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 Notation

Global Notation

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

Markov Decision Processes (Section 2.1)

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

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

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

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: $R(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

$\mathcal{S}$ . . . . . 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.

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

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

Reinforcement Learning (Section 2.2)

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.

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

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

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

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

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 $a_t = a$.

$\hat{Q}^k$ . . . . . The estimate of the optimal value function after k iterations of Value Iteration.

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

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 $a_t = a$.

$\hat{Q}^k$ . . . . . The estimate of the optimal soft value function after k iterations of soft Value Iteration.

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

$\hat{V}^k$ . . . . . The estimate of the optimal value function after k iterations of Value Iteration.

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

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-Armed Bandits (Section 2.3)

x . . . . . TODO: define here and move into correct place in the list

y . . . . . TODO: define here and move into correct place in the list

f_i . . . . . TODO: define here and move into correct place in the list

x^m . . . . . TODO: define here and move into correct place in the list

y^m . . . . . TODO: define here and move into correct place in the list

$\bar{y}_i^m$ . . . . . TODO: define here and move into correct place in the list

N . . . . . TODO: define here and move into correct place in the list

μ_i . . . . . TODO: define here and move into correct place in the list

μ^* . . . . . TODO: define here and move into correct place in the list

$\texttt{cum\_regr}_{\text{MAB}}$. TODO: define here and move into correct place in the list

$\pi(m)$ . . . . . TODO: define here and move into correct place in the list. Maybe keep it as this and ask question

m . . . . . TODO: define here and move into correct place in the list

$\pi_{\text{UCB}}(m)$ . . . . TODO: define here and move into correct place in the list

π^m . . . . . TODO: define here and move into correct place in the list

ψ^m . . . . . TODO: define here and move into correct place in the list

w . . . . . TODO: define here and move into correct place in the list

w^m . . . . . TODO: define here and move into correct place in the list

$\mu_{w,i}$ . . . . . TODO: define here and move into correct place in the list

μ^*_w . . . . . TODO: define here and move into correct place in the list

W . . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

Trial Based Heuristic Tree Search and Monte Carlo 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.

`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 to the MDPs time horizon H.

$N(s)$ . . . . . 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)`.children` 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)`.children` The set of decision nodes that are children of `node`(s).

`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.

$\mathcal{T}$ . . . . . The `THTS++` search tree. $\mathcal{T} \subseteq \mathcal{S} \cup \mathcal{S} \times \mathcal{A}$.

$\mathcal{T}^k$ . . . . . The `THTS++` search tree after k trials have been run.

$\hat{V}_{\text{init}}$ . . . . . The heuristic value function used in `THTS++`, used to initialise the value of a new decision node.

. . . . .	TODO: Below is stuff from Section 2.4.X, which need to write up and sort into the list
n	Number of trials run.
T	Computation time limit.
$\bar{V}_{\text{UCT}}$	TODO: define here and move into correct place in the list
$\bar{Q}_{\text{UCT}}$	TODO: define here and move into correct place in the list
π_{UCT}	TODO: define here and move into correct place in the list
b_{UCT}	TODO: define here and move into correct place in the list
$\bar{V}_{\text{MCTS}}$	TODO: define here and move into correct place in the list
$\bar{Q}_{\text{MCTS}}$	TODO: define here and move into correct place in the list
π_{MCTS}	TODO: define here and move into correct place in the list
b_{MCTS}	TODO: define here and move into correct place in the list
$\tilde{V}_\theta$	TODO: define here and move into correct place in the list
$\hat{V}^{\pi_{\text{rollout}}}$	TODO: define here and move into correct place in the list
π_{rollout}	TODO: define here and move into correct place in the list
$\hat{V}_{\text{MENTS}}$	TODO: define here and move into correct place in the list
$\hat{Q}_{\text{MENTS}}$	TODO: define here and move into correct place in the list
π_{MENTS}	TODO: define here and move into correct place in the list
α_{MENTS}	TODO: define here and move into correct place in the list
λ_s	TODO: define here and move into correct place in the list
ϵ	TODO: define here and move into correct place in the list

Multi-Objective Reinforcement Learning (Section 2.5)

$\boldsymbol{\mathcal{M}}$	TODO: define here and move into correct place in the list
τ	TODO: define here and move into correct place in the list
$\tau_{:h}$	TODO: define here and move into correct place in the list
$\mathbf{V}^\pi$	TODO: define here and move into correct place in the list
$\mathbf{Q}^\pi$	TODO: define here and move into correct place in the list
Δ^D	TODO: define here and move into correct place in the list
u	TODO: define here and move into correct place in the list

$\mathbf{w}$ . . . . . TODO: define here and move into correct place in the list

D . . . . . TODO: define here and move into correct place in the list

Π . . . . . TODO: define here and move into correct place in the list

$U(\Pi; u)$ . . . . . TODO: define here and move into correct place in the list

u_{lin} . . . . . TODO: define here and move into correct place in the list

$CH(\Pi)$ . . . . . TODO: define here and move into correct place in the list

$CS(\Pi; u)$ . . . . TODO: define here and move into correct place in the list

$CCS(\Pi)$ . . . . TODO: define here and move into correct place in the list

$\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi)$ . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

Sampling From Catagorical Distributions 2.6)

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

. . . . . TODO: define here and move into correct place in the list

List of Abbreviations

Markov Decision Processes (Section 2.1)

MDP . . . . . Markov Decision Process

Reinforcement Learning (Section 2.2)

MDP . . . . . Markov Decision Process.

MENTS . . . . Maximum ENtropy Tree Search.

Multi-Armed Bandits (Section 2.3)

MAB . . . . . TODO: define here and move into correct place in the list

UCB . . . . . TODO: define here and move into correct place in the list

EMAB . . . . TODO: define here and move into correct place in the list

CMAB . . . . TODO: define here and move into correct place in the list

Trial Based Heuristic Tree Search and Monte Carlo Tree Search (Section 2.4)

MCTS . . . . . Monte Carlo Tree Search.

MENTS . . . . Maximum ENtropy Tree Search.

THTS . . . . . Trial-based Heuristic Tree Search.

`THTS++` . . . . . An extension of THTS used in this thesis.

UCT . . . . . Upper Confidence Bound applied to Trees.

UCT-MCTS . An MCTS variant of Upper Confidence Bound applied to Trees.

Multi-Objective Reinforcement Learning (Section 2.5)

CHVI . . . . . Convex Hull Value Iteration.

CHVS . . . . . Convex Hull Value Set.

MOMDP . . . Multi-Objective Markov Decision Process.

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

TODO: Introduce that going to introduce notation and give the building blocks this thesis builds off

TODO: comment about notation like charlies, $\mathbb{1}$ for example. Also trends of notation that we use: $\tilde{V}$ is (neural net) function approx, $\bar{V}$ is sample average, $\hat{V}$ is estimating something, $\mathbf{V}$ is a vector, $\mathcal{V}$ is a set, and these notations can be combined, $\boldsymbol{\mathcal{V}}$ is a set of vectors, typeface text $\mathtt{V}$ typically refers to things that

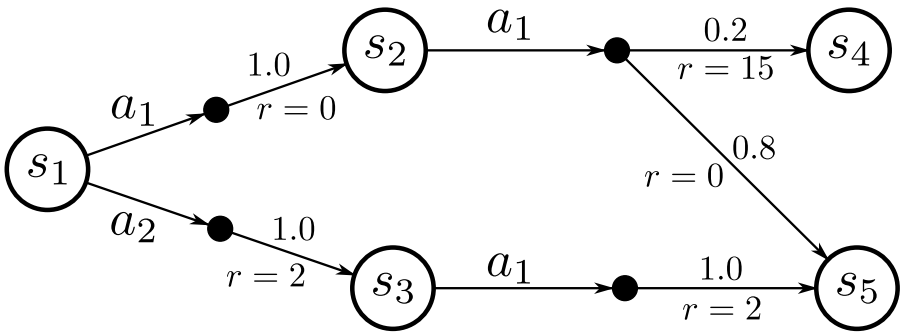

Figure 2.1: An example MDP $\mathcal{M}$, where TODO: description of MDP drawn are more implementation details

2.1 Markov Decision Processes

In this section *Markov Decision Processes* (MDPs) are introduced, along with related defintions 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 recieves 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. TODO: at points we may allude to some designs and ideas that can generalise to more general forms of markov decision processes, but it is not the main focus here.

Definition 2.1.1. *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 2.1. 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 2.1.2. *The set of* successor states $\mathrm{Succ}(s, a)$ *of a state-action pair* (s, a)*, with respect to an MDP, is defined as:*

$$\mathrm{Succ}(s, a) := \{s' \in \mathcal{S} | p(s'|s, a) > 0\}. \tag{2.1}$$

Additionally, let $s' \sim \mathrm{Succ}(s, a)$ *be a shorthand for* $s' \sim p(\cdot|s, a)$*.*

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 2.1.3. *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* $a \in \mathcal{A}.\pi(a|s) \geq 0$ *.*

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 TODO: ref these simulations are used to incrementally build a search tree.

Definition 2.1.4. *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* $s_{t+1} \sim \text{Succ}(s_t, a_t)$*.*

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,*
- *finally, 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.*

TODO: citations in this section? Puttman?

2.2 Reinforcement Learning

This section serves as a brief introduction to fundamental concepts in Reinforcement Learning, and motivates . The field of Reinforcement Learning considers an agent that has to learn how to make decisions by interacting with its environment (Figure 2.2). 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.

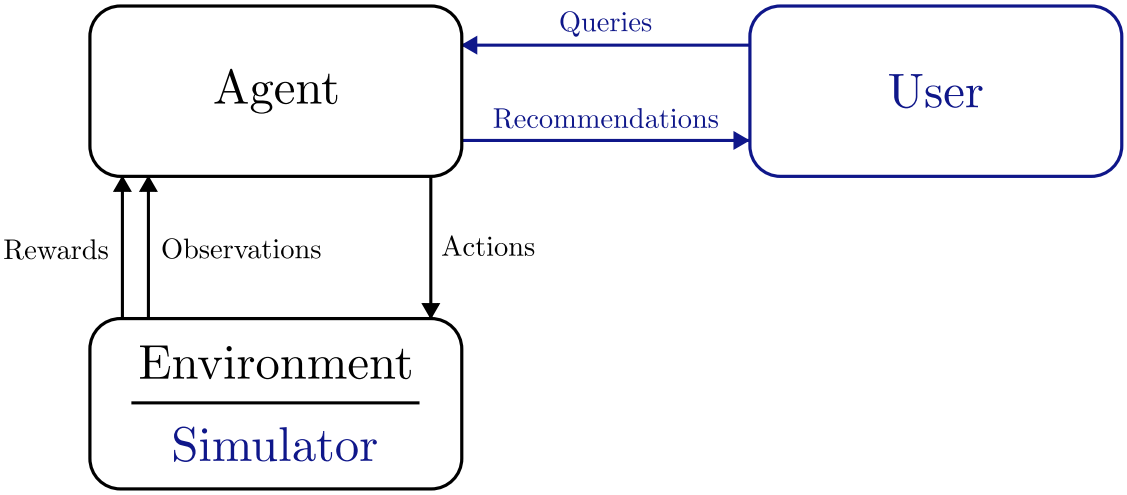

Figure 2.2: An overview of reinforcement learning. Left: depicts the typical scenario where an agent can perform actions in an environment and is given feedback in the form of observations and rewards. Right: shows a similar scenario where the agent instead plans using a simulated environment and is then queried for recommendations about how to act in the real environment. TODO: update scale and caption for updated fig

Classically the agent is considered to interact with its environment directly TODO: cite sutton and barto?, and thus must make a trade-off between exploring new strategies and exploiting learned strategies, commonly known as the *exploration-exploitation trade-off*. If an agent were to try to only exploit, then it may not discover better strategies, and if an agent only explores, then it may miss out on the opportunity to exploit the best known strategy and obtain greater rewards.

Also depicted in Figure 2.2 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.

In this thesis, the environment will always take the form of an MDP (Defintion 2.1.1), and observations will always be *fully-observable*, meaning that the agent is provided with full access to the states of the MDP. TODO: comment about partially observable? and cite?. Moreover, a lot of the work in this thesis concerns the simulation scenario from Figure 2.2, and motivates our research questions around exploration: **Q1 - Exploration**.

Following on from Section 2.1, the remainder of this section defines *value functions* and the objectives of reinforcement learning, covers *Value Iteration*, a tabular dynamic programming approach to reinforcement learning, and finally Subsection 2.2.1 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 2.2.1. *The* value *of a policy π from state s at time t is:*

$$V^{\pi}(s;t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=t}^{H-1} r_t \middle| s_t = s\right]. \tag{2.2}$$

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

$$Q^{\pi}(s,a;t) = R(s,a) + \mathbb{E}_{s'\sim\text{Succ}(s,a)}[V^{\pi}(s';t+1)]. \tag{2.3}$$

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

Definition 2.2.2. *The* Optimal (Q-)Value *of a state(-action pair) is defined as:*

$$V^{*}(s;t) = \max_{\pi} V^{\pi}(s;t) \tag{2.4}$$

$$Q^{*}(s,a;t) = \max_{\pi} Q^{\pi}(s,a;t). \tag{2.5}$$

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

Definition 2.2.3. *The* (standard) reinforcement learning objective function $J(\pi)$ *is defined as:*

$$J(\pi) = V^{\pi}(s_0;0). \tag{2.6}$$

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, can be shown to be deterministic TODO: under conditions, finite? TODO: cite:

Definition 2.2.4. *The* optimal (standard) policy π^* *is the policy maximising the standard objective function:*

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.7}$$

or equivalently, the optimal standard policy can be found from the optimal Q-value function:

$$\pi^*(s) = \arg\max_{a} Q^*(s, a). \tag{2.8}$$

TODO: all refs below I think can be Sutton and Barto, or [Bellman 1957] Bellman, R. 1957. Dynamic Programming. Princeton, NJ, USA: Princeton University Press, 1 edition.

It can be shown that the optimal (Q-)value functions satisfy the *Bellman equations* TODO: refs :

$$V^*(s; t) = \max_{a \in \mathcal{A}} Q^*(s, a; t), \tag{2.9}$$

$$Q^*(s, a; t) = R(s, a) + \mathbb{E}_{s' \sim \text{Succ}(s,a)}[V^*(s'; t + 1)]. \tag{2.10}$$

The Bellman equations admit a *dynamic programming* approach which can be used to computer the optimal value functions, known as *Value Iteration* TODO: ref. 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:

$$\hat{V}^{k+1}(s; t) = \max_{a \in \mathcal{A}} \hat{Q}^{k+1}(s, a; t), \tag{2.11}$$

$$\hat{Q}^{k+1}(s, a; t) = \mathbb{E}_{s' \sim \text{Succ}(s,a)}[R(s, a) + \hat{V}^k(s'; t + 1)]. \tag{2.12}$$

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 Bellman equations can be shown to be *contraction operators* TODO: refs, which can be used to show that $V^k \to V^*$ as $k \to \infty$, and when the state and action spaces are discrete, they will converge in a finite number of iterations. TODO: refs

2.2.1 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. [**?**].

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

$$\mathcal{H}(\pi(\cdot|s)) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log\pi(a|s)] = \sum_{a\in\mathcal{A}} \pi(a|s)\log\pi(a|s). \tag{2.13}$$

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*.

Soft values are defined as follows:

Definition 2.2.5. *The* soft value *of a policy π from state s at time t is:*

$$V^{\pi}_{\text{sft}}(s;t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=t}^{H-1} r_t + \alpha\mathcal{H}(\pi(\cdot|s_i))\middle| s_t = s\right]. \tag{2.14}$$

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

$$Q^{\pi}_{\text{sft}}(s,a;t) = R(s,a) + \mathbb{E}_{s'\sim p(\cdot|s,a)}[V^{\pi}_{\text{sft}}(s';t+1)]. \tag{2.15}$$

Similarly, optimal soft (Q-)values can be defined by taking the maximum over policies again:

Definition 2.2.6. *The* Optimal soft (Q-)Value *of a state(-action pair) is defined as:*

$$V^{*}_{\text{sft}}(s;t) = \max_{\pi} V^{\pi}_{\text{sft}}(s;t), \tag{2.16}$$

$$Q^{*}_{\text{sft}}(s,a;t) = \max_{\pi} Q^{\pi}_{\text{sft}}(s,a;t). \tag{2.17}$$

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

Definition 2.2.7. *The* maximum entropy (or soft) reinforcement learning objective function $J_{\text{sft}}(\pi)$ *is defined as:*

$$J_{\text{sft}}(\pi) = V^{\pi}_{\text{sft}}(s_0; 0). \tag{2.18}$$

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 with knowledge of the optimal soft value and soft Q-value functions, the optimal soft policy is known:

Definition 2.2.8. *The* optimal soft policy π^*_{sft} *is the policy maximising the soft objective function:*

$$\pi^*_{\text{sft}} = \arg\max_{\pi} J_{\text{sft}}(\pi). \tag{2.19}$$

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

$$\pi^*_{\text{sft}}(a|s;t) = \exp\left(\left(Q^*_{\text{sft}}(s,a;t) - V^*_{\text{sft}}(s;t)\right)/\alpha\right). \tag{2.20}$$

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 (2.9) and (2.10) 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.

Similarly to standard reinforcement learning, it can be shown that the optimal soft (Q-)value functions satisfy the *soft Bellman equations* [**?**]:

$$V^*_{\text{sft}}(s;t) = \alpha \log \sum_{a \in \mathcal{A}} \exp\left(Q^*_{\text{sft}}(s,a;t)/alpha\right), \tag{2.21}$$

$$Q^*_{\text{sft}}(s,a;t) = R(s,a) + \mathbb{E}_{s' \sim \text{Succ}(s,a)}[V^*_{\text{sft}}(s';t+1)]. \tag{2.22}$$

Figure 2.3: An example K-armed bandit problem, where TODO: description of image

Again, similarly to standard reinforcement learning, *soft Bellman backups* can be defined that admit an analogous algorithm *Soft Value Iteration* [**?**]:

$$\hat{V}^{k+1}_{\text{sft}}(s;t) = \alpha \log \sum_{a \in \mathcal{A}} \exp\left(\hat{Q}^{k+1}_{\text{sft}}(s,a;t)/alpha\right), \tag{2.23}$$

$$\hat{Q}^{k+1}_{\text{sft}}(s,a;t) = R(s,a) + \mathbb{E}_{s' \sim \text{Succ}(s,a)}[\hat{V}^{k}_{\text{sft}}(s';t+1)]. \tag{2.24}$$

2.3 Multi-Armed Bandits

TODO: This section I still need to go through more finely, and add some more content

TODO: list

- $R(s,a)$ is a random variable in MAB literature, but we're assuming it's a fixed value in RL
- Multi-Armed Bandits routines algos
- Exploring Bandits routines and algos
- Contextual Bandits routines and algos

TODO: Some waffel intro about this being used for decision making under uncertainty, but doesnt encorporate the sequential part of it. However, necessary for foundations, because much of sequential work builds ontop of this work

TODO: Also say that this can be viewed as a single state MDP with $\mathcal{S} = \{\perp\}$ and $\mathcal{A} = \{1, ..., K\}$

TODO: Talk about relevance to sequential decision making, and tree search things, becasue they usually extend MAB algorithms

In the K-armed bandit problem, an agent is tasked with iteratively selecting one of K arms, originally introduced by TODO: author TODO: cite MAB orig. In each iteration, once an arm is selected, say $x \in \{1, ..., K\}$, and a reward y is returned according to an unknown probability distribution f_x. Let n be the number of rounds of this game that are played, and let $\mu_i = \mathbb{E}[y|x = i]$ be the expected reward of pulling arm i.

When analysing an algorithmic strategy for Multi-Armed Bandits (MABs), a quantity known as *regret* is commonly used, which compares the cumulative reward obtained, compared to the maximal reward that could be obtained with full knowledge of $\{f_i\}$.

TODO: some sentence, the process of a MAB is as follows:

- for m in $\{1, ..., n\}$:
- agent selects arm x^m according to policy π
- agent recieves reward $y^m \sim f_{x^i}$

Definition 2.3.1. *The* (cumulative) regret *of the agent in the above process is defined as:*

$$\texttt{cum\_regr}_{\text{MAB}}(\pi, n) = n\mu^* - \sum_{i=1}^{n} y^i, \tag{2.25}$$

where $\mu^* = max_i \mu_i$.

To theoretically analyse algorithms for MAB problems, the quantity of expected regret, $\mathbb{E}[\texttt{cum\_regr}_{\text{MAB}}]$ is considered. In TODO: cite it is shown using information theory that there is a lower bound on the expected regret that an agent can achieve of $\Sigma(\log N)$, and in TODO: cite ucb the Upper Confidence Bound (UCB) algorithm is introduced, achieving a matching upper bound of $O(\log N)$.

Let $N^m(j)$ be the number of times that arm i has been pulled after m rounds, $\bar{y}_i^m$ is the sample average of rewards recieved when pulling arm i:

$$\bar{y}_i^m = \frac{1}{m}\sum_{j=1}^{m} y^j \mathbb{1}[x^j = i]. \tag{2.26}$$

Then the arm selected by the UCB algorithm on the mth round is given by:

$$\pi_{\text{UCB}}(m) = \arg\max_{j\in\{1,..,K\}} \bar{y}_j^m + b_{\text{UCB}}\sqrt{\frac{\log(m)}{N(j)}} \tag{2.27}$$

TODO: define b_{UCB}

TODO: talk about explortaiton-exploitation trade off

TODO: make sure say every arm pulled once to initialise

2.3.1 Exploring Bandits

In the pure exploration K-armed bandit problem TODO: cite, the game is changed slighly. The agent still gets to pull an arm each round, but after it recieves a reward each round it is given the opportunity to output a *recommendation*. In exploring multi-armbed bandits (EMABs), 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 change seperates 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.

TODO: some sentence, the process of a EMAB is as follows:

- for m in $\{1, ..., n\}$:
- agent selects arm x^m according to policy π^m
- agent recieves reward $y^m \sim f_{x^m}$
- agent outputs a recommendation policy ψ^m

Under this regime, 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 an *instantaneous regret*, TODO: define instantaneous regret which would come from following the recommendation policy.

Definition 2.3.2. *The* simple regret *of following the recommendation policy ψ^m on the mth round is:*

$$\texttt{sim\_regr}_{\text{EMAB}}(m) = \mathbb{E}_{i\sim\psi^m}[\mu^* - \mu_i]. \tag{2.28}$$

TODO: talk about the bounds they show in their work

TODO: describe the algorithm which is pulling arms 1,2,...,K,1,2,...,K,1,..., and so on, and then recommending the arm that has the best empirical average.

2.3.2 Contextual Bandits

In the contextual K-armed bandit problem TODO: cite, on each round the algorithm is given a context $w \in \mathcal{W}$, which the algorithm does not choose. If contextual multi-armed bandit problems (CMABs), the distribution that the rewards are drawn from now depend on w, written $f_{w,i}$ for context w for arm i.

TODO: some more words around when this is useful and why, and define CMABs

TODO: some sentence, the process of a CMAB is as follows:

- for m in $\{1, ..., n\}$:
- agent recieves context w^m
- agent selects arm x^m according to policy π
- agent recieves reward $y^m \sim f_{w^m,x^m}$

Similar to MABs and EMABs, the notion of regret is used to analyse CMABs. Specifically, *contextual regret* is defined similarly to cumulative regret TODO: ref, while taking into account the contexts drawn.

Definition 2.3.3. *The* (cumulative) contextual regret *of the agent in the above process is defined as:*

$$\texttt{ctx\_regr}_{\text{CMAB}}(\pi, n) = \sum_{i=1}^{n} \mu^*_{w^i} - y^i, \tag{2.29}$$

where $\mu^*_w = \max_i \mu_{w,i}$.

TODO: add definitions for $\mu_{w,i}$

TODO: Talk about contextual zooming precursor

TODO: Write about contextual zooming

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

`THTS++` [3] is introduced in this section, which is an open-source, parallelised extension of the Trial-based Heuristic Tree Search (THTS) schema [2]. `THTS++` is a generalisation of what is commonly meant by Monte Carlo Tree Search (MCTS), which is presented in Section 2.4.4 in terms of `THTS++`. This thesis considers fully-observable environments, but `THTS++` can be used to implement algorithms that consider *partially-observable* environments, such as POMCP TODO: expand accronym, cite. In Section 2.4.3 the standard Upper Confidence Bound applied to Trees (UCT) algorithm is presented in terms of `THTS++` , and in Section 2.4.5 the Maximum ENtropy Tree Search (MENTS) algorithm is presented.

TODO: want a second para saying things like: mcts is used as a heuristic method to solve MDPs; useful for when state space is large enough that using a tabular method like value iteration is too slow; often used online, to interleve planning an executing; they allow for statistical analysis and are more interperable than deep learning methods; and they can still be used with deep learning methods in algorithms like alphazero

This thesis uses the term MCTS to refer to any algorithm that builds a search tree using *Monte Carlo trials*, so any algorithm defined in terms of `THTS++` is an MCTS algorithm.

2.4.1 Notation

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

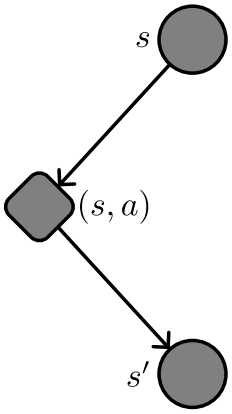

Figure 2.4: Tree diagrams notation, circles will be used to denote *decision nodes* that are associated with states, diamonds will be used to denote *chance nodes* that are associated with state-action pairs and arrows are used to denote parent/child relationships in the tree.

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

In `THTS++` trees consist of *decision nodes* and *chance nodes*. Decision nodes output actions that can be taken by the agent, and chance nodes output *outcomes* that may be random and may depend on the action taken. As such, each decision node has an associated *state* and each chance node has an associated *state-action pair*. Figure 2.4 shows how decision and chance nodes will be depicted in diagrams.

A search tree $\mathcal{T}$ is built using Monte Carlo *trials*. Each trial is split into two phases: the *selection phase* where a trajectory is sampled using a *search policy*, and any newly visited states (and state-action pairs) are added to $\mathcal{T}$ as decision nodes (and chance nodes); and the *backup phase* where value estimates stored in the tree structure are updated. An overview of one trial in `THTS++` is given in Figure 2.5.

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 $s', a' \in \tau\colon h$ 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 the number of times s and (s, a) appear in trajectories sampled in `THTS++` trials.

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 value stored at node $\texttt{node}(s, a)$.

Additionally, let $\texttt{node}(s)\texttt{.children}$ be the set of chance nodes that are children of $\texttt{node}(s)$, and likewise, $\texttt{node}(s, a)\texttt{.children}$ is the set of decision nodes that are children of $\texttt{node}(s, a)$.

`THTS++` introduces the idea of `mcts_mode` into the THTS schema. When running in `mcts_mode`, the trajectory sampled in the selection phase is truncated, and ends when the first state not in the search tree $\mathcal{T}$ is reached. When not running in `mcts_mode`, the trajectory is sampled until the H, the finite horizon of the MDP. These two modes are depicted in blue (`mcts_mode` on) and orange (`mcts_mode` off) in Figure 2.5.

`THTS++` also introduces the notion of a *context* that is sampled for each trial. A context is an arbitrary key-value store that is used to store any relevant data that varies from trial to 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. Moreover, beyond the scope of this thesis, the notion of a context is necessary to encapsulate ideas such as the

state sampled from a *belief state* used in the POMCP algorithm for planning in partially observable environments TODO: cite.

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

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;

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 `mcts_mode`specifying if `THTS++` should operate in MCTS mode.

At the beginning of runnning a `THTS++` algorithm, the search tree is initialised to $\mathcal{T} = \{s_0\}$. 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. 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 $h = H$. If not running in `mcts_mode`, then $h = H$. 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

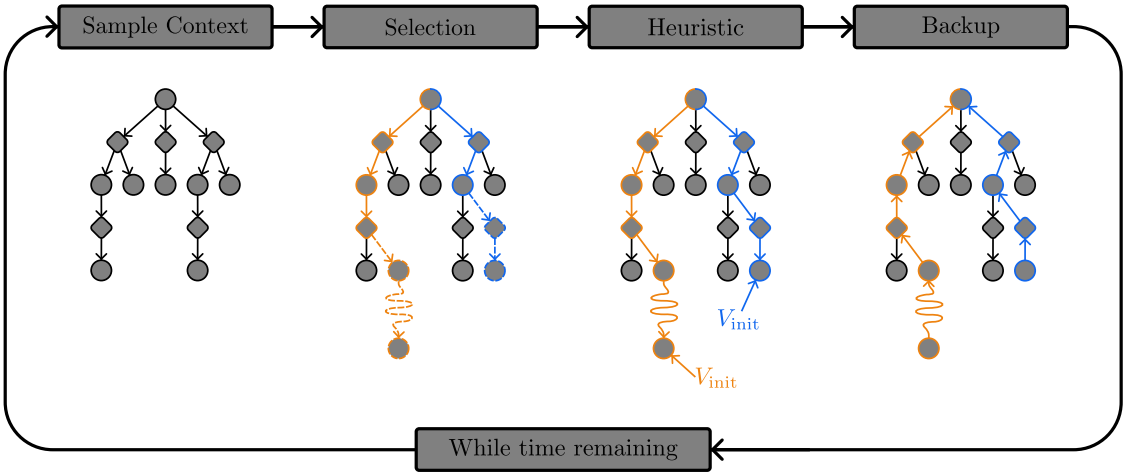

Figure 2.5: Overview of one trial of `THTS++`, where orange shows an example when `mcts_mode` is True, and blue shows an example when `mcts_mode` is False. From left to right: first a context is sampled, which stores any necessary per-trial state (not depicted) and the search tree at the beginning of 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. TODO: fix the slight noise in the dual blue and orange nodes TODO: change blue to add a new chance node too, to highlight that

$\texttt{node}(s_h)\texttt{.V} \leftarrow \hat{V}_{\text{init}}(s_h)$. 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 $t = h-1, ..., 0$. Figure 2.5 depicts this process, and psuedocode is given in Listing 2.1.

```
1  def run_trial(search_tree: 𝒯, search_policy: π_search, heuristic_fn:↩
       V̂_init):
2      context = sample_context()
3      τ_:h = sample_trajectory(𝒯, π_search, context)
4      𝒯 ← 𝒯 ∪ τ_:h
5      node(s_h).V ← V̂_init(s_h, context)
6      for i in {h−1, h−2, ..., 1, 0\}:
7          node(s_i, a_i).backup_q(node(s_i, a_i).children, τ_:h, V̂_init(s_h), context↩
               )
8          node(s_i).backup_v(node(s_i).children, τ_:h, V̂_init(s_h), context)
9
10 def sample_trajectory(search_tree: 𝒯, search_policy: π_search, ↩
       context):
11     cur_state = s_0
12     i = 0
13     while (not mcts_mode or cur_state ∈ 𝒯) and i < H:
14         a_i ∼ π_search(·|s_i, context)
15         r_i ← R(s_i, a_i)
16         i += 1
```

```
        s_i ~ p(·|s_{i-1}, a_{i-1})
    return (s_0, a_0, r_0, s_1, ..., s_{i-1}, a_{i-1}, r_{i-1}, s_i)
```

Listing 2.1: Psuedocode for running a trial in `THTS++`TODO: make each line nicely fit on one line

TODO: make a full figure env for psuedocode listings? Also make a table of listings at start of thesis?

TODO: explicitly define something like `trajectory_nodes`$(\tau_{:h})$ as the set of nodes visited on the trajectory?

2.4.3 Upper Confidence Bounds Applied to Trees (UCT)

Upper Confidence Bounds Applied to Trees (UCT) TODO: cite both papers is a commonly used tree search algorithm, which is based on the Upper Confidence Bounds (UCB) TODO: cite algorithm covered in Section TODO: ref.

In the literature, UCT and MCTS are often used synonomously, however this leaves some of the specifics of the algorithms used as ambiguous. This thesis presents UCT as it was originally presented in TODO: cite. In Subsection TODO: ref the variant of UCT which is commonly referred to as MCTS is given.

UCT can be defined using the THTS schema outlined in Section TODO: ref as follows:

Firstly, UCT as originally presented is run with `mcts_mode`set to False. As such, all sampled trajectories are sampled until timestep H, the finite horizon of the MDP.

At each node a the sampled averages $\bar{V}_{\text{UCT}}$ or $\bar{Q}_{\text{UCT}}$ for value estimates.

The search policy that UCT follows is:

$$\pi_{\text{UCT}}(s) = \arg\max_{a \in \mathcal{A}} \bar{Q}_{\text{UCT}}(s,a) + b_{\text{UCT}} \sqrt{\frac{\log(N(s))}{N(s,a)}} \tag{2.30}$$

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

After sampling a trajectory $\tau_{:H} \sim \pi_{\text{UCT}}$, the sample average value estimates are updated as follows:

$$\bar{Q}_{\text{UCT}}(s_t, a_t) \leftarrow \frac{1}{N(s_t, a_t)} \left((N(s_t, a_t) - 1)\bar{Q}_{\text{UCT}}(s_t, a_t) + \sum_{i=t}^{H-1} R(s_i, a_i) \right) \tag{2.31}$$

$$\bar{V}_{\text{UCT}}(s_t) \leftarrow \frac{1}{N(s_t)} \left((N(s_t) - 1)\bar{V}_{\text{UCT}}(s_t, a_t) + \sum_{i=t}^{H-1} R(s_i, a_i) \right) \tag{2.32}$$

For UCT the `backup_v` and `backup_q` operations implement Equations (TODO: ref) and (TODO: ref) respectively. Because UCT is planning in a finite horizon MDP, the heuristic function will only be called on states that are at the time horizon H. As such, for UCT we can set $\hat{V}_{\text{init}}(s) = 0$.

2.4.4 Monte-Carlo Tree Search

This subsection covers the variant of UCT that is often referred to as just 'MCTS' TODO: cite examples?, although this thesis and the wider literature tends to refer to any algorithm using Monte Carlo trials as an MCTS algorithm. To be unambiguous, this algorithm will be referred to as UCT-MCTS.

UCT-MCTS is commonly presented as having four stages to each trial, depicted in Figure 2.6, which go as follows:

1. selection, which samples a trajectory along a path in the search tree until it finds a state not contained by the search tree;
2. expansion, which creates the new decision node and adds it to the tree;
3. initialisation, which uses a heuristic function to initialise the sample average value estimate at the new leaf node;
4. backup, which updates the value estimates at all nodes visited on the trial.

The selection, expansion and initialisation phases of UCT-MCTS are encapsulated by the selection phase of `THTS++`, where nodes are initalised and added to the tree as necessary.

The main difference between UCT and UCT-MCTS is that `mcts_mode` is turned on in UCT-MCTS, which also means that the heuristic function $\hat{V}_{\text{init}}$ is now used.

Figure 2.6: Overview of one trial of UCT-MCTS, where TODO: re-describe the four stages similarly to how described inline.

There are two common approaches to implementing $\hat{V}_{\text{init}}$ in UCT-MCTS: the first consisting of using a function approximation $\tilde{V}_\theta$ and setting $\hat{V}_{\text{init}} = \tilde{V}_\theta$ TODO: cite some papers doing this, where $\hat{V}_\theta$ aims to approximate the optimal value function V^*; and the second approach consists of using a *rollout policy* TODO: cite some papers doing this. When a rollout policy π_{rollout} is used, a Monte Carlo estimate $\hat{V}^{\pi_{\text{rollout}}}$ of the value function $V^{\pi_{\text{rollout}}}$ is used for $\hat{V}_{\text{init}}$.

Now the remaining details of UCT-MCTS are given. Let $\bar{V}_{\text{MCTS}}$ and $\bar{Q}_{\text{MCTS}}$ refer to the sample average value estimates used by UCT-MCTS. The UCT-MCTS search policy corresponds to the UCT search policy:

$$\pi_{\text{MCTS}}(s) = \arg\max_{a \in \mathcal{A}} Q_{\text{MCTS}}(s,a) + b_{\text{MCTS}} \sqrt{\frac{\log(N(s))}{N(s,a)}}. \tag{2.33}$$

Let $\tau_{:h} \sim \pi_{\text{MCTS}}$, be the trajectory sampled in the selection phase of UCT-MCTS. When using a rollout policy, the truncated trial is completed using the rollout policy $\tau_{h:H} \sim \pi_{\text{rollout}}$ to provide the Monte Carlo estimate at s_h:

$$\hat{V}^{\pi_{\text{rollout}}}(s_h) = \sum_{i=h}^{H-1} r_i. \tag{2.34}$$

In UCT-MCTS 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:

$$\bar{Q}_{\text{MCTS}}(s_t, a_t) \leftarrow \frac{1}{N(s_t, a_t)} \left((N(s_t, a_t) - 1)\bar{Q}_{\text{MCTS}}(s_t, a_t) + \tilde{r} + \sum_{i=t}^{h-1} r_i \right) \tag{2.35}$$

$$\bar{V}_{\text{MCTS}}(s_t) \leftarrow \frac{1}{N(s_t)} \left((N(s_t) - 1)\bar{V}_{\text{MCTS}}(s_t, a_t) + \tilde{r} + \sum_{i=t}^{h-1} r_i \right) \tag{2.36}$$

2.4.5 Maximum Entropy Tree Search

Maximum ENtropy Tree Search (MENTS) TODO: cite, in contrast to UCT, focuses on the maximum-entropy objective, and uses soft Bellman backups (Equations TODO: ref) 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:

$$\pi_{\text{MENTS}}(a|s) = (1 - \lambda_s) \exp\left(\frac{1}{\alpha_{\text{MENTS}}} \left(\hat{Q}_{\text{MENTS}}(s, a) - \hat{V}_{\text{MENTS}}(s) \right) \right) + \frac{\lambda_s}{|\mathcal{A}|}, \tag{2.37}$$

where α_{MENTS} is the temperature paramter used for Equation TODO: ref in MENTS, and $\lambda_s = \min(1, \epsilon / \log(e + N(s)))$, with $\epsilon \in (0, \infty)$ is an exploration parameter.

The value estimates are updated using the soft Bellman backups (TODO: ref) as follows:

$$\hat{Q}_{\text{MENTS}}(s_t, a_t) \leftarrow R(s_t, a_t) + \sum_{s' \in \text{Succ}(s,a)} \left(\frac{N(s')}{N(s_t, a_t)} \hat{V}_{\text{MENTS}}(s') \right), \tag{2.38}$$

$$\hat{V}_{\text{MENTS}}(s_t) \leftarrow \alpha \log \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\alpha} \hat{Q}_{\text{MENTS}}(s_t, a) \right). \tag{2.39}$$

$\hat{V}_{\text{init}}$

In TODO: cite, it is suggested that the heuristic value function is set using a function approximation $\hat{V}_{\text{init}} = \tilde{V}_\theta$, and that the heuristic action function is set to zero: $\hat{Q}_{\text{init}}(s, a) = 0$. They also suggest that if *policy network* $\tilde{\pi}$ is available, the the heuristic action function can be set to $\hat{Q}_{\text{init}}(s, a) = \log \tilde{\pi}(s|a)$.

TODO: Maybe commente here, or refer to where discussed in ch4 (DENTS) work: neither of the options suggested for $\hat{Q}_{\text{init}}$ worked well in practise for us, so we set it depending on the environment.

Figure 2.7: The decision-support scenario for Multi-Objective Reinforcement Learning.]The decision-support scenario for Multi-Objective Reinforcement Learning.

2.5 Multi-Objective Reinforcement Learning

TODO: Link back to some of the multi objective questions

This thesis follows a utility based approach to Multi-Objective Reinforcement learning similar to TODO: cite. For a full review of Multi-Objective Reinforcement Learning see TODO: cite. This work will specifically consider *linear utility* functions and the *decision support scenario* (Figure 2.7), which will be defined more precisely below.

This section defines the multi-objective conterparts to various definitions in Section 2.1 and 2.2. 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.

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 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* $s_{t+1} \sim \text{Succ}(s_t, a_t)$*.*

The notations used for single-objective trajectories (Definition 2.1.4) 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:*

$$\mathbf{V}^\pi(s;t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{i=t}^{H-1} \mathbf{r}_t \middle| s_t = s\right]. \tag{2.40}$$

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

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

In the corresponding point of the single-objective reinforcement learning section (Section 2.2), 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". To resolve this issue, a *utility function* or *scalarisation function* is used to map multi-objective values to scalars.

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

The elements of the D*-dimensional Simplex will be referred to as* weight vectors *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:*

$$u_{\text{lin}}(\mathbf{v}; \mathbf{w}) = \mathbf{w}^\top \mathbf{v}. \tag{2.42}$$

Equiped with a scalarisation function and a weight vector any set of multi-objective values can be ordered, which allows sets of *possibly optimal* policies to be defined. Letting Π be the set of all possible policies, sets of solution policies can be defined:

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 scalarised value is maximised:*

$$U(\Pi; u) = \left\{ \pi \in \Pi \;\middle|\; \exists \mathbf{w} \in \Delta^D . \forall \pi' \in \Pi : u(\mathbf{V}^\pi(s_0; 0); \mathbf{w}) \geq u(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}) \right\}. \tag{2.43}$$

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

Undominated sets often have an infinite cardinality, and as such are infeasibil 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 2.5.8. *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:*

$$\forall \mathbf{w} \in \Delta^D . \exists \pi \in CS(\Pi;u) . \forall \pi' \in \Pi : u(\mathbf{V}^{\pi}(s_0;0);\mathbf{w}) \geq u(\mathbf{V}^{\pi'}(s_0;0);\mathbf{w}). \quad (2.44)$$

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}*.*

To compute a coverage set, 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.

Definition 2.5.9. *The* (multi-objective) value set *with respect to a set of policies* $\Pi' \subseteq \Pi$ *is defined as:*

$$\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi') = \{\mathbf{V}^{\pi}(s_0;0) | \pi \in \Pi'\}. \quad (2.45)$$

Because this thesis considers the decision support scenario, the objective of an multi-objective algorithm will be to compute a $\boldsymbol{\mathcal{V}}\mathbf{als}(CCS(\Pi))$, however, coverage sets are not unique. In the case of the linear utility function, multi-objective values that obtain the same scalarised value will lie on a hyperplane (see Figure TODO: ref). As a result, the vectors in $\boldsymbol{\mathcal{V}}\mathbf{als}(CCS(\Pi))$ will geometrically form a *(parital) convex hull* (also see Figure 2.8). Given this, it is most common to compute the multi-objective values that lie at the vertices of the geometric convex hull (also see Figure 2.8).

For the remainder of this thesis, the primary objective of multi-objective algorithms will be to compute the value set $\boldsymbol{\mathcal{V}}\mathbf{als}(CCS(\Pi))$ that lies at the vertices of the geometric convex hull, which will be referred to as the *Convex Hull Value Set* (CHVS). TODO: make this a defn, going to use the acronym a lot

Figure 2.8: The geometry of Convex Coverage Sets, shown with $D = 2$. In all images, the points depicted correspond to a value set $\boldsymbol{\mathcal{V}}\mathbf{als}(\Pi)$. Left: demonstrates that values obtaining the same scalarised value, for a linear utility function with weight vector $\mathbf{w}$, lie on a hyperplane with a normal vector of $\mathbf{w}$. Right: depicts a (geometric) partial convex hull. Any set of vectors $\boldsymbol{\mathcal{V}} \subseteq \boldsymbol{\mathcal{V}}\mathbf{als}(\Pi)$ that contains a point touching each edge of the geometric convex hull is a valid (value set of a) convex coverage set, and the points that are filled denotes one of such sets. Finally, the circles, which are at the extreme points of the geometric convex hull mark the (value set of the) convex coverage set that is typically computed.

2.5.1 Convex Hull Value Iteration

Convex Hull Value Iteration (CHVI) [**?**] is a tabular dynamic programming algorithm similar to Value Iteration TODO: ref. In CHVI the value functions of value iterations are replaced by sets of vectors, which are estimates of the convex hull value set.

CHVI maintains estimates of CHVS's at each state $\hat{\boldsymbol{\mathcal{V}}}_{\text{CHVI}}(s;t)$.

One important operation for CHVI is `cvx_prune`, which returns that undominated set of vectors from a given set of vectors $\boldsymbol{\mathcal{V}}$:

$$\texttt{cvx\_prune}(\boldsymbol{\mathcal{V}}) = \{\mathbf{v} \in \boldsymbol{\mathcal{V}} | \exists \mathbf{w} \in \Delta^D . \forall \mathbf{v}' \in \boldsymbol{\mathcal{V}} - \{\mathbf{v}\} . \mathbf{w}^\top \mathbf{v} > \mathbf{w}^\top \mathbf{v}'\}. \tag{2.46}$$

The `cvx_prune` operation can be implemented using *linear programming* TODO: cite, and an example of its operation on a set of vectors is given in Figure 2.9.

Additionally, to define a multi-objective version of value iteration, an arithmetic over sets of vectors needs to be defined. An example of the following arithmetic is given in Figure 2.10 Given the sets of vectors $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$, define multiplication by a

Figure 2.9: An example of the `cvx_prune` operation. The circles and triangles form a vector set $\boldsymbol{\mathcal{V}}$, and the circles denote the set `cvx_prune`($\boldsymbol{\mathcal{V}}$).

scalar s, addition with a vector $\mathbf{x}$ and addition between sets as follows:

$$\mathbf{x} + s\boldsymbol{\mathcal{V}} = \{\mathbf{x} + s\mathbf{v} | \mathbf{v} \in \boldsymbol{\mathcal{V}}\} \tag{2.47}$$

$$\boldsymbol{\mathcal{U}} + \boldsymbol{\mathcal{V}} = \{\mathbf{u} + \mathbf{v} | \mathbf{u} \in \boldsymbol{\mathcal{U}}, \mathbf{v} \in \boldsymbol{\mathcal{V}}\}. \tag{2.48}$$

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

$$\hat{\boldsymbol{\mathcal{V}}}^{k+1}_{\text{CHVI}}(s;t) = \texttt{cvx\_prune}\left(\bigcup_{a \in \mathcal{A}} \hat{\boldsymbol{\mathcal{Q}}}^{k+1}_{\text{CHVI}}(s,a;t)\right), \tag{2.49}$$

$$\hat{\boldsymbol{\mathcal{Q}}}^{k+1}_{\text{CHVI}}(s,a;t) = \mathbb{E}_{s' \sim \text{Succ}(s,a)}[\mathbf{R}(s,a) + \hat{\boldsymbol{\mathcal{V}}}^{k}_{\text{CHVI}}(s';t+1)]. \tag{2.50}$$

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

Extracting actions and policies from the computed CHVSs is not explicitly discussed in [**?**]. However, vectors can be *tagged* with actions which allows for a policy to be read out after selecting a value in a CHVS. Consider TODO: having a CHVS for V and Q, set Qtagged to Q with the action in the Q function, and then say that V is computed same as equation XXX, and for convex prune, if there is (v,a) and (v,a'), then break ties randomly. An example of using this tagging to read out policies is given in Figure 2.11.

TODO: some hidden stuff here to consider?

TODO: talk about the POMDP action tagging things

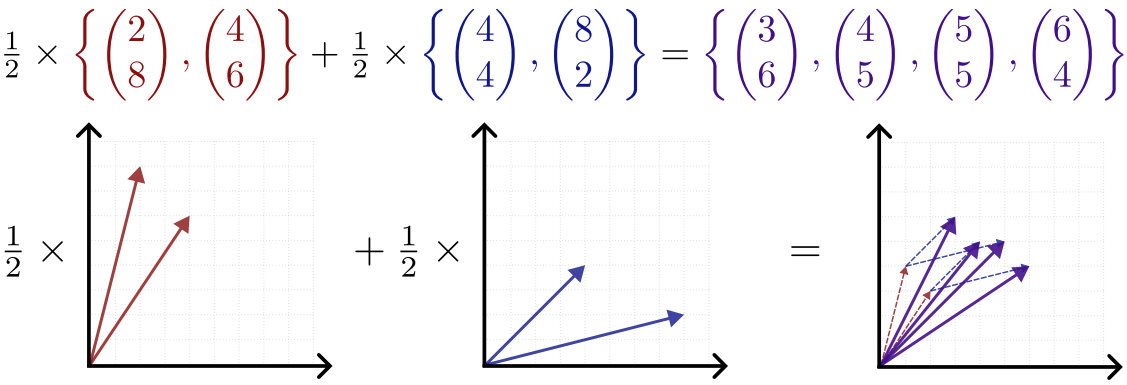

Figure 2.10: An example of arithmetic over sets of vectors, where TODO: describe specifics more specifically, and note that addition by a vector is not shown here, but is similar. Also something looks a bit off about the purple vectors, maybe want to add a light grid behind the graphs? Just to give perspective. Reference that this demonstrates the scalar multiplication from equation (xxx) and addition between sets in equation (xxx). And yes, want the grid, but do later.

Figure 2.11: An example of using tagging with convex hull value sets. An MOMDP is shown, with its associated CHVS's. TODO: colour code the tagging and use arrows to show two different policies being read out from the initial state, and write the corresponding description here.

2.6 Sampling From Catagorical Distributions

TODO: still need to do another pass through this

TODO: this section needs figs still

Much of the work in this thesis will involve sampling from catagorical distributions. Let $f : \{1, ..., m\} \to \mathbb{R}$ be the probability mass function of a catagorical distribution with m categories. Suppose that we want to sample $i \sim f$. A naive method to sample from f will take $O(m)$ time, where a value is sampled from Uniform$(0, 1)$ is often used as follows:

```
def sample_catagorical(f):
    threshold ~ Uniform(0, 1)
    i = 0
    accumulated_mass = 0
    while (accumulated_mass < threshold):
        i += 1
        accumulated_mass += f(i)
    return i
```

TODO: add label and caption for listing

However, the *Alias method* TODO: cite1, cite2 can instead be used, with $O(m)$ preprocessing time to construct an *Alias table*, and can sample from f in $O(1)$ time. In Figure TODO: ref we provide an example of an alias table. A value can be sampled using the alias table by sampling two random numbers, one from Uniform$(\{1, ..., m\})$ and one from Uniform$(0, 1)$. To sample from the alias table, one of the entries is sampled uniformly randomly using the sample from Uniform$(\{1, ..., m\})$, each entry in the table contains three values, `threshold`, `cat_one` and `cat_two`, from which if we let $a \sim$ Uniform$(0, 1)$, we would then return `cat_one` if $a <$`threshold` and `cat_two` otherwise. Psuedocode for this is as follows:

```
def sample_from_alias_table(alias_table):
    index ~ Uniform({1, ..., m})
    cat_one, cat_two, threshold = alias_table[index]
    a ~ Uniform(0, 1)
    if (a < threshold):
        return cat_one
    return cat_two
```

TODO: add label and caption for listing

In TODO: cite it is shown TODO: double check that it's proven that an alias table can be constructed from an arbitrary probability mass function for a 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.

TODO: add fig, and in caption verify that the example alias table maintains the correct masses for each category

Following TODO: cite, we can construct an alias table as follows:

```
1 def build_alias_table(f):
2     pass
```

TODO: add label and caption for listing

TODO: write the build alias table psuedocode

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

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 [1]
- 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.

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/2111.0182

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

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

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] 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.

[2] 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.

[3] Michael Painter. THTS++,https://github.com/MWPainter/thts-plus-plus.