On Monte Carlo Tree Search With Multiple Objectives

Michael Painter

Pembroke College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2024

Acknowledgements

TODO: acknowledgements here

Abstract

TODO: abstract here

Contents

List of Figures

List of Tables

List of Code Listings

List of Abbreviations

CHVI	Convex Hull Value Iteration.
CHVS	Convex Hull Value Set.
CMAB	Contextual Multi-Armed Bandit (problem).
CZ	Contextual Zooming.
EMAB	Exploring Multi-Armed Bandit (problem).
MAB	Multi-Armed Bandit (problem).
MCTS	Monte Carlo Tree Search.
MCTS-1	A specific and common presentation of Monte Carlo Tree Search, presented in Section 2.4.
MDP	Markov Decision Process.
MENTS	Maximum ENtropy Tree Search.
MOMDP	Multi-Objective Markov Decision Process.
THTS	Trial-based Heuristic Tree Search.
`THTS++`	An extension of THTS used in this thesis.
UCB	Upper Confidence Bound (algorithm).
UCT	Upper Confidence Bound applied to Trees.

List of Notation

General Notation

$\mathbb{1}$ . . . . . The indicator function, where $\mathbb{1}(A) = 1$ when A is true, and $\mathbb{1}(A) = 0$ when A is false.

$\hat{A}$ . . . . . The hat notation is used to denote an estimate, i.e. $\hat{A}$ is an estimate for some optimal value A^*

$\bar{A}$ . . . . . The bar notation is used to denote sample averages, i.e. $\bar{A}$ is a sample average of a number of samples $A_1, ..., A_k$, or $\bar{A} = \frac{1}{n}\sum_{i=1}^{k} A_i$.

$\tilde{A}$ . . . . . The tilde notation is used to denote a function approximator, such as a neural network.

$\mathcal{A}$ . . . . . Calligraphic font is used to denote variables that are sets or tuples (ordered sets).

$\mathbf{A}$ . . . . . Bolt font is used to denote vectors, and A_i is used to denote the ith component of the vector $\mathbf{A}$.

$\mathtt{A}$ . . . . . Typewriter text is used to refer to variables relating to an algorithm (or the analysis of an algorithm).

$\boldsymbol{\mathcal{A}}$ . . . . . The notations above will also be combined at times, so $\boldsymbol{\mathcal{A}}$ is used to denote a set that contains vectors.

$\mathbb{E}$ . . . . . The expectation operator. $\mathbb{E}[f]$ denotes the expectation of a distribution f, and $\mathbb{E}_{x\sim f}[g(x)]$ denotes the expected value of $g(x)$ under the distribution f.

$x \sim f$ . . . . . If $f : X \to [0, 1]$ specifies a probability distribution, then $x \sim f$ denotes that the variable x is sampled the distribution f.

(Multi-Objective) Markov Decision Processes (Defined in Sections 2.2 and 2.5)

$\mathcal{A}$ . . . . . A (finite) set of actions.

a_t . . . . . The action at the tth timestep of a trajectory.

D . . . . . : The dimension of rewards in a MOMDP.

H . . . . . The finite-horizon time bound of an MDP.

$\mathcal{M}$ . . . . . A Markov Decision Process, which is a tuple $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, R, p, H)$.

$\boldsymbol{\mathcal{M}}$ . . . . . A Multi-Objective Markov Decision Process, which is a tuple $\boldsymbol{\mathcal{M}} = (\mathcal{S}, s_0, \mathcal{A}, \mathbf{R}, p, H)$.

p . . . . . The next-state transition distribution of an MDP. $p(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$.

π . . . . . : A policy, mapping a state $s \in \mathcal{S}$ to a probability distribution over actions $\mathcal{A}$.

R . . . . . The reward function of an MDP: $R(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

$\mathbf{R}$ . . . . . The D dimensional reward function of a MOMDP: $\mathbf{R}(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^D$.

r_t . . . . . The reward at the tth timestep of a trajectory.

$\mathcal{S}$ . . . . . A (finite) set of states.

$\text{Succ}(s,a)$ . . . . The set of successor states of a state-action pair (s,a), with respect to an MDP: $\text{Succ}(s,a) = \{s' \in \mathcal{S} | p(s'|s,a) > 0\}$.

s_0 . . . . . $s_0 \in \mathcal{S}$ is the initial starting state of an MDP.

s_t . . . . . The state at the tth timestep of a trajectory.

$\texttt{terminal}(s)$ . . Denotes if a state s is terminal or not. That is, if the transition distribution and reward function of the MDP is such that once reached, the state will never be left and no more reward will be received.

τ . . . . . A trajectory, or sequence, of states, actions and rewards that are sampled according to a policy π and an MDP $\mathcal{M}$: $\tau = (s_0, a_0, r_0, s_1, ..., s_{H-1}, a_{H-1}, r_{H-1}, s_H)$.

τ . . . . . A multi-objective trajectory, of states, actions and rewards that are sampled according to a policy π and an MDP $\mathcal{M}$: $\tau = (s_0, a_0, \mathbf{r}_0, s_1, ..., s_{H-1}, a_{H-1}, \mathbf{r}_{H-1}, s_H)$.

$\tau_{i:j}$ . . . . . A truncated trajectory, starting at timestep i, and ending at timestep j: $\tau_{i:j} = (s_i, a_i, r_i, s_{i+1}, ..., s_{j-1}, a_{j-1}, r_{j-1}, s_j)$.

$\tau_{i:j}$ . . . . . A multi-objective truncated trajectory, starting at timestep i and ending at timestep j: $\tau_{i:j} = (s_i, a_i, \mathbf{r}_i, s_{i+1}, ..., s_{j-1}, a_{j-1}, \mathbf{r}_{j-1}, s_j)$.

Reinforcement Learning (Section 2.3)

$J(\pi)$ . . . . . The objective function for (standard) reinforcement learning: $J(\pi) = V^{\pi}(s_0; 0)$.

π^* . . . . . The optimal standard policy, that maximises the objective function $J(\pi)$.

Q^* . . . . . The optimal Q-value function. $Q^*(s, a; t)$ denotes the maximal expected value that can be achieved by any policy, starting from state $s_t = s$, with action $a_t = a$.

Q^{π} . . . . . The Q-value function of a policy π. $Q^{\pi}(s, a; t)$ denotes the expected cumulative reward that policy π will obtain, starting from state $s_t = s$, starting by taking action $a_t = a$.

V^* . . . . . The optimal value function. $V^*(s; t)$ denotes the maximal expected value that can be achieved by any policy, starting from state $s_t = s$.

V^{π} . . . . . The value of a policy π. $V^{\pi}(s; t)$ denotes the expected cumulative reward that policy π will obtain, starting from state $s_t = s$.

Maximum Entropy Reinforcement Learning (Section 2.3.1)

α . . . . . The temperature parameter, or, the coefficient of the entropy term in the maximum entropy (soft) objective.

$\mathcal{H}$ . . . . . The Shannon entropy, of a probability distribution or policy.

$J_{\text{sft}}(\pi)$ . . . . . The objective function for maximum entropy (soft) reinforcement learning: $J_{\text{sft}}(\pi) = V^{\pi}_{\text{sft}}(s_0; 0)$.

π^*_{sft} . . . . . The optimal soft policy, that maximises the soft objective function $J_{\text{sft}}(\pi)$.

Q^*_{sft} . . . . . The optimal soft Q-value function. $Q^*(s, a; t)$ denotes the maximal expected soft value that can be achieved by any policy, from state $s_t = s$ and taking action $a_t = a$.

Q^{π}_{sft} . . . . . The soft Q-value function of a policy π. $Q^{\pi}_{\text{sft}}(s, a; t)$ is the expected value of the policy from $s_t = s$, starting with action $a_t = a$, with an addition of the entropy of the policy weighted by the temperature α.

V^*_{sft} . . . . . The optimal soft value function. $V^*(s; t)$ denotes the maximal expected soft value that can be achieved by any policy, from state $s_t = s$.

V^{π}_{sft} . . . . . The soft value of a policy π. $V^{\pi}_{\text{sft}}(s;t)$ is the expected value of the policy from $s_t = s$, with an addition of the entropy of the policy weighted by the temperature α.

Multi-Objective Reinforcement Learning (Section 2.5)

$CCS(\Pi)$ . . . . A convex coverage set of policies, which contains at least one policy that maximises the linear utility $u_{\text{lin}}(\cdot;\mathbf{w})$ for each weight vector $\mathbf{w}$.

$CCS_{\text{min}}(\Pi)$ . . . The minimal convex coverage set of policies, which has no redundant policies and for each policy in $CCS_{\text{min}}(\Pi)$ there is some weight that it uniquely (with respect to $CCS_{\text{min}}(\Pi)$) obtains the optimal linear utility.

$CH(\Pi)$ . . . . . The undominated set of policies in Π (the set of all possible policies), with respect to the linear utility function u_{lin}. $CH(\Pi) = U(\Pi; u_{\text{lin}})$.

$CS(\Pi;u)$ . . . . A coverage set of policies, which contains at least one policy that maximises the utility $u(\cdot;\mathbf{w})$ for each weight vector $\mathbf{w}$.

`cvx_prune` . . . An operation that takes an arbitrary set of vectors, and returns the subset that lie at the vertices of the geometric (partial) convex hull.

Δ^D . . . . . The D dimensional simplex, or, the set of possible weightings over the D objectives of a MOMDP. $\Delta^D = \{\mathbf{w} \in \mathbb{R}^D | w_i > 0, \sum_i w_i = 1\}$.

Π . . . . . The set of all possible policies in a MOMDP.

$\mathbf{Q}^{\pi}$ . . . . . The multi-objective Q-value function of a policy π. $Q^{\pi}(s,a;t)$ denotes the expected cumulative vector reward that policy π will obtain, starting from state $s_t = s$, starting by taking action $a_t = a$.

$U(\Pi;u)$ . . . . . The undominated set of policies in Π (the set of all possible policies), with respect to the utility function u. Each policy in $U(\Pi;u)$ achieves a maximal utility for some weighting over the objectives.

u . . . . . A utility function, mapping multi-objective values to scalar values, that depends on a weighting over objectives. The multi-objective value $\mathbf{V}^{\pi}(s,a;t)$ is mapped to $u(\mathbf{V}^{\pi}(s,a;t);\mathbf{w})$, where $\mathbf{w}$ is a weighting over the objectives.

u_{lin} . . . . . The linear utility function: $u_{\text{lin}}(\mathbf{v}; \mathbf{w}) = \mathbf{w}^\top \mathbf{v}$.

$\mathbf{V}^\pi$ . . . . . The multi-objective value of a policy π. $V^\pi(s;t)$ denotes the expected cumulative reward that policy π will obtain, starting from state $s_t = s$.

$\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi')$ . . . . The set of multi-objective values obtained by a set of policies Π'.

$\mathbf{w}$ . . . . . A weighting over objectives that quantifies preferences over the multiple objectives, to be used in a utility function.

Trial Based Heuristic Tree Search (Section 2.4)

`backup_v` . . . . Updates the values at a decision node in the backup phase of a trial `THTS++`, using the decision node's children, the trajectory sampled for the trial and the heuristic value function.

`backup_q` . . . . Updates the values at a chance node in the backup phase of a trial `THTS++`, using the chance node's children, the trajectory sampled for the trial and the heuristic value function.

$H_{\texttt{THTS++}}$ . . . . . The planning horizon used in `THTS++`, with $H_{\texttt{THTS++}} \leq H$.

`mcts_mode` . . . Specifies if `THTS++` will sample a trajectory such that only one decision node is added to the search tree per trial. If not running in `mcts_mode` the `THTS++` will sample a trajectory until the planning horizon $H_{\texttt{THTS++}}$.

$N(s)$ . . . . . The number of visits at the decision node corresponding to state s.

$N(s,a)$ . . . . . The number of visits at the chance node corresponding to state-action pair (s,a).

`node`(s) . . . . . The decision node corresponding to the state s.

`node`(s)`.chldrn` The set of chance nodes that are children of `node`(s).

`node`(s)`.V` . . . The set of variables stored at decision node `node`(s), typically used for estimating values.

`node`(s,a) . . . The chance node corresponding to the state-action pair (s,a).

`node`(s,a)`.chldrn` The set of decision nodes that are children of `node`(s,a).

`node`(s,a)`.Q` . . The set of variables stored at decision node `node`(s,a), typically used for estimating Q-values.

π_{search} . . . . . The search policy used in `THTS++` to sample a trajectory in the selection phase.

$\hat{Q}_{\text{init}}$	The heuristic action function used in `THTS++`, used to provide a Q-value estimate for any state-action pairs that aren't in the search tree.
`sample_context`	A function used in `THTS++` that creates a context, or key-value story, and samples any initial values to be stored in the context.
`sample_outcome`	A function used in `THTS++` to sample outcomes (successor states) from the environment (MDP).
$\mathcal{T}$	The `THTS++` search tree. $\mathcal{T} \subseteq \mathcal{S} \cup \mathcal{S} \times \mathcal{A}$.
$\hat{V}_{\text{init}}$	The heuristic value function used in `THTS++`, used to initialise the value of a new decision node.

1 Introduction

Contents

TODO: chapter structure (i.e. in the introduction section I give some background in the field(s), cover the main contributions of this thesis, etc, etc).

1.1 Overview

TODO: list

- Give some context around MCTS (and talk about exploration and exploitation), and why we might use it
  - Larger scale than tabular methods
  - Can do probability and theory stuff (and some explainability, by looking at stats in the tree the agent used)
  - Can use tree search with neural networks to get some of the above (and use for neural network training as in alpha zero)

- Argument from DENTS paper for exploration > exploitation (in context of planning in a simulator)
- Give high level overview of Multi-Objective RL, and why it can be useful
- Give an idea of how my work fits into MCTS and MORL as a whole
- Discuss research questions/issues with current literature (i.e. introduce some of the ideas from contributions section below)

1.2 Contributions

TODO: Inline acronyms used, or make sure that they're defined before hand

Throughout this thesis, we will consider the following questions related to Monte Carlo Tree Search and Multi-Objective Reinforcement Learning:

Q1 - Exploration: When planning in a simulator with limited time, how can MCTS algorithms best explore to make good decisions?

Q1.1 - Entropy: Entropy is often used as an exploration objective in RL, but can it be used soundly in MCTS?

Q1.2 - Multi-Objective Exploration: How can Multi-Objective MCTS methods explore to find optimal actions for different objectives?

Q2 - Scalability: How can the scalability of (multi-objective) MCTS methods be improved?

Q2.1 - Complexity: MCTS algorithms typically run in $O(nAH)$, but are there algorithms that can improve upon this?

Q2.2 - Multi-Objective Scalability: With respect to the size of environments, how scalable are Multi-Objective MCTS methods?

Q2.3 - Curse of Dimensionality: With respect to the number of objectives, to what extent do Multi-Objective MCTS methods suffer from the curse of dimensionality?

Q3 - Evaluation: How can we best evaluate a search tree produced by a Monte Carlo Tree Search algorithm?

Q3.1 - Tree Policies: Does it suffice to extract a policy from a single search tree for evaluation? TODO: going to have to run some extra experiments for that, but I probably should do that for completeness anyway

Q3.2 - Multi-Objective Evaluation: Can we apply methods from the MORL literature to theoretically and empirically evaluate Multi-Objective MCTS?

TODO: some words about how below is the contributions we're making in this thesis and expand these bullets a bit more

- Max Entropy can be misaligned with reward maximisation (**Q1.1 - Entropy**)
- Boltzmann Search Policies - BTS and DENTS (**Q1.1 - Entropy**, and with extra results **Q3.1 - Tree Policies**)
- Use the alias method to make faster algorithms (**Q2.1 - Complexity**)
- Simple regret (**Q1 - Exploration**)
- Use of contexts in THTS to make consistent decisions in each trial (**Q1.2 - Multi-Objective Exploration**)
- Contextual regret introduced in CHMCTS (**Q2.2 - Multi-Objective Scalability**, **Q3.2 - Multi-Objective Evaluation**)
- Contextual Zooming and CHMCTS (designed for **Q1.2 - Multi-Objective Exploration**, runtimes cover **Q2.3 - Curse of Dimensionality**, results **Q3.2 - Multi-Objective Evaluation**)
- Simplex maps (**Q1.2 - Multi-Objective Exploration**, **Q2.2 - Multi-Objective Scalability**, **Q2.3 - Curse of Dimensionality**)
- Contextual Simple Regret (**Q3.2 - Multi-Objective Evaluation**)

1.3 Structure of Thesis

TODO: a paragraph with a couple lines to a paragraph about each chapter. This is the high level overview/intro to the thesis paragraph. I.e. this section is "this is the story of my thesis in a page or two"

1.4 Publications

TODO: update final publication when submit

The work covered in this thesis also appears in the following publications:

- Painter, M; Lacerda, B; and Hawes, N. "Convex Hull Monte-Carlo Tree-Search." In *Proceedings of the international conference on automated planning and scheduling. Vol. 30. 2020*, ICAPS, 2020.
- Painter, M; Baioumy, M; Hawes, N; and Lacerda, B. "Monte Carlo Tree Search With Boltzmann Exploration." In *Advances in Neural Information Processing Systems, 36, 2023*, NeurIPS, 2023.
- Painter, M; Hawes, N; and Lacerda, B. "Simplex Maps for Multi-Objective Monte Carlo Tree Search." In *TODO*, **Under Review at conf_name**.

2
Background

Contents

This chapter introduces the fundamental concepts that will be used throughout this thesis, in particular the `THTS++` schema is introduced in Section 2.4.2, in which monte carlo tree search algorithms will be defined. Section 2.1 begins with multi-armed bandit problems and decision theory, which provides a fundamental building block used in monte carlo tree search methods (Section 2.4). Sections 2.2 and 2.3 provide a framework for sequential decision making under uncertainty.

Section 2.4 introduces monte carlo tree search methods methods that can be used for solving problems in sequential decision making under uncertainty. Section 2.5 extends the frameworks to include multiple objectives. And Section 2.6 discusses sampling from categorical distributions efficiently.

2.1 Multi-Armed Bandits

This section introduces the K-armed bandit problem [22], which is a foundational problem considered in decision theory. In the multi-armed bandit (MAB) problem, an agent is tasked with deciding to pull one of K arms, and the decision typically needs to be made multiple times. For example, the MAB problem could be used in the context of clinical trials, where each "arm" corresponds to a treatment option. As such, the MAB problem is presented in rounds, where an arm is selected each round. In each round an random outcome is observed in form of a reward.

The distribution of rewards is unknown ahead of time, and so an agent needs to explore to gather information about what rewards can be obtained by pulling each arm, and if an agent does not explore then it may miss out on discovering the best strategy. Conversely, the agent should want to exploit, using the information that it has obtained, so that it gather high rewards. In the clinical trials example, exploitation is desirable so that patients receive the best treatment. The problem of balancing these two aspects is known as the *exploration-exploitation trade off*.

The remainder of this section will formalise the MAB problem, and give details of the widely used Upper Confidence Bound (UCB) algorithm [2]. Sections 2.1.1 and 2.1.2 considers two variations on the MAB problem that will be relevant in this thesis.

Figure 2.1 shows how the operation of a K-armed bandit problem proceeds. Formally, let $f(1), ..., f(K)$ be the probability distributions for the rewards of each of the K arms, with expected values of $\mu(1), ..., \mu(K)$ respectively. On the mth round, if the arm $x^m \in \{1, ..., K\}$ is pulled, then a reward $y^m \sim f(x^m)$ is received. To specify a *strategy* σ, on each round m, a probability distribution σ^m is over $\{1, ..., K\}$ is given, which is sampled to decide which arm to pull, i.e. $x^m \sim \sigma^m$.

Parameters: K probability distributions for the rewards of each arm $f(1), ..., f(K)$.

- For each round $m = 1, 2, ...$:
  - the agent selects an arm $x^m \sim \sigma^m$ to pull;
  - the agent receives a reward $y^m \sim f(x^m)$ from the environment;
  - if the environment sends a stop signal, then the game ends, otherwise the next round starts.

Figure 2.1: The procedure of a multi-armed bandit problem, following a strategy σ.

To assess algorithmic strategies for MAB problems, a quantity known as *regret* is commonly used, which compares the cumulative reward obtained, compared to the maximal expected reward that could be obtained with full knowledge of $\{f(i)\}$.

Definition 2.1.1. *The* (cumulative) regret *of strategy σ after n rounds in the MAB process is:*

$$\texttt{cum\_regr}_{\text{MAB}}(n, \sigma) = n\mu^* - \sum_{m=1}^{n} y^m, \tag{2.1}$$

where $\mu^* = max_i \mu(i)$.

To theoretically analyse algorithms for MAB problems, the quantity of expected regret, $\mathbb{E}[\texttt{cum\_regr}_{\text{MAB}}(n, \sigma)]$ is considered. Lai and Robbins [18] show using information theory that there is a lower bound on the expected regret that an agent can achieve of $\Omega(\log n)$. And Auer [2] introduces the Upper Confidence Bound (UCB) algorithm, which achieves a matching upper bound of $O(\log n)$ on the expected regret.

To define the UCB strategy for pulling the arms, a few quantities need to be defined first. Let $N^m(x)$ be the number of times that arm x has been pulled after m rounds. And let $\bar{y}^m(x)$ be the average reward that has been received as a result of pulling arm x after m rounds. Mathematically:

$$N^m(x) = \sum_{i=1}^{m} \mathbb{1}[x^i = x], \tag{2.2}$$

$$\bar{y}^m(x) = \frac{1}{N^m(x)} \sum_{i=1}^{m} y^i \mathbb{1}[x^i = x]. \tag{2.3}$$

The strategy followed by the UCB algorithm on the mth round is given by:

$$x^m_{\text{UCB}} = \arg\max_{i \in \{1,..,K\}} \bar{y}^{m-1}(i) + b_{\text{UCB}} \sqrt{\frac{\log(m)}{N^{m-1}(i)}}, \tag{2.4}$$

$$\sigma^m_{\text{UCB}}(x) = \mathbb{1}[x = x^m_{\text{UCB}}], \tag{2.5}$$

where b_{UCB} is a bias parameter used to control how much the strategy explores. Additionally, each arm is pulled once in the first K rounds by the UCB strategy, to avoid division by zero in Equation 2.4. Alternatively, the division by zero can be considered to give a value of ∞, to give the same effect.

2.1.1 Exploring Bandits

In the pure exploration problem for K-armed bandits [7], the format of each round is changed slightly. The agent still gets to pull an arm each round (according to the *exploration strategy* σ), but after it recieves a reward on each round it is given the opportunity to output a *recommendation strategy* ψ. On the mth round, the recommendation strategy takes the form of ψ^m, a probability distribution over the K arms. In exploring multi-armed bandit (EMAB) problems, the emphasis is now on the algorithm being able to provide the best recommendations possible, rather than trying to exploit pulling the best arm each round. In essence, this separates the needs to explore and exploit, the agent needs to explore with its arm pulls, and output an exploiting recommendation at the end of each round. Figure 2.2 gives an overview of the EMAB problem.

The *(expected) instantaneous regret* for pulling an arm $x \in \{1, ..., K\}$ is given by:

$$\texttt{inst\_regr}(x) = \mu^* - \mu(x), \tag{2.6}$$

where again $\mu^* = max_i \mu(i)$.

Under the exploring regime of EMABs, the performance of an algorithm can be analysed by considering the quantity of *simple regret* of the recommendation policy. The simple regret is the expected value of the instantaneous regret which would come from following the recommendation policy.

Parameters: K probability distributions for the rewards of each arm $f(1), ..., f(K)$.

- For each round $m = 1, 2, ...$:
  - the agent selects an arm $x^m \sim \sigma^m$ to pull;
  - the agent receives a reward $y^m \sim f(x^m)$ from the environment;
  - the agent outputs a recommendation distribution ψ^m, over the arms $\{1, ..., K\}$;
  - if the environment sends a stop signal, then the game ends, otherwise the next round starts.

Figure 2.2: The procedure of an exploring multi-armed bandit problem, following an exploration strategy σ, and recommendation strategy ψ.

Definition 2.1.2. *The* simple regret *of following the exploration strategy σ and recommending a distribution ψ^m on the mth round is:*

$$\texttt{sim\_regr}_{\text{EMAB}}(m, \sigma, \psi^m) = \mathbb{E}_{i \sim \psi^m}[\mu^* - \mu(i)]. \tag{2.7}$$

Bubeck et al. [7] analyse a range of exploration and recommendation strategies in their work. Two of the recommendation strategies considered are the *empirical best arm* (EBA) and *most played arm* (MPA):

$$x^m_{\text{EBA}} = \arg\max_{i \in \{1,..,K\}} \bar{y}^m(i), \tag{2.8}$$

$$\psi^m_{\text{EBA}}(x) = \mathbb{1}[x = x^m_{\text{EBA}}], \tag{2.9}$$

$$x^m_{\text{MPA}} = \arg\max_{i \in \{1,..,K\}} N^m(i), \tag{2.10}$$

$$\psi^m_{\text{MPA}}(x) = \mathbb{1}[x = x^m_{\text{MPA}}]. \tag{2.11}$$

In addition to the UCB strategy, a *uniform exploration strategy* is considered:

$$x^m_{\text{uniform}} = (m \mod K) + 1, \tag{2.12}$$

$$\sigma^m_{\text{uniform}}(x) = \mathbb{1}[x = x^m_{\text{uniform}}]. \tag{2.13}$$

It is shown that exploring with UCB leads to a polynomial simple regret bound for both recommendation strategies: $\texttt{sim\_regr}_{\text{EMAB}}(m, \sigma_{\text{UCB}}, \psi^m_{\text{EBA}}) = O(m^{-k_{\text{UCB,EBA}}})$

Parameters: the set of arms $\mathcal{X}$, the set of contexts $\mathcal{W}$ and a mapping from $\mathcal{W} \times \mathcal{X}$ to probability distributions: $f(w, x)$.

- For each round $m = 1, 2, ...$:
  - the agent recieves a context $w^m \in \mathcal{W}$ from the environment;
  - the agent selects an arm $x^m \sim \sigma^m(w^m)$ to pull;
  - the agent receives a reward $y^m \sim f(w^m, x^m)$ from the environment;
  - if the environment sends a stop signal, then the game ends, otherwise the next round starts.

Figure 2.3: The procedure of a contextual multi-armed bandit problem, following a strategy σ.

and $\texttt{sim\_regr}_{\text{EMAB}}(m, \sigma_{\text{UCB}}, \psi^m_{\text{MPA}}) = O(m^{-k_{\text{UCB,MPA}}})$ for some appropriate constants $k_{\text{UCB,EBA}}, k_{\text{UCB,MPA}} > 0$. It is also shown that exploring with the uniform strategy (and recommendating the empirical best arm) leads to an exponential simple regret bound, $\texttt{sim\_regr}_{\text{EMAB}}(m, \sigma_{\text{UCB}}, \psi^m_{\text{EBA}}) = O(e^{-k_{\text{uniform}} m})$ for an appropriate constant $k_{\text{uniform}} > 0$.

2.1.2 Contextual Bandits

In the contextual multi-armed bandit (CMAB) problem [15, 28], before an arm is chosen, the agent is provided with a context w. In the CMAB problem the set of arms, now $\mathcal{X}$, and the set of contexts, $\mathcal{W}$ are compact sets (i.e. the sets are closed and bounded). The distribution of rewards now also depends on the context w and arm pulled x, and is written $f(w, x)$, with mean $\mu(w, x)$. The distribution of arms pulled on the mth round can now depend on the context: $\sigma^m(w)$. Figure 2.3 gives an overview of the CMAB problem.

Given this setup some further assumptions are required for the problem to be more tractable. Slivkins [28] make a fairly general assumption that some *metric* d over the *similarity space* $\mathcal{W} \times \mathcal{X}$ is known and is such that the following Lipschitz condition holds:

$$|\mu(x, y) - \mu(x', y')| \leq d((x, y), (x', y')). \tag{2.14}$$

For d to be a metric, the following conditions must hold for some arbitrary $z = (x, y), z' = (x', y'), z'' = (x'', y'')$ with $z \neq z' \neq z''$:

$$d(z, z) = 0, \tag{2.15}$$

$$d(z, z') > 0, \tag{2.16}$$

$$d(z, z') = d(z', z), \tag{2.17}$$

$$d(z, z'') \leq d(z, z') + d(z', z''). \tag{2.18}$$

Similar to MABs and EMABs, the notion of regret is used to analyse CMABs. Specifically, *contextual regret* is defined similarly to cumulative regret [15, 28], while taking into account the contexts drawn.

Definition 2.1.3. *The* (cumulative) contextual regret *of the strategy σ in the process given in Figure 2.3 is defined as:*

$$\texttt{ctx\_regr}_{\text{CMAB}}(\pi, n) = \sum_{m=1}^{n} \mu^*(w^m) - y^m, \tag{2.19}$$

where $\mu^*(w) = \sup_i \mu(w, i)$.

Slivkins [28] also introduces the Contextual Zooming (CZ) algorithm. CZ runs over a fixed number of rounds T, and achieves the contextual regret of $O(T^{\frac{1+c}{2+c}} \log(T))$, where c is the *(Lebesgue)covering dimension* of the similarity space $\mathcal{W} \times \mathcal{X}$. For the purpose of this thesis the covering dimension (a notion about topological spaces), will always be equal to the typical notion of dimension. That is, if $\mathcal{W} \times \mathcal{X}$ has covering dimension c, then $\mathcal{W} \times \mathcal{X}$ can be mapped to a subspace of $\mathbb{R}^c$.

Throughout the CZ algorithm, a set of balls in the similarity space is maintained, called *active balls*. Let A^m denote the set of active balls at the start of the mth round. A ball with center (w, x) and radius r is given by $\{(w', x') \in \mathcal{W} \times \mathcal{X} \mid d((w, x), (w', x')) < r\}$.

The CZ algorithm operates using two rules each round m: the *selection rule* is used to select a relevant ball B^m from the set of active balls A^m; then, an *activation rule* is optionally applied that adds a new ball with smaller radius.

Now the selection rule is described in more detail. Firstly, let $N^m(B)$ be the number of times that a ball $B \in A^m$ has been selected in the first m rounds, and $\bar{y}^m(B)$ the average reward received after m rounds when selecting ball B.

$$N^m(B) = \sum_{i=1}^{m} \mathbb{1}[B = B^m], \tag{2.20}$$

$$\bar{y}^m(B) = \frac{1}{N^m(B)} \sum_{i=1}^{m} y^m \mathbb{1}[B = B^m]. \tag{2.21}$$

Let $r(B)$ denote the radius of ball B and let $\texttt{dom}^m(B)$ be the domain of ball B on the mth round, which is the subset of B that excludes all balls of smaller radius, or:

$$\texttt{dom}^m(B) = B - \left(\bigcup_{B' \in A^m : r(B') < r(B)} B' \right). \tag{2.22}$$

The *confidence radius* $\texttt{conf}^m(B)$ of ball B on the mth round is:

$$\texttt{conf}^m(B) = 4\sqrt{\frac{\log T}{1 + N^{m-1}(B)}}. \tag{2.23}$$

Let $\texttt{relevant}^m(w, A^m)$ denote the set of balls that are relevant for context w on the mth round, which is the set of balls that contain an arm x such that (w, x) is in the domain of the ball:

$$\texttt{relevant}^m(w, A^m) = \{B \in A^m | \exists x \in \mathcal{X}.(w, x) \in \texttt{dom}^m(B)\}. \tag{2.24}$$

The ball B^m selected on the mth round with context w^m can now be given as:

$$I^m(B) = \bar{y}^{m-1}(B) + r(B) + \texttt{conf}^m(B), \tag{2.25}$$

$$B^m = \max_{B \in \texttt{relevant}^m(w^m, A^m)} r(B) + \min_{B' \in A^m} \left(I^m(B') + d(B, B') \right), \tag{2.26}$$

where $d(B, B')$ is the distance between the centers of balls B, B' according to the metric d.

Once a ball has been selected, the arm sampled by CZ, x^m_{CZ}, is sampled randomly from the domain of ball B^m (from the set $\{x \in \mathcal{X} | (w^m, x) \in \texttt{dom}^m(B^m)\}$), which concludes the selection rule.

The activation rule is then used if the confidence radius from Equation (2.23) has shrunk to smaller than the radius of ball selected this round, i.e. if $\texttt{conf}^{m+1}(B^m) \leq r(B^m) < \texttt{conf}^m(B^m)$. In this case, a new ball B' is added to the set of active balls, with radius $r(B') = \frac{1}{2} r(B^m)$ and center (w^m, x^m). If a new ball is added, then $A^{m+1} = A^m \cup \{B'\}$ and otherwise $A^{m+1} = A^m$.

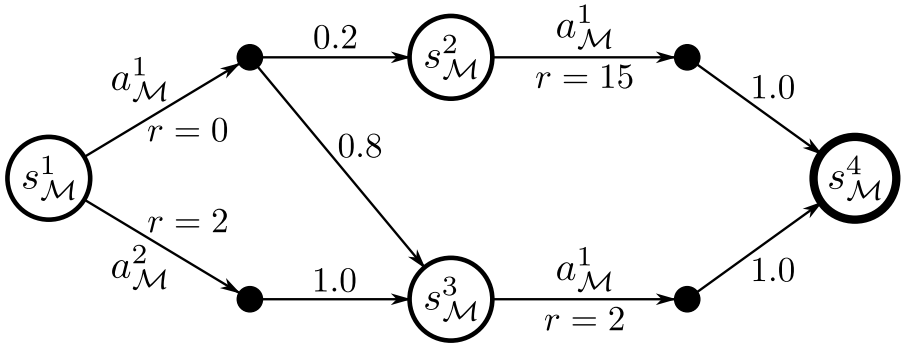

Figure 2.4: An example MDP $\mathcal{M}$, where $s_0 = s^1_{\mathcal{M}}$, the finite horizon is $H = 2$, and terminal states are marked with a thicker border. Edges are appropriately marked with actions and rewards, or marked with transition probabilities.

2.2 Markov Decision Processes

In this section *Markov decision processes* (MDPs) are introduced, along with related definitions of *policies* and *trajectories*. MDPs give a mathematical framework for problems concerning sequential decision making under uncertainty, and in this thesis will be the framework used to model the environment that agents act in. An MDP contains, among other things, a set of states and actions. States are sampled according to a transition distribution which depends on the current state and current action being taken (the Markov assumption). Any time an action is taken from a state the agent receives an instantaneous reward, according to a reward function that depends on the state and action taken.

This thesis is concerned with discrete, finite, fully-observable and finite-horizon Markov decision processes, meaning that the state and action spaces are discrete and finite, and any *trajectories* (sequences of states, actions and rewards) are of a finite length.

Definition 2.2.1. *A* Markov decision process *(MDP) is a tuple* $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, R, p, H)$*, where* $\mathcal{S}$ *is a set of states,* $s_0 \in \mathcal{S}$ *is an initial state,* $\mathcal{A}$ *is a set of actions,* $R(s, a)$ *is a reward function* $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$*,* $p(s'|s, a)$ *is a next state transition distribution* $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *and* $H \in \mathbb{N}$ *is a finite-horizon time bound.*

An example MDP is shown in Figure 2.4. Notationally, it is convenient to define the set of successor states, that is the set of states that could be reached after

taking an action from the current state of the MDP:

Definition 2.2.2. *The set of* successor states $\mathrm{Succ}(s, a)$ *of a state-action pair* (s, a)*, with respect to an MDP, is defined as:*

$$\mathrm{Succ}(s, a) := \{s' \in \mathcal{S} | p(s'|s, a) > 0\}. \tag{2.27}$$

Additionally, throughout this thesis, MDPs will be defined with *terminal states*, which are states that once reached will never be left, and no more reward can be obtained.

Definition 2.2.3. *A state* $s \in \mathcal{S}$ *is a* terminal *state (or a* trap *state), if the transition distribution always returns the same state (i.e.* $p(s|s, a) = 1$ *for all actions* a*), and if no more reward is obtained from that state (*$R(s, a) = 0$ *for all* a*). The function* `terminal`(s) *returns a boolean in* $\{\mathrm{True}, \mathrm{False}\}$ *which will be used to denote if a state* s *is terminal or not.*

To define a strategy that an agent will follow in an MDP, and agent defines a *policy*. A policy maps each state in the state space to a probability distribution over the action space. To "follow" a policy, actions are sampled from the distribution. Often it is desirable to define deterministic policies, which always produce the same action when given the same state, and can be represented as *one-hot* distributions.

Definition 2.2.4. *A* (stochastic) policy $\pi : \mathcal{S} \to (\mathcal{A} \to [0, 1])$ *is a mapping from states to a probability distributions over actions and* $\pi(a|s)$ *is the probability of sampling action* a *at state* s*. The policy* π *must satisfy the conditions: for all* $s \in \mathcal{S}$ *we have* $\sum_{a \in \mathcal{A}} \pi(a|s) = 1$ *and for all actions* $a \in \mathcal{A}$ *that* $\pi(a|s) \geq 0$ *.*

Additionally, a deterministic policy *is defined as a one-hot policy, that is, the policy* π *is deterministic iff it can be written as* $\pi(a|s) = \mathbb{1}[a = a']$ *for some* $a' \in \mathcal{A}$*.*

Moreover, the following notations are used for policies:

- $a \sim \pi(\cdot|s)$ *denotes sampling an action* a *from the distribution* $\pi(\cdot|s)$*;*
- $\pi(s) = a'$ *is used as a shorthand to define the deterministic policy* $\pi(a|s) = \mathbb{1}[a = a']$*;*

- *$\pi(s)$ is used as a shorthand for the action $a' \sim \pi(\cdot|s)$ in the case of a deterministic policy.*

Given an MDP $\mathcal{M}$ and a policy π it is then possible to sample a sequence of states, actions and rewards, known as a *trajectory*. A trajectory *simulates* one possible sequence that could occur if an agent follows policy π in $\mathcal{M}$, and in Section 2.4 these simulations are used to incrementally build a search tree.

Definition 2.2.5. *A* trajectory *τ, is a sequence of state, action and rewards, that is induced by a policy π and MDP $\mathcal{M}$ pair. Let the trajectory be $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{H-1}, a_{H-1}, r_{H-1}, s_H)$, where $a_t \sim \pi(\cdot|s_t)$, $r_t = R(s_t, a_t)$ and $s_{t+1} \sim p(\cdot|s_t, a_t)$.*

The following notations will also be used for trajectories:

- *$\tau \sim \pi$ denotes a trajectory that is sampled using the policy π, where the MDP $\mathcal{M}$ is implicit;*
- *$\tau_{i:j}$ denotes the* truncated trajectory *$\tau_{i:j} := (s_i, a_i, r_i, s_{i+1}, ..., s_{j-1}, a_{j-1}, r_{j-1}, s_j)$, between the timesteps $0 \leq i < j \leq H$ inclusive;*
- *$\tau_{:j} := \tau_{0:j}$ denotes a trajectory that is trunacted on only one end,*
- *given a trajectory τ, the following set notation is used, $s \in \tau$, $(s, a) \in \tau$ as a shorthand for $s \in \{s_i | i = 0, ..., H\}$ and $(s, a) \in \{(s_i, a_i) | i = 0, ..., H-1\}$ respectively,*

and finally, when states, actions and rewards a timestep indexed (i.e. s_t, a_t, r_t), they will always correspond to a trajectory.

2.3 Reinforcement Learning

This section serves as a brief introduction to fundamental concepts in reinforcement learning, and motivates an alternative lens on reinforcement learning that this thesis will consider. The field of reinforcement learning considers an agent that has to learn how to make decisions by interacting with its environment (Figure 2.5).

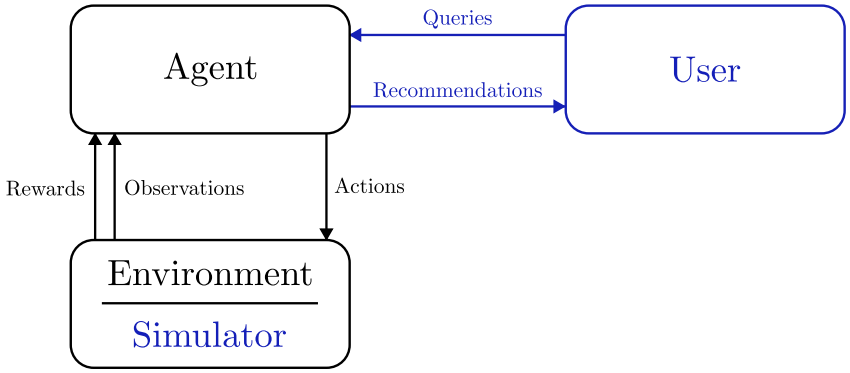

Figure 2.5: An overview of reinforcement learning. In black depicts the standard agent-environment interface for reinforcement learning [29], where an agent can perform actions in an environment and is given feedback in the form of observations and rewards. In blue the environment is replaced by a simulated environment which the agent can use for planning, and is queried by a user for recommendations on how to act in the real environment.

The agent can take actions in the environment, recieving in return *observations* and *rewards*, which can be used to update internal state and used to make further decisions, and the goal of the agent is to maximise the rewards that it recieves.

Classically the agent is considered to interact with its environment directly [29], and thus must make a trade-off between exploring new strategies and exploiting learned strategies. That is, reinforcement learning agents must consider the *exploration-exploitation trade-off* discussed in Section 2.1 too.

Also depicted in Figure 2.5 is a scenario where the agent is equiped with a simulator that it can use to plan and explore, and is either asked to recommend a strategy after a planning/learning phase, or is occassionally queried to recommend actions. This scenario more closely resembles how reinforcement learning is used in the modern era with greater amounts of compute power available, and interactions with the simulator occur at orders of magnitude quicker. Hence, in this scenario, the only significant real-world cost comes from following the recommendations output, to be used in the real-world environment. This changes the nature of the exploration-exploitation trade off, almost separating the two issues, where there is an emphasis on exploring during the planning phase, and the problem of providing good recommendations is concerned with pure exploitation. This

perspective on reinforcement learning motivates the research quesitons around exploration: **Q1 - Exploration**.

In this thesis, the environment will always take the form of an MDP (Defintion 2.2.1), and observations will always be *fully-observable*, meaning that the agent is provided with full access to the states of the MDP.

Following on from Section 2.2, the remainder of this section defines *value functions* and the objectives of reinforcement learning and covers *Value Iteration*, a tabular dynamic programming approach to reinforcement learning. Finally Section 2.3.1 covers *maximum-entropy reinforcement learning*.

The value of a policy π is the expected cumulative reward that will be obtained by following the policy:

Definition 2.3.1. *The* value *of a policy π from state s at time t is:*

$$V^{\pi}(s;t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=t}^{H-1} r_t \middle| s_t = s\right]. \tag{2.28}$$

The Q-value *of a policy π, from state s, with action a, at time t is:*

$$Q^{\pi}(s,a;t) = R(s,a) + \mathbb{E}_{s'\sim p(\cdot|s,a)}[V^{\pi}(s';t+1)]. \tag{2.29}$$

From the definition of the values functions the optimal value functions can be defined by taking the maximum value over all policies:

Definition 2.3.2. *The* Optimal (Q-)Value *of a state(-action pair) is defined as:*

$$V^{*}(s;t) = \max_{\pi} V^{\pi}(s;t) \tag{2.30}$$

$$Q^{*}(s,a;t) = \max_{\pi} Q^{\pi}(s,a;t). \tag{2.31}$$

Value functions can also be used to define an objective function:

Definition 2.3.3. *The* (standard) reinforcement learning objective function $J(\pi)$ *is defined as:*

$$J(\pi) = V^{\pi}(s_0;0). \tag{2.32}$$

The objective of (standard) reinforcement learning can then be stated as finding $\max_{\pi} J(\pi)$.

The *optimal policy* is the policy that maximises the objective function J, and can be shown to be deterministic [20]:

Definition 2.3.4. *The* optimal (standard) policy π^* *is the policy maximising the standard objective function:*

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.33}$$

or equivalently, the optimal standard policy can be in terms of the optimal Q-value function:

$$\pi^*(s) = \arg\max_{a} Q^*(s, a). \tag{2.34}$$

It can be shown that the optimal (Q-)value functions satisfy the *Bellman equations* [5, 20]:

$$V^*(s; t) = \max_{a \in \mathcal{A}} Q^*(s, a; t), \tag{2.35}$$

$$Q^*(s, a; t) = R(s, a) + \mathbb{E}_{s' \sim p(\cdot|s,a)}[V^*(s'; t + 1)]. \tag{2.36}$$

The Bellman equations admit a *dynamic programming* approach which can be used to computer the optimal value functions, known as *Value Iteration* [5, 20]. In Value Iteration, a table of value estimates $\hat{V}(s; t)$ are kept for each s, t. Given any initial estimate of the value function $\hat{V}^0$, the *Bellman backup* operations are:

$$\hat{V}^{k+1}(s; t) = \max_{a \in \mathcal{A}} \hat{Q}^{k+1}(s, a; t), \tag{2.37}$$

$$\hat{Q}^{k+1}(s, a; t) = \mathbb{E}_{s' \sim p(\cdot|s,a)}[R(s, a) + \hat{V}^k(s'; t + 1)]. \tag{2.38}$$

In each iteration of Value Iteration, these values are computed for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $t \in \{0, 1, ..., H\}$. The value estimates computed from the Bellman backups will converge to the optimal values in a finite number of iterations: $V^k \to V^*$.

2.3.1 Maximum Entropy Reinforcement Learning

In *maximum-entropy reinforcement learning*, the objective function is altered to include the addition of an entropy term. The addition of an entropy term is motivated by wanting to learn stochastic behaviours, that better explore large state spaces, and learn more robust behaviours under uncertainty by potentially learning multiple solutions rather than a single deterministic solution [10].

Let $\mathcal{H}$ denote the (Shannon) entropy function [24]:

$$\mathcal{H}(\pi(\cdot|s)) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log\pi(a|s)] = \sum_{a\in\mathcal{A}} \pi(a|s)\log\pi(a|s). \tag{2.39}$$

In the maximum entropy objective, the relative weighting of entropy terms is included using a coefficient α, called the *temperature*. In the maximum entropy objective, analogues of the value functions can be defined, which are typically referred to as *soft (Q-)values*, and similarly the maximum entropy objective is often referred to as the *soft objective*.

Definition 2.3.5. *The* soft value *of a policy π from state s at time t is:*

$$V^{\pi}_{\text{sft}}(s;t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=t}^{H-1} r_t + \alpha\mathcal{H}(\pi(\cdot|s_i))\middle| s_t = s\right]. \tag{2.40}$$

The soft Q-value *of a policy π, from state s, with action a, at time t is:*

$$Q^{\pi}_{\text{sft}}(s,a;t) = R(s,a) + \mathbb{E}_{s'\sim p(\cdot|s,a)}[V^{\pi}_{\text{sft}}(s';t+1)]. \tag{2.41}$$

The optimal soft (Q-)values can be defined by taking the maximum over policies, similarly to the standard objective:

Definition 2.3.6. *The* optimal soft (Q-)Value *of a state(-action pair) is defined as:*

$$V^{*}_{\text{sft}}(s;t) = \max_{\pi} V^{\pi}_{\text{sft}}(s;t), \tag{2.42}$$

$$Q^{*}_{\text{sft}}(s,a;t) = \max_{\pi} Q^{\pi}_{\text{sft}}(s,a;t). \tag{2.43}$$

In maximum entropy reinforcement learning, the objective is to find a policy with maximal soft value.

Definition 2.3.7. *The* maximum entropy (or soft) reinforcement learning objective function $J_{\text{sft}}(\pi)$ *is defined as:*

$$J_{\text{sft}}(\pi) = V^{\pi}_{\text{sft}}(s_0; 0). \tag{2.44}$$

The objective of maximum entropy (or soft) reinforcement learning can then be stated as finding $\max_\pi J_{\text{sft}}(\pi)$.

The optimal soft policy is defined as the policy that maximises the soft objective function J_{sft}, and it can be computed from the optimal soft (Q-)values.

Definition 2.3.8. *The* optimal soft policy π^*_{sft} *is the policy maximising the soft objective function:*

$$\pi^*_{\text{sft}} = \arg\max_{\pi} J_{\text{sft}}(\pi). \tag{2.45}$$

Given V^*_{sft} *and* Q^*_{sft} *the optimal soft police is known to take the form [10]:*

$$\pi^*_{\text{sft}}(a|s; t) = \exp\left(\left(Q^*_{\text{sft}}(s, a; t) - V^*_{\text{sft}}(s; t)\right)/\alpha\right). \tag{2.46}$$

Equations similar to the Bellman equations, aptly named the *soft Bellman equations*, can be defined for maximum entropy reinforcement learning. These equations differ to equations (2.35) and (2.36) by the replacement of the max operation with a *softmax* or *log-sum-exp* operation, and explain why the maximum entropy analogues are referred to as the *soft* versions of their standard reinforcement learning counterparts. The *soft Bellman equations* are [10]:

$$V^*_{\text{sft}}(s; t) = \alpha \log \sum_{a \in \mathcal{A}} \exp\left(Q^*_{\text{sft}}(s, a; t)/alpha\right), \tag{2.47}$$

$$Q^*_{\text{sft}}(s, a; t) = R(s, a) + \mathbb{E}_{s' \sim p(\cdot|s,a)}[V^*_{\text{sft}}(s'; t + 1)]. \tag{2.48}$$

Analogous to standard reinforcement learning, the soft Bellman equations can be used in *soft Bellman backups* for a *Soft Value Iteration* algorithm [10]:

$$\hat{V}^{k+1}_{\text{sft}}(s; t) = \alpha \log \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\alpha}\hat{Q}^{k+1}_{\text{sft}}(s, a; t)\right), \tag{2.49}$$

$$\hat{Q}^{k+1}_{\text{sft}}(s, a; t) = R(s, a) + \mathbb{E}_{s' \sim p(\cdot|s,a)}[\hat{V}^{k}_{\text{sft}}(s'; t + 1)]. \tag{2.50}$$

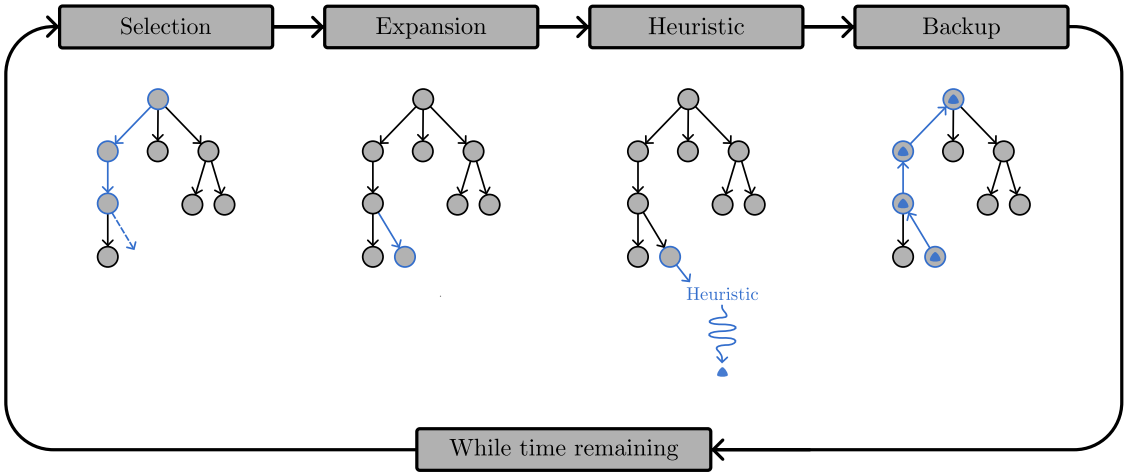

Figure 2.6: Overview of one trial of MCTS-1, depicting the four commonly presented stages [6]. First, in the selection phase a search policy is used to sample a path down the tree. Next a new node is added to the search tree. Then the new node is initialised using a heuristic, often using a *simulation* using a *rollout policy* as depicted. Finally, values (triangles) are updated through *backups* along the path sampled in the selection phase.

2.4 Trial-Based Heuristic Tree Search and Monte Carlo Tree Search

When considering MDPs with large state-action spaces, using tabular dynamic programming methods becomes infeasibly slow. One approach for handling large state-action spaces is to use *heuristic methods*, that consider a subset of the state-action space and utilise heuristic estimates for the value of states. Often these heuristic methods build a *search tree* from the initial state of the MDP.

Monte Carlo Tree Search (MCTS) refers to heuristic algorithms that build a search tree using *Monte Carlo Trials*, where nodes are added to the tree based off randomly sampled trajectories. Two advantages of using MCTS methods in the modern day are: that they allow for statistical analysis, which can provide *probabilistic guarantees* for the performance of the algorithm; and they offer interpretability, as the search tree can be inspected post-hoc to identify why the algorithm made certain decisions.

MCTS trials are commonly presented using four stages, selection, expansion, simulation/heuristic and backup phases [6], which are depicted in Figure 2.6. To avoid confusion with the above definition of MCTS, this thesis will refer to this

specific form of MCTS as MCTS-1. Each of the four phases of MCTS-1 are described in more detail below:

1. *Selection phase* - a *search policy* is used to traverse the search tree from the root node until it reaches a state not contained in the search tree;

2. *Expansion phase* - a new child node is added to the search tree;

3. *Simulation/Heuristic phase* - the new child node's value is initialised using a *heuristic*, often taking the form of a Monte Carlo return, obtained using a *simulation* with a *rollout policy*;

4. *Backup phase* - the value from the heuristic or simulation is used to backup values through the path traversed in the selection phase.

Trial-Based Heuristic Tree Search (THTS) [14] generalises heuristic tree search methods used for planning, such as Trial-Based Real Time Dynamic Programming [4] and LAO* [11]. However, THTS differs slightly from how MCTS-1 is presented above. In a THTS trial, after the expansion/heurisitc phases are run, the trial "switches back to the selection phase and alternate between those phases", until the planning horizon is reached. This is similar to the trial depicted in blue in Figure 2.8.

`THTS++` [21] is introduced in Section 2.4.2, which is an open-source, parallelised extension of the THTS schema (including being able to implement MCTS-1 algorithms), and was built to facilitate the work in this thesis. Section 2.4.3 covers the Upper Confidence Bound applied to Trees (UCT) algorithm [16, 17], which is a commonly used MCTS algorithm, and Section 2.4.4 presents the Maximum ENtropy Tree Search (MENTS) algorithm [32], which is related to the algorithms introduced in Chapter 4 of this thesis.

2.4.1 Notation

To simplify notation in the presentation and analysis of `THTS++` algorithms, this thesis assumes that states and state-action pairs have a one-to-one correspondance with nodes in the search tree. This assumption is purely to simplify notation for

a clean presentation, and any results discussed in this thesis generalise to when this assumption does not hold.

Specifically, this allows the notation for value functions to avoid explicitly writing the timestep parameter, so that $V^\pi(s)$ can be written instead of $V^\pi(s;t)$.

2.4.2 Trial Based Heuristic Tree Search

This subsection introduces `THTS++` , which extends the Trial-Based Heuristic Tree Search (THTS) schema [14], and aims to provide a modular library which can implement any MCTS algorithm. The changes made to the original schema allow algorithms following the four-phase MCTS-1 trials described previously to be encorporated. Additionally, the notion of a per-trial *context* is added, which will be used in Chapters 5 and 6 for multi-objective MCTS algorithms. Additionally, beyond the scope of this thesis, contexts allow `THTS++` to implement algorithms for *partially-observable* environments, such as Partially Observable Monte Carlo Planning [25]. The remainder of this section will be presented in the context of planning for fully-observable MDPs, but `THTS++` generalises to partially-observable environments if *observation-action histories* are used in place of states.

In `THTS++` trees consist of *decision nodes* and *chance nodes*. Decision nodes sample actions that can be taken by the agent, and chance nodes sample *outcomes* (MDP states) that depend on the action taken and the transition function of the MDP. As such, each decision node has an associated *state* and each chance node has an associated *state-action pair*. Figure 2.7 shows how decision and chance nodes will be depicted as circles and diamonds in diagrams.

A search tree $\mathcal{T}$ is built using Monte Carlo *trials* and an overview of one trial in `THTS++` is given in Figure 2.8. Each `THTS++` trial is also split into four phases:

1. *Context sampling* - a *context* is sampled for the trial;

2. *Selection phase* - a trajectory is sampled using a *search policy*, any newly visited states (and state-action pairs) are added to $\mathcal{T}$ as decision nodes (and chance nodes);

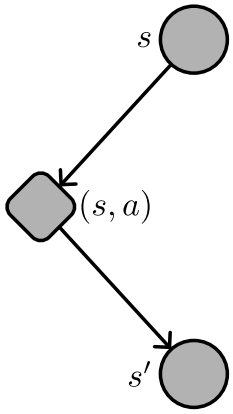

Figure 2.7: Tree diagrams notation, circles will be used to denote *decision nodes* that are associated with states, diamonds will be used to denote *chance nodes* that are associated with state-action pairs and arrows are used to denote parent/child relationships in the tree.

3. *Heuristic phase* - any new leaf nodes added in the selection phase are initialised using a *heuristic function*;

4. *Backup phase* - value estimates in the search tree are updated, along the path sampled in the selection phase.

Note that the selection and expansion phases of MCTS-1 are encapsulated by the selection phase of `THTS++`.

Definition 2.4.1. *A* search tree $\mathcal{T}$ *is a subset of the state and state-action spaces, that is* $\mathcal{T} \subseteq \mathcal{S} \cup \mathcal{S} \times \mathcal{A}$*, where for each* $s \in \mathcal{T}$*, there exists some truncated trajectory* $\tau_{:h}$ *such that* $s_h = s$*, each* $s' \in \tau_{:h}$ *is also in the tree,* $s' \in \mathcal{T}$*, and each* $s', a' \in \tau$*:* h *is also in the tree,* $(s', a') \in \mathcal{T}$*.*

A *decision node* refers to any state that is in the search tree: $s \in \mathcal{T}$. A *chance node* refers to any state-action pair that is in the search tree: $(s, a) \in \mathcal{T}$. And a *node* is used to refer to any decision or chance node in the tree. Sometimes the notation $\texttt{node}(s)$ and $\texttt{node}(s, a)$ will be used to make it clear that a node is being discussed, rather than a state or state-action pair.

$N(s)$ and $N(s, a)$ denote the number of times $\texttt{node}(s)$ and $\texttt{node}(s, a)$ have been visited, or equivalently, the number of times s and (s, a) appear in trajectories sampled in `THTS++` trials.

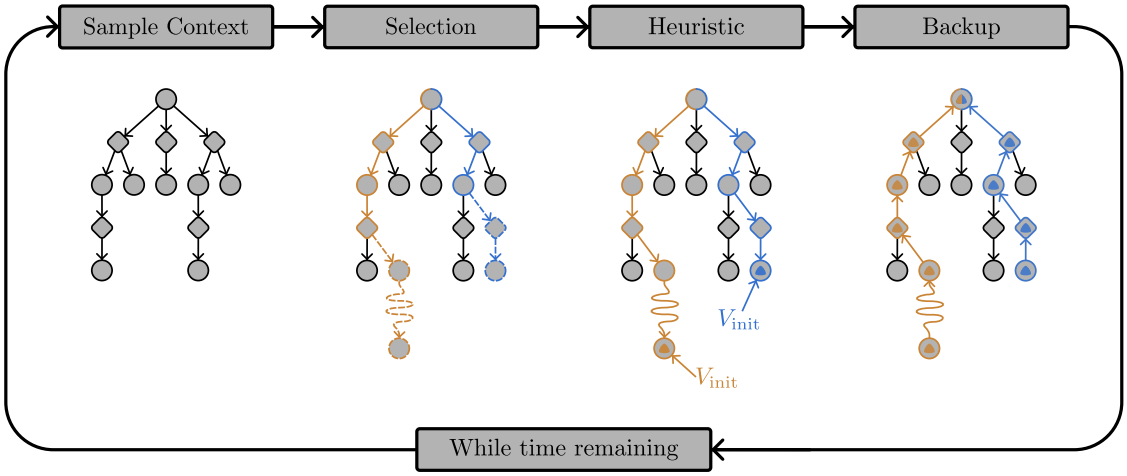

Figure 2.8: Overview of one trial of `THTS++`, where orange shows an example when `mcts_mode` is True, and blue shows an example when `mcts_mode` is False. From left to right: first a context is sampled, which stores any necessary per-trial state (not depicted) and the search tree at the beginning of the trial is shown; second shows the selection phase, where a trajectory is sampled and where dashed lines indicate any new nodes added; third shows that new leaf nodes are initialised using the $\hat{V}_{\text{init}}$ heuristic function; and finally on the right, shows the backup phase, where the arrows directions are changed to show that information is being propogated back up the tree to update the values in the nodes (triangles).

Each decision and chance node will generally store value estimates that are algorithm dependent. $\texttt{node}(s)\texttt{.V}$ is used to denote the set of values stored at node $\texttt{node}(s)$, and $\texttt{node}(s,a)\texttt{.Q}$ for the set of values stored at node $\texttt{node}(s,a)$.

Additionally, let $\texttt{node}(s)\texttt{.chldrn}$ be a mapping from actions to the chance nodes that are children of $\texttt{node}(s)$. Likewise, $\texttt{node}(s,a)\texttt{.chldrn}$ is a mapping from outcomes (successor states) to the decision nodes that are children of $\texttt{node}(s,a)$.

`THTS++` introduces the idea of `mcts_mode` into the THTS schema. When `mcts_mode` is True, a single decision node is added to the search tree on each trial, similarly to MCTS-1. When it is False, the trajectory is sampled until a terminal state or the planning horizon is reached. These two modes of operation are depicted in blue and orange respectively in Figure 2.8.

Definition 2.4.2. *When running in* `mcts_mode`*, the trajectory sampled in the selection phase is truncated, and will end if a state not in the search tree $\mathcal{T}$ is reached, similarly to* MCTS-1 *[6]. When not running in* `mcts_mode`*, the trajectory*

is sampled until a terminal state is reached, or the planning horizon is reached, similarly to THTS [14].

There are two main benefits to running an algorithm in `mcts_mode`. The first is that it uses significantly less memory, which can be a concern if using a large planning horizon H and the tree search is going to run for a long time. The second is that if `mcts_mode` is used with an informative heuristic function, then it allows the algorithm to avoid wasting resouces (time and memory) on parts of the search tree that are not promising.

In contrast, when no informative heuristic is available, a random simulation is often used in MCTS-1 algorithms. In such cases, where memory allows, running with `mcts_mode` set to False can be beneficial, because states that would have been visited in the simulation phase of MCTS-1 are added to the tree and avoids throwing away potentially useful information.

`THTS++` also introduces the notion of a *context* that is sampled for each trial. The context is passed to every subsequent function call in a trial of `THTS++`, and can be used to store temporary state. This context will go unused for the remainder of the chapter, but will be useful in Chapters 5 and 6 when *contextual tree search* is discussed.

Definition 2.4.3. *A* context *is an arbitrary key-value store that is used to store any relevant data that varies from trial to trial.*

To specify an algorithm in the `THTS++` schema, the following need to be specified:

Value Estimates: What value estimates will be stored at each decision and chance node. That is, $\texttt{node}(s)\texttt{.V}$ and $\texttt{node}(s, a)\texttt{.Q}$ need to be defined;

Search policy: A policy π_{search} used to sample a trajectory for the trial, which can use values in the current search tree $\mathcal{T}$, and values from the heuristic action function (below);

Outcome Sampler: A function `sample_outcome` that samples outcomes according to the environment, given a current state and action. In this thesis, this will always sample a state from the transition distribution of the MDP: $s' \sim p(\cdot|s, a)$;

Heuristic value function: A function $\hat{V}_{\text{init}}$ used as a heuristic to initialise values for new decision nodes added to the tree;

Heuristic action function: A function $\hat{Q}_{\text{init}}$ used as a heuristic for Q-values when a state-action pair is not in the current search tree;

Backup functions: Two functions `backup_v` and `backup_q` which updates the values in decision and chance nodes respectively. These functions can use values from their children, from the sampled trajectory and from the heuristic value function;

Context sampler: A function `sample_context` which creates a context key-value store, and samples any initial values to be stored in the context;

MCTS mode: A boolean (which will also be denoted `mcts_mode`) specifying if `THTS++` should operate in `mcts_mode`;

Planning horizon: A (problem dependent) planning horizon for the tree search $H_{\texttt{THTS++}}$, which is no longer than the horizon of the MDP, $H_{\texttt{THTS++}} \leq H$.

Figure 2.8 depicts a trial in `THTS++`, and psuedocode for running a trial is given in Listing 2.1. At the beginning of runnning a `THTS++` algorithm, the search tree is initialised to $\mathcal{T} \leftarrow \{s_0\}$. When the components detailed above are provided, the operation of a trial in `THTS++` is as follows:

- Firstly, a context is sampled using the `sample_context` function, which is available to be used by any other function in the trial.
- A trajectory is sampled $\tau_{:h} \sim \pi_{\text{search}}$ according to the search policy (which may use $\hat{Q}_{\text{init}}$ as necessary) and `sample_outcome`.

  - If running in `mcts_mode`, then $\tau_{:h}$ is such that $s_t \in \mathcal{T}$ for $t = 0, ..., h-1$ and $s_h \notin \mathcal{T}$, or $h = H_{\texttt{THTS++}}$, or s_h is terminal.
  - If not running in `mcts_mode`, then $h = H_{\texttt{THTS++}}$, or s_h is terminal.

- The search tree is updated to include any new nodes from the sampled trajectory, $\mathcal{T} \leftarrow \mathcal{T} \cup \tau_{:h}$.
- The heuristic value function is used to initialise the value of the new leaf node $\texttt{node}(s_h)\texttt{.V} \leftarrow \hat{V}_{\text{init}}(s_h)$.
- Finally, for the backup phase, the `backup_q` and `backup_v` functions are used to update the values of $\texttt{node}(s_t, a_t)\texttt{.Q}$ and $\texttt{node}(s_t)\texttt{.V}$ for $t = h-1, ..., 0$.

2.4.3 Upper Confidence Bounds Applied to Trees (UCT)

Upper Confidence Bound applied to Trees (UCT) [16, 17] is a commonly used tree search algorithm, which is based on the Upper Confidence Bound (UCB) algorithm covered in Section 2.1. UCT is a good example of a common paradigm in tree search algorithms for sequential decision making, where each node in the tree is tasked with making a single decision in the sequence, and so adapts methods from the MAB literature. More specifically, UCT can be viewed as running UCB on a non-stationary MAB at every decision node, where it is non-stationary because the rewards obtained depend on the decisions that children make. The remainder of this subsection will outline how UCT can be defined using the the `THTS++` schema given in Section 2.4.2.

In UCT `mcts_mode` is set to True. And at each decision and node of UCT a sample average $\bar{V}_{\text{UCT}}$ and $\bar{Q}_{\text{UCT}}$ is used for a value estimate. The search policy that UCT then follows is given by:

$$\pi_{\text{UCT}}(s) = \arg\max_{a \in \mathcal{A}} \bar{Q}_{\text{UCT}}(s, a) + b_{\text{UCT}} \sqrt{\frac{\log(N(s))}{N(s, a)}}, \tag{2.51}$$

where b_{UCT} is a *bias* parameter that controls the amount of exploration UCT will perform. In Equation (2.51), when $N(s, a) = 0$ there is a division by zero, which is taken as ∞, and ties are broken uniformly randomly. Note that like UCB (Section

```
def run_trial(search_tree: 𝒯,
              search_policy: π_search,
              heuristic_fn: V̂_init,
              heuristic_action_fn: Q̂_init,
              mcts_mode,
              planning_horizon: H_THTS++):
    # context sampling
    ctx = sample_context()
    # simulation phase
    τ_:h = sample_trajectory(𝒯, π_search, Q̂_init, mcts_mode, H_THTS++, ctx)
    𝒯 ← 𝒯 ∪ τ_:h
    # heuristic phase
    node(s_h).V ← V̂_init(s_h, ctx)
    # backup phase
    for i in {h−1, h−2, ..., 1, 0}:
        node(s_i, a_i).backup_q(node(s_i, a_i).chldrn, τ_:h, V̂_init(s_h), ctx)
        node(s_i).backup_v(node(s_i).chldrn, τ_:h, V̂_init(s_h), ctx)

def sample_trajectory(search_tree: 𝒯,
                      search_policy: π_search,
                      heuristic_action_fn: Q̂_init,
                      mcts_mode,
                      planning_horizon: H_THTS++,
                      ctx):
    i = 0
    while ((not mcts_mode or s_i ∈ 𝒯)
           and (i < H_THTS++)
           and (not terminal(s_i))):
        a_i ~ π_search(·|s_i, ctx)
        r_i ← R(s_i, a_i)
        s_i+1 ← sample_outcome(s_i, a_i, ctx)
        i += 1
    return (s_0, a_0, r_0, s_1, ..., s_i-1, a_i-1, r_i-1, s_i)

def sample_outcome(s, a, ctx):
    s' ~ p(·|s, a)
    return s'
```

Listing 2.1: Psuedocode for running a trial in THTS++.

2.1), this results in every action being taken once to obtain an initial value estimate, and as such, setting $\hat{Q}_{\text{init}}$ is unnecessary for UCT.

There are two common approaches to implementing $\hat{V}_{\text{init}}$ in UCT: the first consisting of using a function approximation $\tilde{V}$ and setting $\hat{V}_{\text{init}} = \tilde{V}$ [26, 27, 19] , where $\hat{V}$ aims to approximate the optimal value function V^*; the second approach consists of using a *rollout policy* [6, 9, 13, 8]. When a rollout policy π_{rollout} is used, a Monte Carlo estimate $\hat{V}^{\pi_{\text{rollout}}}$ is used to estimate the value function

$V^{\pi_{\text{rollout}}}$ and is used for $\hat{V}_{\text{init}}$.

Let $\tau_{:h} \sim \pi_{\text{UCT}}$, be the trajectory sampled in the selection phase of UCT, meaning that a decision node for s_h is added to the search tree, and needs to be initialised with $\hat{V}_{\text{init}}$. When using a rollout policy, the truncated trajectory is completed using the rollout policy, $\tau_{h:H} \sim \pi_{\text{rollout}}$, to provide the Monte Carlo estimate at s_h:

$$\hat{V}^{\pi_{\text{rollout}}}(s_h) = \sum_{i=h}^{H-1} r_i. \tag{2.52}$$

If no informative policy is available to be used for π_{rollout}, then uniformly random policy is often used [6, 1].

In UCT the backups `backup_v` and `backup_q` update the (sample average) value estimates. Letting heuristic value for the leaf node be $\tilde{r} = \hat{V}_{\text{init}}(s_h)$, they are computed as follows:

$$\bar{Q}_{\text{UCT}}(s_t, a_t) \leftarrow \frac{1}{N(s_t, a_t)} \left((N(s_t, a_t) - 1)\bar{Q}_{\text{UCT}}(s_t, a_t) + \tilde{r} + \sum_{i=t}^{h-1} r_i \right) \tag{2.53}$$

$$\bar{V}_{\text{UCT}}(s_t) \leftarrow \frac{1}{N(s_t)} \left((N(s_t) - 1)\bar{V}_{\text{UCT}}(s_t, a_t) + \tilde{r} + \sum_{i=t}^{h-1} r_i \right) \tag{2.54}$$

2.4.4 Maximum Entropy Tree Search

Maximum ENtropy Tree Search (MENTS) [32], in contrast to UCT, focuses on the maximum-entropy objective, and uses soft Bellman backups to update its value estimates. In its original presentation `mcts_mode` is set to True, and it uses the soft value estimates $\hat{V}_{\text{MENTS}}$ and $\hat{Q}_{\text{MENTS}}$. The MENTS search policy is given by:

$$\pi_{\text{MENTS}}(a|s) = (1 - \lambda_s) \exp\left(\frac{1}{\alpha_{\text{MENTS}}} \left(\hat{Q}_{\text{MENTS}}(s, a) - \hat{V}_{\text{MENTS}}(s) \right) \right) + \frac{\lambda_s}{|\mathcal{A}|}, \tag{2.55}$$

where $\lambda_s = \min(1, \epsilon / \log(e + N(s)))$, with $\epsilon \in (0, \infty)$ is an exploration parameter, and α_{MENTS} is the temperature paramter used in MENTS for the maximum entropy objective (the coefficient of the entropy term in Equation (2.40)).

In MENTS the backups `backup_v` and `backup_q` update the soft value estimates and are updated using soft Bellman backups (Equations (2.49) and (2.50)) as follows:

$$\hat{Q}_{\text{MENTS}}(s_t, a_t) \leftarrow R(s_t, a_t) + \sum_{s' \in \text{Succ}(s,a)} \left(\frac{N(s')}{N(s_t, a_t)} \hat{V}_{\text{MENTS}}(s') \right), \tag{2.56}$$

$$\hat{V}_{\text{MENTS}}(s_t) \leftarrow \alpha \log \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\alpha} \hat{Q}_{\text{MENTS}}(s_t, a) \right). \tag{2.57}$$

Figure 2.9: The decision-support scenario for multi-objective reinforcement learning [12].

In [32], the heuristic value function left as an arbitrary evaluation function, but is set using a function approximation $\hat{V}_{\text{init}} = \tilde{V}$ in experiments. The heuristic action function is set to zero, $\hat{Q}_{\text{init}}(s, a) = 0$, but they also suggest that if *policy network* $\tilde{\pi}$ is available, then the heuristic action function can alternatively be set to $\hat{Q}_{\text{init}}(s, a) = \log \tilde{\pi}(s|a)$.

2.5 Multi-Objective Reinforcement Learning

This thesis follows a utility based approach to multi-objective reinforcement learning similar to the review of Hayes et al. [12]. This work will specifically consider *linear utility* functions and the *decision support scenario* which is depicted in Figure 2.9. In the decision support scenario, an algorithm computes a *solution set* consisting of multiple *possibly optimal* solutions, from which a user then picks their most preferred option to be used. This scenario is useful when a user's preferences over the multiple objectives is unknown or difficult to specify in advance.

This section defines the multi-objective counterparts to various definitions given in Section 2.2 and 2.3. Outside of this section, the prefix "multi-objective" may be dropped where it should be clear from context, however bold typeface will consistently be used to denote any vector variables or functions. In Section 2.5.1, a multi-objective extension of Value Iteration is given.

To specify problems with multiple objectives, the reward function of an MDP changed to give a vector of rewards, rather than a scalar reward:

Definition 2.5.1. *A* Multi-Objective Markov Decision Process *(MOMDP) is a tuple* $\boldsymbol{\mathcal{M}} = (\mathcal{S}, s_0, \mathcal{A}, \mathbf{R}, p, H)$*, where* $\mathcal{S}$ *is a set of states,* $s_0 \in \mathcal{S}$ *is an initial state,* $\mathcal{A}$ *is a set of actions,* $\mathbf{R}(s, a)$ *is a vector reward function* $\mathcal{S} \times \mathcal{A} \to \mathbb{R}^D$*, where* D *is*

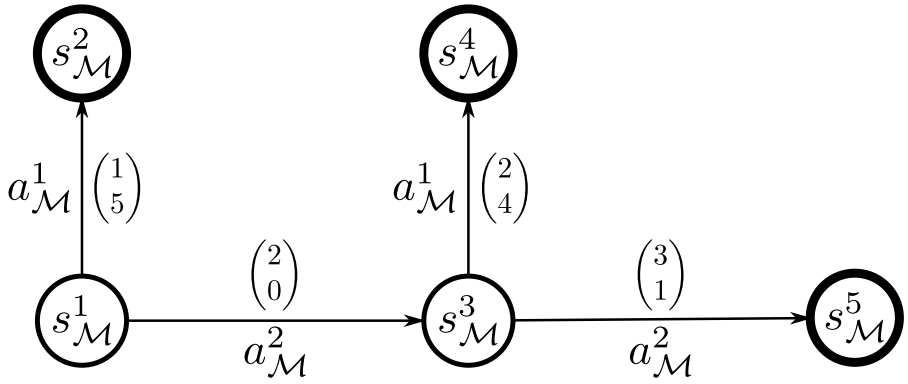

Figure 2.10: An example MOMDP $\boldsymbol{\mathcal{M}}$, where $s_0 = s^1_{\mathcal{M}}$, the finite horizon is $H = 2$ and terminal states are marked with thicker borders. This particular MOMDP is deterministic, so all transition probabilities are one.

the dimension of the rewards and the MOMDP, $p(s'|s,a)$ is a next state transition distribution $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ and $H \in \mathbb{N}$ is a finite-horizon time bound.

Now multi-objective trajectories are defined:

Definition 2.5.2. *A* multi-objective trajectory *τ, is a sequence of state, action and vector rewards, that is induced by a policy π and MOMDP $\boldsymbol{\mathcal{M}}$ pair. Let the trajectory be $\tau = (s_0, a_0, \mathbf{r}_0, s_1, a_1, \mathbf{r}_1, ..., s_{H-1}, a_{H-1}, \mathbf{r}_{H-1}, s_H)$, where $a_t \sim \pi(\cdot|s_t)$, $\mathbf{r}_t = \mathbf{R}(s_t, a_t)$ and $s_{t+1} \sim p(\cdot|s_t, a_t)$.*

The notations used for single-objective trajectories (Definition 2.2.5) will also be used for multi-objective trajectories too. Such as, $\tau \sim \pi$ for sampling trajectories using policies, and $\tau_{i:j}$ for truncated trajectories.

Similarly, multi-objective variants of the (Q-)value of a policy are defined:

Definition 2.5.3. *The* multi-objective value *of a policy π from state s at time t is:*

$$\mathbf{V}^\pi(s;t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=t}^{H-1} \mathbf{r}_t \middle| s_t = s\right]. \tag{2.58}$$

The multi-objective Q-value *of a policy π, from state s, with action a, at time t is:*

$$\mathbf{Q}^\pi(s,a;t) = \mathbf{R}(s,a) + \mathbb{E}_{s'\sim p(\cdot|s,a)}[\mathbf{V}^\pi(s';t+1)]. \tag{2.59}$$

In the corresponding point of the single-objective reinforcement learning section (Section 2.3), the the optimal (Q-)value functions and the objective of single-objective reinforcement learning were defined. However, in a multi-objective setting there is no longer a *total ordering* over values, and so there maybe be multiple vectors that could be "optimal". For example, consider two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$ which are such that $u_1 > v_1$ and $u_2 < v_2$. To resolve this issue, a *utility function* or *scalarisation function* is used to map multi-objective values to scalar values.

Definition 2.5.4. *The* (D-dimensional) Simplex *consists of the set of D-dimensional vectors, whose entries are non-negative and sum to one. More formally, the D dimensional simplex is*

$$\Delta^D = \{\mathbf{w} \in \mathbb{R}^D | w_i > 0, \sum_i w_i = 1\}. \tag{2.60}$$

The elements of the D-dimensional Simplex will be referred to as weight vectors *(or just* weights*) in this thesis, as they will be used to specify preferences over the D dimensions of the reward function.*

Definition 2.5.5. *A* utility function *(or* scalarisation function*)* $u : \mathbb{R}^D \times \Delta^D \to \mathbb{R}$ *is used to map from a multi-objective value $\mathbf{v} \in \mathbb{R}^D$ and a weighting over the objectives $\mathbf{w} \in \Delta^D$ to a scalar value. That is, according to the utility function $u(\cdot; \mathbf{w})$ the multi-objective value $\mathbf{v}$ is mapped to the scalar value $u(\mathbf{v}; \mathbf{w})$.*

Of particular interest in this thesis is the *linear utility function* where the scalar value takes the form of a dot-product:

Definition 2.5.6. *The* linear utility function u_{lin} *is the utility function defined by:*

$$u_{\text{lin}}(\mathbf{v}; \mathbf{w}) = \mathbf{w}^\top \mathbf{v}. \tag{2.61}$$

Equipped with a utility function and a weight vector any set of multi-objective values can be mapped to scalars and ordered. A policy is *possibly optimal*, or *undominated*, with respect to utility function u if it achieves a maximal utility for some weight $w \in \Delta^D$. In contrast, a policy is *dominated* if for every weight there is another policy that achieves a better utility.

Let Π be the set of all possible policies for a given MOMDP. When constructing a solution set, a subset of Π, it makes sense that it should not contain any dominated policies. This leads to the first notion of a solution set, called an *undominated set*, consisting of all undominated policies:

Definition 2.5.7. *The* undominated set *of policies* $U(\Pi; u) \subseteq \Pi$*, with respect to a utility function* u*, is the set of policies for which there is a weight vector* $\mathbf{w} \in \Delta^D$ *where the utility (scalarised value) is maximised:*

$$U(\Pi; u) = \left\{ \pi \in \Pi \;\middle|\; \exists \mathbf{w} \in \Delta^D . \forall \pi' \in \Pi : u(\mathbf{V}^{\pi}(s_0; 0); \mathbf{w}) \geq u(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}) \right\}. \tag{2.62}$$

In particular, the convex hull *of policies* $CH(\Pi)$ *is the undominated set with respect to the linear utility function* u_{lin}*. That is* $CH(\Pi) = U(\Pi; u_{\text{lin}})$*.*

Question: How do you feel about removing the timestep in the value functions in this section too? So it can be $\mathbf{V}^{\pi}(s)$ rather than $\mathbf{V}^{\pi}(s; t)$? I think it would make some definitions like undominated sets a bit cleaner. Alternatively, maybe I should write it $u_{\mathbf{w}}$.

When computing solutions sets, it is often useful to first compute the multi-objective values that could be obtained, and then later use the data structures used by the algorithm to read out the selected policy. This thesis will refer to the set of values obtained by a set of policies as a *value set*:

Definition 2.5.8. *The* (multi-objective) value set *with respect to a set of policies* $\Pi' \subseteq \Pi$ *is defined as* $\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi') = \{\mathbf{V}^{\pi}(s_0; 0) | \pi \in \Pi'\}$.

Undominated sets often have an infinite cardinality, and as such are infeasible to compute. However, in undominated sets there are often many reduntant policies that obtain the same scalarised values. Instead of computing an undominated set, it is more feasible to compute a *coverage sets* which contain at least one policy that maximises the scalarised value given any weight vector $\mathbf{w}$:

Definition 2.5.9. *A set $CS(\Pi; u) \subseteq U(\Pi)$, is a* coverage set *with respect to a utility function u, if for every weight vector $\mathbf{w} \in \Delta^D$, there is a policy $\pi \in CS(\Pi; u)$ that maximises the value of $u(\cdot; \mathbf{w})$. That is, for $CS(\Pi; u)$ to be a coverage set, the following statement must be true:*

$$\forall \mathbf{w} \in \Delta^D . \exists \pi \in CS(\Pi; u) . \forall \pi' \in \Pi : u(\mathbf{V}^{\pi}(s_0; 0); \mathbf{w}) \geq u(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}). \quad (2.63)$$

Again, in particular, any set $CCS(\Pi)$ is a convex coverage set *if it is a coverage set with respect to the linear utility function u_{lin}.*

Note that it is still possible for there to be multiple coverage sets. Algorithms that compute convex coverage sets typically compute a unique minimal convex coverage set that contains no redundant policies, that is, each policy has some weight for which it uniquely gives an optimal solution in the set.

Definition 2.5.10. *A set $CCS_{\text{min}}(\Pi) \subseteq CH(\Pi)$ is minimal if the following holds*

$$\forall \pi \in CCS_{\text{min}}(\Pi) . \exists \mathbf{w} \in \Delta^D . \forall \pi' \in CCS_{\text{min}}(\Pi) - \{\pi\} . u_{\text{lin}}(\mathbf{V}^{\pi}(s_0; 0); \mathbf{w}) > u_{\text{lin}}(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}). \quad (2.64)$$

Now the geometry of convex coverage sets is considered, to see how something like $\boldsymbol{\mathcal{V}}\mathbf{als}(CCS_{\text{min}}(\Pi))$ can be computed. Firstly, any multi-objective values that obtain the same linear utility will lie on a hyperplane, whose normal is the weight vector used in the linear utility function (see (a) in Figure 2.11). As a result, the values of the convex hull, $\boldsymbol{\mathcal{V}}\mathbf{als}(CH(\Pi))$ lie on a *geometric (partial) convex hull* (see (b) in Figure 2.11). Moreover, any points in $\boldsymbol{\mathcal{V}}\mathbf{als}(CH(\Pi))$ that lie on the edge of the geometric convex hull are redundant, as the values corresponding to the neighbouring vertices of geometric convex hull will always achieve the same or greater utility (see (c) in Figure 2.11).

Because of the ambiguity between the convex hull of policies, and the geometric convex hulls of values, this thesis will refer to any value set that forms a geometric convex hull as a *convex hull value set* (CHVS) and will refer to the set $\boldsymbol{\mathcal{V}}\mathbf{als}(CCS_{\text{min}}(\Pi))$ as the *optimal convex hull value set*. Question: should I make this paragraph a definition to emphasise it a bit more?

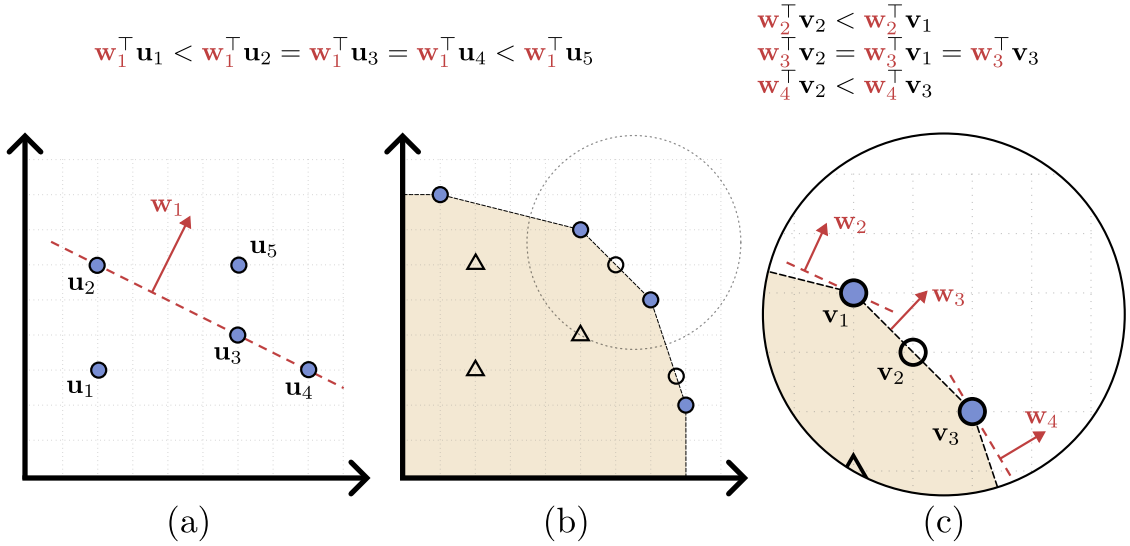

Figure 2.11: The geometry of convex coverage sets, shown with $D = 2$. In each images the points depicted correspond to a value set $\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi')$, for some set of policies Π'. (a) demonstrates that values ($\mathbf{u}_2$,$\mathbf{u}_3$,$\mathbf{u}_4$) obtaining the same utility lie on a hyperplane. Moving in the direction of the weight vector increases the utility ($\mathbf{u}_5$) and likewise moving in the opposite direction decreases utility ($\mathbf{u}_1$); (b) circles correspond to values that are on the convex hull, so form the set $\boldsymbol{\mathcal{V}}\mathbf{als}(CH(\Pi'))$, the blue circles form the minimal convex coverage set, $\boldsymbol{\mathcal{V}}\mathbf{als}(CCS_{\text{min}}(\Pi'))$, and triangles are not part of the convex hull. All of the unfilled shapes would be pruned by `cvx_prune`(Equation (2.65)); (c) shows a magnification of (b) with additional labels, it shows that the value $\mathbf{v}_2$ is redundant because one of $\mathbf{v}_1$ or $\mathbf{v}_3$ will achieve an equal or greater utility. Additionally, is shows $\mathbf{w}_2$ and $\mathbf{w}_4$ which are weights that the values $\mathbf{v}_1$ and $\mathbf{v}_3$ uniquely maximise the utility within $\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi')$

Given a value set, the `cvx_prune` operation removes any vectors that are dominated or redundant. That is, for every vector that remains after the pruning operation, there is some weight for which it *uniquely* gives the maximal utility for. Given set of vectors $\boldsymbol{\mathcal{V}}$ it is defined as:

$$\texttt{cvx\_prune}(\boldsymbol{\mathcal{V}}) = \{\mathbf{v} \in \boldsymbol{\mathcal{V}} | \exists \mathbf{w} \in \Delta^D . \forall \mathbf{v}' \in \boldsymbol{\mathcal{V}} - \{\mathbf{v}\} . \mathbf{w}^\top \mathbf{v} > \mathbf{w}^\top \mathbf{v}'\}. \tag{2.65}$$

Note that the `cvx_prune` operator computes the values of a minimal convex coverage set, that is, for some set of policies Π' the `cvx_prune` operation satisfies $\texttt{cvx\_prune}(\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi')) = \boldsymbol{\mathcal{V}}\mathbf{als}(CCS_{\text{min}}(\Pi'))$. `cvx_prune` can be implemented using *linear programming* [23], and an example of its operation on a set of vectors can be seen in (b) of Figure 2.11, where `cvx_prune` would prune all points but the blue circles.

$$\frac{1}{2} \times \left\{ \begin{pmatrix} 2 \\ 8 \end{pmatrix}, \begin{pmatrix} 4 \\ 6 \end{pmatrix} \right\} + \frac{1}{2} \times \left\{ \begin{pmatrix} 4 \\ 4 \end{pmatrix}, \begin{pmatrix} 8 \\ 2 \end{pmatrix} \right\} = \left\{ \begin{pmatrix} 3 \\ 6 \end{pmatrix}, \begin{pmatrix} 4 \\ 5 \end{pmatrix}, \begin{pmatrix} 5 \\ 5 \end{pmatrix}, \begin{pmatrix} 6 \\ 4 \end{pmatrix} \right\}$$

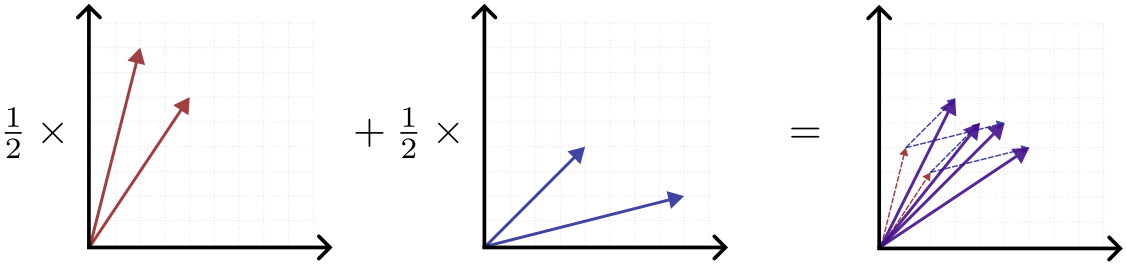

Figure 2.12: An example of arithmetic over vector sets. Two vector sets are shown in red and blue, which are multiplied by $\frac{1}{2}$ using Equation (2.66), and then added using Equation (2.67). On the right the resulting vector set is shown in purple, which are the sum of appropriately scaled red and blue vectors.

2.5.1 Convex Hull Value Iteration

Convex Hull Value Iteration (CHVI) [3] is a tabular dynamic programming algorithm similar to Value Iteration (Section 2.3) that will compute optimal convex hull value sets. In CHVI the value estimates of Value Iteration are replaced by CHVSs, which are estimates of the optimal CHVS. These estimates will be denoted $\hat{\boldsymbol{\mathcal{V}}}_{\text{CHVI}}(s;t)$ and $\hat{\boldsymbol{\mathcal{Q}}}_{\text{CHVI}}(s,a;t)$.

Firstly, to define a multi-objective version of Value Iteration, an arithmetic over vector sets needs to be defined. An example of the following arithmetic is given in Figure 2.12 Given the vector sets $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$, define multiplication by a scalar s, addition with a vector $\mathbf{x}$ and addition between sets as follows:

$$\mathbf{x} + s\boldsymbol{\mathcal{V}} = \{\mathbf{x} + s\mathbf{v} | \mathbf{v} \in \boldsymbol{\mathcal{V}}\} \tag{2.66}$$

$$\boldsymbol{\mathcal{U}} + \boldsymbol{\mathcal{V}} = \{\mathbf{u} + \mathbf{v} | \mathbf{u} \in \boldsymbol{\mathcal{U}}, \mathbf{v} \in \boldsymbol{\mathcal{V}}\}. \tag{2.67}$$

Now to define the multi-objective Bellman backups used in CHVI, let $\hat{\boldsymbol{\mathcal{V}}}^0_{\text{CHVI}}(s;t) = \{\mathbf{0}\}$, where $\mathbf{0} = (0, ..., 0) \in \mathbb{R}^D$. The CHVI backups are then:

$$\hat{\boldsymbol{\mathcal{V}}}^{k+1}_{\text{CHVI}}(s;t) = \texttt{cvx\_prune}\left(\bigcup_{a \in \mathcal{A}} \hat{\boldsymbol{\mathcal{Q}}}^{k+1}_{\text{CHVI}}(s,a;t) \right), \tag{2.68}$$

$$\hat{\boldsymbol{\mathcal{Q}}}^{k+1}_{\text{CHVI}}(s,a;t) = \mathbb{E}_{s' \sim p(\cdot|s,a)}[\mathbf{R}(s,a) + \hat{\boldsymbol{\mathcal{V}}}^{k}_{\text{CHVI}}(s';t+1)]. \tag{2.69}$$

This again parallels the Bellman backups use in single-objective Value Iteration, where the max operation is replaced by the `cvx_prune` operation over the set of achievable values from the current state s: $\bigcup_{a \in \mathcal{A}} \hat{\mathcal{Q}}^{k+1}_{\text{CHVI}}(s, a; t)$.

Once the algorithm has terminated, and a weight $\mathbf{w}$ is provided, a policy can be extracted by computing scalar Q-values from the CHVSs [3]:

$$Q_{\mathbf{w}}(s, a; t) = \max_{\mathbf{q} \in \hat{\mathcal{Q}}_{\text{CHVI}}(s,a;t)} \mathbf{w}^\top \mathbf{q}, \tag{2.70}$$

$$\pi_{\text{CHVI}}(s; t, \mathbf{w}) = \arg\max_{a \in \mathcal{A}} Q_{\mathbf{w}}(s, a; t). \tag{2.71}$$

If the user would rather select a particular value from the CHVS, say $\mathbf{v} \in \hat{\boldsymbol{\mathcal{V}}}_{\text{CHVI}}(s_0; 0)$, rather than provide a weight, one can be computed. Let $\mathbf{v}_\perp$ be a *reference point*, defined at each index as:

$$(v_\perp)_i = \min\{v'_i | \mathbf{v}' \in \hat{\boldsymbol{\mathcal{V}}}_{\text{CHVI}}(s_0; 0)\}. \tag{2.72}$$

The vector that runs from $\mathbf{v}_\perp$ to $\mathbf{v}$ provides an appropriate weighting over objectives for which $\mathbf{v}$ will produce the largest utility out of $\hat{\boldsymbol{\mathcal{V}}}(s_0; 0)$, as is demonstrated in Figure 2.13. Hence, to extract a policy that achieves the value $\mathbf{v}$ the weight $\mathbf{w}$ should be set to:

$$\mathbf{w} = \frac{\mathbf{v} - \mathbf{v}_\perp}{\|\mathbf{v} - \mathbf{v}_\perp\|_2}, \tag{2.73}$$

and then follow $\pi_{\text{CHVI}}(s; t, \mathbf{w})$ as defined in Equation (2.71).

2.6 Sampling From Catagorical Distributions

Question: Maybe heres a good time to ask about some notation stuff. So I used m in MABs section, and I reuse it here. Is that ok? I was trying to keep as much notation unique as possible, but noticed I've reused a couple variables. I was hoping its ok because I'm using them "locally".

Some of the work in this thesis will involve sampling from catagorical distributions, and sometimes it will be helpful to do so efficiently, as in Chapter 4. Let $f : \{C_1, ..., C_m\} \to [0, 1]$ be the probability mass function of a catagorical distribution with m categories. Suppose that we want to sample $c \sim f$. A naive

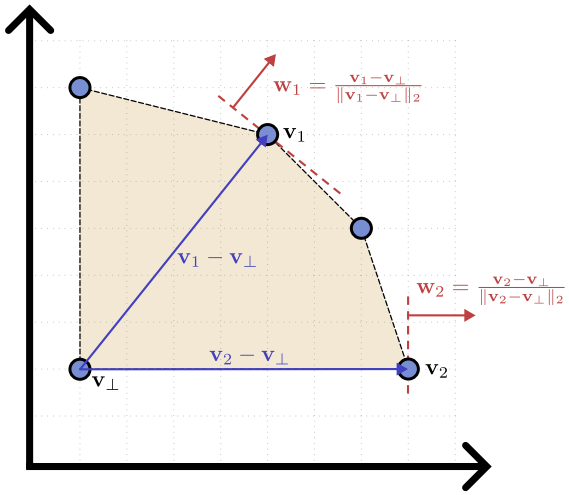

Figure 2.13: A visualisation, reusing the convex hull from Figure 2.11, demonstrating how to compute a weight vector that can be used to extract a specific value from a vector set. The hyperplane running through $\mathbf{v}_i$ with normal $\mathbf{w}_i = \frac{\mathbf{v}_i - \mathbf{v}_\perp}{\|\mathbf{v}_i - \mathbf{v}_\perp\|_2}$ is tangential to the geometric convex hull. Hence $\mathbf{v}_i$ is the optimal value for the weight $\mathbf{w}_i$ and following the CHVI policy in Equation (2.71) will give the desired value $\mathbf{v}_i$.

```
1 def sample_catagorical(f):
2     threshold ~ Uniform(0, 1)
3     i = 0
4     accumulated_mass = 0
5     while (accumulated_mass < threshold):
6         i += 1
7         accumulated_mass += f(i)
8     return i
```

Listing 2.2: Psuedocode for naively sampling from a catagorical distribution.

method to sample from f will take $O(m)$ time, where a continuous uniform random value is sampled, $u \sim \text{Uniform}(0, 1)$, and f is linearly scanned until a probability mass of u has been passed. See Listing 2.2 for psuedocode of the naive method.

However, the *alias method* [31, 30] can instead be used to sample from a catagorical distribution. The alias method uses $O(m)$ preprocessing time to construct an *alias table*, and after can sample from f, using the table, in $O(1)$ time. An alias table consists of m entries, accounting for an mth of the probability mass of f. Each entry takes the form $(c_0, c_1, \texttt{thrsh})$, where $c_0, c_1 \in \{C_1, ..., C_m\}$ are categories and $\texttt{thrsh} \in [0, 1]$ gives the ratio of the $\frac{1}{m}$ probability mass to assign to c_0 and c_1. Sampling from the alias table can be performed by sampling two random numbers, one to index into the table and one to compare against `thrsh`,

```
1 def sample_from_alias_table(alias_table):
2     index ~ Uniform({1, ..., m})
3     (c_0, c_1, thrsh) = alias_table[index]
4     u ~ Uniform(0, 1)
5     if (u <= thrsh):
6         return c_0
7     return c_1
```

Listing 2.3: Psuedocode for sampling from an alias table.

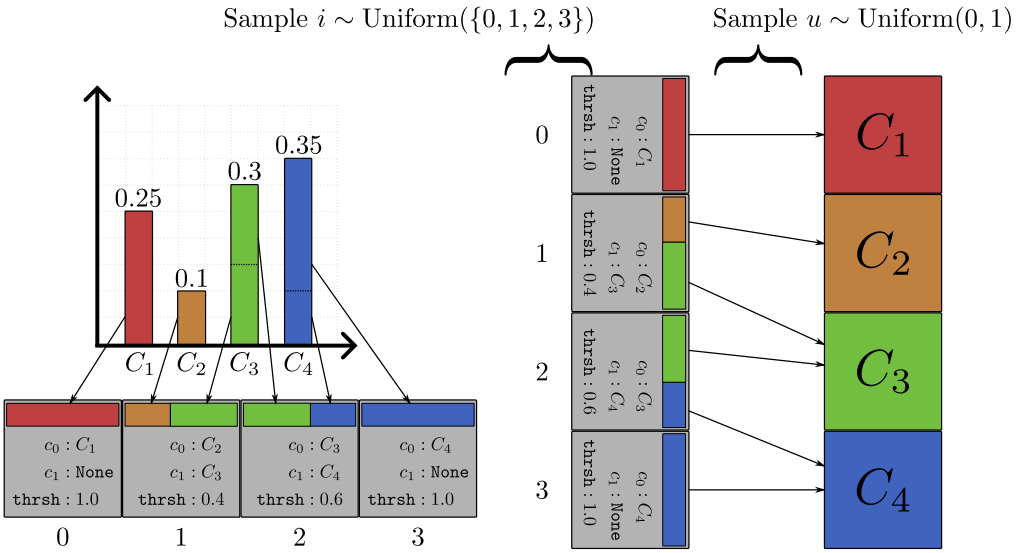

Figure 2.14: Example of building and sampling from an alias table, on a distribution with four categories. Left: shows how the probability mass is split into the four entries of the table. Right: shows how two random variables can be used to sample from the table in $O(1)$ time, where i is used to index into the table, and u is used to compare with `thrsh` and decide which of the two arrows from the entry to follow.

from which $c \sim f$ can be returned by looking it up in the table. Listing 2.3 gives psuedocode for sampling from an alias table, and Figure 2.14 visualises the construction and sampling of an alias table.

Vose [30] proves that an alias table can always be constructed from an arbitrary catagorical distribution, such that the probability of sampling any catagory from the alias table is identical to the probability of sampling it from the original probability mass function.

Psuedocode for constructing an alias table is given in Listing 2.4, following [31, 30]. An alias table can be constructed by keeping two sets of categories, one set for "big" categories that currently have more than $\frac{1}{m}$ mass to be added to the table, and one for "small" catagories that have at most $\frac{1}{m}$ mass left to be added

```
def build_alias_table(f):
    # lists of categories and mass remaining
    small = []
    big = []

    # fill big and small lists
    for c in f:
        if f(c) > 1/m:
            big.append((c, f(c)))
        else:
            small.append((c, f(c)))

    # construct alias table
    alias_table = []
    while len(small) > 0:
        # create entry
        c0, c0_mass_remaining = small.pop()
        if mass_remaining == 1/m:
            alias_table.append((c0, None, 1.0))
            continue
        c1, c1_mass_remaining = big.pop()

        # add entry to table (accounting for 1/mth of total mass)
        thrsh = m * c0_mass_remaining
        alias_table.append((c0, c1, thrsh))

        # put c1 back in lists appropriately
        c1_mass_remaining -= c0_mass_remaining
        if c1_mass_remaining > 1/m:
            big.append((c1, c1_mass_remaining))
        else:
            small.append((c1, c1_mass_remaining))

    return alias_table
```

Listing 2.4: Psuedocode for constructing an alias table.

to the table. To create a new entry in the table, take one category from the small set, and take a category from the big set to complete the remainder of the $\frac{1}{m}$ mass that the small set didn't fill. After creating a new entry, move the big category to the little set if it no longer has more than $\frac{1}{m}$ mass to be added.

3
Literature Review

Contents

TODO: currently this is a copy and paste of what I originally wrote for background chapter 2. Deleted parts which are irrelevant for litreview here (and vice versa for the background section).

TODO: I'm also going to use this as a space to paste papers I should write about as they come up while writing later chapters

3.1 Reinforcement Learning

TODO: Intro should say that look at Sutton and Barto and something else for deep RL, for a more complete overview. Here we will just discuss papers that consider entropy in their work, as thats the most relevant part for this thesis.

TODO: list

- Talk about entropy and some of that work (probably a subsection)

TODO: In the entropy bit talk add this, removed from ch2: Note that there are other forms of entropy, such as relative and Tsallis entropy, which can be used in place of Shannon entropy (TODO cite). For the work considered in this thesis, the other forms of entropy can be used by replacing the definition of $\mathcal{H}$ by the relevant definition.

TODO: talk about some deep learning methods here

3.2 Multi-Armed Bandits

TODO: Maybe dont need to cover this in litrev, but should talk about exploring bandits, UCT and contextual bandits either in background or in litrev

TODO: linUCB for contextual bandits

3.3 Trial-Based Heuristic Tree Search and Monte-Carlo Tree Search

3.3.1 Trial Based Heuristic Tree Search

TODO: THTS paper, talk about the differences that the paper has to our presentation of `THTS++`

TODO: cut from ch2: Finally we will briefly point out the differences between `THTS++`and the original THTS schema in subsection

TODO: talk about how these methods are still relevant with deep learning because of algorithms that use both, such as alpha zero

3.3.2 Monte-Carlo Tree Search

TODO: list

- Talk about the things that are ambiguous from literature (e.g. people will just say UCT, which originally presented doesn't run in `mcts_mode`, but often assumed it does)

- Should talk about multi-armed bandits here?

https://inria.hal.science/inria-00164003/document

https://pdf.sciencedirectassets.com/271585/1-s2.0-S0004370211X0005X/1-s2.0-S000437021100052X/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEOP

TODO: removed from ch2, this can be litrev: add polynomial UCT here? and or prioritised UCT from alpha go here?

TODO: removed from ch2: add stuff about regret here, give the O(logn) bound for UCT, and also talk about the papers that

3.3.3 Maximum Entropy Tree Search

TODO: MENTS, RENTS and TENTS

3.4 Multi-Objective Reinforcement Learning

TODO: list

- Should talk about multi-objective and/or contextual multi-armed bandits here?
- Bunch of the work covered in recent MORL survey [12]
- Mention some deep MORL stuff, say that this work (given AlphaZero) is adjacent work

TODO: removed from ch2: comment here or in literature review about there being more types of scalariation function that arent necessarily weighted by a weight, and ESR vs SER stuff

TODO: talk about more of the MORL survey, including some of the other motivating scenarios

TODO: Talk about the better way of doing CHVI? https://www.jmlr.org/papers/volume13/lizot and Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. Also by Lizotte. But also say that its approximate for $D > 2$

and so we stick with the slower one? But the CHVS operations can be computed more efficiently in $D = 2$ using this stuff. Did it for the python implementation.

TODO: also need to talk about the MO sequential decision survey. Define Pareto Front. Say that the Pareto Front "Definition 3 If the utility function u is any monotonically increasing function, then the Pareto Front (PF) is the undominated set [Roijers et al., 2013]:"

TODO: need to talk about this because some prior/related work uses PF rather than CH

TODO: cover some inner loop and outer loop things? Roijers computing CCS work? See what MORL survey says about it

TODO: Maybe talk about the *witness algorithm* for Partially Observable MDPs as its very similar to CHVI stuff

3.5 Multi-Objective Monte Carlo Tree Search

TODO: I think this whole section can just go in litrev

TODO: list

- Define the old methods (using the CH object methods, so clear that not doing direct arithmetic)
- Mention that old method could be written using the arithmetic of CHMCTS (but they don't)
- Different flavours copy UCT action selection, but with different variants
- Link back to contributions and front load our results showing that all of the old methods don't explore correctly

TODO: There has been some prior work in multi-objective MCTS which we will outline here

TODO: Write out implementations of prior works using THTS

TODO: define pareto front

TODO: perez algorithms

TODO: https://ieeexplore.ieee.org/document/8107102

TODO: https://www.roboticsproceedings.org/rss15/p72.pdf TODO: https://arxiv.org/abs/211

TODO: https://proceedings.mlr.press/v25/wang12b/wang12b.pdf

TODO: https://ifmas.csc.liv.ac.uk/Proceedings/aamas2021/pdfs/p1530.pdf TODO: https://link.springer.com/article/10.1007/s10458-022-09596-0

TODO: Some stuff we wrote that didnt use in ch2. Might want to talk about it with eval:

Moreover, in the case of MCTS algorithms, by having the user select a preferred policy, it implicitly forces the user to chose a preference over the objectives, as the policy corresponds to a weight vector that it is optimal for. As MCTS algorithms are often used in an online fashion, where planning is interleaved with execution, this implicitly selected weight can be used for any online execution needed, effectively reducing the multi-objective problem into a single-objective problem.

4

Monte Carlo Tree Search With Boltzmann Exploration

Contents

4.1 Introduction

TODO: list

- high level overview of DENTS work
- discuss how DENTS answers the research questions from introduction chapter
- state clearly that we're in single objective land here
- Comment about work exploring multi-armed bandits motivating this work

4.2 Boltzmann Search

TODO: list

- Recall MENTS
- Define BTS using THTS functions
- Define DENTS using THTS functions
- Discuss alias method variant (and complexity analysis) in a subsection?

4.3 Toy Environments

TODO: list

- Define D-chain stuff from the paper
- Define the D-chain with entropy trap
- Front load some results still

4.4 Theoretical Results

TODO: list

- add theoretical results

4.5 Empirical Results

TODO: list

- DChain
- GridWorlds
- Go

4.6 Full Results

TODO: there's a lot of figures for the D-chain environment, work out how to best fit them in? Or put them in this seperate section?

5 Convex Hull Monte Carlo Tree Search

Contents

5.1 Introduction

TODO: list

- high level overview of CHMCTS work
- discuss how CHMCTS answers the research questions from introduction chapter
- moving into multi-objective land now
- Comment about CHVI and prior MOMCTS work motivating this

5.2 Contextual Tree Search

TODO: list

- Discuss need for context when doing multi-objective tree Search
  - Use an example env where left gives (1,0) and right gives (0,1), optimal policy picks just left or just right, but hypervolume based methods wont
  - Use previous work on these examples and show they dont do well bad
- Discuss how UCT = running a non-stationary UCB at each node, so given above discussion, there is work in contextual MAB
- Introduce contextual regret here

5.3 Contextual Zooming for Trees

TODO: list

- Give contextual zooming for trees algorithm
- Discussion on the contextual MAB to non-stationary contextual MAB stuff (CZT is to CZ what UCT is to UCB) (and what theory carry over)

5.4 Convex Hull Monte Carlo Tree Search

TODO: list

- Give convex hull monte carlo tree search
- Contextual zooming with the convex hull backups

TODO: Repeat definition of cprune, and add to abbreviations there

5.5 Results

TODO: list

- Results from CHMCTS paper
- Get same plots from C++ code, but compare expected utility, rather than the confusing hypervolume ratio stuff

6

Simplex Maps for Multi-Objective Monte Carlo Tree Search

Contents

6.1 Introduction

TODO: list

- high level overview of simplex maps work
- discuss how simplex maps answer the research questions from introduction chapter
- staying in multi-objective land now
- Motivated by CHMCTS being slow

6.2 Simplex Maps

TODO: list

- Define simplex map interface
- Give details on how to efficiently implement the interface with tree structures
- (Good diagram is everything here I think)

6.3 Simplex Maps in Tree Search

TODO: list

- Come up with better title for section
- Use simplex maps interface to create algorithms from the dents work
- Give a high level idea of what δ parameter is (used in theory section)

6.4 Theoretical Results

TODO: list

- Convergence can build ontop of DENTS results
- Runtime bounds (better than $O(2^D)$ which is what using convex hulls has)
- Simplex map has a diameter δ (i.e. the furthest away a new context could be from a point in the map)
- Bounds can then come from that diameter (which is a parameter of the simplex map/algorithm) and DENTS results

6.5 Empirical Results

TODO: list

- Results from MO-Gymnasium
- Compare algorithms using expected utility

7 Conclusion

Contents

TODO: Something about we'll conclude by looking back at contributions and possible future work.

7.1 Summary of Contributions

TODO: go through each of the research questions and contributions, and write about how the work answers the research questions

7.2 Future Work

TODO: outline some avenues of potential future work

Appendices

A

List Of Appendices To Consider

- Multi Armed Bandits, maybe
- MAybe from of the things in background are more appropriate as appendices?

Bibliography

[1] Bruce Abramson. Expected-outcome: A general model of static evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 12(2):182–193, 1990.

[2] P Auer. Finite-time analysis of the multiarmed bandit problem, 2002.

[3] Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*, pages 41–47, 2008.

[4] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995.

[5] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN 9780486428093.

[6] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[7] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory: 20th International Conference, ALT 2009, Porto, Portugal, October 3-5, 2009. Proceedings 20*, pages 23–37. Springer, 2009.

[8] Tristan Cazenave, Flavien Balbo, Suzanne Pinson, et al. Monte-carlo bus regulation. In *12th international IEEE conference on intelligent transportation systems*, volume 340345, 2009.

[9] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280, 2007.

[10] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

[11] Eric A Hansen and Shlomo Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[12] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

[13] Thomas Keller and Patrick Eyerich. Prost: Probabilistic planning based on uct. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 119–127, 2012.

[14] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 135–143, 2013.

[15] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 681–690, 2008.

[16] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[17] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1:1–22, 2006.

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

[19] Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. A0c: Alpha zero in continuous action space. *arXiv preprint arXiv:1805.09613*, 2018.

[20] Mausam Natarajan and Andrey Kolobov. *Planning with Markov decision processes: An AI perspective*. Springer Nature, 2022.

[21] Michael Painter. THTS++. URL `https://github.com/MWPainter/thts-plus-plus`. Available at `https://github.com/MWPainter/thts-plus-plus`.

[22] Herbert Robbins. Some aspects of the sequential design of experiments. 1952.

[23] Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 52:399–443, 2015.

[24] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[25] David Silver and Joel Veness. Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, 23, 2010.

[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[27] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[28] Aleksandrs Slivkins. Contextual bandits with similarity information. In *Proceedings of the 24th annual Conference On Learning Theory*, pages 679–702. JMLR Workshop and Conference Proceedings, 2011.

[29] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

[30] Michael D Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on software engineering*, 17(9):972–975, 1991.

[31] Alastair J Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[32] Chenjun Xiao, Ruitong Huang, Jincheng Mei, Dale Schuurmans, and Martin Müller. Maximum entropy monte-carlo planning. *Advances in Neural Information Processing Systems*, 32, 2019.