School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

Planning to Explore in Reinforcement Learning

Jared William Swift

Submitted in accordance with the requirements for the degree of
BSc Computer Science with Artificial Intelligence (Ind)

2022/23

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (01/05/2023)
Link to online code repository	URL	Sent to supervisor and assessor (01/05/2023)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) JARED WILLIAM SWIFT

Summary

Efficient exploration is imperative to success in Reinforcement Learning. However, random approaches, which are not inefficient, are often favoured in practice due to their ease of implementation and robustness. Model-based approaches to exploration are typically more sample efficient, but much of the current approaches are either very optimistic or computationally expensive, and rely on learning a model entirely from scratch and assume that it will eventually become correct.

Within this work we explored a model-based approach to exploration that leverages an initial model, uses a *reasonable* amount of optimism, alongside intrinsic motivation, to explore and learn whilst not assuming that the model will eventually become correct. Exploration was driven by enabling the planner to hypothesise where the model may be incorrect, through additional actions denoted *Meta Actions* and then realising these hypotheses through experience.

Various implementations were developed based on this idea, and were evaluated in a collection of gridworld-like domains, and our method of exploration was shown to perform better than the most ubiquitous random method, ϵ-greedy, in most cases; moreover, the agents with *Meta Actions* available to them often outperformed those without them, showing that they can be a useful exploration mechanism.

Acknowledgements

I'd like to thank my supervisors, Dr. Mehmet Dogar and Dr. Matteo Leonetti, for their constant support and insight throughout this project; I learned so much during our conversations. I'd like to thank Dr. Mehmet Dogar for supervising this project, when he was not due to have any project students this year. I'd like to thank Dr. Matteo Leonetti once more, for supporting and mentoring me throughout almost my entire undergraduate degree, and providing me with many opportunities to program service robots, where I found my passion and research interest. I'd like to thank my assessor, Dr. Sebastian Ordyniak for his valuable intermediate feedback. I'd like to thank my girlfriend, for supporting me throughout my undergraduate degree, and listening to my ideas. I'd like to thank my parents and family for everything; I wouldn't be where I am today without them.

Contents

Chapter 1

Introduction

Reinforcement Learning (RL) [53] is based on the idea of learning through experience; it is trial-and-error learning. A decision-making agent learns how to behave in an environment by interacting with it and receiving positive and negative reinforcement. Experience can only be gained by choosing to try new things and observing the consequences; this is known as exploration. However it's often infeasible to try everything, especially in tasks of practical interest, therefore exploration needs to be done in such a way that it gains maximal knowledge about the environment with minimal learning time and costs [58]. However, the agent also needs to utilise its learned knowledge; this is known as exploitation. This gives rise to the so-called *exploration versus exploitation trade-off*; the agent needs to balance exploration and exploitation in such a way that enables both the acquisition and utilisation of knowledge. Consider a human trying to navigate a maze, as in Figure 1.1. Initially, the human has no knowledge about the maze, so they must choose some path to try; if the path they try is realised to lead to a dead end, they might receive some negative reinforcement, and learn that this path is not a good one to take. To escape the maze, the human must try new things; they must explore the maze and discover where the dead ends are, and which paths seem promising, but they also need to exploit the knowledge that they have gained, and follow the promising paths.

Conversely to RL, Automated Planning (or just Planning) [17; 29; 38] uses embedded knowledge of the environment, in the form of a model, to determine the optimal sequence of successive actions to fulfil a given goal. However, planning relies on the model to accurately represent the environment; which cannot be guaranteed, due to approximations, abstractions, and human-error, and therefore planning can be quite fragile. As an example, consider Figure 1.1 again. Imagine the human is given a map of the maze, but it shows the true path out of the maze to be a dead end. The human might look at the map and decide it's impossible to escape the maze.

Planning and RL take different approaches to decision-making; relying on previously obtained, or embedded, knowledge versus obtaining knowledge through experience. However, they may be combined, which is known as Model-Based RL (MBRL); which is most commonly encapsulated in the form of planning over a learned model, which has been shown to be very effective in recent years [45; 31]. Coming back to the example in Figure 1.1, the human could reason to try the true path that seems like it is a dead end, and realise that it is not actually a dead end, and update the map for the next person.

Exploration is a widely studied topic in RL, since it necessitates learning. Thus, strategies for exploration vary widely. A particularly interesting avenue of research is the use of models within exploration - from using approximate models to constrain exploration, such as DARLING [30], to using optimistic models to guide exploration, such as OIM [57], to using ensembles of models to derive novel states to explore towards, such as MAX [43]. In practice, however, model-free exploration methods based on randomness, such as ϵ-greedy [52; 60] are

ubiquitous.

Exploration in RL is still an open problem; many advanced, complex, techniques have been proposed, but the most widely used techniques remain the simplest. Thus, within this work we explore the development of a framework that synergises planning and learning in order to drive exploration by making intelligent hypotheses about the environment, informed by the inherently inaccurate, but still useful, model, previous experience and environmental observations, with the goal being to mitigate model inaccuracies on the quality of learned behaviour, and an overarching aim of distancing the field of exploration from randomness.

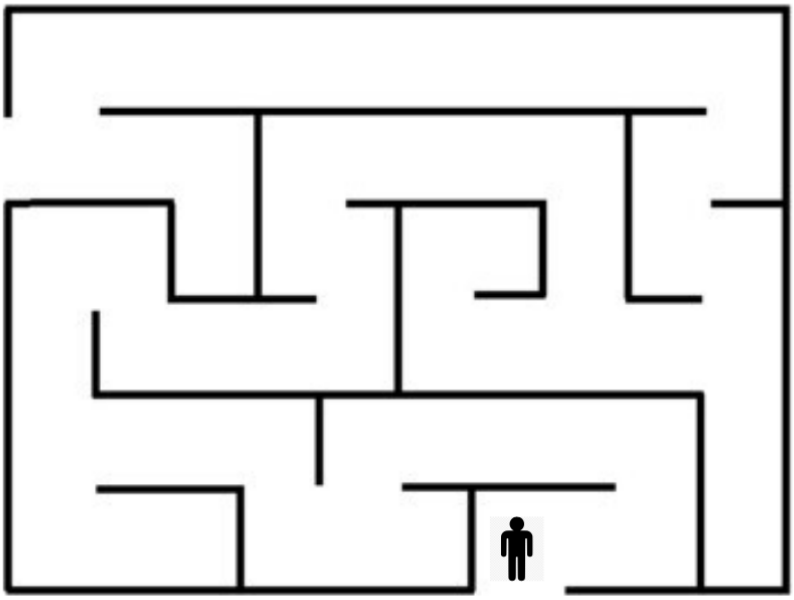


Figure 1.1: Maze Navigation

Chapter 2

Background Research & Notation

Within this chapter, we begin by introducing the formal method for representing sequential decision making problems. We then proceed to formally introduce Planning and discuss some algorithms of interest. We finish by formally introducing Reinforcement Learning, the various encapsulations of RL and providing a short survey of exploration methods in RL; this survey is the main bulk of our literature review.

2.1 Markov Decision Processes

The Markov Property states that the future is conditionally independent of the past given the present. A sequential decision-making problem that satisfies the Markov Property is known as a Markov Decision Problem, and can be modelled by a Markov Decision Process (MDP) [36]. Where an agent is able to fully observe its state, the problem can be modelled as an MDP. Conversely, where an agent can only partially observe its state, the problem can be modelled by a Partially Observable Markov Decision Process (POMDP). Consider an agent playing a perfect information game, such as Chess or Go, the agent (and its opponent) are able to fully observe the state of the game with ease. However, a robot navigating through a maze may not be able to observe its exact state due to uncertainty in its sensors. Furthermore, an MDP can be stationary or non-stationary which refer to whether its dynamics stay the same, or change temporally. An MDP that contains absorbing states, that is any state that upon entering the task terminates, then is said to be episodic.

Within this work we assume that the agent is fully able to observe its state, the environment is stationary and can be discretised in some way and tasks are episodic. Hence we consider **stationary**, **finite**, **undiscounted** MDPs. Thus, an MDP is a 4-tuple, $\text{MDP} = (S, A, T, R)$ where:

- S is a finite set of states.
- A is a finite set of actions.
- $T : S \times A \times S \to [0, 1]$ is the transition function, which determines the probability of transitioning from a state $s \in S$ to $s' \in S$ with an action $a \in A$.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, which determines the reward signal, $r \in \mathbb{R}$ received by the agent from transitioning from a state $s \in S$ to $s' \in S$ with the action $a \in A$. This reward is extrinsic to the agent; it comes from the environment.

Definition 2.1. An MDP is said to be deterministic if $\forall s, s' \in S, \forall a \in A$, $T(s, a,' s) \in \{0, 1\}$.

Definition 2.2. An MDP is said to be stochastic if it is not deterministic

2.1.1 Policies, Value and Quality

A policy, denoted by π, is a mapping of states to actions that describes a behaviour within an MDP. The desired behaviour within an MDP is to maximise the cumulative reward, this is known as the optimal policy and denoted π^*. The cumulative reward is given by:

$$G_t = \sum_{k=t+1}^{T} R_k \tag{2.1}$$

which represents the cumulative reward received from time t to some finite time T. A Value Function, or state-value function, denoted $V^\pi(s)$, measures the expected cumulative reward that can be received starting in each state $s \in S$ and following a policy π:

$$V^\pi(s) = \mathbb{E}^\pi\left[G_t|s_t = s\right] = \mathbb{E}^\pi\left[\sum_{k=t+1}^{T} R_k|s_t = s\right] \tag{2.2}$$

Where $\mathbb{E}$ represents the expected value that the agent follows π, t is an arbitrary time step and T is a finite time horizon. The optimal Value Function, V^*, is the one that is maximum for every $s \in S$ and is given by:

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.3}$$

This can also be represented in a recursive form, known as the Bellman Optimality equation for V:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + V^*(s')] \tag{2.4}$$

Similarly, a Q-Function, or action-value function, denoted $Q^\pi(s, a)$, measures the expected cumulative reward that can be received starting in each state $s \in S$, taking an action $a \in A$ and thereafter following a policy π:

$$Q^\pi(s, a) = \mathbb{E}^\pi\left[G_t|s_t = s, a_t = a\right] = \mathbb{E}^\pi\left[\sum_{k=t+1}^{T} R_k|s_t = s, a_t = a\right] \tag{2.5}$$

Where $\mathbb{E}$ represents the expected value that the agent follows π, t is an arbitrary time step and T is a finite time horizon. The optimal Q-Function, Q^*, is the one that is maximum for every $s, a \in S \times A$ and is given by:

$$Q^*(s, a) = \max_\pi Q_\pi(s, a) \tag{2.6}$$

Similarly to the Value Function, this can also be represented in a recursive form, known as the Bellman Optimality equation for Q:

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \max_{a'} Q^*(s', a')] \tag{2.7}$$

An interesting observation is that π^* can be derived from both V^* and Q^*. In the case of V^*, π^* can be derived by determining at each state, which actions yield the next state with the maximum value. In the case of Q^*, π^* can be derived by simply taking the action with the highest Q-value at each state [53].

2.2 Planning

Planning involves reasoning on a model of an environment, in order to produce a sequence of successive actions that will achieve a specified goal whilst satisfying some constraints or achieving some objectives, such as minimising cost [17; 29; 38].

In the context of MDPs, the goal is to maximise cumulative reward. Formally, let $s_t \in S$ be some start state at a discrete time, t. The goal of the planner is to produce a sequence of actions, $P = (a_t, a_{t+1}, ..., a_{t+n})$, such that $\pi^*(s_t) = P(t)$, for all t.

Planning has been a widely studied topic in AI for many years and as such there exist many planning methods and algorithms. While the use of Planning is central to this work, particular planning methods and algorithms are not; therefore, we only consider heuristic search methods and planning by Dynamic Programming.

2.2.1 Heuristic Search

Heuristic search (or informed search) [16] is a class of search algorithms that use problem specific knowledge in the form of heuristics to guide the search. A heuristic, denoted $h(s)$, is a function that estimates the cost of transitioning from the current state, s, to the goal state. We consider best-first search algorithms, which aim at expanding states that appear best according to an evaluation function, denoted $f(s)$.

2.2.1.1 A* Search

A* Search (or just A*) [19] is a heuristic search algorithm that is a form of best-first search. A cost function, $g(s)$, is introduced, which indicates the cost to get from the start state to a state s. The main idea is that the evaluation function can be a combination of the cost to get to the state being evaluated, $g(s)$, and the estimated cost to get from the state being evaluated to the goal state, thus:

$$f(s) = g(s) + h(s) \tag{2.8}$$

The evaluation function indicates the estimated cost of the minimum cost solution that passes through s [38]. A* works by guiding the search to expand states with the lowest value for $f(s)$ first, in order to determine a sequence of states which is the path (or in our case, the plan). A* is a complete algorithm; it will always find a solution if one exists. However, the optimality of A* depends on the choice of the heuristic: the heuristic must be admissible, which means that it must never overestimate the cost of getting from the current state to the goal state; in this sense, the heuristic must always be optimistic.

2.2.2 Dynamic Programming

Dynamic Programming (DP) [5; 6] refers to a collection of methods that can be used to solve MDPs, by computing the optimal policy with respect to them. The two main algorithms that we consider are Policy Iteration and Value Iteration; both of which are forms of Generalised Policy Iteration (GPI). GPI refers to any process that involves Policy Evaluation and Policy Improvement acting with each other. As it happens, most DP and RL methods are forms of GPI [53].

2.2.2.1 Policy Evaluation and Improvement

Given a policy, π, it can be evaluated by computing its Value Function, V^π:

$$V^\pi = \sum_a \pi(a|s) \sum_{s'} T(s,a,s')[R(s,a,s') + V^\pi(s')] \tag{2.9}$$

Policy Improvement is the process of generating an improved policy from a suboptimal policy [6]. A policy π could be improved by contemplating taking an action $a \in A$ in a state $s \in S$, such that $\pi(s) \neq a$ and then continue following π thereafter. We can evaluate this change by computing the Q-function for s, a:

$$Q^\pi(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + V^\pi(s')] \tag{2.10}$$

Then comparing $Q^\pi(s,a)$ with $V^\pi(s)$. If $Q^\pi(s,a) > V(s)$, then the policy is update and improved as such: $\pi(s) = a$.

2.2.2.2 Policy Iteration

Policy Iteration (PI) [5; 20] is a method for computing the optimal policy of an MDP. Assuming an arbitrary initial policy, π, at each step π is evaluated, yielding a Value Function V^π (or a Q-Function Q^π). The policy is then updated greedily with respect to V^π (or Q^π). This iterative process leads to monotonic improvements to π.

2.2.2.3 Value Iteration

Value Iteration (VI) [5] is a method for computing the optimal estimate for the Value or Q-Function of an MDP. As the estimate approaches optimality, the policy that is greedy with respect to the Value or Q-Function also approaches optimality [56]. In the context of Value Functions, assuming an arbitrary initial estimate, V, at each step V is updated as such:

$$V(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + V(s')] \tag{2.11}$$

for all $s \in S$. This update originates from the Bellman optimality equation for V, Equation 2.4. For Q-Functions, assuming an arbitrary initial estimate, Q, at each step Q is updated as such:

$$Q(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \max_{a'} Q(s',a')] \tag{2.12}$$

for all $s \in S$, $a \in A$. This update originates from the Bellman optimality equation for Q, Equation 2.7.

2.3 Reinforcement Learning

Within an RL setting, formalised by an MDP, an agent learns how to behave in an environment by interacting with it through actions, at discrete, sequential, time steps, and observing the affects through its new state and a scalar reward signal, as seen in Figure 2.1. The reward

signal may be delayed, meaning that the consequences of actions may not be known until long after they are taken [4]. This gives rise to the (temporal) credit assignment problem [33], the problem of determining which actions led to an outcome and assigning credit among them; it's often the case that a sequence of actions led to an outcome, rather than a single action. The ultimate goal of the agent is to learn a policy, π^*, that maximises the expected cumulative long-term reward [53]; as we outlined in Chapter 1, exploration, exploitation and the trade-off between the two, are key to this. The two main instantiations of RL are model-free and model-based, each of which we will delve deeper into within this section.

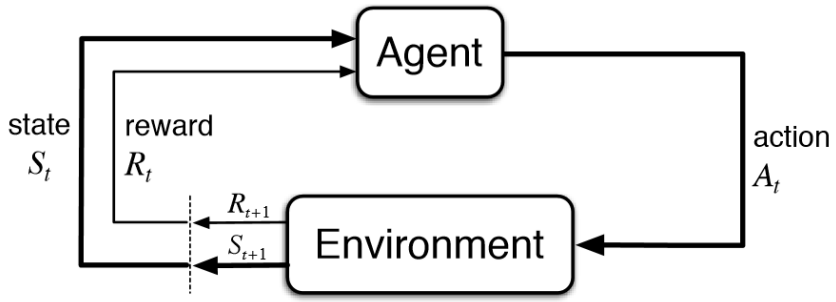


Figure 2.1: The RL Loop [53]

2.3.1 Model-free RL

Model-free (or direct) is where the agent learns a policy directly from experience gained by interacting with the environment; it is purely trial-and-error. Model-free learning tends to be quite flexible to varying problems, since no assumptions are made about the environment's dynamics. Furthermore, since the environment's dynamics do not need to be stored, learned or considered, model-free learning is scalable and does not suffer from the *curse of dimensionality* (in the non-tabular cases) and can be computationally efficient, as there is generally no lookahead deliberation process involved. However, model-free learning can be sample-inefficient; the agent may need to take many actions before discovering the optimal policy. Moreover, in real-world domains, model-free learning may not even be feasible, due to the cost of experience. The most common model-free approach is Temporal Difference Learning.

2.3.1.1 Temporal Difference Learning

Temporal Difference (TD) Learning [55; 39; 40] aims at solving the problem of temporal credit assignment by combining ideas from Monte Carlo methods, which learn directly from experience in the environment, and Dynamic Programming methods, which bootstrap estimates from other previously learned estimates; it is a form of GPI [53].

The core idea of TD Learning is to update the estimate of the Value Function, V_π, whenever there is a difference between temporally successive predictions. This difference is known as the TD error. The updated estimate is bootstrapped from the previous estimate, meaning that TD learning essentially learns a prediction from another prediction.

The simplest TD Learning algorithm is TD(0), which updates it's estimates as such:

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + V(s_{t+1}) - V(s_t)] \tag{2.13}$$

The TD target is the current prediction of the expected cumulative reward: $r_{t+1} + V(s_{t+1})$, the TD error is $r_{t+1} + V(s_{t+1}) - V(s_t)$ and $0 \leq \alpha \leq 1$ is the learning rate, which is used to control how much weight is given to the TD error when updating the estimate [49].

TD Learning can also be extended to learn a Q-Function, which stores the value of state-action pairs rather than the value of states. For each state-action pair, it stores an estimate of the expected cumulative reward starting in that state and taking an action, and following a fixed policy. This can be done using Q-Learning and SARSA.

Q-Learning [60; 59] is an off-policy TD Learning method that learns a Q-function. It is off-policy, as the update rule assumes a greedy policy - this means that the value of the optimal policy is learned independently of the agent's actions following the current policy. Q-values are iteratively updated using the Bellman equation, the update rule is as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.14}$$

SARSA (State Action Reward State Action) [37] is an on-policy TD Learning method that learns a Q-function. It is on-policy, as the update rule assumes that the current policy continues being followed - this means that the value of the current policy being followed is learned. Q-values are iteratively updated using the Bellman equation, the update rule is as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.15}$$

2.3.2 Model-Based Learning

Model-based (or indirect) RL is where an agent uses a known or learned model of the environments dynamics (in the form of an MDP) in order to learn an optimal policy. The agent may learn the model and policy at the same time, as in the Dyna family [50; 51] or learn a policy by planning over a known model, as in AlphaZero [44] or learn a policy by planning over a learned model, as in MuZero [41]. Model-based RL offers improved real-world sample efficiency, potential for directed, informed exploration and better asymptotic performance, the prospect of transfer learning, safer learning and explainability [34]. These benefits come at a computational cost, typically in planning; however we take the stance that this added computational cost is certainly worth it, particularly for the benefit of reducing learning costs. Since it's not always the case that a model is known as a priori, and even if it is it might be inaccurate, model-learning is a key component of model-based RL.

2.3.2.1 Model Learning

Model Learning can be difficult due to uncertainty caused by limited data and stochasticity. Uncertainty can be overcome through sufficient exploration; sampling a transition multiple times can give a better estimate of its probability distribution.

Model learning can be viewed as a supervised learning problem [22]. In discrete environments, exact models of the environment's dynamics can be learned in the form of a *tabular maximum likelihood model* [51] which maintains a table with an entry for every possible

transition. In the stochastic case, for each transition the table will store:

$$T(s,a,s') = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')} \tag{2.16}$$

Where $n : S \times A \times S \to \mathbb{Z}$ represents the number of times a transition has been observed. This also works for the deterministic case, which will result in all transitions mapping to either 0 or 1; but this leads to unnecessary computations; it's better to manually update T if the domain is known to be deterministic. A key drawback to this approach is the lack of scalability to large state spaces.

In continuous domains an exact model cannot be learned, due to the infinite number of states (and potentially actions). An approximate model can be learned by using state aggregation (which discretises the state space) and a tabular maximum likelihood model [27].

In continuous domains its impossible to learn an exact tabular model, and it may also be infeasible for discrete domains with large state spaces, therefore an approximate model must be learned; most commonly this is done through Function Approximation methods, such as Linear Regression [54; 47] and Gaussian Processes [13].

2.4 Exploration Methods in Reinforcement Learning

Within this section, we will provide a brief survey of exploration methods in RL. Whilst Thrun distinguished between undirected and directed exploration [58], and Amin et al. distinguished between reward-free and reward-based exploration [1], there is no definitive taxonomy of exploration methods; as such, we will use our own categories, which distinguishes between model-free, model-based and hybrid approaches.

2.4.1 Model-Free Methods

Model-free exploration methods are characterised by the fact that they do not use, or learn, a model to aid exploration. This means that they can be robust to varying domains, where designing or learning a model could prove difficult. However, they must rely on other methods to drive exploration, which tend to be grounded in randomness.

Purely random exploration comes in the form of a Random Walk, or unguided random search [2], which arises from randomly sampling actions with uniform probability. Exploration occurs naturally when the agent moves away from the goal, rather than closer to it. This is perhaps the most naive exploration method, since it is entirely random, and thus it is very inefficient and rarely used in practice.

Boltzmann exploration samples an action according to the Boltzmann distribution:

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}} \tag{2.17}$$

Where the temperature, $0 \leq \tau \leq 1$, is a hyperparameter that determines how frequently random actions are chosen; as τ approaches 0, the policy approaches greediness [58; 1]. When there are large differences between Q-values, little exploration occurs, however when Q-values are close to each other, considerable exploration occurs. This means that Boltzmann

exploration has a particular reliance on the initialisation of the Q-values, and the early exploration [61]. Moreover, Boltzmann exploration does not consider uncertainty in Q-values [10], which can lead to actions being chosen which appear to be promising, but in reality are not, due to epistemic uncertainty.

ϵ-greedy [60; 52] uses a hyperparameter, $0 \leq \epsilon \leq 1$ to explicitly balance between exploration and exploitation. With probability ϵ the agent explores by taking a random action, which is sampled with uniform probability, with probability $1 - \epsilon$ the agent exploits by taking the best action, which is selected greedily with respect to the current policy. Uniformly choosing actions means that actions which are known to be sub optimal can be continually evaluated long after they are realised to be so.

Pure random exploration is very inefficient, and certainly not a practical exploration method. Boltzmann exploration and ϵ-greedy offer better exploration strategies than pure randomness, but they are not without their faults. Moreover, model-free methods in general tend to lack temporal persistence which can lead to indecisiveness during exploration, for instance moving back and forth between two states, leading to inefficient exploration. Extensions have been proposed to Boltzmann exploration and ϵ-greedy that overcome some of their respective issues, such as Boltzmann-Gumbel exploration [10] and ϵz-greedy [12], although the standard methods seem to favoured in practice.

2.4.2 Model-Based Methods

Model-based exploration methods are those that use, or learn, a model to drive exploration.

2.4.2.1 Optimistic

Optimistic exploration methods are based on the *optimism in the face of uncertainty* (OFU) principle [23].

Explicit, Explore or Exploit (E^3) [24] maintains a partial model of the environment through collecting statistics, akin to the *tabular maximum likelihood* approach. A distinguishment is made between *known* and *unknown* states; a state is *known* after it has been visited an arbitrary number of times, thus its learned dynamics must be close to the true dynamics. If the current state is *unknown*, then *balanced wandering* takes place, and the agent explores by selecting the action that has been taken the least number of times from that state. If the current state is *known*, then an exploitation policy and an exploration policy are computed. One of these policies is chosen and followed for a finite time horizon; exploitation is chosen if the policy will induce large cumulative reward, exploration is chosen otherwise.

R-MAX [7] is an extension and generalisation of E^3. It begins with an optimistic initial model such that every transition leads to some imaginary state, *The Garden of Eden State*, that returns the maximal reward, denoted R_{max}. R-MAX uses the same concept of *known* and *unknown* states as in E^3; after a state becomes known, then the model is updated with the collected statistics regarding that state. The agent always follows the optimal policy according to the model. R-MAX removes the need for explicit contemplation between exploring and exploiting as in E^3; exploration and exploitation occur naturally.

Optimistic Intial Model (OIM) [57] begins with an optimistic model, the same as R-MAX. The model is updated through observations similarly to E^3. The agent maintains two

Q-functions; one for the real extrinsic rewards, and one for intrinsic exploration rewards. Whilst exploring, the agent acts optimally with respect to the combination of the two Q-functions; thus the latter Q function can be seen as an exploration bonus. During this, the agent will either act near-optimally, or will gain new information it can utilise. After exploration is terminated, perhaps after a certain number of time steps, the agent acts optimally with respect to the Q-function based on extrinsic rewards.

E^3, R-MAX and OIM are provably able to achieve near-optimal performance in polynomial time in discrete environments, however they don't tend to be very efficient in practice. R-MAX and OIM in particular show that optimism is definitely a useful heuristic, but they are very optimistic in perhaps an unrealistic way; an interesting question to ask is whether this unrealistic level of optimism necessary, and if a *reasonable* level of optimism is just as useful. These methods also assume that the model will eventually become correct, which may not be the case in complex tasks, and in the worst case will learn the entire model, leading to wasted exploratory steps.

2.4.2.2 Intrinsically Motivated

Model-Based Active Exploration (MAX) [43] is purely an exploration framework, which pure exploitation could follow from. An ensemble of dynamics models is maintained and trained on new observations. At each time step a model is sampled from the ensemble, and a one-step policy is derived from the model such that it maximises the intrinsic reward function, which indicates novel states using expected information gain, based on the Jensen-Shannon Divergence.

Plan2Explore [42] separates learning in to two separate phases. During the first phase, the agent learns a global model of the environment, in the form of a latent dynamics model, whilst maintaining and training an ensemble of models on new observations. To explore, the agent trains an exploration policy in the world model that aims to seek out novel states; novelty is defined by estimating latent disagreement in the ensemble of models with the global model. In the second phase, the agent adapts to downstream tasks by using the extrinsic reward signals.

MAX and Plan2Explore both perform task-agnostic exploration, which they claim means they can generalise to downstream tasks easily. However, only Plan2Explore was shown to do this, and in-fact it was shown to generalise to a variety of downstream tasks in a zero-shot manner. However, both of these approaches use model-ensemble approaches which can be very computationally expensive; an interesting question to ask is whether this is really necessary.

2.4.3 Hybrid Methods

Go-Explore [15; 14] consists of two phases. In phase one, an *archive* of states is maintained, and trajectories that can be used to reach them, initially this contains only the start state. Exploration is done by sampling a state from the archive, planning to that state (Go) and from there exploring randomly (Explore). The *archive* is updated when novel states are encountered, or when better trajectories to states already within the *archive* are discovered. This repeats until the end of an episode. In phase two, promising trajectories are made more robust to noise using imitation learning. Go-Explore does not explicitly define what exploration strategy should be used whilst exploring, however random exploration was used in the empirical evaluation.

PEG [21] builds upon Go-Explore, offering a method for choosing which goals should be sampled from the *archive* in the Go phase; the goal that has the highest expected exploration value is chosen.

Domain Approximation for Reinforcement Learning (DARLING) [30] constrains exploration by computing a *partial policy* from the model, and performs model-free RL, in the form of ϵ-greedy, within it. A *partial policy* is a function that maps states to possible actions, and is constructed from a subset of possible plans according to some metric and threshold (for instance all plans that are shorter than 1.5 times the length of the shortest plan).

Go-Explore and PEG offer a simple framework that bootstraps from exploration performed in previous episodes, meaning that unnecessary exploration is reduced. However, when evaluating, ϵ-greedy was used, which might lead to the agent returning to states it had already explored previously; if this framework were combined with an intelligent exploration strategy, it could be much more powerful. DARLING constrains exploration to a set of reasonable states, based on an initial, potentially inaccurate, model. This work is particularly interesting to us, as unlike the other methods we have discussed in our review, it leverages an initial model; whereas all other works discussed here learn from scratch.

2.4.4 Conclusions

While model-free methods, such as ϵ-greedy, are widely used in practice, they are inefficient and often introduce hyperparameters that need to be tuned; their ubiquity is perhaps due to their ease of implementation. Model-based methods offer a more intelligent approach to exploration and tend to be based on optimism or intrinsic motivation; optimistic methods seem to be over-optimistic and methods that use intrinsic motivation seem to be computationally expensive. However, model-based, and some hybrid, methods often assume that the model will become correct and don't leverage an initial model, which if used could speed up learning. The work of DARLING is particularly interesting to us as it leverages an initial model.

Chapter 3

Methods

Within this chapter, we describe our method. We begin by discussing the motivations for our method, which are grounded in human reasoning, then we formalise our method and supply an illustrative example. We finish by discussing the various implementations that were produced in order to evaluate the main contribution of our method.

3.1 A Human Thought Process

When human's make decisions, they are often driven by previous experiences and some form of internal model of the world that we maintain. However, sometimes we question ourselves and think "what if this other decision is better?"; this leads to us making different decisions, which are driven by reasoning. As an example, consider the route that you travel to work everyday. At some point you might question yourself and think "what if there is a faster route?", which may lead to you taking a longer route, by distance, because you think that there is a possibility that it could be faster. It might turn out that the longer route is indeed slower, and you may never try it again. However, it might be that you realise that the longer route is much faster, due to lack of congestion for example, and thereafter you follow it every morning. This is the decision-making and reasoning process that we try to emulate.

3.2 Unifying Optimism and Intrinsic Motivation

We propose a framework that synthesises planning and reinforcement learning that aims to achieve efficient exploration through optimism and intrinsic motivation. We acknowledge that model inaccuracies are inevitable, but inaccurate models are still useful, and use this as a basis to drive exploration. We enable the planner to hypothesise changes to the model that would be most beneficial to it. The planner produces plans with these hypotheses in mind, and the correctness of them is realised through experience in the real environment, leading to the model being updated accordingly.

This approach utilises the principle of *optimism in the face of uncertainty*; we state that until we have gained experience in the environment, we are uncertain about our model, and thus we make optimistic hypotheses about the environment. Furthermore, this approach is based on intrinsic motivation; the planner has some intrinsic motivation to seek out states it is uncertain about. However, extrinsic rewards are also considered, as the planner always produces plans with maximising the cumulative reward in mind.

We enable the planner to make these hypotheses by equipping it with additional actions, which we denote Meta Actions. The use of Meta Actions is the main contribution of this work.

3.3 Meta Actions

These actions do not cause the agent to act in and affect the environment, and thus do not return any observation in terms of a new state and a reward, but rather cause changes directly to the model when called upon. An important factor and one of the main difficulties is deciding what Meta Actions should be exposed to the planner and when the planner should be able to invoke the Meta Actions. Therefore, we state three conditions which all must hold for a Meta Action to be invoked: a Meta Action must be admissible, feasible and reasonable.

Definition 3.1. A Meta Action is admissible if applying it to the model leads to a better policy with respect to the model.

It's important that hypothesised changes are optimistic; that they benefit the planner in some way, otherwise they are not very useful. Thus, admissibility is important, as defined in Definition 3.1, if applying a Meta Action does not result in any benefit to the planner, but rather it negatively affects the planner, then it should not be called. This means that planning must be done with finding the sequence of actions that maximises reward in mind.

Definition 3.2. In a deterministic domain, a Meta Action is feasible if the target state-action pair that it is to be applied to has not been previously observed nor has that Meta Action been previously called on it.

Definition 3.3. In a stochastic domain, a Meta Action is feasible if either:

- The target state-action pair that it is to be applied to has not been previously observed within the current episode nor has that Meta Action been called on it within the current episode.
- The target state-action pair that it is to be applied to has not been previously observed within the current or previous N episodes, nor has that Meta Action been called on it within the current or previous N episodes.

If the agent were to make hypothesises that contradict its own observations, it would induce hallucinatory behaviour. Furthermore, by the optimistic nature of our approach, its possible that the agent could infinitely hypothesise changes to the model, namely in the form of increased rewards. Therefore, feasibility is important, as defined in Definitions 3.2 and 3.3. Within deterministic settings the transitions and rewards that the agent observes are the true ones; thus we can be certain that the model is correct with respect to the observations of the agent, hence Definition 3.2 ensures that observations in the entire lifetime of the agent are not contradicted. Definition 3.3 gives two alternate conditions for feasibility in stochastic settings, each of which are explored throughout the implementations to see which is stronger. This definition ensures that the agent cannot contradict recent observations (those within the current or previous N inclusive episodes), and thus hallucinate, whilst ensuring that changes to the model cannot be infinitely hypothesised. It allows Meta Actions to be called multiple times on state action pairs, including those of which have been previously observed, this is because we can never be completely certain of the correctness of the model due to the aleatoric uncertainity introduced due to the stochasticity.

Definition 3.4. A Meta Action is reasonable if it has been embedded by-hand or learned through experience.

Often, the change to the model that would be of most benefit the planner, is to simply add a transition from the current state to the goal state. However, this is almost never going to be a change that is realised to be correct, it is too optimistic, and instead would lead to behaviour akin to taking a random-walk through the state space until the goal is reached and the Meta Actions have been exhausted, or perhaps it would induce exploration that is akin to R-MAX or OIM. Hence, reasonability, as defined in Definition 3.4 is key.

3.4 Framework

We assume that the environment has a discrete state space, S, or a state space that can be discretised, a finite action space, A, with deterministic or stochastic dynamics which can be described by a transition function, T, and deterministic rewards which can be described by a reward function, R. Therefore, we assume that the environment can be modelled as an MDP $E = (S, A, T, R)$, within which the agent acts at discrete time steps. The goal of the framework is to produce a policy, π^* which maximises the cumulative reward received when acting in the environment E. Additionally, we assume that an attempt at modelling the environment has been made and embedded in an MDP $M = (S, A, T', R')$, where the same conditions hold for T' and R' as for T and R. We do not expect or require M to be accurate.

We perform model-learning by maintaining T' as a *tabular maximum likelihood model*, and keep track of observed transitions using a function n which maps state-action-state triples to an integer indicating how many times that transition has been observed. We chose this approach of model-learning, as we assumed a discrete/discretised state space and this method offers a simple implementation of model-learning.

We learn a tabular Q-Function, Q, through Q-Learning, which the policy π is derived from. Q-Learning was chosen over SARSA, since it allows the optimal policy to be learnt independently of the current policy being followed; this suited our framework well, since we acknowledge that the policies followed during the exploration may not be optimal.

Our framework consists of two distinct phases: a planning phase where exploration takes place and a model-free phase. Since we only consider episodic tasks, the agent is given a finite number of episodes, N_p, for the planning phase, thereafter until termination , the model-free learning takes over and the idea is that given sufficient model-free episodes Q can converge to Q^*, and thus π^* can be derived. Algorithm 1 gives a relatively high-level overview of the framework.

3.4.1 The Planning Phase

The planning phase has three distinct steps: planning, acting and learning, which can be seen in Figure 3.1. The goal of the planning phase is to perform exploration, and provide a good estimate for Q^* which the model-free learning can bootstrap and derive a policy from.

Algorithm 1 High-level Framework Pseudocode

Input: N, number of episodes
Input: N_p, number of planning episodes
Input: $M = (S, A, T, R)$, model
Input: s_s, s_g, start, goal state
Output: π, final policy

```
Q(s, a) ← 0, ∀s ∈ S, ∀a ∈ A
s ← s_s
for i = 0 to N do
    Planning ← (i < N_p)
    while s ≠ s_g do
        if Planning is true then
            a ← Plan(s, s_g)
        else
            a ← arg max_a Q(s, a)
        end if
        Take action a in state s, observe reward r and new state s'
        Learn Q according to observation.
        if Planning is true then
            Learn M according to observation.
        end if
        s ← s'
    end while
end for
π(s) ← arg max_a Q(s, a), ∀s ∈ S, ∀a ∈ A
return π
```

3.4.1.1 Planning

The planner constructs a temporary model, M', which is identical to M. It then plans on M' to produce a plan, P, from the current state s to some goal, terminal, state s_g. The planner has access to the action space, A, as well as the additional Meta Actions. Whilst considering the Meta Actions, the planner may create additional temporary models, in order to evaluate the benefit of calling a particular Meta Action. If a Meta Action is chosen, then the temporary model M' is updated, and planning continues. P is maintained by the planner until M is updated, when re-planning occurs. This ensures that unnecessary planning does not take place, saving on computational costs. However, this means that some mechanism needs to be in place for determining if the model has been altered since the last plan was generated; this can be a simple Boolean flag. P is stored in a first-in, first-out (FIFO) data structure, such as a Queue. Thus, when the planner is invoked by the agent it simply removes and returns the top action.

3.4.1.2 Acting

At discrete time steps, t, the agent samples an action a from the Planner, and executes it. If the action is a Meta Action, then it does not result in any interactions with the environment, and thus no observations or "time steps" are made. Otherwise, at time $t + 1$ it observes its new state s' and the scalar reward signal.

3.4.1.3 Learning

Learning only occurs of the action taken was not a Meta Action. The observation table is updated with the observed transition: $n(s, a, s') \leftarrow n(s, a, s') + 1$, and if necessary the transition function, T' of M is updated by Equation 2.16. Furthermore, the reward function, R', of M is updated with the received reward, if necessary. Q is updated according to the new state and reward received using Equation 2.14.

3.4.2 The Model-Free Phase

The model-free phase has two distinct steps: acting and learning, which can be seen in Figure 3.2. The goal of the model-free phase is to use pure model-free learning to bootstrap from the Q values learned during exploration, and get as close as possible to Q^*, so that π^* can be derived.

3.4.2.1 Acting

At discrete time steps, t, the agent greedily selections an action a with respect to the Q, it selects the action according to the current policy, and executes it. At time $t + 1$ it observes its new state s' and the scalar reward signal.

3.4.2.2 Learning

Q is updated according to the new state and reward received using Equation 2.14. Eventually, the updates will not result in changes to the policy, π, and therefore it will be continually followed until termination.

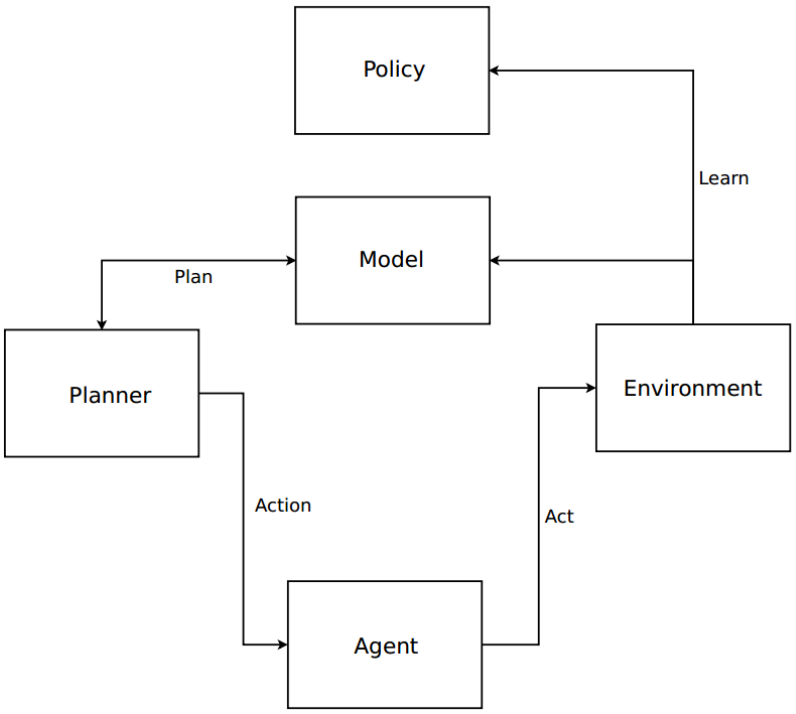


Figure 3.1: Planning Phase

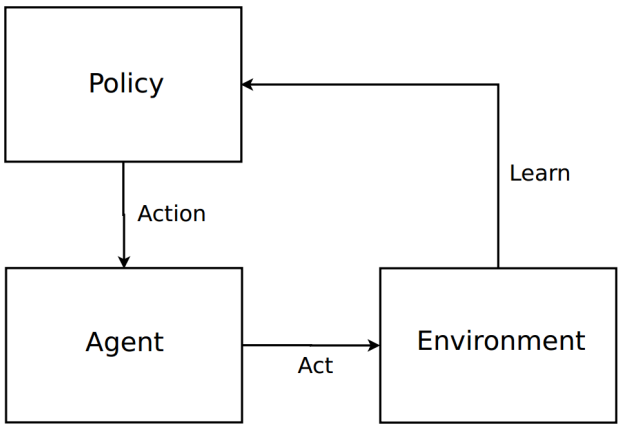
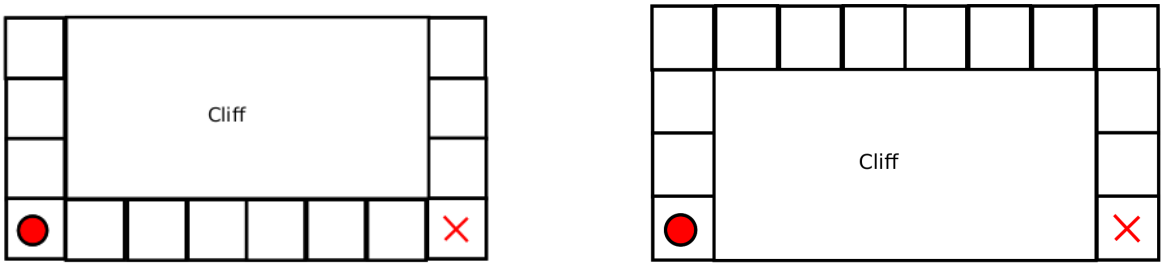


Figure 3.2: Model-Free Phase

3.5 An Illustrative Example

Consider the deterministic domain in Figure 3.3a. This is a modified version of the cliff-walking domain [53]. The red circle located at (0,0) is the agent; the goal is located in the bottom right corner, (0, 7). If the agent transitions into one of the "cliff" states, they are returned to the start state. Let's suppose that for some reason, perhaps due to changes in the environment, the agent is seeded with the inaccurate model shown in Figure 3.3b. A pure planning approach would fail, as it would continually plan a path that goes through the cliff, due to the inaccurate model. A pure model-free learning approach would probably be successful, as this is a very simple domain, however in reality domains can be much more complicated than this; which is where model-free methods begin to struggle.



(a) Modified Cliff-Walking Domain (b) Agent's Model

Figure 3.3: Comparison of Modified Cliff-Walking Domain and Agent's Model

Assuming an implementation of our framework, where reasonable Meta Actions are embedded, that allow the agent to hypothesise changes to transitions and rewards originating in the current state, and targeting an adjacent state. Initially, the most beneficial changes to the model would be to remove the cliff across the bottom row, as shown in Figure 3.4. The agent then attempts to follow the plan going across the bottom of the grid, after which it realises that the hypotheses were correct. The planner may make further hypotheses which lead to the agent trying alternate paths, for instance hypothesising that the cliff is not present across the second row and that the reward through that row is increased; meaning that it would be a better path than the previous one. This process continues, with the planner making

hypotheses and the agent verifying them, until no more hypotheses can be made, or the agent runs out of planning steps, after which the model-free learning takes over, and the produce policy matches the initial plan.

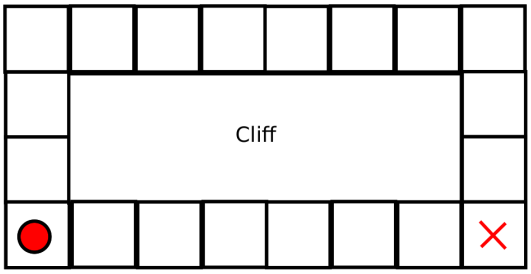


Figure 3.4: Hypothesised Model

3.6 Implementations

The high-level ideas of the framework led to various implementations. The main differences among the implementations lie in the choice of planning algorithm, thus how the planning algorithm hypothesises changes, the available Meta Actions and the source of reasonable Meta Actions; learned versus embedded. Furthermore, some implementations had simplifications applied to them to deal with specific domains. We note that these implementations are not definitive, and much improvements could be made, but they are designed with the goal of proving the usefulness of Meta Actions.

3.6.1 RL-A* Meta

This was the initial implementation of the framework. The chosen planner was a basic A* planner, which limited the implementation to deterministic domains. A* was chosen due to ease of implementation and its use of an evaluation function, f, which provided a good means of evaluating Meta Actions. As discussed in Section 2.2.1.1, the evaluation function, f, is the combination of the heuristic function, h, and the cost function. For the cost function, it was intuitive to utilise the reward function. Namely, we define the cost of being in a state, s, as the sum of inverted rewards that it took to arrive at that state. Namely:

$$f(s_t) = -\sum_{k=0}^{t-1} \left[R(s_k, a_k, s_{k+1}) \right] + h(s_t) \tag{3.1}$$

The choice of heuristic, h, relies on domain specific knowledge, therefore we do not define it here. However, it remains that the heuristic must be admissible. At each state, the next one to be expanded is chosen such that it maximises f, taking into account the Meta Actions.

3.6.2 RL-A* Meta, with short term memory

This implementation was an extension of RL-A* Meta, which aimed to scale to stochastic domains. The stochastic nature meant that the evaluation function once again needed to be

modified, as such:

$$f(s_t) = -\sum_{k=0}^{t-1} \left[(1 - T(s_k, a_k, s_{k+1}))R(s_k, a_k, s_{k+1}) \right] + h(s_t) \tag{3.2}$$

Since transitions were not guaranteed, the cost was weighted using the probability of the transition not occurring. This meant that transitions with a higher probability of occurring were preferred. To ensure feasibility of Meta Actions, a table was maintained which kept track of which actions had been called on which state-action-state triples within the previous N episodes; we refer to this as the short-term memory. This encouraged the agent to try Meta Actions again that it had tried in the past, but "forgotten" that it had done so; if it got unlucky previously due to stochasticity, it could try again and discover a good policy it may not have been able to discover before.

3.6.3 RL-VI Meta

The main problem with the RL-A* agents is the reliance on a good heuristic function, which can be difficult to design and hence, they are limited to domains where a good heuristic function can be easily designed. Therefore, this implementation aims to deal with varying domains. We opted for planning by dynamic programming, namely through Value Iteration (VI). VI was chosen because it allowed for us to easily evaluate plans through the Value Function. We chose VI over Policy Iteration, as it is generally faster, and we needed to perform it many times. Despite its nature, in our setting repeatedly performing VI is not too expensive, as real and hypothetical changes to the model are not too different, which means that it can converge in few iterations. We used a slightly modified version of VI that allowed it to evaluate the Meta Actions. Namely the updated rule was modified as such:

$$V(s) = \max_a \sum_{s'} \max_{M'} T''(s, a, s')[R''(s, a, s') + V(s')] \tag{3.3}$$

Where $M' = (S, A, T'', R'')$ represents each candidate MDP; this includes the original MDP, and those that can be produced by applying each Meta Action.

3.6.4 RL-VI Meta, with learned Meta Actions

The overall implementation is the same as RL-VI Meta, except Meta Actions are learned and obtained through experience, rather than embedded by-hand in the model. Meta Actions are learned when a discrepancy is noticed between the model and the real environment. For instance, when a reward is received that wasn't expected, a Meta Action is learned that increases reward to the value of the unexpected reward. Furthermore, when a transition is experienced that is unexpected, a Meta Action is learned that enables the discrepancy to be emulated; for instance, if the agent expects to move a single state vertically on an "UP" action, but it actually moves to the right, then it will learn to increase the transition probability of moving right on the "UP" action.

Chapter 4

Empirical Evaluation

Within this chapter, we evaluate our various implementations in a collection of gridworld-like domains. The goal of the evaluation is to see how effectively our agents can explore and learn against baseline methods.

4.1 Baselines

In order to truly evaluate our framework implementations, we needed some baselines to compare against. Firstly, we chose ϵ-greedy, with simulated annealing, due to its ubiquity. Secondly, we chose "PRL", a model-based exploration method that always acts greedily with respect to current model during exploration, which is updated through observations; this is essentially our framework minus the Meta Actions, and is explained quite well by Algorithm 1. This was chosen as we wanted to evaluate the usefulness of Meta Actions.

4.2 Domains

The domains that we chose to evaluate within were all gridworld-based. This decision was made due to the ease of modelling such domains, especially in a tabular manner. Furthermore, this enabled us to easily understand how the agents explore. Thus, the actions available to the agents are common between tasks (unless specified otherwise); up, down, left and right. Where Meta Actions are not learned, reasonable ones are specified that enable the agent to hypothesise changes between rewards and transitions among adjacent states. The Manhattan Distance [26] is used as a heuristic in agents that utilise A*.

Gridworld is a deterministic domain, shown in Figure C.1. The agent starts in the middle of the room at the bottom of the grid. There are three doors to leave the room, in the left, right and top of the room. All of the doors are open, if they were closed then the agent would not know how to open them. The goal of the agent is to navigate to the other side of the top door. In our experiments, the model that the agents were seeded with indicated that the door leading to the goal was closed. The agent receives a reward of -1 at every time step.

Cliff-Walking [53] is a deterministic domain, shown in Figures C.2 and C.3. The agent starts in the bottom left corner of the grid and needs to navigate to the goal state at the bottom right of the grid. However, there is a cliff along the bottom of the grid, which the agent needs to avoid. In our experiments, the model that the agents were seeded with indicated that the cliff was bigger than it actually is; it was along the bottom two rows of the grid. The agent receives a reward of -1 at every time step, unless it steps into the cliff which induces a reward of -100 and returns the agent to the start state (without terminating the episode).

Windy-Gridworld [53] is a deterministic domain, shown in Figure C.4 The agent starts in the left middle of the grid, and must navigate towards a goal state on the right-hand side of the grid. However, the is an upward wind within some columns which varies in strength. The wind shifts the next state upwards by the strength of the wind. In our experiments, the model that

the agents were seeded with did not capture the wind. The agent receives a reward of -1 at every time step.

Stochastic Gridworld is the same as Gridworld, except there is some stochasticity introduced; the top door has a probability of 0.4 of being closed, within each episode. In our experiments, the model that the agents were seeded with indicated that the door leading to the goal was closed, with probability 1. The reward formulation and the actions available to the agent remain the same along with the seeded model.

Frozen Lake [9] is a stochastic domain, shown in Figure C.5. The agent must navigate from the start state in the top-left corner to the goal state in the bottom-right corner. However, the frozen lake is slippery, meaning that the agent moves in the intended direction with probability $\frac{1}{3}$ and in either of the perpendicular directions with probability $\frac{1}{3}$ each. Furthermore, there are holes where the ice has been broken or melted and entering these holes terminate the episode. The model that the agents were seeded with did not capture the slipperiness of the frozen lake, and also suggests that there is only a single path to the goal state. Rewards are sparse; reaching the goal state returns a reward of +1, whilst at all other time steps the agent receives no reward.

Stochastic Windy-Gridworld is the same as Windy-Gridworld, except there is stochasticity introduced; the wind (if there is any) is stochastic. The reward formulation and the actions available to the agent remain the same along with the seeded model.

4.3 Results & Analysis

Within this section, we present results of evaluating our implementations against the aforementioned domains. All results were generated by aggregating over 20 runs, where each run produced a sliding window view, with a window size 5. Thus, each point plotted represents the average over the previous 5 episodes averaged over the 20 runs. An outlier to this was the Frozen Lake domain, where a window size of 1 was used. These decisions were made to reduce variance in the results due to randomness, and overall produce a more robust set of results. We also plot the 95% confidence intervals. Throughout all experiments, a discount factor of 1.0 was used, due to the episodic nature of the domains. Moreover, a learning rate of 0.6 was used; this was tuned by-hand and turned out to produce the best results for all agents. For the model-based agents, a maximum of 10 planning steps was allotted.

Figures 4.1 and 4.2 present the learning curves for each agent in each of the deterministic and stochastic domains, respectively. Tables 4.3 and 4.4 provide summaries of these results, including the minimum, mean, maximum and final rewards, as well as the standard deviation.

Gridworld In the Gridworld domain, the agent that performed best on average was RL A* Meta (RLAM), closely followed by RL VI Meta (RLVIM). The poorest performing was the RL agent, which accumulated a fair amount of negative reward in the initial episodes. During the Planning phase, RLAM and RLVIM were each able to discover the optimal policy, however the formed did it much quicker. The PRL and RL VI Meta Learn (RLVIML) agents followed the exact same path throughout the planning phase, and didn't diverge from it; this was because the PRL agent didn't experience any discprencies between its model and the domain and thus didn't replan, whilst the RLVIML agent did not actually learn any Meta Actions to

utilise. Each of the model-based agents suffered considerable performance decreases whilst switching to the learnt Q values. All of the agents were able to discover the optimal policy within the 100 episode limit.

Windy Gridworld In the Windy Gridworld domain, the RLVIM agent outperformed the others on average, although it was closely followed by the RLAM and PRL agents. The RL agent again struggled during its initial exploration, accruing large negative rewards. While the RLAM and PRL agents were able to discover the optimal policy during exploration, the RLVIM agent was not, although it seems to have explored more thoroughly since it suffered less during the switch to the model-free learning. The RLVIML agent was not successful at all; the Meta Actions that it learned somehow led to it becoming stuck in local optima during planning and unable to complete even a single episode. Interestingly, the RLAM and PRL agents were able to achieve a maximum reward of -15.0, which is in-fact optimal, however the agent performing the best in the end was the RL agent; this may be due to insufficient exploration.

Cliff-Walking In the Cliff-Walking domain, the RLVIM agent was once again the best performing on average, by some distance. It seems to have performed the best during the exploratory phase; its attempts at reasoning about the environment led to it discovering the true-cliff, enabling it to avoid it later on, and thus it didn't see too much of a drop in performance when switching to model-free learning. The RL agent again struggled with its initial exploration, with its inital performance being around a factor of a 100 worst than the other agents; this is likely due to dithering, causing the agent to step into the cliff. The RLAM and PRL agents did not perform much exploration, leading to large dips in performance once again. Furthermore, the RLVIML agent wasn't successful in learning any Meta Actions, so it performed very similarly to PRL. By the final episode all agents, but the RL agent, were able to discover the optimal policy.

Stochastic Gridworld In the Stochastic Gridworld, the PRL agent performed the best on average; although it did not perform much exploration in comparison to the other agents. Compellingly, the RLVIM agent did not suffer from the performance dip witnessed in previous domains; perhaps this is because it indeed achieved sufficient exploration. The RL agent once again accumulated considerable negative reward during its initial exploration. The RLVIML and RL A* Meta short term memory (RLAM STM) agents were completely unsuccessful. Interestingly, the successful agents learned a policy that didn't use the door, although it was open more often than not.

Stochastic Windy Gridworld In the Stochastic Windy Gridworld domain, the pure RL agent achieved the best performance, followed closely by the PRL agent. In this domain, the RLVIM agent accumulated more negative reward than the RL agent during their respective initial explorations. The RLVIML and RL A* Meta short term memory (RLAMSTM) agents were completely unsuccessful, again. However, the other agents learned relatively stable policies, and were able to deal with the stochasticity well.

Frozen Lake In the Frozen Lake domain, the PRL agent achieved the best performance, although all of the agents performed generally poor; achieving a maximum of 3.75% success. In this domain, the RLVIML agent outperformed RLVIM by a small margin, this is interesting and indicates that useful Meta Actions were learned. The RLAMSTM agent failed to complete a single episode.

In conclusion, the model-based agents generally outperformed the pure RL agent; and the Meta agent variations tended to outperform the plain PRL agent. For the most part, all of the agents were able to learn good policies with a limited number of episodes. Learning reasonable Meta Actions didn't prove to be very successful, however using embedded Meta Actions was quite successful. The RLAMSTM agent failed in every domain; this may be due to the cost function that we formulated for the agent, and it might've been more successful if we determinised the model and used our deterministic version of A* (see Section 3.6.1). All of the model-based agents typically displayed considerable decreases in performance when switching to model-free learning - this is caused by some states having never been visited during exploration.

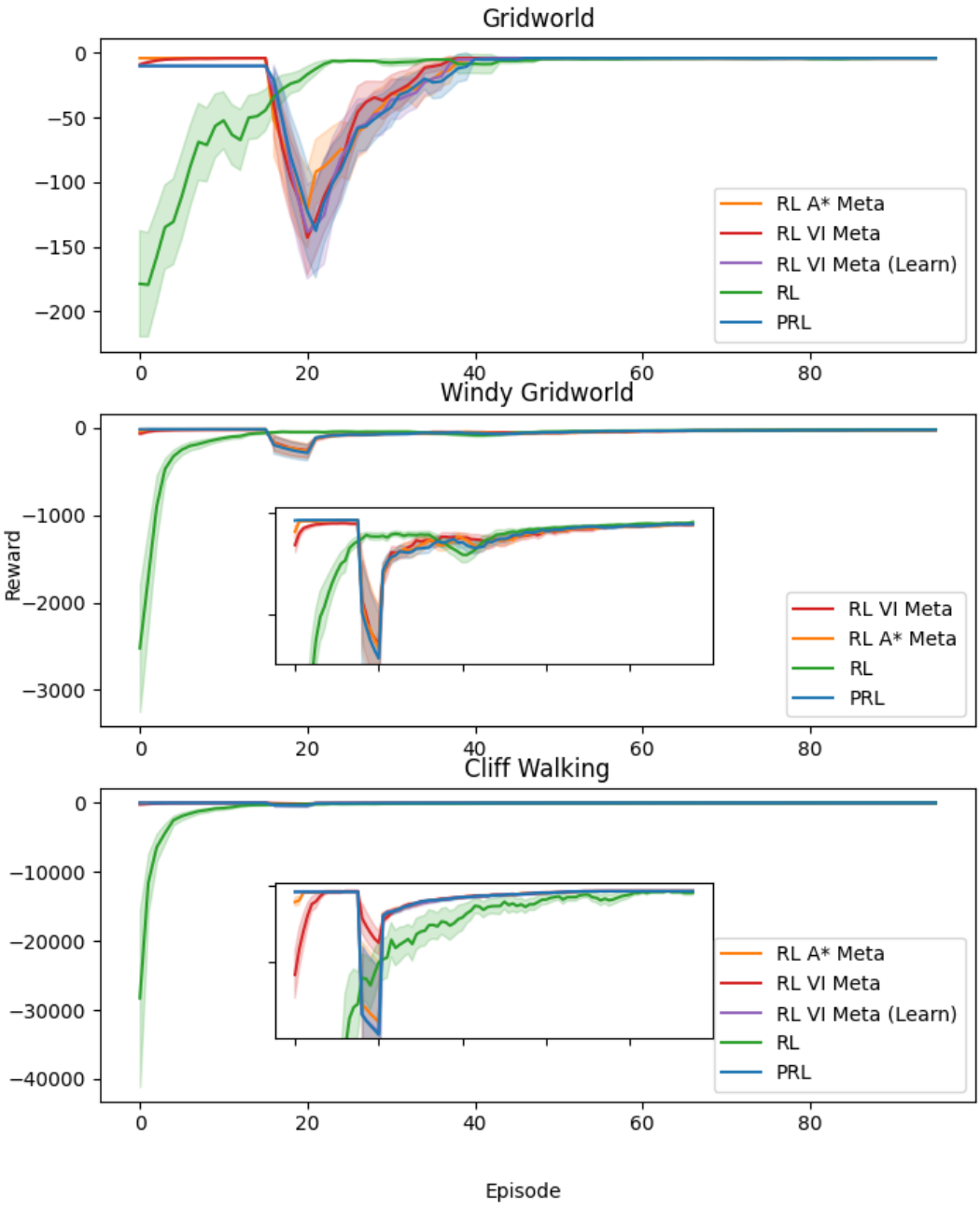


Figure 4.1: Deterministic Results

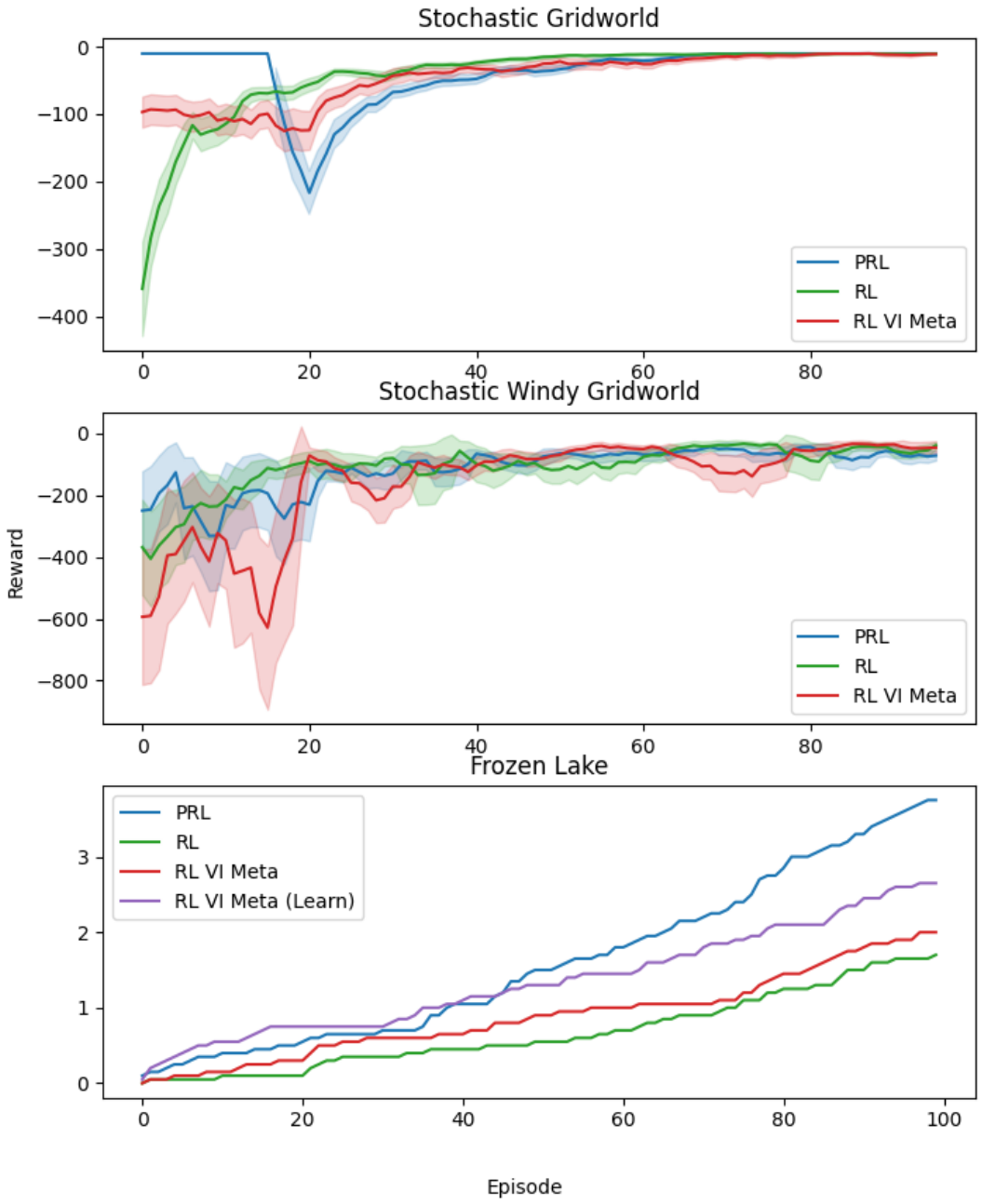


Figure 4.2: Stochastic Results

Agent	Gridworld		Windy Gridworld		Cliff-Walking	
RLVIM	Min	-209.72	Min	-259.3	Min	-231.64
	Max	-10.0	Max	-20.9	Max	-13.0
	Mean	-16.24	Mean	-50.152	Mean	-34.258
	Std. Dev	29.743	Std. Dev	44.944	Std. Dev	37.456
	Final	-4.0	Final	-23.62	Final	-13.0
RLVIM (Learn)	Min	-138.42	Min	DNF	Min	-383.74
	Max	-4.0	Max	DNF	Max	-13.0
	Mean	-18.111	Mean	DNF	Mean	-40.66
	Std. Dev	30.005	Std. Dev	DNF	Std. Dev	76.498
	Final	-4.0	Final	DNF	Final	-13.0
RLAM	Min	-120.58	Min	-267.0,	Min	-357.08
	Max	-4.0	Max	-15.0	Max	-13.0
	Mean	-16.111	Mean	-50.375	Mean	-39.78
	Std. Dev	26.631	Std. Dev	47.226	Std. Dev	70.261
	Final	-4.0	Final	-23.8	Final	-13.0
PRL	Min	-137.58	Min	-286.64	Min	-388.7
	Max	-4.0	Max	-15.0	Max	-13.0
	Mean	-18.017	Mean	-51.018	Mean	-40.635
	Std. Dev	28.838	Std. Dev	51.16	Std. Dev	77.127
	Final	-4.0	Final	-23.16	Final	-13.0
RL	Min	-179.46	Min	-2521.54	Min	-28279.32
	Max	-4.0	Max	-19.16	Max	-13.2
	Mean	-21.012	Mean	-109.037	Mean	-709.696
	Std. Dev	38.013	Std. Dev	317.461	Std. Dev	3170.597
	Final	-4.06	Final	-19.16	Final	-17.54

Table 4.3: Reward Summary for Deterministic Domains

Agent	Stoch. Gridworld		Stoch. Windy Gridworld		Frozen Lake	
RLVIM	Min	-125.0	Min	-628.3	Success Rate	2.0%
	Max	-9.92	Max	-32.9		
	Mean	-44.666	Mean	-155.688		
	Std. Dev	36.761	Std. Dev	153.302		
	Final	-10.92	Final	-45.5		
RLVIM (Learn)	Min	DNF	Min	DNF	Success Rate	2.65%
	Max	DNF	Max	DNF		
	Mean	DNF	Mean	DNF		
	Std. Dev	DNF	Std. Dev	DNF		
	Final	DNF	Final	DNF		
RLAM (STM)	Min	DNF	Min	DNF	Success Rate	DNF
	Max	DNF	Max	DNF		
	Mean	DNF	Mean	DNF		
	Std. Dev	DNF	Std. Dev	DNF		
	Final	DNF	Final	DNF		
PRL	Min	-216.72	Min	-331.2	Success Rate	3.75%
	Max	-10.0	Max	-43.1		
	Mean	-36.766	Mean	-112.641		
	Std. Dev	43.71	Std. Dev	70.109		
	Final	-10.02	Final	-71.1		
RL	Min	-359.3	Min	-405.1	Success Rate	1.7%
	Max	-10.22	Max	-32.4		
	Mean	-42.7	Mean	-107.711		
	Std. Dev	59.999	Std. Dev	76.601		
	Final	-10.22	Final	-38.3		

Table 4.4: Reward Summary for Stochastic Domains

Chapter 5

Discussion

5.1 Conclusion

Exploration is a very important topic in RL, in tasks of interest its particularly important to do exploration well; in terms of learning time and cost. Model-free exploration methods are widely used, although tend to be inefficient. An alternative is model-based exploration which often uses optimism or intrinsic motivation. Optimistic methods tend to be over-optimistic while intrinsically motivated methods rely on expensive computation. Furthermore, in most cases models are learned entirely from scratch and are assumed to eventually become correct.

We proposed an approach to exploration that leverages an initial model, optimism and intrinsic motivation; whilst not relying on the model eventually becoming correct. This approach used Meta Actions, which are the main contribution of this work, which when given to a Planner enable it to hypothesise changes to the model. A framework was developed that utilises a Planner equipped with these Meta Actions to drive exploration; this framework was encapsulated in multiple implementations - the must successful of which was the RL VI Meta agent. With our empirical evaluation we showed that the use of Meta Actions can be useful in exploration, mostly under the condition of embedded reasonability, and allow our framework to overcome model inaccuracies, where it wouldn't be able to do so without; moreover, we showed that leveraging planning to drive exploration is much more sample efficient than ubiquitous model-free exploration methods such as ϵ-greedy.

Meta Actions can be useful during exploration. We hope that this work opens the door for future research to fully understand how we can best use and learn Meta Actions.

5.2 Limitations and Future Work

The limitations of this work are mostly due to assumptions that were made to enable simplifications. Within this section we discuss these assumptions, alongside other limitations, and potential solutions as well as general ideas for future work.

Stochastic Rewards and Bandits We assumed a deterministic reward function. This assumption is one that certainly does not hold in all domains, most notably within Bandit scenarios, where there is a single state and multiple actions [28]. Stochastic Rewards could be considered by extending model-learning to learn a *tabular maximum likelihood model* for the reward function alongside the transition function.

Currently, the Meta Actions available regarding the rewards simply enable the reward to be increased. This could be modified to increase the probability of such rewards. Furthermore, in the Bandit setting, our approach of choosing when to call Meta Actions, by considering which would most benefit the planner, would probably not work; due to the single state nature, every change would most benefit the planner. Therefore in this case, a better way of choosing

when to call Meta Actions needs to be considered - this could be through information theory, such as minimising uncertainty, or perhaps a count-based approach.

Planning A key benefit of using an A* planner is that it is fast, at the cost of having to design a good, admissible, heuristic, and determinising the domain; our approach to stochasticity described in Section 3.6.2 was unsuccessful. In many domains, designing a good heuristic by-hand is very difficult. However recent works, such as [18], have shown that heuristics can be learned directly - which is a potentially interesting extension to this work, which would allow a very simple planner to be used in perhaps a wide range of domains. However, the loss of accuracy due to determinisation could outweigh the value of decreasing computation.

Whilst Value Iteration natuarally deals with stochasticity, is generally slow due to its exhaustive nature, and as state spaces grow it may become very inefficient. Furthermore, as state spaces grow, Value Iteration may become inefficient. Therefore, alternate, faster, planning algorithms, that work under stochasticity, could also be considered such as Upper Confidence Trees (UCT) [25]; which is the UCB algorithm [3] applied to tree search.

Continous State and Action Spaces We assumed discrete domains, or domains that offered discretisation. However, discretisation might not be sufficient; it's difficult to find a balance between coarse and fine grain state and action boundaries that maintain accuracy and efficiency. Thus, it's likely that function approximation based approaches would be a better fit for modelling. However, this introduces new issues; planning in continuous MDPs can be difficult and computationally expensive, cFVI [32] could be a potential option and UCT has also been extended to work in continuous state and action spaces [11].

Partial Observability We assumed fully observable domains which is not always possible, particularly in real-life tasks of interests, such as those the robotics domain. Therefore, our approach could be extended to POMDPs, where planning would take place in belief space, rather than state space. Exactly solving POMDPs is an intractible problem, however various approaches have been suggested for approximate planning in POMDPs, such as POMCP [46] and POMCPOW [48], the latter of which works under continuous belief and action spaces.

Learning Meta Actions The method of learning Meta Actions that we proposed was rather simple and it wasn't vert effective during our experiments. However, we believe that learning Meta Actions is much more powerful than embedding them. An alternative approach to learning Meta Actions could leverage a Neural Network, trained through the replay buffer at the end of each episode, that takes that takes as input a state, and suggests modifications to the transitions and rewards relating to that state.

Definition of Feasibility Whilst the definition of feasibility, given in Section 3.3, is sufficient to prevent infinite hypothesising, through only allowing Meta Actions to be called once on a state-action pair, a stronger emphasis could be put on preventing contradictions to be made (particularly in the stochastic case, where we only consider the observations pertaining to the current episode or previous N episodes). A formidable approach could be to

follow the idea of known and unknown states, present in E^3 [24] and R-MAX [7], only allowing Meta Actions to be called on *unknown* states/state-action pairs; however this would introduce a hyperparameter, m, to define after how many observations a state/state-action pair becomes known.

Task-Agnostic Exploration Our exploration approaches are goal-conditioned and task-specific. A useful line of investigation might be to consider task-agnostic exploration, where we explore the state space independent of any task, and then afterwards use extrinsic reward to adapt to downstream tasks, as in Plan2Explore [42]. This could be done by choosing goals to plan and explore towards through intrinsic motivation, for instance seeking novel states that have high levels of uncertainty associated with them or seeking states with a long planning horizon. This could lead to generalisation to a variety of tasks in a given domain with minimal learning beyond the initial exploratory phase; this would take place in the model-free phase that we outlined within our framework.

Further Benchmarking We restricted our benchmarking to gridworld-like domains. Whilst this enabled us to clearly evaluate and show the usefulness of using Meta Actions, it would have been worthwhile to benchmark our implementations in a wider range of domains, such as classic control tasks; this would've provided us with a more in-depth empirical evaluation, further showing the usefulness of our framework. Moreover, we only considered discrete tasks, it would be interesting to see how our framework performs in continuous domains (discretised, or with the modifications described pertaining to continous state and action spaces). These benchmarks could be done using OpenAI Gym [9] and bsuite domains [35].

Beyond Episodic Tasks We only considered tasks with a finite time horizon associated with them. An interesting further line of work could be to consider tasks that have an infinite time horizon: continuous tasks. Task-agnostic exploration could naturally enable continuous tasks to be learned.

Theoretical Analysis A theoretical analysis of Meta Actions would be very beneficial. In particular, we would like to prove that if there is a model inaccuracy, it will be discovered.

References

[1] Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. A survey of exploration methods in reinforcement learning. *CoRR*, abs/2109.00157, 2021.

[2] Charles William Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, 1986.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[4] Andrew G Barto, Richard S Sutton, and Christopher JCH Watkins. Learning and sequential decision making. *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 539–602, 1990.

[5] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[6] Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[7] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3(null):213–231, mar 2003.

[8] British Computer Society. Code of conduct for bcs members, 2022.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[10] Nicolò Cesa-Bianchi, Claudio Gentile, Gábor Lugosi, and Gergely Neu. Boltzmann exploration done right. *CoRR*, abs/1705.10257, 2017.

[11] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 433–445, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[12] Will Dabney, Georg Ostrovski, and Andre Barreto. Temporally-extended ε-greedy exploration. In *International Conference on Learning Representations*, 2021.

[13] Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 465–472, Madison, WI, USA, 2011. Omnipress.

[14] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth Stanley, and Jeff Clune. First return, then explore. *Nature*, 590:580–586, 02 2021.

[15] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019.

[16] Stefan Edelkamp and Stefan Schrödl. *Heuristic Search: Theory and Applications*. Morgan Kaufmann, Amsterdam, 2011.

[17] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, Amsterdam, 2004.

[18] Ricardo Luna Gutierrez and Matteo Leonetti. Meta-reinforcement learning for heuristic planning. *CoRR*, abs/2107.02603, 2021.

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

[20] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[21] Edward S. Hu, Richard Chang, Oleh Rybkin, and Dinesh Jayaraman. Planning goals for exploration. In *The Eleventh International Conference on Learning Representations*, 2023.

[22] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992.

[23] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[24] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232, 2002.

[25] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, page 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[26] Eugene F Krause. Taxicab geometry. *The Mathematics Teacher*, 66(8):695–706, 1973.

[27] Leonid Kuvayev and Richard S. Sutton. Model-based reinforcement learning with an approximate, learned model. 1996.

[28] Tor Lattimore and Csaba Szepesvari. Bandit algorithms. 2017.

[29] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[30] Matteo Leonetti, Luca Iocchi, and Peter Stone. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence*, 241:103 – 130, September 2016.

[31] Sergey Levine and Vladlen Koltun. Guided policy search. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1–9, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[32] M. Lutter, S. Mannor, J. Peters, D. Fox, and A. Garg. Value iteration in continuous actions, states and time. In *International Conference on Machine Learning (ICML)*, 2021.

[33] Marvin Minsky. Steps toward artificial intelligence. *Proc. IRE*, January 1961.

[34] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[35] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020.

[36] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley amp; Sons, Inc., USA, 1st edition, 1994.

[37] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[38] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[39] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[40] A. L. Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.

[41] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[42] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. *CoRR*, abs/2005.05960, 2020.

[43] Pranav Shyam, Wojciech Jaskowski, and Faustino Gomez. Model-based active exploration. *CoRR*, abs/1810.12162, 2018.

[44] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel,

Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[45] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

[46] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[47] Alexander Strehl and Michael Littman. Online linear regression and its application to model-based reinforcement learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[48] Zachary Sunberg and Mykel J. Kochenderfer. POMCPOW: an online algorithm for pomdps with continuous state, action, and observation spaces. *CoRR*, abs/1709.06196, 2017.

[49] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988.

[50] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.

[51] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, jul 1991.

[52] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael Mozer, and Michael E. Hasselmo, editors, *NIPS*, pages 1038–1044. MIT Press, 1995.

[53] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[54] Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. *CoRR*, abs/1206.3285, 2012.

[55] Richard Stuart Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, 1984. AAI8410337.

[56] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan Claypool Publishers, 2010.

[57] István Szita and András Lőrincz. The many faces of optimism: A unifying approach. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1048–1055, New York, NY, USA, 2008. Association for Computing Machinery.

[58] Sebastian Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[59] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Mach. Learn.*, 8:279–292, 1992.

[60] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[61] Marco Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, 1999.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

Overall, I am very pleased with this project. From the beginning this project was quite ambitious; however, I was able to achieve the goals initially set out at the start of the project.

Although this project was successful, this doesn't by any means imply that it was easy. Having no experience with RL nor Planning meant that I had a lot to learn in a short amount of time. It was sufficient for me to understand Planning at a high-level, however I had to *really* understand RL. Sutton and Barto's "Reinforcement Learning: An Introduction" [53] was a constant companion to me, alongside many other resources and research papers. Over the course of this project, I became obsessed with RL and understanding how and why it works; thus, I developed a strong intuition for RL which was foundational in the success of this project.

My conversations with my supervisors were really insightful, and I often found myself fascinated within our conversations. However, I found that often I struggled to articulate my ideas, which led to us not being on the same page; this is definitely something that I need to improve upon. Similarly, I had difficulties articulating my ideas on paper, therefore writing the report was very time consuming, although I got there in the end.

Coming up with ideas is a challenging and interesting process. The overall idea of the framework and use of Meta Actions was there from the beginning. However, encapsulating that idea involves coming up with various other ideas. I am most pleased with the idea that I came up with for the modified version of Value Iteration; it actually makes sense as a solution, due to the evaluative nature of VI.

In terms of implementing my ideas, I did not struggle, for the most part. Although when I did run into problems, which are inevitable in software development, it could be quite difficult to discover exactly where things were going wrong. Perhaps, if I had more logging-style functionality, it would be easier to trace back unexpected behaviour, and solve issues more quickly. However, I was a bit stubborn and didn't do that.

My project planning and management was definitely my weakest point. I often found myself lacking direction and contemplating what I was actually supposed to be doing. If I planned more thoroughly in the beginning and actually stuck to it, I could have been a lot more successful, and perhaps had more time for further research.

I often found myself going back and improving my ideas. On the one hand this was good, because I was able to identify weaknesses in my approaches and improve them. On the other hand, this was to my detriment. I kept going back and changing things, when I should've been spending more times evaluating my ideas; even when told by my supervisors that I should be doing this, I couldn't help myself.

In conclusion, this project was a success. However, it could have been more successful if I planned better and set strict deadlines for development.

A.2 Personal reflection and lessons learned

On a personal level, I am very pleased with my growth throughout this project. I was able to go from knowing nothing about RL, to being quite knowledgeable about it. Furthermore, I realised that I want to pursue a career where I can continue research into RL.

The biggest lesson that I learned was the importance of project planning. I believe my project could've been much more successful if I maintained a clearer idea of the project formulated through a plan. In future projects, I am certainly going to put more time and effort into planning.

Another big lesson that I learned was that benchmarking should be a continual part of research, not something that should be done at the end. Furthermore, standard benchmarks should be considered rather than developing benchmarks on our own. I learned this the hard way, I introduced my own "biases" when developing my own benchmarks which looking back now having done *proper* benchmarking, led to misconceptions about performance.

Research never ends. This lesson was a hard one to learn. As I mentioned in my critical self-evaluation, I often found myself returning to my algorithms and making improvements, due to my perfectionist nature. However, one needs to understand that research is, potentially, infinite; there will always be some improvement to be made, but they don't all have to be made in the current piece of work, they can be left for future research.

A.3 Legal, social, ethical and professional issues

The legal, social, ethical and professional issues mostly relate to the potential use cases of this project, rather than the project itself. This project did not produce a *complete* framework that can be deployed straight into the real-world, however in this section we will discuss under the assumption that this is a possibility.

A.3.1 Legal issues

In general, the major legal concern relating to AI is liability; who is to blame for a mistake made by an AI? If our framework was used by some company, and it resulted in an accident where someone got hurt, who would be to blame? Would it be us, who developed the framework, the engineer who implemented and trained it, or the company itself? Another legal concern pertains to data. If a company used data (potentially belonging to individuals), then they should have the legal authority to use such data.

Within our project, use of external libraries and figures were referenced, where appropriate, and used with their associated licenses in mind. This ensured use of external libraries and figures was done legally.

A.3.2 Social issues

A commonly discussed issue in the context of AI is that it will "replace humans", leading to unemployment. However, this can be omitted by offering retraining to those individuals who have had their jobs automated in order to ensure that they remain skilled and employable.

A.3.3 Ethical issues

The main ethical concerns with AI are bias and fairness. In the context of RL, bias could be introduced during learning. Bias could be avoided and fairness guaranteed by not only optimising to achieve the maximum cumulative reward, but also to maximise some fairness objective/minimise some bias objective. However, humans are normally responsible for constructing the reward function of an environment, which can ultimately influence the behaviour of any agents that learn through it.

A.3.4 Professional issues

This project was carried out in-line with the British Computing Society Code of Conduct [8]. Most notably, we ensured that we did not misrepresent our proposed methods by presenting false results. Furthermore, we ensured that any external work was attributed correctly, through thorough referencing and providing links to external material in the appendices.

Appendix B

External Material

https://github.com/ibrahim-elshar/gym-windy-gridworlds https://numpy.org/
https://www.gymlibrary.dev/

Appendix C

Domains

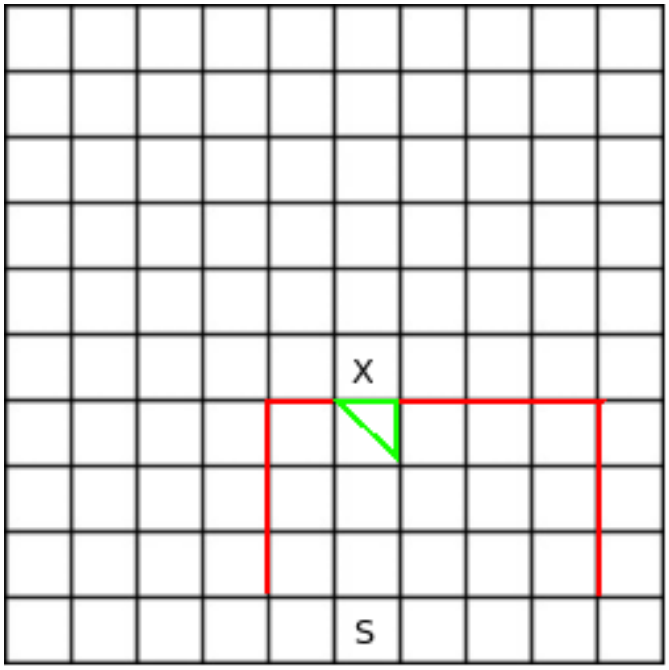


Figure C.1: Gridworld Domain

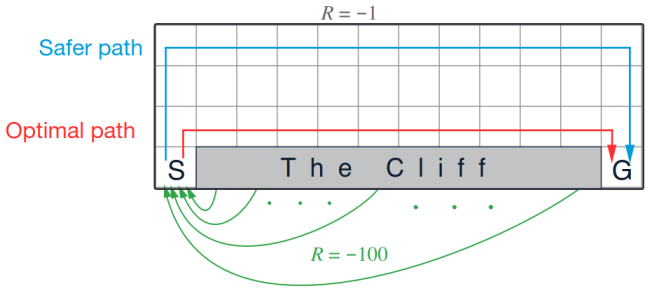


Figure C.2: Cliff-Walking Domain [53]

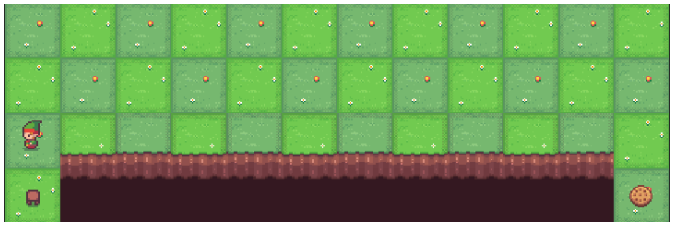


Figure C.3: Cliff-Walking Domain [9]

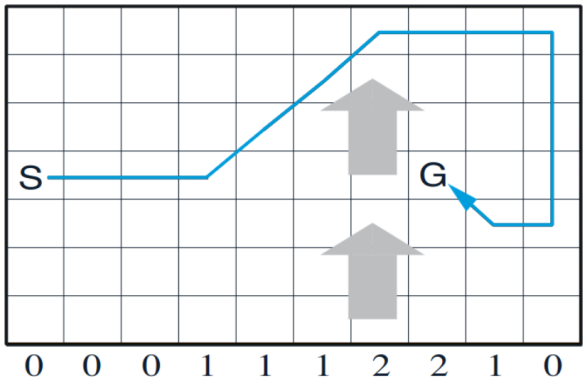


Figure C.4: Windy Gridworld Domain [53]

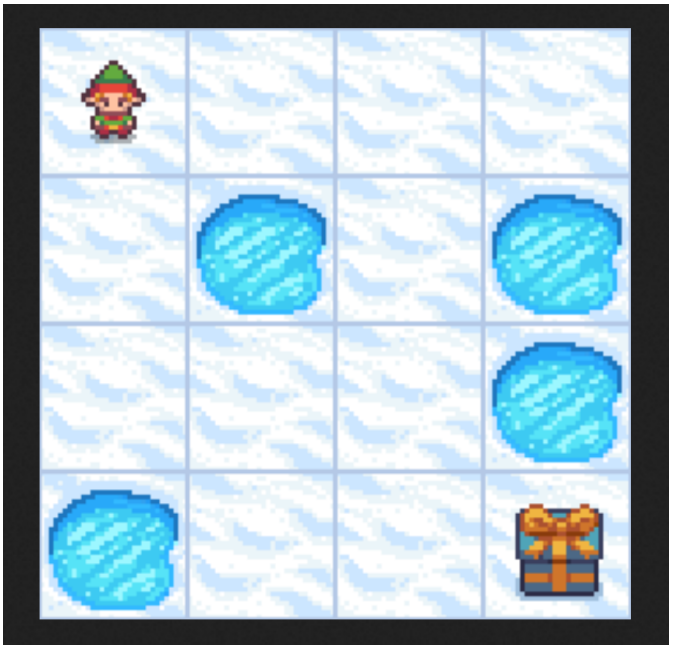


Figure C.5: Frozen Lake Domain [9]