policies is denoted as Π^{SR} (Π^{SD}). At each time step t, the agent is in some state S_t, it takes an action $A_t \sim \pi(\cdot|S_t)$ according to a possibly randomized policy π, it transitions to a new state $S_{t+1} \sim p(\cdot|S_t, A_t)$, and it receives a reward $R_t \sim q(\cdot|S_t, A_t)$. As far as the goal of RL is finding the policy which maximizes the cumulative reward collected.An MDP paired with a policy π is called Markov Reward Processes (MRPs) denoted by the tuple $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma, \mu \rangle$. For $s \in \mathcal{S}, \mathcal{P}^\pi(\cdot|s)$ is a probability measure over $\mathcal{S}$. P^π is a probability density function gained by marginalizing the transition model of the MDP, $\mathcal{P}$ over the actions:*

$$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s, a), \quad \forall s, s' \in S \tag{2.15}$$

$P^\pi(s'|s)$ gives the probability of ending up in state s', starting from state s, in one time step. Correspondingly, $\mathcal{R}^\pi(\cdot|s, s')$ for all $s, s' \in S$ is a probability measure over with the corresponding density function $\mathbb{R}$ obtained from the reward model of MDP as:

$$R^\pi(r|s, s') = \sum_{a \in A} \pi(a|s) R(r|s, s', a), \quad \forall s, s' \in S \tag{2.16}$$

MRPs are suitable model for uncontrolled processes. We are left with Markov Chains or Markov Processes by removing the concept of reward from the MRPs.

Optimal Policies

Theorem 1. *For any Markov Decision Process*

- *There is an optimal policy π^* which is better or equal to all other policies $\pi \in \Pi$;*
- *All optimal policies achieve the optimal value function, $V^{\pi *}(s) = V^*(s)$;*

- *All optimal policies achieve the optimal action-value function,* $Q^{\pi*}(s,a) = Q^*(s,a)$*;*
- *There is always a deterministic optimal policy for any MDP*

Therefore, according to the theorem, by finding the optimal Q function, Q^*, and taking "greedy" action in each state, we can discover the optimal policy of an MDP which means:

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}}\, Q^*(s,a) \quad \forall s \in S \tag{2.17}$$

2.3 Planning in Markov Decision Processes

As far as an MDP is defined as the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu, \gamma \rangle$, we can solve the MDP (find the optimal policy), if we know all the elements of the tuple [33]. In Artificial Intelligence, planning is solving a decision making problem without ever making a decision. Some of the most used planning algorithms are dynamic programming, policy iteration, and value iteration.

2.3.1 Dynamic Programming

By finding an optimal policy, we can solve an MDP. For finding this policy, the usual approach is to find the optimal value function (or action-value function) and drive from it the optimal policy. When the state transition model ($\mathcal{P}$) of the MDP is known, Dynamic Programming (DP) is the most common approach used to solve MDPs. In general, DP [9, 10] is a common technique to solve problems which can be divided into sub-problems or problems that satisfy the following two properties:

- Problems must have an optimal substructure *i.e.*, the solution can be decomposed into sub-problems;
- Problems must have overlapping sub-problems *i.e.*, sub-problems recur repeatedly, and solutions can be cached and reused.

As far as sub-problems are solved, the main problem is also solved, and this recursive problem solving is a definition of the Bellman operator. In fact MDPs have both these properties:

- Bellman equation gives recursive decomposition;

- Value function stores and reuse solutions.

DP algorithms require full knowledge of the MDP, and that is the main disadvantage of this approach. As far as the number of states and/or actions may be infinite or the dynamics of the process may be unknown, it is difficult in MDPs to model real problems, and sometimes it is impossible.

2.3.2 Policy Iteration

By alternating the policy evaluation and policy improvement phases, policy iteration solves MDPs [29]. The algorithm starts from a random policy, π_0 than the policy iteration phase tried to extract the value function of the current policy. Also, the closed-form solution could be used to get a specific solution, but as we know, this comes with a high computational cost. As a matter of choice, we can apply recursively to the Bellman Expectation Operator. By applying this operator recursively, we get an approximation of the value function of the policy, but for most applications, we look for an approximation of the value function. In addition, in modified policy iteration [60], the algorithm does not wait until convergence, defending that a sufficiently good approximation is enough.

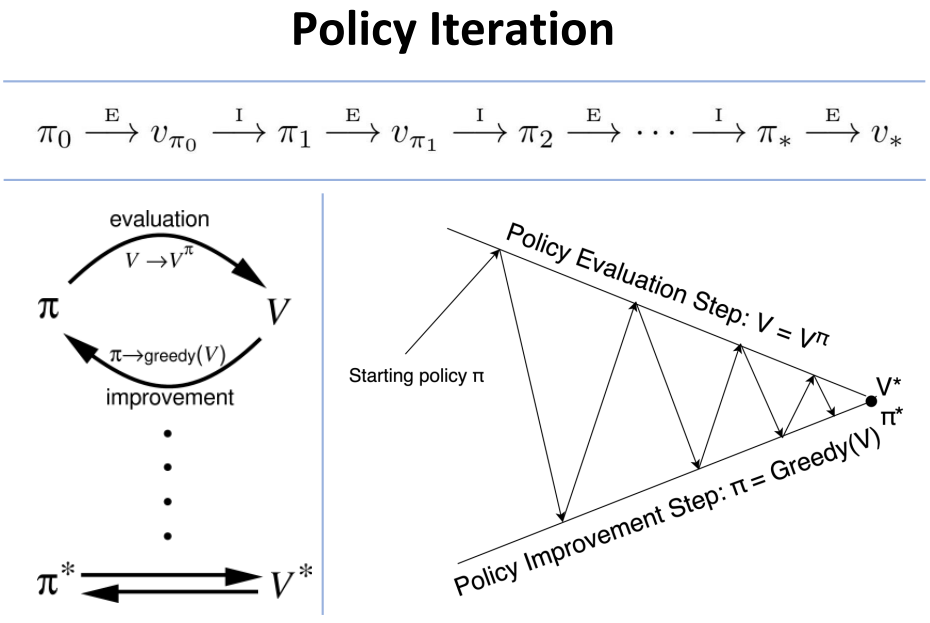

Figure 2.2: Policy iteration algorithm: it is a sequence of policy evaluation and a greedy policy improvement steps until the optimum is reached [73].

After the policy evaluation phase, policy improvement generates the greedy policy from the value function, by selecting, in each state, the action that maximizes it.

$$\pi^{t+1}(s) = \underset{a \in A}{\operatorname{argmax}}\, Q^{\pi^t}(s, a), \quad \forall s \in S \tag{2.18}$$

Theorem 2. *Let π, π' be two deterministic policies such that*

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s), \quad \forall s \in S \tag{2.19}$$

then the policy π' is not worse than π:

$$V^{\pi'}(s) \geq V^{\pi}(s), \quad \forall s \in S \tag{2.20}$$

Of course, by evaluating the policy employing the value function, to generate the greedy policy, we need the state transition model, $\mathcal{P}$. If the model is not available, Q-iteration is used. Here, instead of evaluating the value function of the policy, we evaluate the action-value function in a similar way.

Policy iteration at each iteration has a policy evaluation step that is ensured to converge only at the limit, so this is the main drawback of the policy iteration, which causes the reason why the value iteration algorithm has been introduced.

2.3.3 Value Iteration

Value iteration is based on the iterative application of the Bellman Optimality Operation discussed before, which finds the optimal value function (and from it the optimal policy) for finite MDPs without the intermediate use of policies. It should be noted that there also exist Approximate Value Iteration algorithms for continuous MDPs, such as FQI, which is discussed in 2.5.4. Value iteration starts from an initial value function, V^0, and then applies the Bellman operator iteratively until a stopping condition. Obviously, the stopping condition might be a maximal number of iterations or based on some minimal metric of the distance between two subsequent estimations of the value function. Finally, the optimal policy is found in the same way as

in policy iteration, by taking the greedy policy induced by the optimal value function. Also, the state transition model is required; otherwise, we need to estimate the optimal action-value function instead. The convergence to the optimal value function is guaranteed. Principally, given two consecutive approximations of the optimal value function, we can bound the error *w.r.t* the true optimal value function:

$$\| V^{t+1} - V^{s} \|_{\infty} < \epsilon \quad \Rightarrow \quad \| V^{t+1} - V^{*} \|_{\infty} < \frac{2\epsilon\gamma}{1-\gamma} \tag{2.21}$$

Value iteration focuses on the value function only, while policy iteration represents explicitly the policy. Therefore, intermediate value functions may not correspond to any policy. Both have polynomial time complexity for MDPs with fixed discount factor [49]. If we consider a single iteration, we will see policy iteration is more computationally demanding *w.r.t.* value iteration, since it requires evaluating the policy and performing the greedy improvement, but it tends to converge in a smaller number of iterations. However, Linear Programming (LP) becomes impractical at a much smaller number of states than DP methods do, besides DP; also, LP can be employed to recover the optimal value function.

2.4 Reinforcement Learning Taxonomy

When we are trying to solve a MDP, according to the problem setting, several algorithms can be applied. The goal of this section is to introduce some RL terminologies.

2.4.1 Reinforcement Learning Bandit vs. MDP

In RL, when the environment is unknown, one can describe it by observations, states, actions, reward, state transition probability, and conditional observation probabilities. Bandit, MDP, and POMDP model the environment by considering different elements differently. Bandit only explores which actions are optimal regardless of the state. In the classical Multi-Armed Bandit (MAB) [13, 36] policies, we assume the rewards are *i.i.d.* for each action in all the time [91]. Multi-Armed Bandits (MABs) are MDPs with only one state, so the actions do not change the state of the environment while in RL

in general, the actions can change the state in the environment. In addition, one-state RL (MAB) explores only the stochastic rewards, however MDP explores state transition probability too [44].

2.4.2 Model-based vs. Model-free

The goal of RL is to maximize rewards for our actions, and these rewards are dependent on the policy and system dynamics (model). If the agent has certain knowledge about the environment's model, and the model is known, it means the algorithm is model-based. On the other hand, if the learning process occurs without any accurate information of the transition model, it means we have a model-free algorithm.

2.4.3 Infinite Horizon vs. Episodic

If we have a unique infinite trajectory according to the experience which is presented to the algorithm, it means we have a task with an infinite horizon. On the other hand, the task is episodic if we have trajectories of finite length, which means every trajectory is a different episode.

2.4.4 Episodic vs. Non-Episodic

Algorithms that need a full episode to perform an update are episodic. However, non-episodic algorithms can learn even from incomplete episodes, which are the only ones applicable to MDPs where episodes can be infinite.

2.4.5 On-line vs. Off-line

On-line algorithms update their information (value functions and/or policy) for finding the problem's solution during the generation of data while off-line algorithms require a full dataset to perform the learning.

2.4.6 Greedy vs. $\epsilon-$Greedy

These algorithms are the simplest possible algorithms for the trading of exploration and exploitation, but they suffer from the difficulty, which is they have sub-optimal regret. Absolutely the regret of both Greedy and Epsilon Greedy increases linearly by the time.

It is understandable, as the Greedy will lock on one action that happened to have good results at one point in time, but it is not, in reality, the optimal action. So Greedy will keep exploiting this action while ignoring the others, which might be better. As a result, it *exploits* too much. The $\epsilon-$ Greedy, on the other hand, *explores* too much because even when one action seems to be the optimal one, the methods keep allocating a fixed percentage of the time for exploration, thus missing opportunities and increasing total regret. In contrast, the decaying $\epsilon-$Greedy methods try to decrease the percentage dedicated to exploration as time goes by.

2.4.7 On-policy vs. Off-policy

As far as we have two policies, one is used to interact with the environment, which is called *behavioral*, and the second one that is learned from the algorithm is *target*. If the two policies are the same, we speak about On-policy learning algorithms. If the two policies are different, we are referring to Off-policy learning algorithms.

Q-Learning

Q-learning is an off-policy RL algorithm that aims to find the best action to take given the current state. It is considered off-policy because Q-learning function learns from actions outside the current policy. In general, it seeks to learn a policy that maximizes the total reward. In this regard, Q-value can be updated using the following rule:

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)] \tag{2.22}$$

In Q-Learning, the agent learns optimal policy by using absolute greedy policy and behaves using other policies such as ϵ-greedy policy. As far as the updated policy is different from the behavior policy, so Q-Learning is off-policy.

SARSA

One of the algorithms which can be used for learning a MDP policy is State-action-reward-state-action (SARSA). In this algorithm, updating the Q-value depends on the current state of the agent S_1, the action the agent

selects A_1, the reward R which agent gets for choosing this action, the next state S_2 that the agent arrives after taking that action, and the next action A_2 the agent selects in its new state. The acronym for the quintuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ is SARSA [71].

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)] \tag{2.23}$$

Updating policy is how the agent learns the optimal policy, behavior policy is how the agent behaves. In SARSA, the agent learns optimal policy and behaves using the same policy, such as ϵ-greedy policy. So as opposed to Q-learning, SARSA is on-policy because the updated policy is the same as the behavior policy.

2.5 Learning Policy

As discussed in section 2.2.6, a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$ explains how a learner interacts with the environment, which maps recognized environmental states to actions. An optimal policy, π^*, is a policy that maximizes the total reward it receives in the long run. There are many possible approaches to learn such policy [74]. Here we discuss some of them.

2.5.1 Temporal Difference Methods

Temporal Difference (TD) methods, such as Q-learning [73, 85] and SARSA [74], learn by backing up experienced rewards through time. An estimated action-value function, $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is learned, where $Q(s, a)$ is the expected return found when executing action a from state s, and greedily following the current policy after that. According to the Q, which selects the action with the highest value for the current state, the current best policy will be generated. Exploration means that when the agent chooses an action to learn more about the environment, it must be balanced with exploitation, which means when the agent picks what it believes to be the best action. ϵ -greedy action select in, *i.e.*, the agent selects a random action with chance ϵ, and the current action is selected with probability $1-\epsilon$, is a simple approach that balances the two [75].

2.5.2 Batch Learning Methods

Batch learning methods (*e.g.*, Least-Squares Policy Iteration [35] and FQI [22] are offline so they do not attempt to learn as the agent interacts with the environment. Batch methods can save several interactions with the environment and use the data multiple times for learning, so they are designed to be more sample efficient. Moreover, these kinds of methods allow a distinct separation of the learning mechanism from the exploration mechanism, which is useful to understand whether to attempt to gather more data about the environment or exploit the current best policy.

2.5.3 Other approaches

Some other approaches such dynamic programming (see Section 2.3.1) and model-based or model-free are also available for learning an optimal a policy, and model-based or model-free, which discussed in Section 2.4.2.

The Batch Learning Problem

According to Sutton and Barto [73], the goal in the batch learning problem is finding a policy that maximizes the sum of expected rewards in the agent-environment loop. In addition, in this setting, the agent is not allowed to interact with the system during learning. It means that the learner only receives a set $f = \{(s_t, a_t, r_{t+1}, s_{t+1}) | t = 1, ..., p\}$ of p transitions $(s, a, r, s^{'})$ sampled from the environment instead of observing a state s, then selects an action a and modify the policy based on the subsequent state $s^{'}$ and reward r [39]. In the generalization case for the batch learning setting, the learner cannot consider any assumption on the transitions' sampling procedure. We only can sample arbitrarily, which can be purely random or uniform from the sate-action space $S \times A$. In the end, the agent uses only this information to find a policy for interacting with the environment. The policy cannot be improved in this application phase, even with new observation. The learner cannot be anticipated to find an optimal policy because it should not interact with the environment; also, the given set of transitions mostly is finite [39]. The goal here is to obtain the best possible policy from the given data instead of learning an optimal policy. According to figure 2.3, the whole procedure separate into three phases, the exploration phase that collects samples of the form $< s, a, s^{'}, r >$ (sampling phase), and the offline algorithm which is

on the base of samples compute action-value function (learning phase) and application of the learned policy [39, 42].

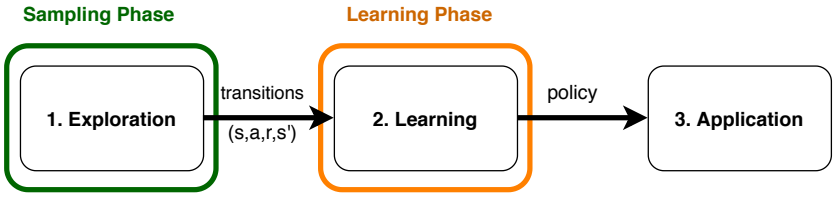

Figure 2.3: Three main phases of the process of batch learning in RL. The first phase is collecting transitions with an arbitrary sampling strategy(sampling phase). The second stage corresponds to the application of (batch) RL algorithms in order to learn the best possible policy from the set of transitions(learning phase). The third phase is the application of the learned policy. Note that in this step, the exploration is not part of the batch learning task. During the application phase, that is not part of the learning task either, policies stay fixed and are not improved further [39]

2.5.4 Fitted Q-Iteration

Fitted Q-iteration (FQI) algorithm [22] is a model-free, off-policy batch mode RL algorithm that belongs to the family of approximate Value Iteration. Its main goal is to learn a good approximation of the optimal action-value function $Q^*(s, a)$, by iteratively extending the optimization horizon [21]. FQI can be applied in supervised learning techniques. It considers a full dataset f containing the information that gets from experience because the algorithm is offline. Each sample in the dataset describes an interaction with the environment, and includes four tuples:

- s_t^i belongs to current state;
- a_t^i denotes to action performed;
- r_{t+1}^i is the immediate reward;
- s_{t+1}^i expresses the next state.

$$f = \{(s_t^i, a_t^i, r_{t+1}^i, s_{t+1}^i) | i = 1, 2, ..., |f|\}. \tag{2.24}$$

The data represents all the experience the an agent has collected *i.e.* exploited to infer an estimate of the optimal action-value function $Q^*(s, a)$. In FQI, the agent cannot directly interact with the environment, which generates the samples and updates a policy or a value function.

In continuous or very large discrete state and/or action spaces, the dataset does not contain all possibilities. To deal with this issue FQI performs a Supervised Learning regression with the action-value function as the output that must be learned. If the regressor can predict the value of any state-action pair, it can also estimate the action-value function. At each iteration of the algorithm, the horizon in which the optimization of the action-value function is performed increases one step. $Q_1^*(s,a)$ is estimated at the first iteration, so the action-value function becomes optimal only concerning the next step, which means the immediate reward. By applying the regressor to the training set, an approximation of this function can be obtained. The current state and action pair (s_t, a_t) is considered as an input and immediate reward (r_{t+1}^i) is a target. Generally, for N^{th} iteration, the estimated function will be $Q_N^*(s,a)$, which is the optimal action-value function according to N steps. Then the training set will be once again the couple (s_t, a_t) of current state and action; on the other side, the target will be iteratively computed as the immediate reward and the discounted best value over $N-1$ steps. As expected, the first step is separately determined by the action, while the following $N-1$ steps have the optimal value given by $Q_{N-1}^*(s_{t+1}, a_{t+1})$ which is the function that was approximated in the previous iteration.

2.6 Regression

In ML, regression is a supervised learning approach in which we estimate the model that encompasses the relationship between input variables and the output. This allows us to predict or estimate the conditional expectation of the output variable, given a set of input variables' values. In the simplest way, it is linear, where we are dealing with a single variable, and we drew this relationship to fit best such that the Mean Square Error (MSE) between the predicted value and the real value of the output would be minimized. Non-linear forms of regression include polynomial regression or any other type of function that provides non-linearity [24]. In this work, we used regression methods in order for prediction purposes, and in the following, we explain briefly about the methods we used.

2.6.1 Gaussian Processes Regression

GPs [88] is a nonlinear Bayesian approach to perform regression. It is a non-parametric method in the sense that it is of a particular functional form; accordingly, it calculates the probability distribution over a set of functions that fits the data rather than calculating the probability distribution of a specific function's parameters [16, 87]. In the Bayesian setting, a GPs can be considered as a prior probability for making inference [46, 88]. An example of prediction using GPs is presented in Figure 2.4.

Let the covariance matrix of multivariate Gaussian parameters be the Gram matrix of points in the dataset with any kernel, and we can sample from the Gaussian distribution. Then, another covariance matrix is constructed, which explains the correlations between all input and output variables considering the dataset points in the desired domain [3]. The inference of continuous values using a GPs prior is known as Gaussian process regression [64].

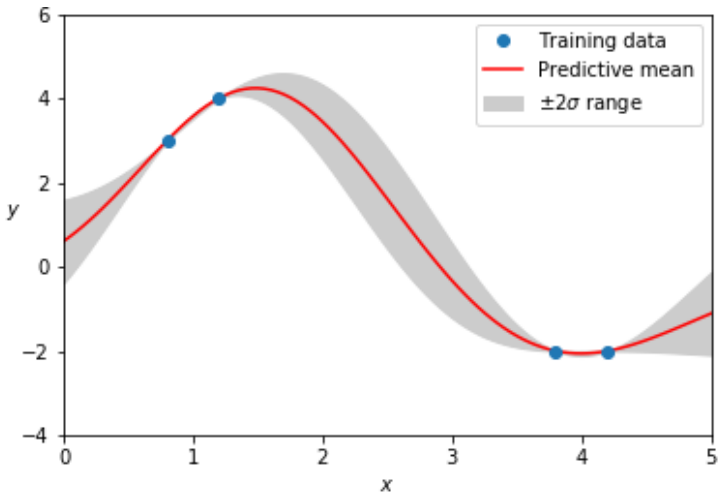

Figure 2.4: An example of prediction using GPs. The gray region accounts for the uncertainty of the model to predict the output considering input values.

2.6.2 Extra Tree Regression

It is a tree-based ensemble method for regression problems. The main idea that randomized decision trees perform as well as the classical trees has introduced by Mingers *et. al* in 1989 [50].

The Extra-Trees algorithm constructs a group of unpruned decision or re-

gression trees according to the typical top-down approach. Its only difference with other tree-based ensemble methods is that in the Extra Tree algorithm, the nodes are split in some cut-points, which are selected entirely randomly. In addition, it uses the whole learning sample instead of a bootstrap replica to grow the trees. Extra tree regression, in the extreme case, constructs totally randomized trees whose structures are independent of the output values of the learning sample. One advantage of this randomization is that it can be tuned to a particular problem we are dealing with by selecting the appropriate hyperparameter, hence it is very accurate. Besides it is computational efficient [26].

Chapter 3

Transfer Learning in Reinforcement Learning

In this chapter, we provide the reader with the background and related works in the transfer learning. In Section 3.1, we provide a brief overview of the TL in general. Since the scope of this work is the TL in RL domain, the rest of the chapter is focused on the TL in RL. In Section 3.5 we discuss the related works done in TL; and since the objective of this work is to design an active transfer of samples we dedicated Section 3.5.1 to discuss state-of-the-art algorithms in the sub-field of *Transfer of Samples* in RL.

3.1 Overview

TL is a broad research area that serves well many ML and DM applications; many ML and DM algorithms are built upon a primary assumption indicating that the data used for training the algorithms share the same feature space and distribution with the future data. However, when dealing with real-world problems, this assumption may not hold for many of the applications. For instance, consider a classification task which suffers from the scarcity of training data in the domain of interest while we have access to a sufficient amount of data from another domain with different feature space and/or data distribution. For such a problem, proper TL would improve the performance of learning while avoiding expensive efforts such as data-labeling [55].

The intuition behind TL, is that not only the generalization can happen *within tasks*, but also it may happen *across tasks* [75]. As depicted in Figure 3.1a, traditional ML algorithms learn each task independently, while TL algorithms (see Figure 3.1b) aim at transferring the knowledge obtained from some source tasks (the tasks learned before) to a target task in case of access to fewer but high-quality training data for the target task.

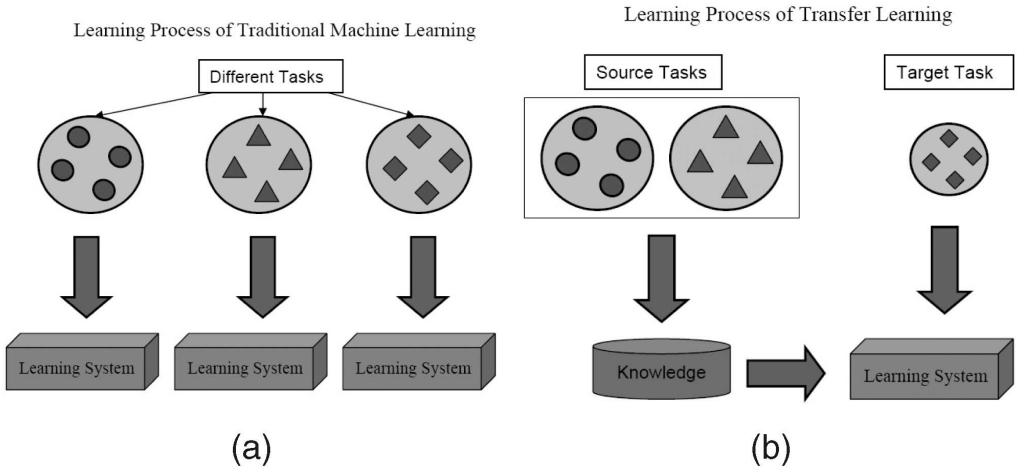

Figure 3.1: Different learning processes between (a) traditional machine learning and (b) transfer learning [55].

For more explanations of the TL in various fields, we refer the reader to [55,86] surveys, and the rest of the work focuses on the applications of TL in RL.

3.2 Motivation

In RL problems [73], agents aim at maximizing their obtained reward (maybe time-delayed) through taking a sequence of actions. The popularity of the RL framework comes from its learning methods' capability, which makes it powerful to handle extremely complex problems [75]. Nevertheless, when RL agents begin learning *tabula rasa*, mastering difficult tasks could become extremely long that it is infeasible. In short, the advantages of TL algorithms notably when dealing with *budget* constraint are as follow:

- Accelerate the learning process.
- Minimize the required data from the target task.

In the following, we will discuss the taxonomies and state-of-the-art concerning this work, and we refer the reader to the surveys [42, 75] for broader information regarding TL in RL.

3.3 Evaluation Methods

This section explains some of the concepts related to the evaluation of TL algorithms. The first evaluation metric is autonomy. It is said that the agent in TL algorithms should perform the following steps to be fully autonomous:

1. According to the target task, an appropriate source task is selected.
2. Learn the relation between source task and target task
3. Transfer knowledge from the source task(s) to the target task.

TL research considers each of them independently, and until now, no TL methods are capable of accomplishing all three goals. The majority of algorithms, however, consider the first two steps performed by a human, and ignore the time spent learning the source tasks.

In addition, there are many metrics to measure transfer's effectiveness [75]:

1. *Jumpstart*: Transfer from the source tasks can cause an improvement of the initial performance in the target task
2. *Asymptotic Performance*: The final learned performance of an agent in the target task may be improved via transfer.
3. *Total Reward*: The total reward accumulated by an agent, can be improved transferring from the source tasks, for learning without performing the transfer.
4. *Transfer Ratio*: The ratio of the total accumulated rewards with the transfer to the total accumulated rewards without transfer.
5. *Time to Threshold*: The number of samples needed by an agent to achieve a pre-specified performance level may be reduced via transfer.

One problem of these metrics is that during different times one algorithm can perform well in some metrics and bad in others, so evaluation under multiple metrics is a good suggestion. There is a visual representation of these metrics in figure 3.2 [75].

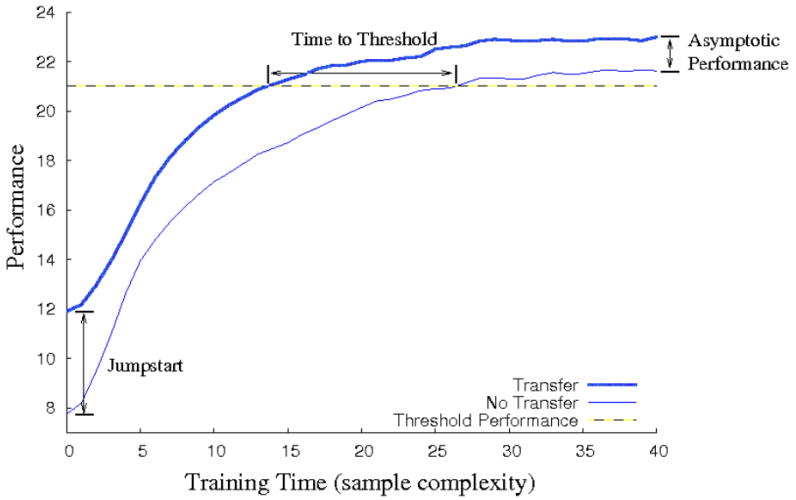

Figure 3.2: Visual representation of the performance measures, in this case the transfer algorithm outperforms the no transfer one. The plot shows the advantages to the jumpstart, asymptotic performance, time to threshold and total reward (the area under the learning curve) [75].

3.4 Negative Transfer

Negative transfer is one of the main challenges in TL. Intuitively, it is the negative impact of using a part of the knowledge that deteriorates the learning algorithm's performance with respect to learning from scratch. Therefore, it happens in case the transfer introduces incorrect learning biases and results, which can be a result of a similar source and target task.

Although we can say that it is possible to promote learning in a target task faster by transferring the knowledge, there are few methods for determining when the method might fail or the usefulness of a given transfer approach. These methods, such as Master [74] and region transfer [43] try to measure task similarity, and according to that, the agent decides about when and what to transfer; however, many of them do not guarantee the effectiveness of transferring. One of the methods for controlling negative transfer is IWFQI [82] proposed by Tirinzoni *et. al* in 2018 which is discussed in section 3.5.1.

Figure 3.3 depicts an example of negative transfer. It considers the pair of tasks, one source and one target, which are extremely similar, but direct transfer of a policy or action-value function is harmful [75].

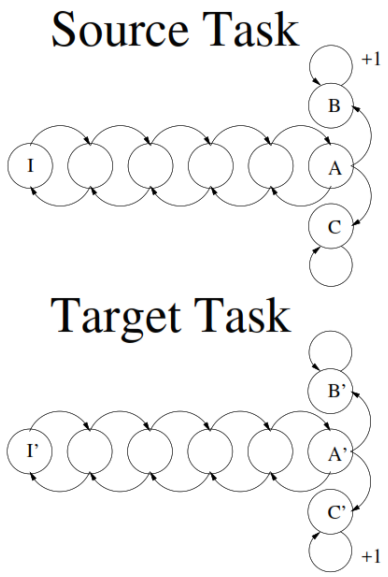

Figure 3.3: This figure represents a pair of tasks which are likely to result in negative transfer for TL methods [75].

According to Figure 3.3, the agent begins from state I and action East. Other states can go East or West, except for state A, which also has North and South actions. Once the agent plays North or South in state A, it will remain in state B or C, respectively, and follow self-transitioning. Except for the self-transition in state B, no transition has a reward. Then consider the target task, which is the same as the source task, except that the self-transition from C' is the only reward transition in the MDP. Q^*(I', East) in the target task is the same as Q^*(I, East) in the source task. As expected, the optimal policy in the target task is different at only a single state, A', and only at states A', B', and C' the optimal action-value function is different [75]. In this figure, transferring the knowledge involving these different states leads to negative transfer, which concerns mostly the reuse of policies or value functions, while other kinds of knowledge (such as samples) might still be transferred without harming the learning process.

3.5 Related Works

In this section, we provide the related works regarding TL in the domain of RL. In Section , we discuss the state-of-the-art algorithms concerning the *Transfer of Samples*, and the recent advancement in *Trade-off between Information and Reward* are presented in Section 3.5.2. Moreover, in Sections 3.5.3, 3.5.4 and 3.5.5 we provide the other related works in *Transfer of Value Function*, *Policy Transfer* and *Modeling Global Task Dynamics* respectively.

3.5.1 Transfer of Samples

In this section, we review the state-of-the-art methods related to our work in the sub-field of *Transfer of Samples* in RL.

Taylor *et al.* in 2008 propose TIMBREL [74] as a method to transfer the samples in model-based RL. They assumed that the state-action space of tasks might be different and employed intertask mappings to map source samples to target. Since TIMBREL does not acknowledge the differences in the transition or reward models, it can potentially result in considerable negative transfers if the tasks are different.

In [17], Crammer *et. al* extend the classical generalization bounds to the case where samples from source tasks are directly transferred. They investigate the *trade-off* between the total number of samples transferred and the total number of source tasks. According to their findings, the *variance* decreases as the total number of samples transferred increases and potentially introduces *bias* due to the differences between the tasks. Asymmetrically, *bias* decreases, and *variance* increases as the total number of samples is transferred due to the limited number of samples.

Garcke & Vanck in 2014 [25] is another related work concerning the transfer of samples from different datasets. They proposed a method that employs importance weighting aiming at correcting the distribution shift. Their idea leverages from density-ratio estimation, for example, work [68], and since they suppose a given parametric form for the weight function, it has to be estimated directly from the data.

Another related work is the RBT method proposed by Lazaric *et al.*, 2008 [43]. They build the algorithm under the assumption that tasks have similar transition models and reward functions. In order to reduce the learning

complexity of the batch RL, RBT reduces the number of samples an agent requires to collect from the target task by selecting samples from those source tasks, which are mostly similar to the target task.

The reported results confirm that RBT effectively reduces the complexity of learning even in the situations that some source tasks are extremely different from the target task. To do so, RBT computes two criteria: *Task Compliance* and *Sample Relevance*. Task Compliance determines the degree in which each source task is different from the target task in the form of a *model identification problem*. Using these criteria, RBT can transfer samples from each source task proportionally. On the other hand, since Task Compliance is not able to suggest which samples in a given source task are actually better to transfer, RBT is empowered with Sample Relevance measurement, which indicates which samples to be transferred.

The work [40] by Laroche & Barlier in 2017 introduced an algorithm named as SDT for a particular set of Transfer RL problems. SDT is designed for the problems in which all tasks share the same transition models, and only the reward functions differ from task to task. SDT aims to reuse the learned transition model from one task for other tasks, and for each sample, an immediate reward estimator is learned. Besides, because of the knowledge of the transferred dynamics from tasks, *the optimism in the face of uncertainty* principle enables SDT to explore a new task efficiently.

To solve a target task, most of the works focus on selecting the most relevant source samples but ignoring the differences between the task models. However, in 2018 Tirinzoni *et al.* proposed a novel model-based algorithm named as IWFQI [82] to transfer samples. IWFQI automatically estimates the relevance (*i.e.* importance weight) of each source sample for solving the target task and transfer all the samples by employing Importance Sampling (IS) [54]. To maximize the transformation of the information, they employ two different GPs for reward and transition models. Then, the transferred samples, along with their estimated relevance, are used by a batch RL (in their case FQI) to solve the target task. The contribution of each sample to the learning process is proportional to its estimated relevance.

To test their proposed algorithm, they conducted some experiments and compared the performance of IWFQI versus SDT, RBT, and FQI discussed earlier. The results confirm that IWFQI accomplishes better learning perfor-

mance, and even in the case of substantial differences among some of the source tasks with target tasks, it is truly robust to the negative transfer (see Section 3.4).

In 2019, Tirinzoni *et al.* proposed an approach for reusing samples in policy search methods [80]. The proposed method takes a complex case situation into account, in which the complete trajectories (*i.e.* the followed steps during each episode) between environments that are not similar in transition models should be transferred by the agent. The proposed method makes use of IS techniques for transferring the source samples while it corrects the bias introduced due to the differences in the source distributions. To reduce the sample complexity and convergence time while transferring entire trajectories for improving the gradient estimates of a policy search algorithm, they employed ideas from Multiple Importance Sampling (MIS), introduced by Veach & Guibas in 1995 [84].

To reduce the variance of the MIS estimator while avoiding introducing any bias, they employed two techniques. The first technique is *Per-decision Estimators* (Precup, 2000 [58]) *i.e.* a common variance reduction approach. The other one is *Regression-based Control Variates*. The control variates are among the widely utilized variance reduction techniques for general Monte Carlo estimators [28].

3.5.2 Trade-off between Information and Reward

The intention of this section is to discuss two recent works in 2020 concerning the strategies for the trade-off between information and rewards for identifying tasks.

Tirinzoni *et. al* in [78] proposed a novel phased confidence-based algorithm for structured bandits problems in which the arms' rewards are potentially correlated. It exploits the provided structure, utilizes sets of confidence over the target bandit problem's parameters, and quickly dumps all sub-optimal arms. It promptly dumps all the sub-optimal arms by exploiting the provided structure and utilizing sets of confidence over the target bandit problem's parameters.

To do so, at the start of each phase, the algorithm builds a set of bandit models that are compatible with the already computed confidence intervals, and

matching optimal arms are pulled in order till the end of the phase. Their interesting finding is that to detect sub-optimality, unlike existing works, which confide only in the obtained samples from arm i, the proposed method exploits the information obtained during pulling other arms, which reduces the number of pulls to the sub-optimal arm i. Moreover, as a result of the algorithm's requirements to know the horizon n, they designed a practical anytime extension (under the same assumptions in [41]) that derives a constant-regret bound which scales better in the relevant structure-dependent quantities; and, for specific structures which follow these assumptions, they derive matching lower bound which confirms the optimality of the proposed algorithm in the constant-regret regime.

At the start of each phase, the algorithm builds a set of bandit models that are compatible with the already computed confidence intervals, and matching optimal arms are pulled in order till the end of the phase. The interesting finding on their work detecting sub-optimality. Unlike existing works, which confide only in the obtained samples from arm i, the proposed method exploits the information obtained during pulling other arms which reduces the number of pulls to the sub-optimal arm i.

The other work concerning the trade-off between information and reward is the one conducted by Tirinzoni *et. al* in [79], which is about sequential transfer in RL in the situations in which the agent has access to a generative model of the state-action pairs. The main objective of this work is to investigate information to determine the optimal solution very fast, but sacrificing the good initial behaviour of the agent. At first, according to a given set of solved tasks that encompass an approximation of the target task, the proposed methodology through the so-called *Policy Transfer from Uncertain Models (PTUM)* algorithm pursues the most informative state-action pairs in order to instantly singles out a precise solution. To evidence the advantages of utilizing the methodology with such a piece of prior knowledge, they derived PAC bounds [67] on its sample complexity. Afterward, for the sake of learning the approximate tasks, the proposed *Sequential Transfer* algorithm reduces the transfer setting into a hidden Markov model and apply spectral methods, *i.e.* the tensor decomposition approach introduced in [4] for recovering the parameters.

3.5.3 Transfer of Value Function

TL of value function reuse the knowledge learned in one task to improve learning another related task.

In 2018, Tirinzoni *et. al* proposed a method [81], they addressed the problem of transferring value functions in RL. In this work, they proposed an approach using Gaussian and Gaussian mixtures and analyzed them by deriving a finite-sample in different domains. This approach used source tasks to learn from a prior distribution over all optimal value functions and employed variational approximation for estimating the corresponding posterior in a new target task. As a result, both algorithms achieved excellent performance in simple tasks, but mixtures of Gaussian deal better in a complex environment.

Another method for transferring value function is [6], which introduces a transfer framework that considers the reward function changes between tasks in the same environment's dynamics. The proposed approach is based on two concepts: Firstly, Successor Features (SFs), which means a value function design that separates the dynamics of the environment from the rewards. Secondly, Generalised Policy Improvement (GPI) *i.e.* a generalization of dynamic programming's policy improvement operation by considering a set of policies instead of a single one. In the end, the whole idea allows the free exchange of information across tasks. Two theorems proposed in this approach, show improvement in transferring the value function in practice.

According to [6], a framework [5] introduced based on two ideas SFs and GPI which is a novel way to transfer skills. The method contribution is to extend the SFs and GPI in two ways. Another method assumption is that the rewards for all the tasks of interest can be calculated as linear combinations of a particular set of features. At the same time, they relaxed this assumption and theoretically proved that the framework could be extendable for any set of tasks that are only different in the reward function. Besides, they show that one can use the reward functions themselves as features for future tasks hence removing the requirement to define a set of characteristics previously. Now it make sense to combine SFs and GPI with deep learning. Afterward, they empirically verified the method usefulness on a 3D environment where observations are images from a first-person perspective. They show that the transfer raised by SFs and GPI leads to very good policies on invisible

tasks almost directly. Moreover, they discuss the way of learning policies specialized to the new tasks such that it can be added to the agent's set of skills, and then be reused in the future.

Another related work is presented in [47], which uses structure mapping, *i.e.* finding the mapping between the source and target tasks and thus building the transfer automatically. It proposes an optimized version of the structure mapping engine and uses a heuristic search to find the most desirable maximal mapping. The source and target task features are the input of the algorithm, which are introduced as qualitative dynamic Bayes networks. They apply this method to the Keepaway task from RoboCup, a simulated soccer for comparing the result of the automated transfer to that from the non-automated transfer.

3.5.4 Transfer of Policy

MAXQINIT [1] addresses the problem of which knowledge should be transferred in lifelong RL. They formalized the answer through two families of results. First, they distinguish the initial policy that maximizes anticipated performance over the distribution of tasks for more complex classes of policies and task distributions. They then use these results to introduce MAXQINIT, an algorithm for updating initialization in enduring learning that trades off between tempered optimism and decreasing sample complexity. Running experiments explores the performance of the jump start policies and showcase the practicality of MAXQINIT for improvement of the algorithms in lifelong RL in domains in terms of speed.

A probabilistic approach introduced by [47] tries to reuse policies that are from past learned similar policies in order to improve RL. This method employs past policies as a probabilistic bias, where the learning agent faces three choices:

- The exploitation of the ongoing learned policy
- The exploration of random unexplored actions
- The exploitation of past policies

The new algorithm and its elements use an exploration strategy to incorporate the new reuse bias; also, they provide a similarity function for calculating

the extent to which the past policies are similar to a new one. The empirical results indicate that policy reuse improves learning performance over other methods that do not re-use policies.

3.5.5 Modeling Global Task Dynamics

Hidden Parameter Markov Decision Process (HiP-MDP) [19] addresses the domain in which we have similar but not identical tasks by making a model with few numbers of latent parameters, which uses data and a semiparametric regression approach to learn the structure of the new environment, such that the agent can efficiently adapt to the variations in the new task. It builds the model upon two assumptions.

1. there exists a finite number of latent parameters that, knowing such, would allow fully understanding the dynamics of the tasks.
2. the parameter values of the task remain fixed in the experiment.

The agent can be adapted to the dynamics of the new task rapidly by keeping and updating its distribution (or belief) over the hidden model parameters. The algorithm applies a point estimate for needed policy in the new environment based on the parameter values or does the planning in the belief space. The model uses the structure of Indian Buffet Process [27] and Gaussian processes [88]. They validated the method by an experiment on acrobat [70] domain challenge.

According to the formulation of the HiP-MDP [19], the method proposed in [32] try to model the joint uncertainty in the latent parameters and the state space. So this approach leverages a latent embedding to approximate the true dynamics of a task. They also replace the original Gaussian Process-based model with a Bayesian Neural Network, and this adjustment provides robust and efficient learning and also the scope of the is expended by higher dimensions and more complex dynamics.

Chapter 4

Active Transfer in Reinforcement Learning

This chapter includes the discussion about the TL method that we propose in this work. The algorithm transfers the samples from the source task to the target task. To do so, an evaluation metric is needed to decide which samples should be transferred. In Section 4.1, we explain the approach to find the best criteria as a *score* for ranking samples in the source tasks. Then in Section 4.2, we discuss how we employ such criteria in the algorithm. Finally the AWFQI algorithm is explained in Section 4.3.

4.1 Score Function Criteria

Active transfer learning encompasses selecting a set of state-action pairs to transfer to the target MDP, such that the policy performance would be maximized. The number of state-action pairs is constrained by a user-defined budget. At first, we need to decide which state-action candidates we should select, such that querying them from target MDP results in optimizing policy in target MDP. This criterion creates the motivation for the scoring function for AWFQI algorithm in a sense that we score each state-action pair according to this criterion so that we can use it in later stages. It should be noted that the underlying assumption here is that we only have access to the samples from source MDPs and not their generate models. On the other hand, we

have access to the generative model of the target MDP.

4.1.1 Problem Setting

To design score functions, we start from a simplified setting in which we have access to a finite set of discreet MDPs, which we have access to their generative models, and the target task is among them. Then we need to identify which of the MDPs is the target one. Then, in Section 4.2, we relax these assumptions and extend the score functions to the setting where we only have samples from a finite set of continuous source MDPs without necessarily having the target among them, and we have only access to the generative model of the target MDP.

We suppose each MDP $\mathcal{M}_\theta$ is parameterized by some parameter $\theta \in \Theta$ characterizing its transition probabilities and reward distribution (the other components are fixed and shared among all MDPs). We assume we are given a finite set of possible MDP parameters $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$. Each MDP $\mathcal{M}_\theta$, for $\theta \in \Theta$, is fully known. However, the identity of true MDP that the agent is facing $\mathcal{M}_{\theta^*}$, parameterized by $\theta^* \in \Theta$, is unknown. We further suppose that we have access to a generative model of the true environment which takes as input any state-action pair (s, a) and returns a sample next state $S' \sim p_{\theta^*}(\cdot|s, a)$ and a sample reward $R \sim q_{\theta^*}(\cdot|s, a)$. Then, for a given budget $n > 0$, our goal is to choose n state-action pairs to query the generative model, which allows us either to identify the true parameters θ^* or to compute the optimal policy π^*.

4.1.2 Proposed Approach

A high-level procedure for solving our problem is outlined in Procedure 1, which is composed of four main steps. We described these steps in the next paragraphs.

Procedure 1

Input: Set of possible MDP parameters Θ, budget n, prior belief b_θ^0

Output: Policy π^*

1: Choose n state-action pairs: $\mathcal{D}_{sa} = \{(s_i, a_i) \mid i = 1, \dots, n\} \leftarrow$ SAMPLE-SA(Θ, n)
2: Obtain next states and rewards: $\mathcal{D} = \{(s_i, a_i, S_i', R_i) \mid i = 1, \dots, n\} \leftarrow$ GENERATIVEMODEL($\mathcal{D}_{sa}$)
3: Update belief of each MDP: $b_\theta \leftarrow$ UPDATEBELIEF(Θ, $\mathcal{D}$, b_θ^0) $\forall \theta \in \Theta$
4: Compute optimal policy: $\pi^* \leftarrow$ COMPUTEPOLICY(Θ, b_θ)

Choosing the state-action pairs

The function SAMPLE-SA(Θ, n) takes as input the set of MDP parameters Θ and the budget n and produces a dataset $\mathcal{D}_{sa}$ of state-action pairs at which the agent should query the generative model. This is the key step of the algorithm. Intuitively, state-action pairs should be chosen such that the dataset obtained after querying the generative model is informative for identifying or solving the unknown MDP $\mathcal{M}_{\theta^*}$. Different approaches to solve this step are discussed in Section 4.1.3.

Querying the generative model

Once we have the set $\mathcal{D}_{sa}$ computed at the previous step, we call the generative model of the true environment ($\mathcal{M}_{\theta^*}$) to obtain a sample next state and reward for each $(s, a) \in \mathcal{D}_{sa}$. The output is another dataset $\mathcal{D} = \{(s_i, a_i, S_i', R_i) \mid i = 1, \dots, n\}$ where $S_i' \sim p_{\theta^*}(\cdot|s_i, a_i)$ is a sample from the transition probability at (s_i, a_i), while $R_i \sim q_{\theta^*}(\cdot|s_i, a_i)$ is a sample from the reward distribution at (s_i, a_i). We notice that $p_{\theta^*}(\cdot|s, a)$ is a categorical distribution over the finite set $\mathcal{S}$ for each $(s, a) \in \mathcal{S} \times \mathcal{A}$ and drawing a sample from it is simple. On the other hand, we could model $q_{\theta^*}(\cdot|s, a)$ as a Gaussian distribution with mean $r_{\theta^*}(s, a)$ and variance $v_{\theta^*}(s, a)$. Therefore, drawing a sample from q_{θ^*} reduces to drawing a samples from $\mathcal{N}(r_{\theta^*}(s, a), v_{\theta^*}(s, a))$.

Updating the belief over the true MDP

The algorithm keeps a probability distribution over Θ representing its belief over which of the given MDPs is the true one. The initial belief b_θ^0 is provided as input. In the simplest (non-informative) settings, such belief could be

uniform over Θ, *i.e.*, $b^0_\theta = \frac{1}{K}\ \forall \theta \in \Theta$. Once we are given the set $\mathcal{D}$ computed at the previous step, this belief can be updated as follow:

For each $\theta \in \Theta$, we want to compute the probability that the true parameter θ^* is θ. By applying Bayes' rule we have:

$$\begin{aligned}\mathbb{P}(\theta^* = \theta|\mathcal{D}) &= \frac{\mathbb{P}(\mathcal{D}_{sr}|\theta^* = \theta, \mathcal{D}_{sa})\mathbb{P}(\theta^* = \theta)}{\mathbb{P}(\mathcal{D}_{sr}|\mathcal{D}_{sa})} \\ &= \frac{\mathbb{P}(\mathcal{D}_{sr}|\theta^* = \theta, \mathcal{D}_{sa})\mathbb{P}(\theta^* = \theta)}{\sum_{\theta' \in \Theta} \mathbb{P}(\mathcal{D}_{sr}|\theta^* = \theta', \mathcal{D}_{sa})\mathbb{P}(\theta^* = \theta')},\end{aligned} \tag{4.1}$$

where the dataset $\mathcal{D}$ has been decomposed into the two parts $\mathcal{D}_{sa}$ and $\mathcal{D}_{sr}$. Let us forget the normalization term and focus on the numerator. The term $\mathbb{P}(\mathcal{D}_{sr}|\theta^* = \theta, \mathcal{D}_{sa})$ can be further decomposed by exploiting the fact that samples are *i.i.d.*,

$$\begin{aligned}\mathbb{P}(\mathcal{D}_{sr}|\theta^* = \theta, \mathcal{D}_{sa}) &= \prod_{i=1}^{n} \mathbb{P}(S'_i, R_i|\theta^* = \theta, s_i, a_i) \\ &= \prod_{i=1}^{n} \mathbb{P}(S'_i|\theta^* = \theta, s_i, a_i)\mathbb{P}(R_i|\theta^* = \theta, s_i, a_i) \\ &= \prod_{i=1}^{n} p_\theta(S'_i|s_i, a_i)q_\theta(R_i|s_i, a_i).\end{aligned} \tag{4.2}$$

On the other hand, the term $\mathbb{P}(\theta^* = \theta)$ is simply the initial belief of the parameter $\mathbb{P}(\theta^* = \theta) = b^0_\theta$. Putting all together, the updated (unnormalized) belief can be computed as

$$\tilde{b}_\theta = \prod_{i=1}^{n} p_\theta(S'_i|s_i, a_i)q_\theta(R_i|s_i, a_i)b^0_\theta \quad \forall \theta \in \Theta. \tag{4.3}$$

The corresponding final belief can be computed after normalizing,

$$b_\theta = \frac{\tilde{b}_\theta}{\sum_{\theta' \in \Theta} \tilde{b}_{\theta'}} \quad \forall \theta \in \Theta. \tag{4.4}$$

Computing the final policy

This is another key step in the algorithm. Now that we have an updated belief over the identity of the true MDP $\mathcal{M}_{\theta^*}$, we must exploit this information to return the best possible policy.

4.1.3 Active Learning over the State-Action Space (Choosing the State-Action Pairs)

We propose different methodologies for sampling the state-action space. In all cases, we first define a *score* function $\phi \in \mathcal{P}(\mathcal{S} \times \mathcal{A})$ which is a valid probability distribution over the state-action space. Then, we take n *i.i.d.* samples $(S_i, A_i) \sim \phi$.

Uniform baseline

The first very naive approach is to sample different states and actions with equal probability. Thus, for $i = 1, \dots, n$, we sample s_i with probability $\frac{1}{S}$ and a_i with probability $\frac{1}{A}$. Equivalently, the score function for this approach is $\phi(s, a) = \frac{1}{SA}$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.

It should be noted that this approach works only for discrete MDPs since the number of states actions must be finite. For continuous MDPs we need to adapt to a continuous uniform distribution.

Variance-based sampling

Intuitively, the state-action pairs that are most informative for identifying θ^* are those where the MDPs differ the most. Therefore, we consider a score proportional to the *variance* of rewards and next states between models.

Consider the reward function at some state-action pair (s, a). Using the law of total variance,

$$\begin{aligned} \mathbb{V}\mathrm{ar}[R|s,a] &= \mathbb{E}_\theta[\mathbb{V}\mathrm{ar}[R|s,a,\theta]] + \mathbb{V}\mathrm{ar}_\theta[\mathbb{E}[R|s,a,\theta]] \\ &= \underbrace{\mathbb{E}_\theta[v_\theta(s,a)]}_{\text{Intra-MDP}} + \underbrace{\mathbb{V}\mathrm{ar}_\theta[r_\theta(s,a)]}_{\text{Inter-MDP}}. \end{aligned} \tag{4.5}$$

Since our goal is to gather information about θ^*, we maximize the inter-MDP variance. Thus, we set the score for the reward function as

$$\phi_r(s,a) = \frac{\mathbb{V}\text{ar}_\theta[r_\theta(s,a)]}{\sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}}\mathbb{V}\text{ar}_\theta[r_\theta(s',a')]} \tag{4.6}$$

Let us now consider the transition probabilities. In this case we directly take the variance of the probabilities rather than the random next states. The score is

$$\phi_p(s,a) = \frac{\sum_{s'\in\mathcal{S}}\mathbb{V}\text{ar}_\theta[p_\theta(s'|s,a)]}{\sum_{s''\in\mathcal{S}}\sum_{a''\in\mathcal{A}}\sum_{s'''\in\mathcal{S}}\mathbb{V}\text{ar}_\theta[p_\theta(s'''|s'',a'')]}. \tag{4.7}$$

Then, our final score is simply the sum of two terms in Equations 4.6 and 4.7

$$\phi(s,a) = \phi_r(s,a) + \phi_p(s,a) \tag{4.8}$$

Information-based sampling

This approach uses the Mutual Information (MI) between the rewards/transition probabilities and the true parameter θ^* to define a score function that explicitly encodes the expected amount of knowledge gained when sampling a specific state-action pair.

Let us start with the transition probabilities. The MI is defined as:

$$\begin{aligned} I(S';\theta^*|s,a) &= \sum_{\theta\in\Theta}\sum_{s'\in\mathcal{S}}\mathbb{P}(s',\theta|s,a)\log\frac{\mathbb{P}(s',\theta|s,a)}{\mathbb{P}(s'|s,a)\mathbb{P}(\theta|s,a)} \\ &= \sum_{\theta\in\Theta} b_\theta^0 \sum_{s'\in\mathcal{S}} p_\theta(s'|s,a)\log\frac{p_\theta(s'|s,a)}{\mathbb{P}(s'|s,a)} \\ &= \sum_{\theta\in\Theta} b_\theta^0 \sum_{s'\in\mathcal{S}} p_\theta(s'|s,a)\log\frac{p_\theta(s'|s,a)}{\sum_{\theta'\in\Theta} p_{\theta'}(s'|s,a)b_{\theta'}^0}, \end{aligned} \tag{4.9}$$

which can be evaluated in closed-form. Similarly, the MI for the reward function is

$$
\begin{aligned}
I(R;\theta^*|s,a) &= \sum_{\theta\in\Theta}\int_{\mathbb{R}} \mathbb{P}(x,\theta|s,a)\log\frac{\mathbb{P}(x,\theta|s,a)}{\mathbb{P}(x|s,a)\mathbb{P}(\theta|s,a)}\mathrm{d}r \\
&= \sum_{\theta\in\Theta} b_\theta^0 \int_{\mathbb{R}} q_\theta(x|s,a)\log\frac{q_\theta(x|s,a)}{\mathbb{P}(x|s,a)}\mathrm{d}r \\
&= \sum_{\theta\in\Theta} b_\theta^0 \int_{\mathbb{R}} q_\theta(x|s,a)\log q_\theta(x|s,a)\mathrm{d}r \\
&\quad - \sum_{\theta\in\Theta} b_\theta^0 \int_{\mathbb{R}} q_\theta(x|s,a)\log \mathbb{P}(x|s,a)\mathrm{d}r \\
&= \underbrace{H(R|s,a)}_{(a)} - \underbrace{\sum_{\theta\in\Theta} b_\theta^0 H(R|s,a,\theta)}_{(b)},
\end{aligned}
\tag{4.10}
$$

where $H(X) := -\sum_{x\in\mathcal{X}} p(x)\log p(x)$ (or $H(X) := -\int_{\mathcal{X}} p(x)\log p(x)\mathrm{d}x$) is the entropy of a random variable X. As it is well known, the MI can be rewritten as the reduction in entropy before (term (a)) and after (term (b)) observing the value of θ^*. Since the entropy of a Gaussian distribution can be expressed in closed-form, term (b) can be easily computed as:

$$
\sum_{\theta\in\Theta} b_\theta^0 H(R|s,a,\theta) = \frac{1}{2}\sum_{\theta\in\Theta} b_\theta^0 \log(v_\theta(s,a)) + \frac{1}{2}\log(2\pi e). \tag{4.11}
$$

On the other hand, term (a) cannot be in general computed in closed-form since the distribution of R (conditioned on s,a) requires marginalizing over Θ. Therefore, we consider an empirical approximation by drawing m *i.i.d.* samples from $\mathbb{P}(R|s,a)$ as follows: for each $j=1,\dots,m$, first draw $\theta_j \sim b_\theta^0$, then draw $R_j \sim q_{\theta_j}(\cdot|s,a)$. Given these samples, we use a simple maximum-likelihood estimator of the entropy. First, we divide the reward space in K bins $\mathcal{B}_1,\dots,\mathcal{B}_K$ and compute the empirical distribution as:

$$
\hat{p}_k := \frac{1}{m}\sum_{j=1}^{m} \mathbb{1}\{R_j \in \mathcal{B}_k\}. \tag{4.12}
$$

Then, our entropy estimator is:

$$\hat{H}(R|s,a) = -\sum_{k=1}^{K} \hat{p}_k \log \hat{p}_k. \tag{4.13}$$

Hence, the MI can be approximated as:

$$\hat{I}(R;\theta^*|s,a) = \hat{H}(R|s,a) - \frac{1}{2}\sum_{\theta\in\Theta} b_\theta^0 \log(v_\theta(s,a)), \tag{4.14}$$

where we neglected the term $-\frac{1}{2}\log(2\pi e)$ since it does not depend on (s,a). Finally, since $I(S',R;\theta^*|s,a) = I(S';\theta^*|s,a) + I(R;\theta^*|s,a)$ due to the conditional independence between S' and R given (s,a), we set our score to:

$$\phi(s,a) = \frac{I(S';\theta^*|s,a) + I(R;\theta^*|s,a)}{\sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}} I(S';\theta^*|s',a') + I(R;\theta^*|s',a')}. \tag{4.15}$$

This step indeed requires the assumption that $\mathbb{P}(S',R|s,a) = \mathbb{P}(S'|s,a)\mathbb{P}(R|s,a)$.

4.1.4 Near-Optimal Policies for Markov Decision Processes with Discrete Uncertainty

Given the set of possible MDP parameters Θ and a distribution over Θ (our belief b_θ), we ideally would like to find a Markovian policy maximizing the expected return, where the expectation accounts also for the uncertainty over the true MDP:

$$\underset{\pi\in\Pi^{\mathrm{SR}}}{\operatorname{argmax}} \sum_{\theta\in\Theta} b_\theta \mathbb{E}_{\substack{S_{t+1}\sim p_\theta(\cdot|S_t,A_t)\\ A_t\sim\pi(\cdot|S_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(S_t,A_t)\right]. \tag{4.16}$$

Since the problem is very complicated, we employ an approximate solution. Maximum-a-posteriori is the simplest solution to return the optimal policy for the most likely MDP. That is, we first take $\hat{\theta} = \operatorname{argmax}_{\theta\in\Theta} b_\theta$ and then compute the optimal policy for $\mathcal{M}_{\hat{\theta}}$. This last step can be done using standard algorithms such as value iteration. This approach is likely to work well

when the true MDP has been identified with very high probability (*i.e.*, there exists some $\theta \in \Theta$ with $b_\theta \simeq 1$). Unfortunately, it is likely to fail when this does not happen, and the optimal policies for different MDPs significantly differ. Solving the problem presented in Equation 4.16 remains an open question to be worked on in future work.

4.1.5 Score Function Criteria Evaluation

We heuristically evaluate the effectiveness of approaches explained in Section 4.1 using Grid World environment and reporte the results in Figures 4.2 and 4.3.

The Grid World environment [72] is a two-dimensional discrete environment and an instance of a deterministic distribution model. A scheme of a simple Grid World is presented in Figure 4.1.

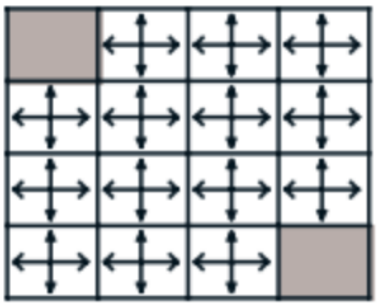

Figure 4.1: An example of a Grid World.

In this environment, the state and action spaces are both discrete. The size of the Grid World corresponds to the number of states along each axis. For example in figure 4.1 the size is four. Given a state s and an action a, we can get an array of the probability of the reward to receive and the next state. The goal of the agent is to reach it to one of the two gray corners. To encourage agent, every move will have a reward of -1 [20]. In this work, we employed a Grid World with only one goal state with $reward = +1$ and $reward = 0$ for the rest of the states. The size of the Grid World is 10×10. Source MDPs have different goal states, and the reward is only positive in that state. The agent start at state $(0, 0)$.

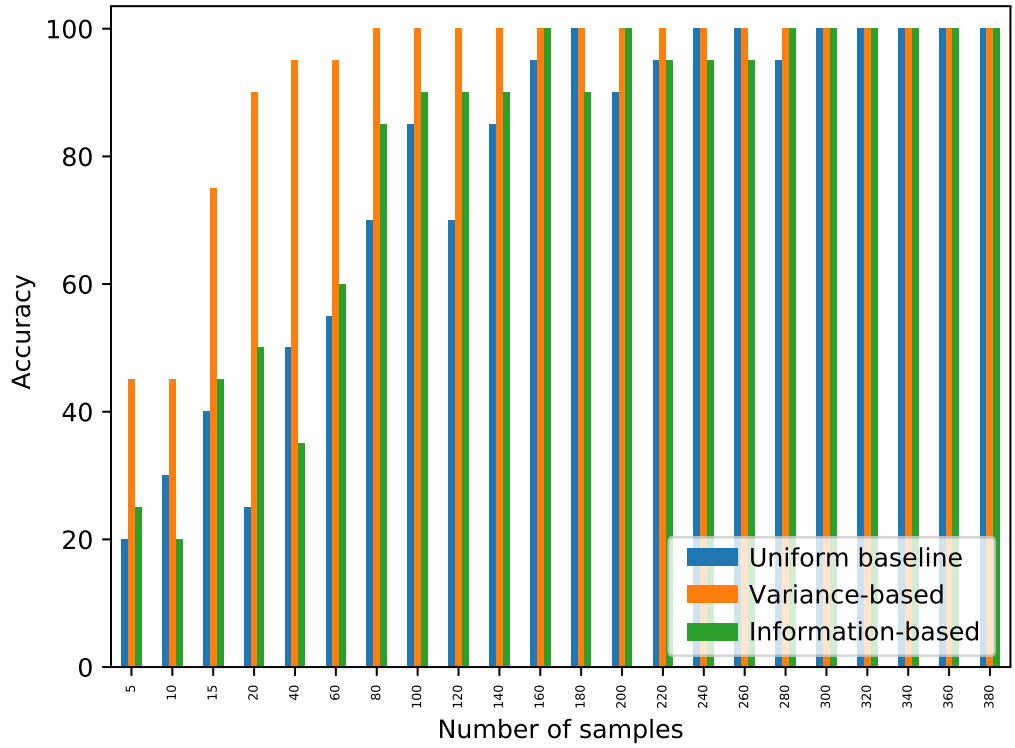

Figure 4.2: Percentage of the time sampling by different scoring criteria correctly identify true MDP.

Figure 4.2 compares the percentage of the times different approaches identify true MDP in 10 independent runs of the algorithm. The x-axis is the number of samples we select as our budget in that run, and the bars associated with that budget show the percentage of times different approaches select the true MDP correctly constrained by that particular budget. As expected and is clear in Figure 4.2, the random selection of the samples lead to different accuracy percentage in different sample sizes. In other words, increasing the budget does not necessarily increase the accuracy (certainty) of the selection. This randomness is more obvious in smaller budget size since as the number of samples increases, the algorithm selects almost all the source samples. This happens because the environment is discrete, and the number of state-action pairs is finite. Sampling, according to MI, also fluctuates occasionally in different sample numbers. However, variance-based sampling showed the best results through the runs, and even with few samples, this method can correctly identify the true MDP.

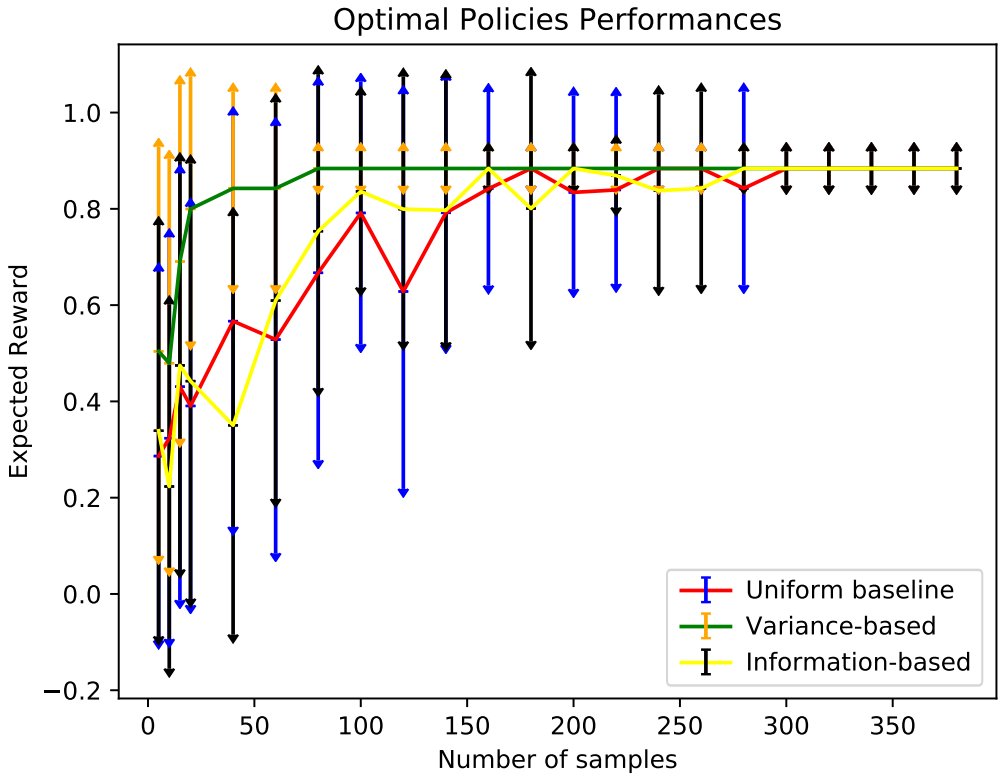

Figure 4.3: Performance of the approach.

We provide another comparison in Figure 4.3, which illustrates the expected reward in different approaches. This experiment is also done in the same setting of Grid World as explained before.

In this figure, the x-axis is the number of samples that can be selected (budget), and the y-axis shows the expected reward or policy performance using different approaches. This figure suggests that the policy performance in variance-based sampling has a distinguishable jumpstart compared to other approaches. However, the asymptotic behavior is similar among different sampling approaches.

4.2 Score Function Construction

In Section 4.1, we presented and analyzed three score functions for a simplified setting (where there is only finite possible known MDPs containing the unknown target) and assessed empirically that the variance-based sampling outperforms the other approaches (according to the results in Section 4.1.5). Hereof, we employ the variance-based approach to design a score function.

Now we move to a more general transfer setting where we consider a set of tasks, *i.e.*, MDPs, $\{\mathcal{M}_j = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_j, \mathcal{R}_j \rangle, j = 0, ..., m\}$, where $\mathcal{M}_0$ stands for the target MDP and $\mathcal{M}_1, ..., \mathcal{M}_m$ denote the source MDPs. We suppose all MDPs share the same state-action space and but have potentially different transition probabilities (dynamics) and rewards. Suppose that, for $j = 0, ..., m$, we are given a dataset of $\mathcal{N}_j$ samples from the j-th MDP, $\mathcal{D}_j = \{\langle s_i, a_i, s'_i, r_i \rangle\}_{i=1}^{\mathcal{N}_j}$, where state-action pairs are drawn from a common distribution $\mu \in \Delta(\mathcal{S} \times \mathcal{A})$. The goal of TL is to employ the samples in $\mathcal{D}_1, ..., \mathcal{D}_m$ such that they boost the learning process in the target task $\mathcal{M}_0$. Unlike the setting discussed in 4.1, we no longer assume that we have access to the generative model of the source MDPs. Moreover, the target MDP is no longer fully known but for which we can access a generative model for at most n samples (*i.e.* budget). In addition, the current transfer setting applies to both discrete and continuous environments.

It should be noted that, only for the sake of comparison, here we designed two score functions: an *ideal* score function that *has* access to the source generative models, and other score function which respects the mentioned assumption as it is expected in *real* TL algorithms. Then we use both versions of score functions in the proposed algorithm (AWFQI), and accordingly, We call the algorithm which uses the ideal score function as AWFQI_oracle and the other version of the algorithm which uses the realistic score function simply as AWFQI, since that is the real algorithm proposed here. With these explanations, we discuss the design of the score functions as follow.

1. Ideal score function
2. Realistic score function

As mentioned before, our motivation for implementing both score functions is being able to compare them and understand to what extent the errors of predictions affect the results.

The main goal is drawing samples from a score function $score : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^+$, in which $\mathcal{S}$ and $\mathcal{A}$ are state and action space respectively. So the score function is a mapping from state-action space to a real value between 0 and 1. Practically, the output of both score functions is a table which potentially can have enough number of rows to be considered as continuous when we want to draw sample from. This table is the posterior of score functions.

For simplifying the exposition, we provide an example where the state space is two dimensional, but the method directly extends to many dimensions.

In the case were the environment has two dimension, the table is composed of eight columns $idx, s_x, s_y, a, var(r|(s,a)), var(s'_x|(s,a)), var(s'_y|(s,a))$ and score *i.e.* $\sum_{col=4}^{7}$.

$$score(s,a) \propto Var[R|(s,a)] + Var[s'_x|(s,a)] + Var[s'_y|(s,a)] \tag{4.17}$$

We can consider only column 4 (variance of rewards) as the final score. In fact, in all the experiences here, we considered only the variance of rewards (column 4, not column 7). Eventually, we pass this table to the algorithm (the STEP_CORE function), where we draw samples according to these scores.

4.2.1 Ideal Score Function

Here we computed the variance of rewards and next states according to their real values obtained from the source MDPs. This score function is implemented only to compare with when we predict these values (Section 4.2.2).

Grid Mode

In this mode, we make a grid with a dx, dy as granularity in both dimensions. Then we obtain the actual values of rewards and next states for all nodes of the grid using source MDPs. Then we compute the variance of these values and make the score table (Figure 4.4). Note that the grids dx and dy can be very small to make the function/table continuous.

idx	s_x	s_y	a	$\text{var}(r \mid (s,a))$	$\text{var}(s'_x \mid (s,a))$	$\text{var}(s'_y \mid (s,a))$	$\text{var}(r \mid (s,a)) + \text{var}(s'_x \mid (s,a)) + \text{var}(s'_y \mid (s,a))$
	grid nodes state actions						

Figure 4.4: Score table in AWFQI_oracle, grid mode, generated using exact rewards and next states for the grid nodes state-actions.

Source Data mode

In this mode, we obtain the actual values of rewards and next states for all the source MDPs state-action pairs (available in the source dataset). Then we compute the variance of these values and make the score table as Figure 4.5. This mode cannot be considered as continuous because it has only the state-action pairs from the source files. Here we could fit a GPs on the score to make the function continuous, but we wanted to avoid any prediction since it is the oracle score function.

idx	s_x	s_y	a	$\text{var}(r \mid (s,a))$	$\text{var}(s'_x \mid (s,a))$	$\text{var}(s'_y \mid (s,a))$	$\text{var}(r \mid (s,a)) + \text{var}(s'_x \mid (s,a)) + \text{var}(s'_y \mid (s,a))$
	Source data state actions						

Figure 4.5: Score table in AWFQI_oracle, source data mode, generated using exact rewards and next states for the state-action pairs in all the source files.

Grid and Source Data Mode (Both)

This score function (table) is the concatenation of the two previous tables, we collect the state-action pairs in a grid with a user-defined dx and dy and also source dataset state-action pairs and obtain the real values of reward and next states for all of them (Figure 4.6).

4.2.2 Realistic Score Function

In this mode, we use GPs to predict the rewards and next states of the state-action pairs. To fit this model, we use source file data only, in all the following modes.

idx	s_x	s_y	a	$\text{var}(r\,\|(s,a))$	$\text{var}(s'_x\|(s,a))$	$\text{var}(s'_y\|(s,a))$	$\text{var}(r\|(s,a))+\text{var}(s'_x\|(s,a))+\text{var}(s'_y\|(s,a))$
	Source data state actions						
	grid nodes state actions						

Figure 4.6: Score table in AWFQI_oracle, grid and source data mode, generated using exact rewards and next states for the state-actions in the grid and in all the source files.

Grid Mode

We first create a grid according to dx and dy which can be small enough for the continuity of the output (score function/table). Then for each state-action pair node in that grid, we predict the values of rewards and next states in all MDPs using their source data (source dataset). Then we compute the variance of them and fill in the 8-column table $(idx, s_x, s_y, a, var(r|(s,a)), var(s'_x|(s,a)), var(s'_y|(s,a)), score(i.e. \sum_{col=4}^{7}))$ as showed in Figure 4.7.

idx	s_x	s_y	a	$\text{var}(\hat{r}\,\|(s,a))$	$\text{var}(\hat{s'}_x\|(s,a))$	$\text{var}(\hat{s'}_y\|(s,a))$	$\text{var}(\hat{r}\,\|(s,a))+\text{var}(\hat{s'}_x\|(s,a))+\text{var}(\hat{s'}_y\|(s,a))$
	grid nodes state actions						

Figure 4.7: Score table in AWFQI, grid mode, generated using predicted rewards and next states for the state-action pairs in the grid.

Source Data Mode

In this mode, we predict the values of rewards and the next states of state-action pairs that are in the source data. We do it by fitting GPs models on reward and next states of available data for each MDP. Then we compute the function/table as before (Figure 4.8).

idx	s_x	s_y	a	$\text{var}(\hat{r}\,\|(s,a))$	$\text{var}(\hat{s'}_x\|(s,a))$	$\text{var}(\hat{s'}_y\|(s,a))$	$\text{var}(\hat{r}\,\|(s,a))+\text{var}(\hat{s'}_x\|(s,a))+\text{var}(\hat{s'}_y\|(s,a))$
	Source data state actions						

Figure 4.8: Score table in AWFQI, source data mode, generated using predicted rewards and next states for the state-action pairs in source data.

Grid and Source data mode (Both)

This score function (table) in this mode is the concatenation of two previous tables; we collect the state-action pairs in a grid with a user-defined dx and dy and also the source dataset state-action pairs (Figure 4.9).

idx	s_x	s_y	a	$\text{var}(\hat{r}\,\|(s,a))$	$\text{var}(\hat{s'}_x\|(s,a))$	$\text{var}(\hat{s'}_y\|(s,a))$	$\text{var}(\hat{r}\,\|(s,a))+\text{var}(\hat{s'}_x\|(s,a))+\text{var}(\hat{s'}_y\|(s,a))$
	grid nodes state actions						
	Source data state actions						

Figure 4.9: Score table in AWFQI, grid mode mix with source data, generated using predicted rewards and next states for the state-action pairs in the grid and in source data.

4.3 Active Weighted Fitted-Q Iteration

We investigated the state-of-the-art algorithms in the transfer of samples discussed in Section 3.5.1 as the candidates for the foundation of our active TL approach. For the variety of reasons we decided to utilize IWFQI [82] as our main backbone in combination with variance-based sampling.

First of all, unlike TIMBREL [74], IWFQI selectively discards samples according to the estimated difference between the MDPs. Contrary to RBT [43], IWFQI does not expect any certain condition for deciding what to transfer, nor does it require any assumption of similarity between the tasks. Moreover, since RBT [43] computes the compliance and relevance criteria such that they mutually account for both the reward and transition models,

it ignores the samples in case one of the models is very different among the tasks. However, IWFQI preserves the part of the sample that is similar, at the cost of introducing a negligible bias. SDT [40] transfers the samples into FQI with the assumption that tasks follow the same transition dynamics. Although similar to IWFQI it learns the reward function at the first iteration and substitutes the predicted values to the reward of samples in the dataset; its assumption regarding the shared dynamics potentially hinders its applicability as a transfer method in most of the real-world tasks. Another advantage of IWFQI is that it handles the *trade-off* discussed in [17] since it transfers all the samples while accounting for the differences between the tasks. They reported a slight bias due to the errors in the estimation of the task models, which can be reduced by increasing the sample size. Finally, as opposed to the method by Garcke & Vanck [25], IWFQI estimates the densities involved and characterizes the weight distribution, taking its expectation as the final estimation.

Due to the reasons as mentioned above, we base our work on IWFQI, and by incorporating the sampling at each step, we call it AWFQI, which will be described in the next section.

4.3.1 AWFQI Algorithm

The high-level representation of AWFQI algorithm is shown in Algorithm 2. Given a set of samples from source tasks, according to the mode discussed in Sections 4.2.1 and 4.2.2 the GenerateScoreFunction computes the variance of the rewards and next states for the (s, a) pairs and after normalizing the variances build the ScoreFunction. After that, the DrawSA function takes as input the ScoreFunction and, according to the provided budget, draw candidate (s, a) pairs to be sampled from the target MDP. In line 3, the CollectTargetSamples function employs the generative model of the target MDP and collects the candidate (s, a) pairs list. In the next step, the CalculateWeights function computes the weights of the rewards and next states for the (s, a) pairs and provides the final dataset to be used by FQI for solving the target task. Finally, in each iteration, FQI updates the policy over the target task, and at the end of the iterations, provides the optimal learned policy π^*.

Algorithm 2 Active Importance Weighted Fitted Q-Iteration (AWFQI)

Input: Target MDP $\mathcal{M}_0$, Samples from Source MDPs $\mathcal{D}_1, ..., \mathcal{D}_m$, budget n, Number of iteration n_iter, The score function mode sf_mode

Output: Policy π^*

ScoreFunction$(\cdot) \leftarrow$ GenerateScoreFunction($\mathcal{D}_1, ..., \mathcal{D}_m$, sf_mode)
SA $\leftarrow$ DrawSA(ScoreFunction$(\cdot)$, n)
$\mathcal{D}_0 \leftarrow$ CollectTargetSamples($\mathcal{M}_0$, SA)
$\tilde{\mathcal{D}^+} \leftarrow$ CalculateWeights($\mathcal{D}_0, ..., \mathcal{D}_m$)
Set iter = 0
$\hat{\mathcal{R}} \leftarrow$ FitInitialModel($\mathcal{R}_0$)
$Q_0 \leftarrow \hat{\mathcal{R}}$
for iter **in** n_iter **do**
    FitModel($\tilde{\mathcal{D}^+}$)
    $Q_{iter+1} \leftarrow$ ComputeValueFunction(Q_{iter})
end for
return Policy π^*

Chapter 5

Empirical Evaluation

In this chapter, we present the experiments in order to evaluate and compare the performance of AWFQI_oracle and AWFQI (real setting with when we do not have source MDPs).

Among the transfer of samples algorithms discussed in 3.5.1, we compare the performance of our proposed AWFQI algorithm to RBT, SDT and IWFQI. To be fair, in these algorithms, we set the policy that generates episodes at the beginning of the algorithms to uniform policy, but we set $\epsilon-$greedy for the updates and evaluation of the policy in the iterations. Moreover, in AWFQI, the budget = batch size $\times$ horizon in all the tests. As for the score, we considered only the variance of the rewards (column 4).

5.1 Experimental Settings

According to the score functions used in oracle and real AWFQI (introduced in Sections 4.2.1 and 4.2.2 respectively), three main settings seem logical to be considered, which are presented in Sections 5.1.2, 5.1.3 and 5.1.4. We tested the algorithms mentioned above with all these settings in the Puddle World environment. The environment setting is provided in Section 5.1.1. In the algorithm, max iteration for updating policy is 60 and for AWFQI algorithm score function implementation we set $(dx, dy) = (0.1, 0.1)$ when applicable.

For each of these settings, we ran the algorithms using two sub-settings. The first one is very similar to the setting provided in [82], in which the number of steps is 6, and in each step, we have 10 episodes (batch-size). In AWFQI the budget in each step is 500 (horizon $\times$ batch size $=$ 500). We ran the algorithms in this setting for 3 independent runs. In the second sub-setting, the number of episodes (batch size) is 1 at each step and the budget for AWFQI in each step is 50 (horizon $\times$ batch size $=$ 50). Since for other algorithms in each step, the policy is updated by adding 50 samples, which is the number of horizon, AWFQI should transfer 50 samples at each step too. We ran the algorithms in this setting for 10 independent runs.

We provided two plots[1]. The first one is the performance of the policy updates in algorithms, through the steps. The next plot shows the collected points on target MDP in all steps by algorithms along with the scatter of source MDPs states on the environment.

5.1.1 Environment

As discussed, the experiments carried out in Section 4.1.5 to evaluate score function criteria are based on Grid World environment. To test the performance of the proposed algorithm we set up our experiment using the Puddle World environment. In the following we detail the characteristics of this environment.

Puddle World was proposed by Sutton *et. al.* [69]. It is a repeated rectangular two-dimensional continuous grid with a goal area that is unknown to the agent and some elliptical puddles. The agent's goal is to reach the goal state while avoiding the puddle areas.

In this work, we employed a modified version of the Puddle World as described in [82]. Here, the Puddle World is a discrete-action, continuous-state (stochastic) problem, with the state-space in the range $[0, 10]^2$. The action-space is discrete and allows the agent to move in the four cardinal directions. At each time-step, the agent receives a reward of -1. A penalization proportional to the distance from all puddles (presented in Equation 5.1) can be added to the reward.

[1] IWFQI algorithm is denoted by WFQI in the legend of the plots.

$$R(s,a) = -1 - 100 \sum_{u \in \mathcal{U}} W_u(s) \tag{5.1}$$

Where $\mathcal{U}$ is the set of puddles and $W_u(s)$ is the weight of puddle u for state s, modeled as a bi-variate Gaussian. Each action moves the agent by α (according to the formula presented in Equation 5.2) in the corresponding direction [82].

$$\alpha = (1 + 5 \sum_{u \in \mathcal{U}} W_u(s'))^{-1} \tag{5.2}$$

The agent takes one of the four actions (*i.e. Right*, *Left*, *Up*, *Down*) at each time step which changes its position by α according to the selected direction. Tirinzoni *et. al* [82] designed two variations of the environment. In the first version *i.e. shared dynamics*, $\alpha = 1$ is fixed. In the other version *i.e. puddle-based dynamics*, α slows down the agent according to the distance from all puddles. In this work, we utilized the *shared dynamics* version of the Puddle World. The sources and target Puddle Worlds utilized for this work are rendered in Figures 5.1 and 5.2.

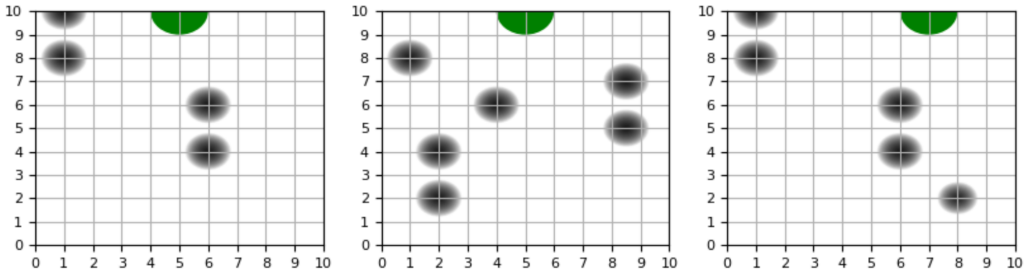

Figure 5.1: Source MDPs

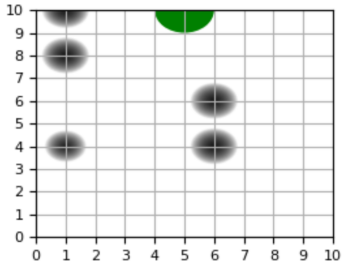

Figure 5.2: Target MDP

In the Puddle World setting, we set horizon = 50. Other general characteristics of the MDPs are reported in Table 5.1.

	Start	Goal	# Puddles	Puddle Mean
Source MDP1	(0,0)	(5,10)	4	(1.0,10.0) (1.0,8.0) (6.0,6.0) (6.0,4.0)
Source MDP1	(0,0)	(5,10)	6	(2.0,2.0) (4.0,6.0) (1.0,8.0) (2.0,4.0) (8.5,7.0) (8.5,5.0)
Source MDP3	(0,0)	(7,10)	5	(8.0,2.0) (1.0,10.0) (1.0,8.0) (6.0,6.0) (6.0,4.0)
Target MDP	(0,0)	(5,10)	5	(1.0,4.0) (1.0,10.0) (1.0,8.0) (6.0,6.0) (6.0,4.0)

Table 5.1: MDPs general characteristics.

5.1.2 Setting 1: Source Data Mode

In this setting, both AWFQI and AWFQI_oracle can potentially transfer only the (s, a) pairs presented in the source data files as described in Section 4.2. In this sense, the score function here is not considered continuous. In other words, the probability that the score function selects (s, a) pairs from the source MDPs samples is proportional to the variance of the rewards of that particular state-action among the source MDPs is the highest.

5.1.3 Setting 2: Grid Mode

In this setting, both AWFQI and AWFQI_oracle transfer samples from a set of (s, a) pairs that are on a user-defined grid. The score function is continuous as we are using grids, so we can make small enough dx and dy in the functions such that we can draw samples from continuous function. These two modes are explained in Section 4.2.

5.1.4 Setting 3: Source Data and Grid Mode (Both)

In this setting, the AWFQI and AWFQI_oracle can potentially collect/transfer (s, a) pairs which are either on a user-defined grid or from the data from the source MDPs. So it is considered continuous 4.2.

5.2 Experimental Results

In this section, we provide the results that we obtained by running the mentioned algorithms and AWFQI-oracle and AWFQI with the settings we explained in the previous section. Note that we have two sub-settings for each setting with minor changes in the number of episodes at each step (see Section 5.1 for detailed explanations).

Our motivation for designing these sub-settings is comparing the performance of algorithms when the number of transferred samples are relatively few versus when this number is high. We can also obtain an insight into how AWFQI is robust to the different numbers of budget, specifically we would like to observe the usefulness of the algorithm when the number of budget is low. We can acknowledge the efficiency of using variance-based score function in AWFQI to be consistent with Figure 4.2. Also, in that figure, we noticed that variance-based performs better than the other algorithms when the number of budget is low.

5.2.1 Setting 1: Source Data Mode

Number of episodes (n_episode = 10)

The following are the results of the experiment when n_episode = 10.

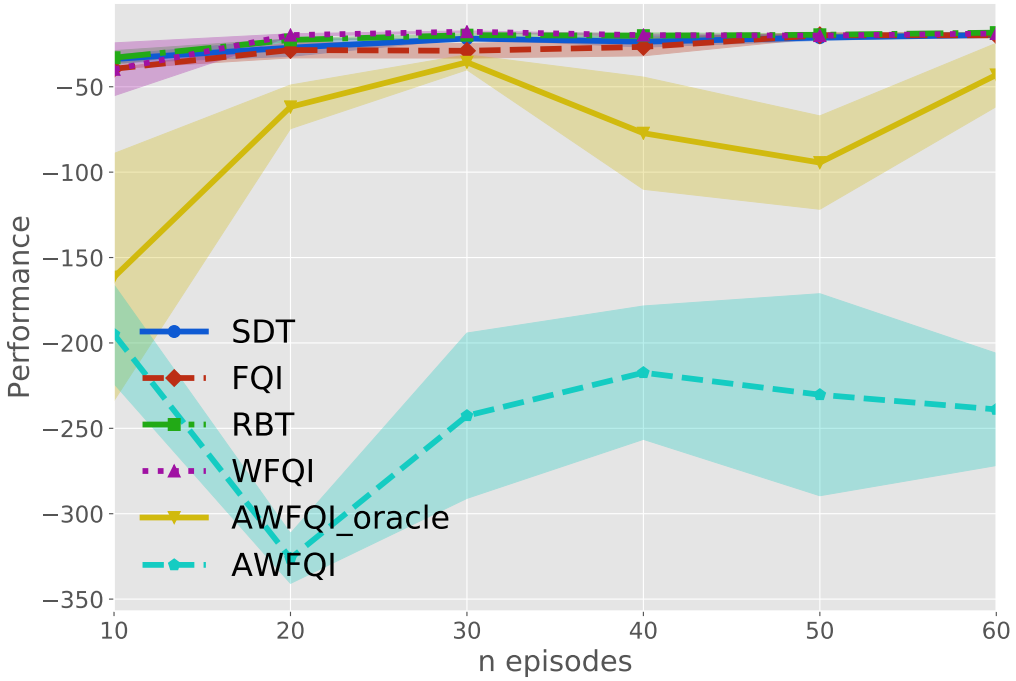

Figure 5.3: The algorithms performance. In this setting, the score function restricts both AWFQI_oracle and AWFQI to collect only the samples that their (s, a) exist in source data. Samples are collected using scores in Figure 4.5 in AWFQI_oracle and Figure 4.8 in normal AWFQI.

Neither of AWFQI_oracle and AWFQI are efficient in this setting because we have forced them to select among only the limited number of (s, a) *i.e.* the state-action pairs of the source MDPs. This leads AWFQI to collect the same points from target MDP in different steps. Particularly, since the source MDP data points are limited, AWFQI tends to select redundant samples when episodes are growing. So we are actually updating the policy using the same samples we have already had in previous steps.

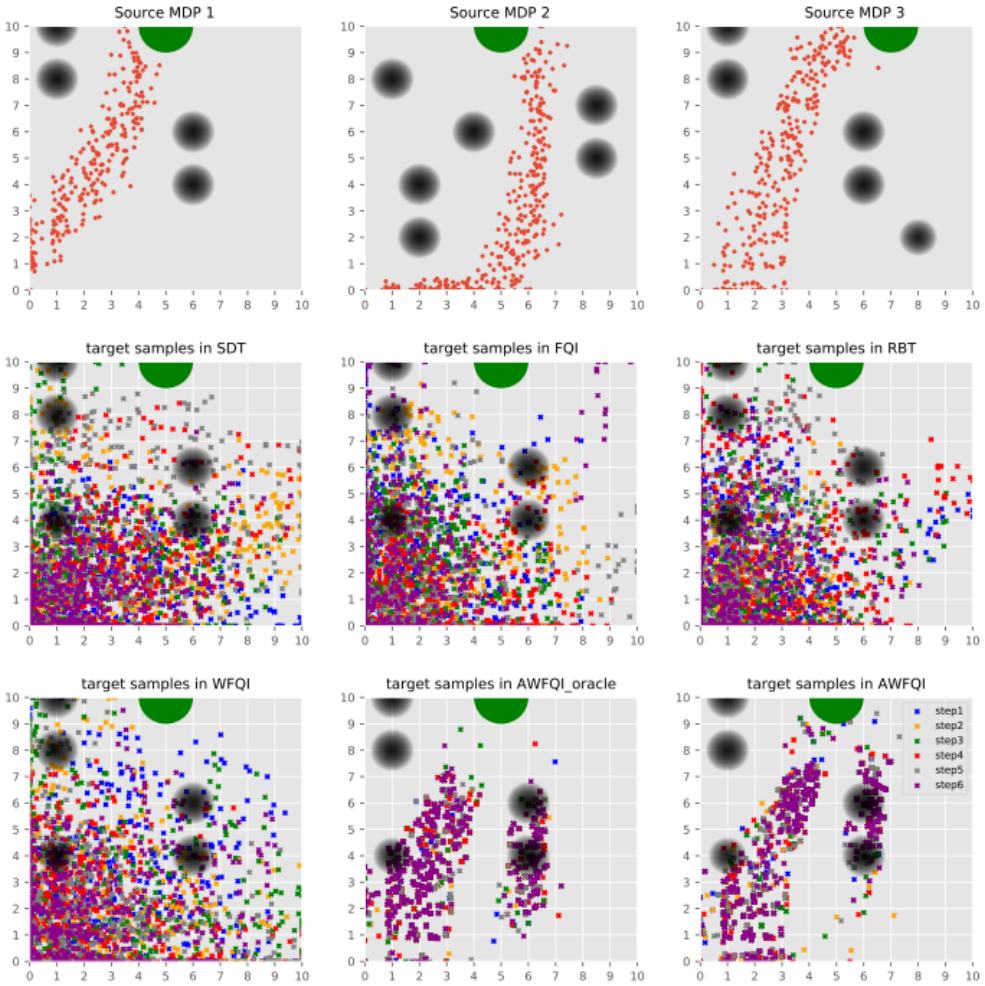

Figure 5.4: All algorithms collected (s, a) on target MDP through steps. In this setting, the score function restricts both AWFQI_oracle and AWFQI to collect only the samples that their state-action pair exist in source data files. Samples are collected using score function in Figure 4.5 in AWFQI_oracle and Figure 4.8 in AWFQI.

As depicted in Figure 5.4, we can see that both AWFQI and AWFQI_oracle are restricted to the samples to the regions which were part of the trajectories collected from the source MDPs which resulted in a bias for those regions and restrict these algorithms to collect samples from other regions which could potentially resulted in better results. It could be a potential reason why these algorithms perform worse than the others.

Number of episodes (n_episode $= 1$)

In this section in all the tests the number of episodes is 1.

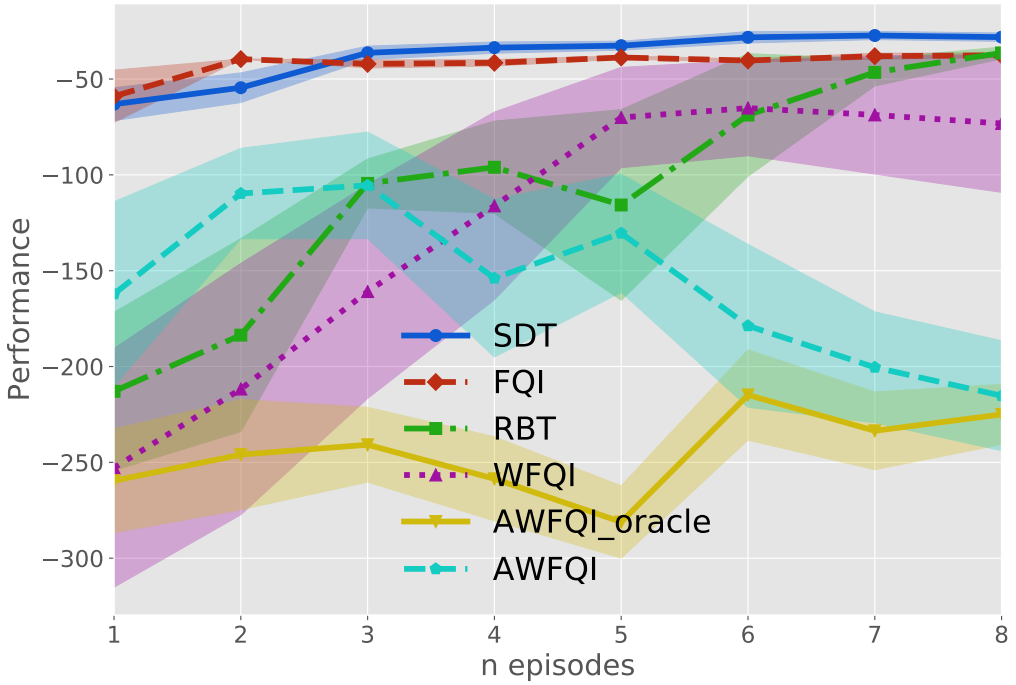

Figure 5.5: The algorithms performance. In this setting, the score function restricts both AWFQI_oracle and AWFQI to collect only the samples that their (s, a) exist in source data. Samples are collected using scores in Figure 4.5 in AWFQI_oracle and Figure 4.8 in normal AWFQI.

Both AWFQI_oracle and AWFQI are not performing satisfactorily here as well. In this setting, the other algorithms were, in general, worse than the previous sub-setting because the number of samples is less here. Compared to AWFQI algorithms, they outperform because they have the opportunity to collect samples randomly (using u niform policy), and they are not restricted to pre-defined (s, a). Besides, here we can see that AWFQI_oracle and IWFQI performance deteriorates compared to before, which could be another consequence of few numbers of samples size (budget).

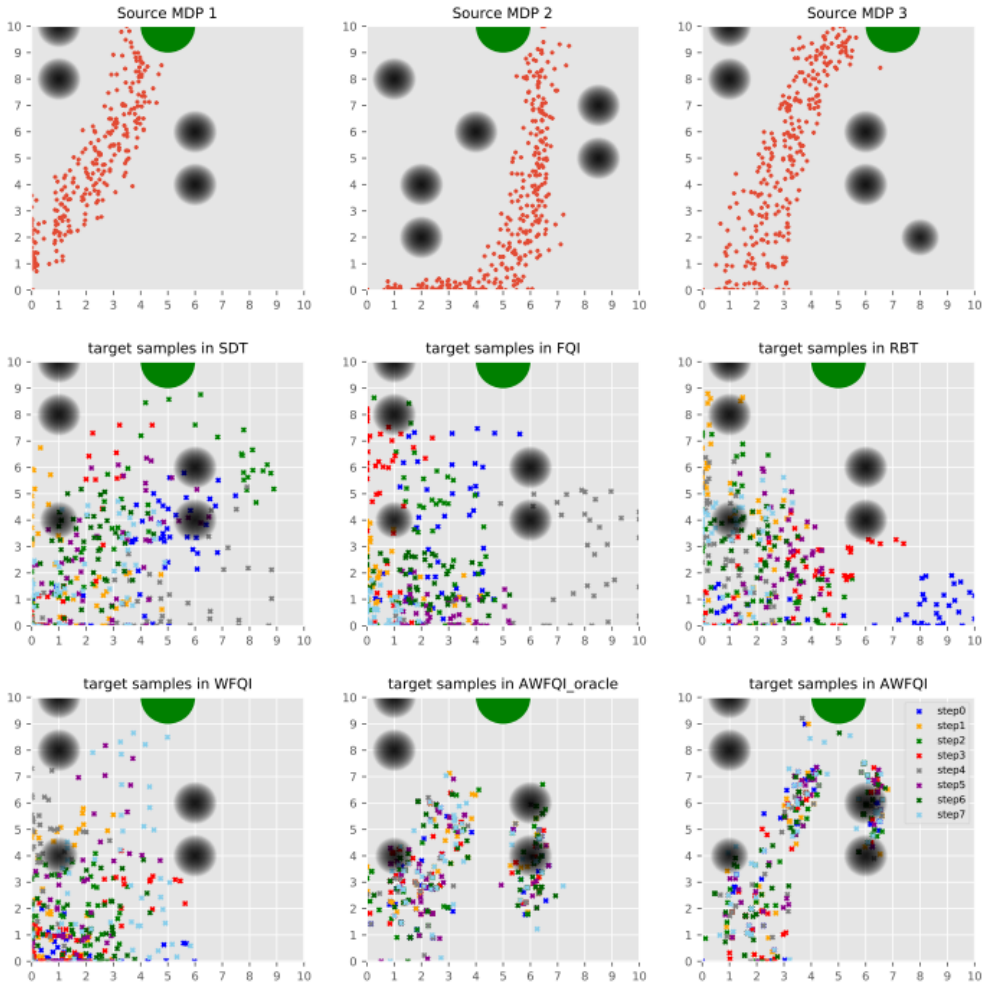

Figure 5.6: All algorithms collected (s, a) on target MDP through steps. In this setting, the score function restricts both AWFQI_oracle and AWFQI to collect only the samples that their state-action exist in source data files. Samples are collected using score function in Figure 4.5 in AWFQI_oracle and Figure 4.8 in AWFQI.

This figure shows the state-action pairs collected through the runs. Due to the limitation of the number of samples in the source MDPs, we notice that the data points collected in AWFQI_oracle and AWFQI in Figure 5.4 and 5.6 are comparable.

5.2.2 Setting 2: Grid Mode

Number of episodes (n_episode = 10)

In this section, in all the tests, the number of episodes is 10.

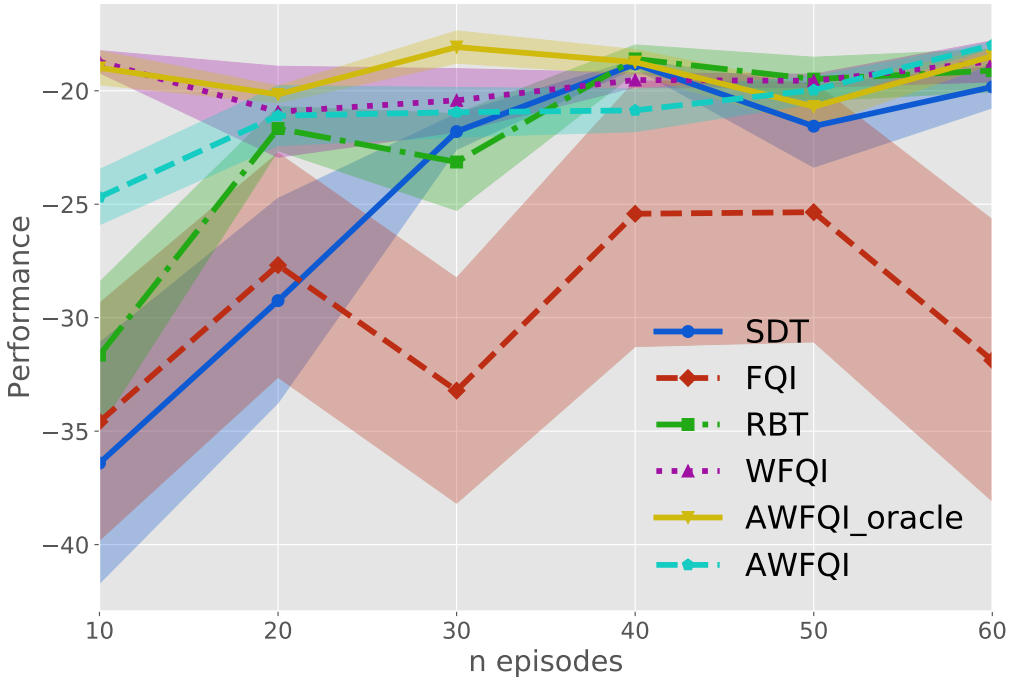

Figure 5.7: The algorithms performance. In this setting, the score function/table contains only the (s, a) in grid, and both AWFQI_oracle and AWFQI collect the samples that their (s, a) exist in the grid. Samples are collected using scores in Figure 4.5 in AWFQI_oracle and Figure 4.8 in AWFQI.

In this setting, both AWFQI_oracle and AWFQI perform better than other algorithms. They have a good jumpstart compared to other algorithms, which shows their satisfactory performance with even very few samples. Also, the difference between AWFQI_oracle and AWFQI seems very little, which suggests the predicted scores were consistent with the true scores.

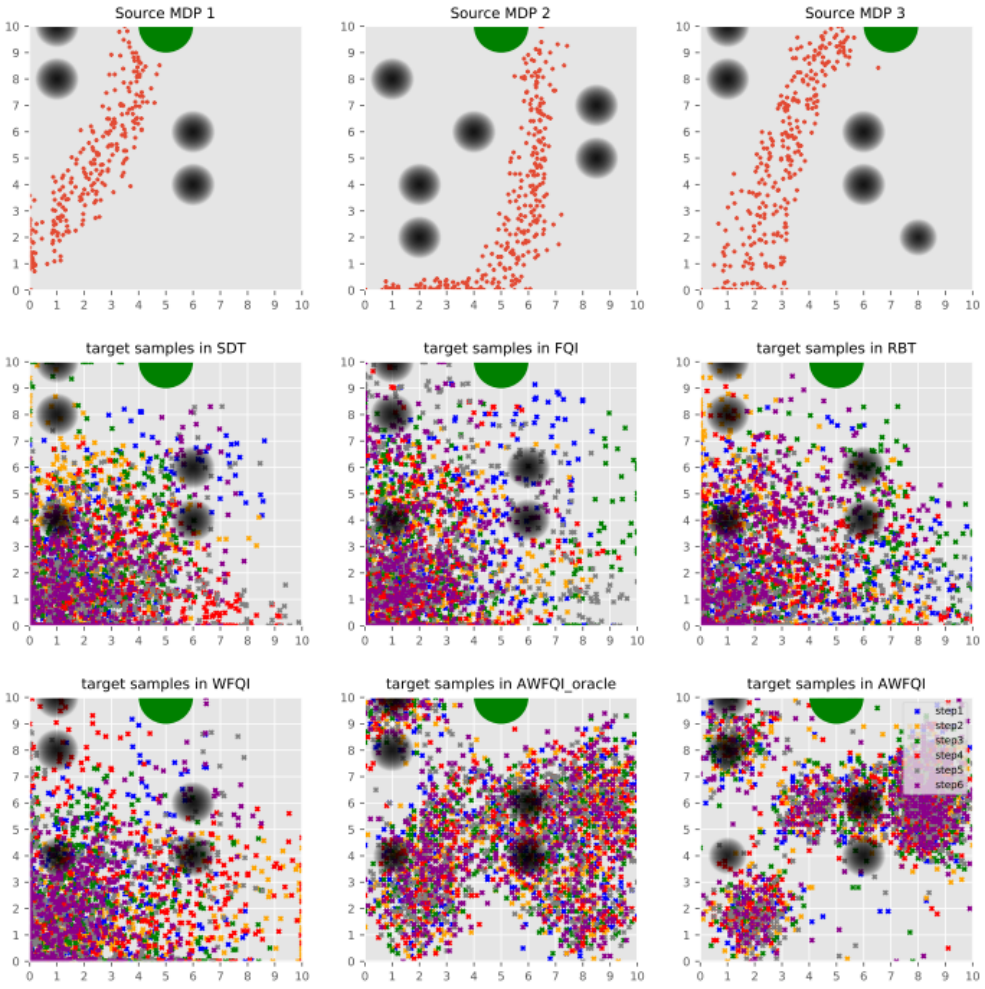

Figure 5.8: All algorithms collected (s, a) on target MDP through steps. In this setting, the score function/table contains only the (s, a) in grid, and both AWFQI_oracle and AWFQI collect the samples that their (s, a) exist in the grid. Samples are collected using scores in Figure 4.5 in AWFQI_oracle and Figure 4.8 in AWFQI.

The sampled state-action pairs in this figure show that both AWFQI_oracle and AWFQI transfer samples from the regions near the goal state. As depicted in Figure 5.8, AWFQI tends not to sample from the bottom-middle part of the grid, where indeed the MDPs are the same. Also, the similarity between the asymptotic behavior of performance of AWFQI_oracle and AWFQI algorithm and the samples they collected show that GPs is working quite well in predicting the variance of rewards in the score function.

Number of episodes (n_episode = 1)

In this section in all the tests the number of episodes is 1.

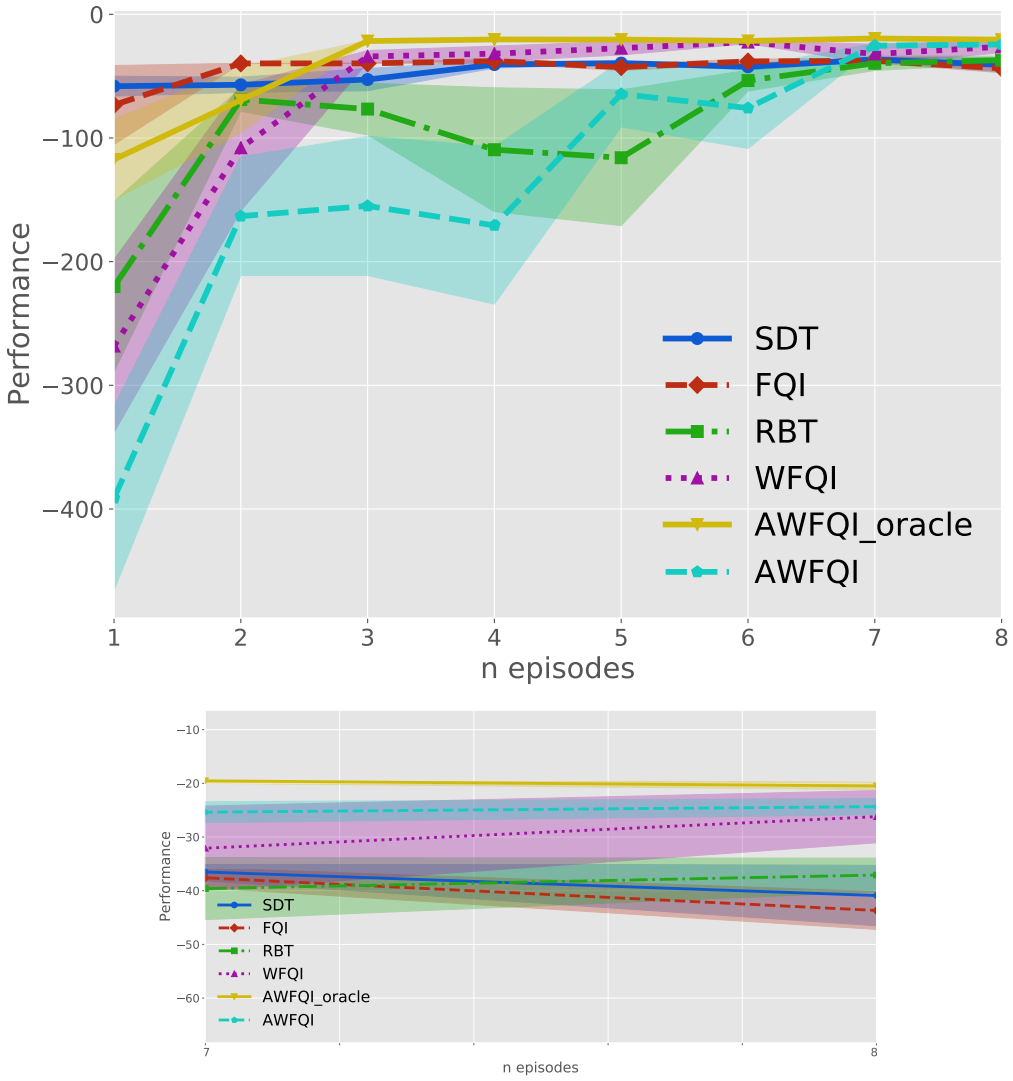

Figure 5.9: The algorithms performance. In this setting, the score function/table contains only the (s, a) in grid, and both AWFQI_oracle and AWFQI collect the samples that their (s, a) exist in the grid. Samples are collected using scores in Figure 4.4 in AWFQI and Figure 4.7 in AWFQI. The sub-plot on the bottom provides a closer view on the asymptotic behaviors of the algorithms.

In this setting, AWFQI's starting point is not as good as the previous setting

(especially we have ups and downs in the plots), but it converges to the same point (as shown in the sub-plot on the bottom of Figure 5.9) with even much less number of budget, since we have only one episode in each step and algorithms add 50 samples for updating their policy. In general, they outperform other algorithms which confirms that the active version with a generative model is indeed more sample efficient.

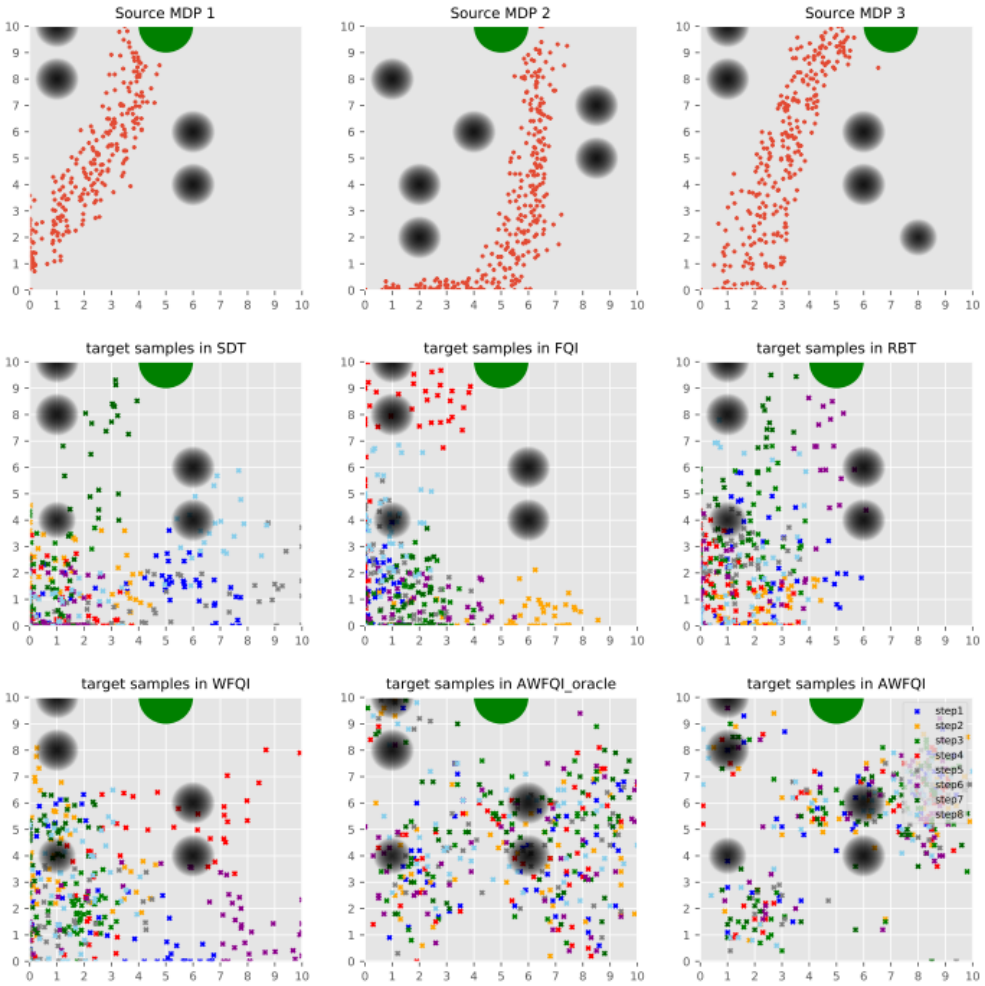

Figure 5.10: All algorithms collected (s, a) on target MDP through steps. In this setting, the score function/table contains only the (s, a) in grid, and both AWFQI_oracle and AWFQI collect the samples that their (s, a) exist in the grid. Samples are collected using scores in Figure 4.4 in AWFQI_oracle and Figure 4.7 in AWFQI_oracle.

This figure confirms the results we obtained in Figure 5.9, and we can see that the points that are collected in steps in AWFQI_oracle and AWFQI are almost in the same regions.

5.2.3 Setting 3: Source Data and Grid Mode (Both)

Number of episodes (n_episode = 10)

In this section, in all the tests, the number of episodes is 10.

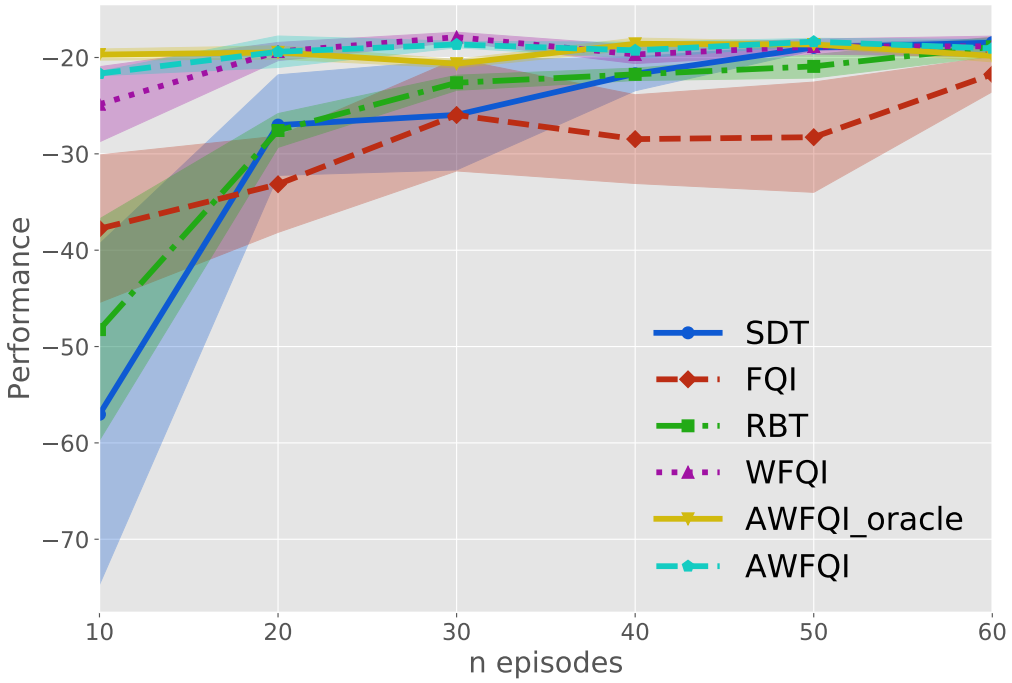

Figure 5.11: The algorithms performance. In this setting, the AWFQI_oracle score function/tables contains (s, a) from grid and source data. AWFQI contains grid state-action pairs and the pairs from source data files. Both types of (s, a) are predicted from source data files. Samples are collected using scores in figure 4.6 in AWFQI_oracle and figure 4.9 in AWFQI.

Here, the figures show good performance for both algorithms. The reason could be we keep both state-action pairs from the source MDPs and the ones from the grid; consequently, the score function can select each of those samples and does not have the limitation as setting 1. Besides, the function is continuous, as we can potentially select very small dx and dy. AWFQI

performs slightly better than AWFQI_oracle version. They have a very higher jumpstart rather than other algorithms and converge to a higher value.

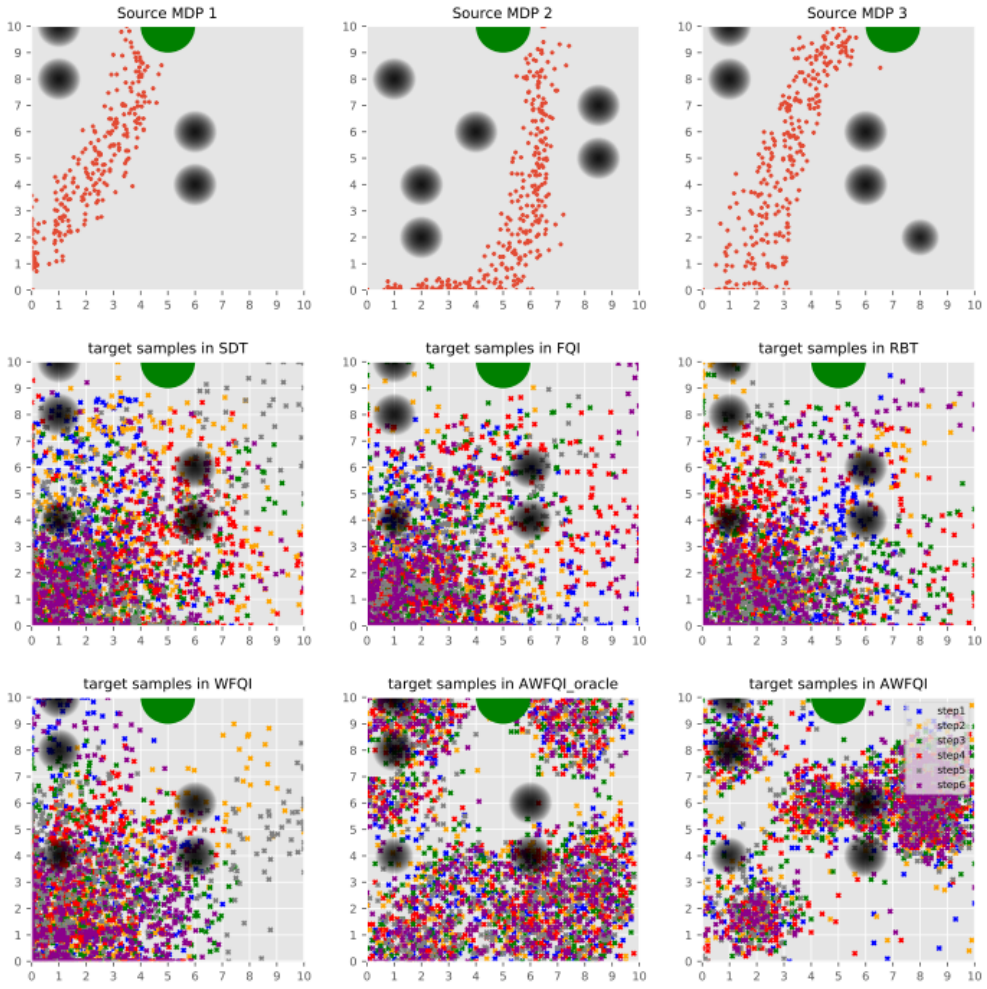

Figure 5.12: All algorithms collected (s, a) on target MDP through steps. In this setting, the AWFQI_oracle score function/tables contains state-action pairs from both grid and source data. AWFQI contains grid state-action pairs and the ones from source data files. Both types of (s, a) are predicted from source data files. Samples are collected using scores in figure 4.6 in AWFQI_oracle and figure 4.9 in AWFQI.

Although the spread of sampling in AWFQI_oracle and AWFQI are different suggested by this figure, the overall performances of these algorithms are higher than the others. It could be because they have collected samples from different regions of the environment.

Number of episodes (n_episode = 1)

In this section in all the tests the number of episodes is 1.

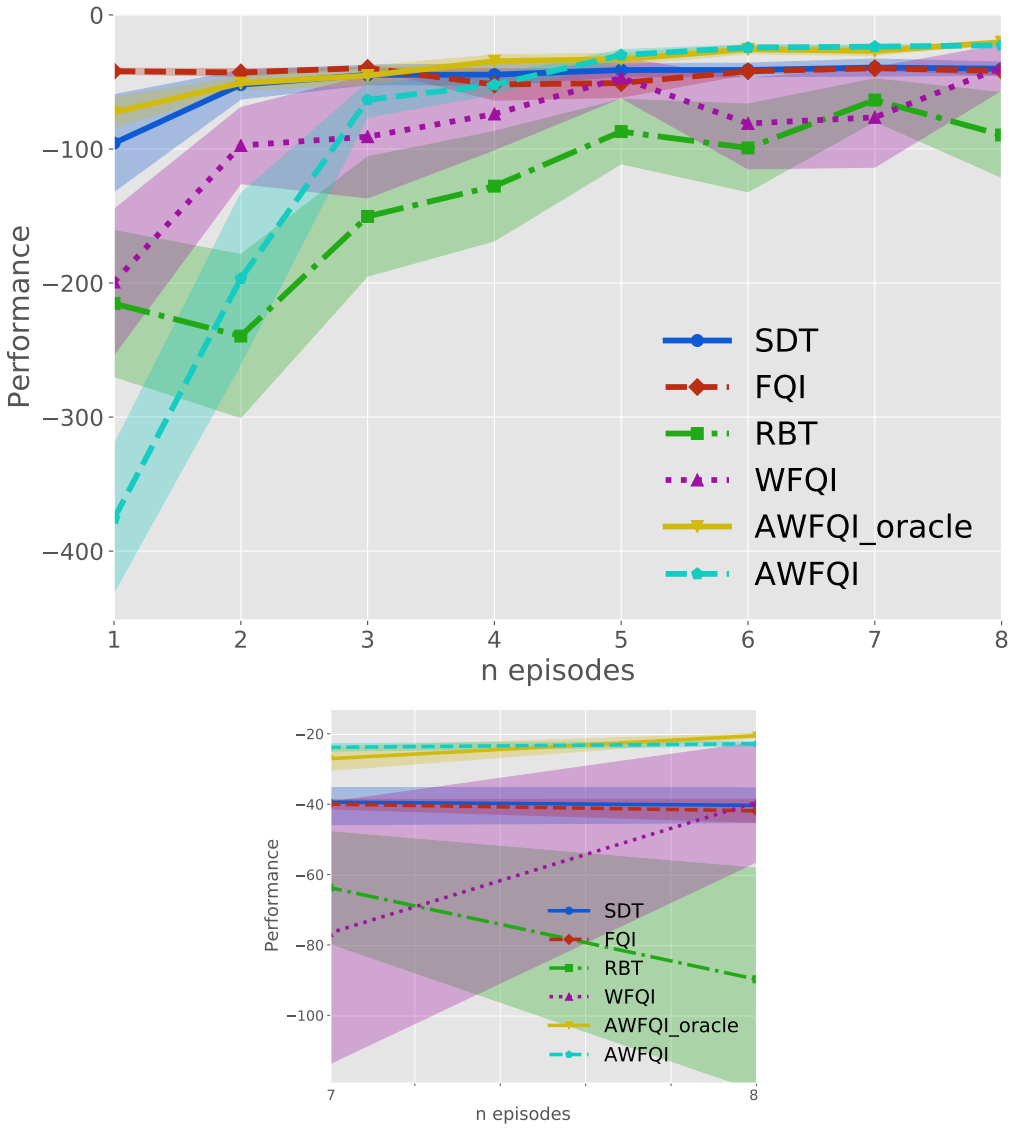

Figure 5.13: The algorithms performance. In this setting, the AWFQI_oracle score function/tables contains (s, a) from grid and source data. AWFQI contains grid state-action pairs and the pairs from source data files. Both types of (s, a) are predicted from source data files. Samples are collected using scores in figure 4.6 in AWFQI_oracle and figure 4.9 in AWFQI. The sub-plot on the bottom provides a closer view on the asymptotic behaviors of the algorithms.

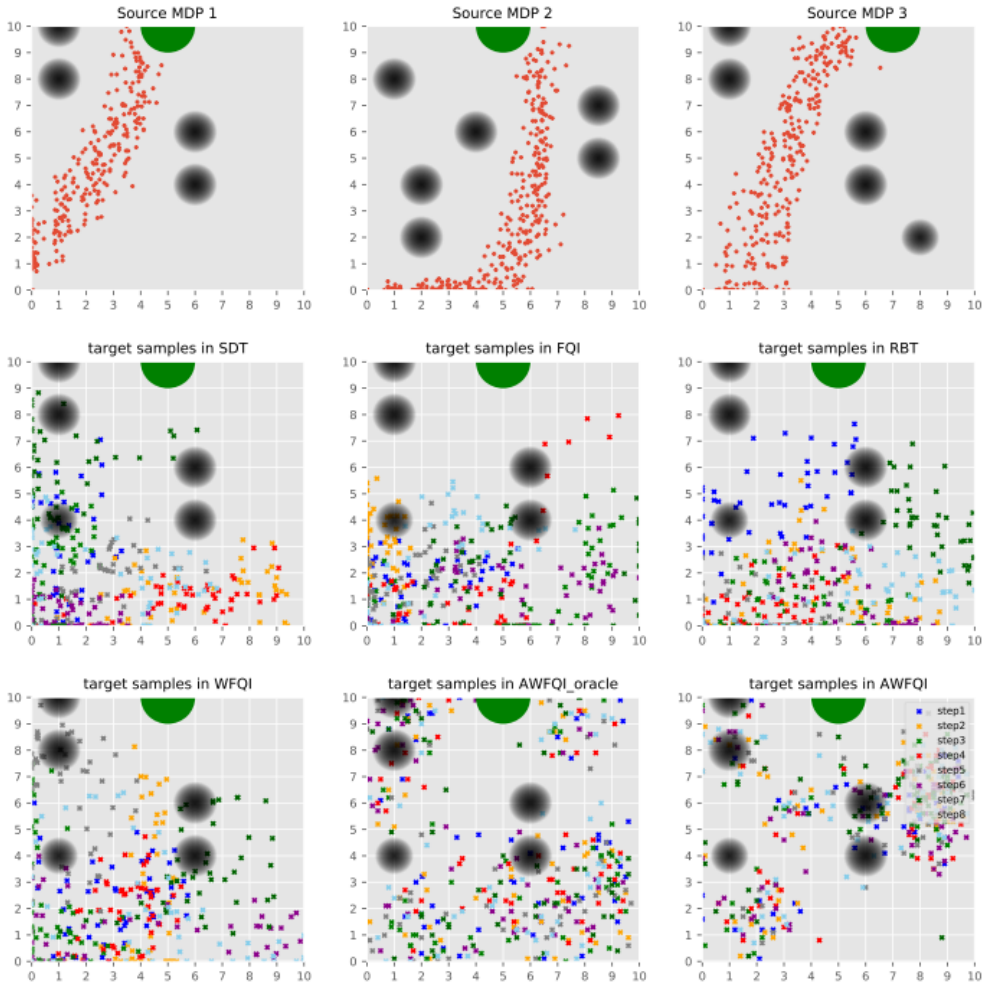

Figure 5.14: All algorithms collected (s, a) on target MDP through steps. In this setting, the AWFQI_oracle score function/tables contains state-action pairs from both grid and source data. AWFQI contains grid state-action pairs and the ones from source data files. Both types of (s, a) are predicted from source data files. Samples are collected using scores in figure 4.6 in AWFQI_oracle and figure 4.9 in AWFQI.

In this setting, even when the number of collected samples is very few, both AWFQI_oracle and AWFQI work well and converge to the same value as before relatively fast. In terms of convergence point, AWFQI converges in the third step, while AWFQI_oracle version converges sooner in step two. The algorithms have worse jumpstart than the previous setting due to fewer budget for transferring the samples but as clearly can be seen in the sub-

plot on the bottom of Figure 5.13 the asymptotic behaviors of the active algorithm outperforms the non-active algorithms.

As presented in Figure 5.14, the collected points in AWFQI are in the optimal trajectory, while AWFQI_oracle does not collect samples from all the path on the trajectory but still performs very well.

All in all, through out the experiments, we could show that the proposed active transfer of samples with a generative model is indeed more sample efficient.

Chapter 6

Conclusions & Future Work

In this work, we proposed an active transfer learning algorithm as a continuation of previous research conducted by Tirinzoni *et. al* IWFQI in TL. We designed a scoring function that scores the samples collected from the source tasks by capturing the dissimilarities among them. In this regard, we implemented several score functions based on Uniform distribution (random), Mutual Information MI and variance among the source tasks and validated the usefulness of these approaches by an experiment which showed the variance-based score function defeated other approaches in terms of accuracy of predicting the target task.

Then, we predict the conditional variance among the source MDPs given the already collected samples by applying GPs regression method. An advantage of using GPs is its potential for estimating the variance in environments such as Puddle World since the rewards surface follows Gaussian distribution. After that, we actively used this function through the runs of the algorithm (AWFQI) for updating the policy and solving the target MDP task. This function can be applied in discrete and continuous domains and does not consider any prior assumption about the distribution of the source or target MDPs. We compared the performance of our method with other state-of-the-art algorithms on transferring the samples such as RBT, SDT, IWFQI, and a non-transfer algorithm (FQI). We showed that our method outperforms these algorithms in many situations, especially when the budget for transferring is limited. We further implemented an oracle version of the AWFQI algorithm

to show the extent to which our method is affected by estimating the source environment in unknown regions. Moreover, we empirically examined the proposed active transfer of samples with access to the generative model, which shows the proposed algorithm's sample efficiency. We noticed that AWFQI performs almost the same as the oracle version.

Nevertheless, as future work, it would be interesting to utilize other regressors or ML models for better estimation of the dissimilarity among the source MDPs. Apart from that, one could investigate more forms of scoring criteria in the score function construction as we tested only three approaches. Besides, it might be interesting to combine multiple distinct measures instead of utilizing only one criterion to devise an ensemble method for calculating the state-action pair's scores. In addition, in some of the observations in the results, we noticed that sometimes the estimated scores work better than the oracle version. It raises the thought to incorporate exploration factor supplementary to the used score function.

Bibliography

[1] David Abel, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael Littman. Policy and value transfer in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 20–29, 2018.

[2] Oguzhan Alagoz, Heather Hsu, Andrew J Schaefer, and Mark S Roberts. Markov decision processes: a tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483, 2010.

[3] Mauricio A Alvarez, Lorenzo Rosasco, and Neil D Lawrence. Kernels for vector-valued functions: A review. *arXiv preprint arXiv:1106.6251*, 2011.

[4] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15:2773–2832, 2014.

[5] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Žídek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *arXiv preprint arXiv:1901.10964*, 2019.

[6] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065, 2017.

[7] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[8] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[9] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[10] Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*. Princeton university press, 2015.

[11] Gareth Bendall and François Margot. Greedy-type resistance of combinatorial problems. *Discrete Optimization*, 3(4):288–298, 2006.

[12] Darrin C Bentivegna, Christopher G Atkeson, and Gordon Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, 2004.

[13] Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

[14] Rich Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pages 657–664, 1995.

[15] Dongkyu Choi, Tolgo Konik, Negin Nejati, Chunki Park, and Pat Langley. Structural transfer of cognitive skills. In *Proceedings of the eighth international conference on cognitive modeling*, pages 115–120. Citeseer, 2007.

[16] Rasmussen CE& Williams CKI. Gaussian processes for machine learning, 2006.

[17] Koby Crammer, Michael Kearns, and Jennifer Wortman. Learning from multiple sources. *Journal of Machine Learning Research*, 9(Aug):1757–1774, 2008.

[18] Robert H Crites and Andrew G Barto. Improving elevator performance using reinforcement learning. In *Advances in neural information processing systems*, pages 1017–1023, 1996.

[19] Finale Doshi-Velez and George Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering

latent task parametrizations. In *IJCAI: proceedings of the conference*, volume 2016, page 1432. NIH Public Access, 2016.

[20] Anders Eriksson, Genci Capi, and Kenji Doya. Evolution of meta-parameters in reinforcement learning algorithm. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 1, pages 412–417. IEEE, 2003.

[21] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Iteratively extending time horizon reinforcement learning. In *European Conference on Machine Learning*, pages 96–107. Springer, 2003.

[22] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[23] Alan Fern, SungWook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In *Advances in neural information processing systems*, pages 847–854, 2004.

[24] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[25] Jochen Garcke and Thomas Vanck. Importance weighted inductive transfer learning for regression. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 466–481. Springer, 2014.

[26] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[27] Thomas L Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12(4), 2011.

[28] J.M. Hammersley and D.C. Handscomb. *Monte Carlo Methods*. Methuen's monographs on applied probability and statistics. Methuen, 1964.

[29] Ronald A Howard. Dynamic programming and markov processes. 1960.

[30] Okhtay Ilghami, Hector Munoz-Avila, Dana S Nau, and David W Aha. Learning approximate preconditions for methods in hierarchical plans. In *Proceedings of the 22nd international conference on Machine learning*, pages 337–344, 2005.

[31] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[32] Taylor W Killian, Samuel Daulton, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in neural information processing systems*, pages 6250–6261, 2017.

[33] Andrey Kolobov. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012.

[34] J Zico Kolter, Pieter Abbeel, and Andrew Y Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Advances in Neural Information Processing Systems*, pages 769–776, 2008.

[35] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[36] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[37] John E Laird, Paul S Rosenbloom, and Allen Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine learning*, 1(1):11–46, 1986.

[38] Gerhard Lakemeyer, Elizabeth Sklar, Domenico G Sorrenti, and Tomoichi Takahashi. *RoboCup 2006: Robot Soccer World Cup X*, volume 4434. Springer, 2007.

[39] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[40] Romain Laroche and Merwan Barlier. Transfer reinforcement learning with shared dynamics. In *Proceedings of the Thirty-First AAAI Confer-*

ence on Artificial Intelligence, AAAI'17, page 2147–2153. AAAI Press, 2017.

[41] Tor Lattimore and Rémi Munos. Bounded regret for finite-armed structured bandits. In *Advances in Neural Information Processing Systems*, pages 550–558, 2014.

[42] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

[43] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 544–551, New York, NY, USA, 2008. Association for Computing Machinery.

[44] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[45] H. Liang, W. Fu, and F. Yi. A survey of recent advances in transfer learning. In *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, pages 1516–1523, 2019.

[46] Weifeng Liu, Jose C Principe, and Simon Haykin. *Kernel adaptive filtering: a comprehensive introduction*, volume 57. John Wiley & Sons, 2011.

[47] Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. volume 1, 01 2006.

[48] Patrick Mannion, Jim Duggan, and Enda Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In *Autonomic road transport support systems*, pages 47–66. Springer, 2016.

[49] Yishay Mansour and Satinder Singh. On the complexity of policy iteration. *arXiv preprint arXiv:1301.6718*, 2013.

[50] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine learning*, 3(4):319–342, 1989.

[51] Emre O Neftci and Bruno B Averbeck. Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, 1(3):133–143, 2019.

[52] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer, 2006.

[53] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[54] Art B Owen. Monte carlo theory, methods and examples. *Monte Carlo Theory, Methods and Examples. Art Owen*, 2013.

[55] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[56] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[57] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[58] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[59] Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming, 1994.

[60] Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[61] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *European Conference on Machine Learning*, pages 699–707. Springer, 2007.

[62] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[63] Burrhus Frederic Skinner. *Science and human behavior*. Number 92904. Simon and Schuster, 1965.

[64] Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[65] Robert F Stengel. *Optimal control and estimation*. Courier Corporation, 1994.

[66] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[67] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(11), 2009.

[68] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density ratio estimation in machine learning*. Cambridge University Press, 2012.

[69] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[70] Richard S Sutton and Andrew G Barto. Reinforcement learning: an introduction cambridge. *MA: MIT Press.[Google Scholar]*, 1998.

[71] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[72] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[73] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[74] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Joint European*

conference on machine learning and knowledge discovery in databases, pages 488–505. Springer, 2008.

[75] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[76] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[77] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646, 1996.

[78] Andrea Tirinzoni, Alessandro Lazaric, and Marcello Restelli. A novel confidence-based algorithm for structured bandits. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3175–3185, Online, 26–28 Aug 2020. PMLR.

[79] Andrea Tirinzoni, Riccardo Poiani, and Marcello Restelli. Sequential transfer in reinforcement learning with a generative model. In *International Conference on Machine Learning*, 2020.

[80] Andrea Tirinzoni, Mattia Salvini, and Marcello Restelli. Transfer of samples in policy search via multiple importance sampling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6264–6274, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[81] Andrea Tirinzoni, Rafael Rodriguez Sanchez, and Marcello Restelli. Transfer of value functions via variational methods. In *Advances in Neural Information Processing Systems*, pages 6179–6189, 2018.

[82] Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. Importance weighted transfer of samples in reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings*

of Machine Learning Research, pages 4936–4945, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[83] Hado Van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement learning*, pages 207–251. Springer, 2012.

[84] Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428, 1995.

[85] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[86] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

[87] William J Welch, Robert J Buck, Jerome Sacks, Henry P Wynn, Toby J Mitchell, and Max D Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[88] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[89] Robert S Woodworth and EL Thorndike. The influence of improvement in one mental function upon the efficiency of other functions.(i). *Psychological review*, 8(3):247, 1901.

[90] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[91] Li Zhou. A survey on contextual multi-armed bandits. *arXiv preprint arXiv:1508.03326*, 2015.