

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	ix
List of Tables	xi
List of Abbreviations	xv
List of Notation	xvii
1 Introduction	1
1.1 Overview	1
1.2 Contributions	2
1.3 Structure of Thesis	4
1.4 Publications	4
2 Background	5
2.1 Markov Decision Processes	6
2.2 Reinforcement Learning	8
2.2.1 Maximum Entropy Reinforcement Learning	12
2.3 Multi-Armed Bandits	14
2.3.1 Exploring Bandits	17
2.3.2 Contextual Bandits	18
2.4 Trial-Based Heuristic Tree Search and Monte Carlo Tree Search . .	19
2.4.1 Notation	21
2.4.2 Trial Based Heuristic Tree Search	21
2.4.3 Upper Confidence Bounds Applied to Trees (UCT)	26
2.4.4 Maximum Entropy Tree Search	28
2.5 Multi-Objective Reinforcement Learning	29
2.5.1 Convex Hull Value Iteration	34
2.6 Sampling From Catagorical Distributions	36

3	Literature Review	41
3.1	Reinforcement Learning	41
3.2	Multi-Armed Bandits	42
3.3	Trial-Based Heuristic Tree Search and Monte-Carlo Tree Search . .	42
3.3.1	Trial Based Heuristic Tree Search	42
3.3.2	Monte-Carlo Tree Search	42
3.3.3	Maximum Entropy Tree Search	43
3.4	Multi-Objective Reinforcement Learning	43
3.5	Multi-Objective Monte Carlo Tree Search	44
4	Monte Carlo Tree Search With Boltzmann Exploration	47
4.1	Introduction	47
4.2	Boltzmann Search	48
4.3	Toy Environments	48
4.4	Theoretical Results	48
4.5	Empirical Results	48
4.6	Full Results	49
5	Convex Hull Monte Carlo Tree Search	51
5.1	Introduction	51
5.2	Contextual Tree Search	51
5.3	Contextual Zooming for Trees	52
5.4	Convex Hull Monte Carlo Tree Search	52
5.5	Results	52
6	Simplex Maps for Multi-Objective Monte Carlo Tree Search	53
6.1	Introduction	53
6.2	Simplex Maps	54
6.3	Simplex Maps in Tree Search	54
6.4	Theoretical Results	54
6.5	Empirical Results	54
7	Conclusion	55
7.1	Summary of Contributions	55
7.2	Future Work	55
Appendices		
A	List Of Appendices To Consider	59
Bibliography		61

List of Figures

2.1	An example MDP \mathcal{M}	7
2.2	An overview of reinforcement learning.	9
2.3	An example K -armed bandit problem.	15
2.4	The multi-armed bandit problem.	16
2.5	Overview of one trial of MCTS.	19
2.6	Tree diagrams notation.	22
2.7	Overview of one trial of THTS++.	23
2.8	The decision-support scenario for Multi-Objective Reinforcement Learning. [6]	29
2.9	An example MDP \mathcal{M}	30
2.10	The geometry of convex coverage sets, shown with $D = 2$. In each images the points depicted correspond to a value set $\mathbf{Vals}(\Pi')$, 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 $\mathbf{Vals}(CH(\Pi'))$, the blue circles form the minimal convex coverage set, $\mathbf{Vals}(CCS_{\min}(\Pi'))$, and triangles are not part of the convex hull. All of the unfilled shapes would be pruned by <code>cvx_prune</code> (Equation (2.44)); (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, it 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 $\mathbf{Vals}(\Pi')$. . .	34
2.11	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 (2.45), and then added using Equation (2.46). On the right the resulting vector set is shown in purple, which are the sum of appropriately scaled red and blue vectors.	35

2.12	A visualisation, reusing the convex hull from Figure 2.10, 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 $\mathbf{w}_i = \frac{\mathbf{v}_i - \mathbf{v}_\perp}{\ \mathbf{v}_i - \mathbf{v}_\perp\ }$ 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 (2.50) will give the desired value \mathbf{v}_i . TODO: add the $\ \cdot\ _2$ bit and add the \mathbf{w}_i bit into diagram	36
2.13	TODO: write caption TODO: shorten threshold, maybe to a symbol? So can clearly read what is says. Maybe also get rid of sampling bits, maybe make arrows bigger too	38

List of Tables

List of Code Listings

2.1	Psuedocode for running a trial in THTS++	26
2.2	Psuedocode for sampling from a catagorical distribution.	37
2.3	Psuedocode for sampling from an alias table.	37
2.4	Psuedocode for constructing an alias table.	39

List of Abbreviations

CHVI	Convex Hull Value Iteration.
CHVS	Convex Hull Value Set.
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.

Multi-Armed Bandits (Section 2.3)

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

List of Notation

Global Notation **TODO: probability?**

$\mathbb{1}$	The indicator function, where $\mathbb{1}(A) = 1$ when A is true, and $\mathbb{1}(A) = 0$ when A is false.
\mathbb{E}	TODO: Expectation?
TODO:	TODO: Trends of notation? From Ch2 intro: "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, $\mathbf{\mathcal{V}}$ is a set of vectors, typeface text V typically refers to things that are more implementation details"

(Multi-Objective) Markov Decision Processes (Defined in Sections 2.1 and 2.5)

\mathcal{A}	A (finite) set of actions.
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 $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, R, p, H)$.
\mathcal{M}	A Multi-Objective Markov Decision Process, which is a tuple $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, \mathbf{R}, p, H)$.
p	The next-state transition distribution of an MDP. $p(s' s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$.
π	: A policy, mapping a state $s \in \mathcal{S}$ to a probability distribution over actions \mathcal{A} . TODO: probably dont need this, not going to use non-locally
R	The reward function of an MDP: $R(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.
\mathbf{R}	The D dimensional reward function of a MOMDP: $\mathbf{R}(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^D$.
\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} : $\tau = (s_0, a_0, r_0, s_1, \dots, s_{H-1}, a_{H-1}, r_{H-1}, s_H)$.
τ	A multi-objective trajectory, of states, actions and rewards that are sampled according to a policy π and an MDP \mathcal{M} : $\tau = (s_0, a_0, \mathbf{r}_0, s_1, \dots, s_{H-1}, a_{H-1}, \mathbf{r}_{H-1}, s_H)$.
$s, (s, a) \in \tau$	TODO: define this here?
$\tau_{i:j}$	A truncated trajectory, starting at timestep i , and ending at timestep j : $\tau_{i:j} = (s_i, a_i, r_i, s_{i+1}, \dots, s_{j-1}, a_{j-1}, r_{j-1}, s_j)$.
$\tau_{i:j}$	A multi-objective truncated trajectory, starting at timestep i and ending at timestep j : $\tau_{i:j} = (s_i, a_i, \mathbf{r}_i, s_{i+1}, \dots, s_{j-1}, a_{j-1}, \mathbf{r}_{j-1}, s_j)$.

Reinforcement Learning (Section 2.2)

$J(\pi)$	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 $a_t = a$.
\hat{Q}^k	The estimate of the optimal value function after k iterations of Value Iteration. TODO: probably dont need this, not going to use non-locally
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 $s_t = s$.
\hat{V}^k	The estimate of the optimal value function after k iterations of Value Iteration. TODO: probably dont need this, not going to use non-locally

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 2.2.1)

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: $J_{\text{sft}}(\pi) = V_{\text{sft}}^\pi(s_0; 0)$.

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_{sft}^π The soft Q-value function of a policy π . $Q_{\text{sft}}^\pi(s, a; t)$ 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_{sft}^π The soft value of a policy π . $V_{\text{sft}}^\pi(s; t)$ 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)

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

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}^π	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}^π	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 $s_t = s$.
\mathbf{w}	A weighting over objectives that quantifies preferences over the multiple objectives, to be used in a utility function.
$CS(\Pi; u)$	TODO: define here and move into correct place in the list
$CCS(\Pi)$	TODO: define here and move into correct place in the list
$CCS_{\min}(\Pi)$	TODO: define here and move into correct place in the list
$\mathbf{Vals}(\Pi)$	TODO: define here and move into correct place in the list
cvx_prune	TODO: define here and move into correct place in the list

TODO: TODO: Sort below into more reasonable things to look up

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
$\text{cum_regr}_{\text{MAB}}$.	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)

α_{MENTS}	The temperature parameter used in MENTS, which is the coefficient of the entropy terms in the maximum entropy objective.
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.

ϵ	An exploration parameter used in MENTS.
λ_s	An exploration parameter used in MENTS.
<code>mcts_mode</code> . . .	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 <code>mcts_mode</code> 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) .
<code>node(s)</code>	The decision node corresponding to the state s .
<code>node(s).chldrn</code>	The set of chance nodes that are children of <code>node(s)</code> .
<code>node(s).V</code> . . .	The set of variables stored at decision node <code>node(s)</code> , typically used for estimating values.
<code>node(s, a)</code> . . .	The chance node corresponding to the state-action pair (s, a) .
<code>node(s).chldrn</code>	The set of decision nodes that are children of <code>node(s)</code> .
<code>node(s, a).Q</code> . .	The set of variables stored at decision node <code>node(s, a)</code> , typically used for estimating Q-values.
$\tilde{\pi}$	A policy (neural) network, used to initialise soft Q-value estimates in MENTS, if available.
π_{MENTS}	The search policy used in MENTS.
π_{rollout}	A policy used to sample a trajectory that is used to estimate the value of a state being added to a search tree.
π_{search}	The search policy used in THTS++ to sample a trajectory in the selection phase.
π_{UCT}	The search policy used in UCT.
\hat{Q}_{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.
$\hat{Q}_{\text{MENTS}}(s, a)$. .	A soft Q-value estimate at the chance node (s, a) in MENTS.
$\bar{Q}_{\text{UCT}}(s, a)$. . .	The sample-average value estimate used to estimate the Q-value at a chance node (s, a) in UCT.
<code>sample_context</code>	A function used in THTS++ that creates a context, or key-value story, and samples any initial values to be stored in the context.

<code>sample_outcome</code>	A function used in THTS++ to sample outcomes (successor states) from the environment (MDP).
\mathcal{T}	The THTS++ search tree. $\mathcal{T} \subseteq \mathcal{S} \cup \mathcal{S} \times \mathcal{A}$.
\tilde{V}	A function approximation to V^* .
$\hat{V}^{\pi_{\text{rollout}}}(s)$	An estimate of $V^{\pi_{\text{rollout}}}(s)$ computed using the return from a single trajectory sampled using policy π_{rollout} . It is used to initialise the value of a state being added to the search tree.
\hat{V}_{init}	The heuristic value function used in THTS++ , used to initialise the value of a new decision node.
$\hat{V}_{\text{MENTS}}(s)$	A soft value estimate at the decision node s in MENTS .
$\bar{V}_{\text{UCT}}(s)$	The sample-average value estimate used to estimate the value at a decision node s in UCT .

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

1

Introduction

Contents

1.1	Overview	1
1.2	Contributions	2
1.3	Structure of Thesis	4
1.4	Publications	4

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**)

TODO: Would like to do the comparing different types of eval, even if not listing it as a research question (compare giving it X seconds per decision and evaluating that policy (SLOW), and comparing policy extracted from the tree)

TODO: can make an argument that the best bound achieved by theory is given by letting temperature go to max. Which is consistent with the exploring bandits results

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; Baïoumy, 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

2.1	Markov Decision Processes	6
2.2	Reinforcement Learning	8
2.2.1	Maximum Entropy Reinforcement Learning	12
2.3	Multi-Armed Bandits	14
2.3.1	Exploring Bandits	17
2.3.2	Contextual Bandits	18
2.4	Trial-Based Heuristic Tree Search and Monte Carlo Tree Search	19
2.4.1	Notation	21
2.4.2	Trial Based Heuristic Tree Search	21
2.4.3	Upper Confidence Bounds Applied to Trees (UCT)	26
2.4.4	Maximum Entropy Tree Search	28
2.5	Multi-Objective Reinforcement Learning	29
2.5.1	Convex Hull Value Iteration	34
2.6	Sampling From Catagorical Distributions	36

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, $\mathbf{\mathcal{V}}$ is a set of vectors, typeface text \mathbf{V} typically refers to things that

are more implementation details

2.1 Markov Decision Processes

TODO: chatgpt the intro stuff

TODO: talk about seq decision making under uncertainty at points

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 2.1.1. 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 $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, $p(s'|s, a)$ is a next state transition distribution $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ and $H \in \mathbb{N}$ is a finite-horizon time bound. *TODO: I would quite like to make this indefinite horizon. Then also this would specify a planning horizon H , and heuristic function not unnecessary for mcts mode being false*

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:

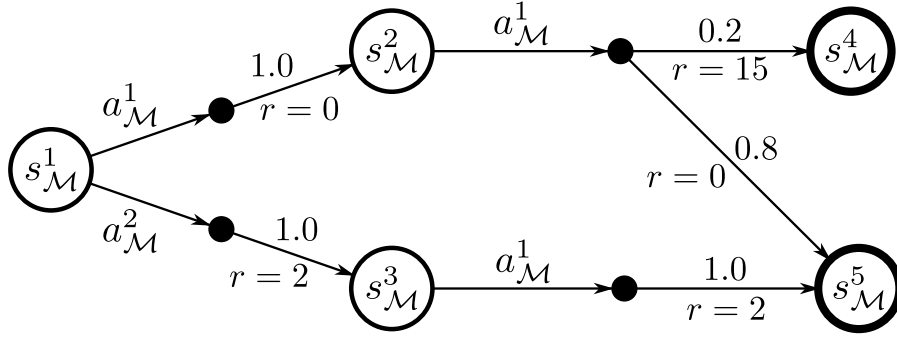


Figure 2.1: An example MDP \mathcal{M} , where TODO: description of MDP drawn

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

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

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 $\pi : \mathcal{S} \rightarrow (\mathcal{A} \rightarrow [0, 1])$ 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 2.4 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 $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{H-1}, a_{H-1}, r_{H-1}, s_H)$, 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 $\tau_{i:j} := (s_i, a_i, r_i, s_{i+1}, \dots, s_{j-1}, a_{j-1}, r_{j-1}, s_j)$, between the timesteps $0 \leq i < j \leq H$ inclusive;
- $\tau_{:j} := \tau_{0:j}$ denotes a trajectory that is truncated 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 $s \in \{s_i | i = 0, \dots, H\}$ and $(s, a) \in \{(s_i, a_i) | i = 0, \dots, H-1\}$ respectively.

TODO: citations in this section? Puttman?

TODO: Define the return of a trajectory? Make sure return is defined somewhere?

2.2 Reinforcement Learning

TODO: chatgpt the intro stuff

TODO: talk about seq decision making under uncertainty at points

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, receiving in return

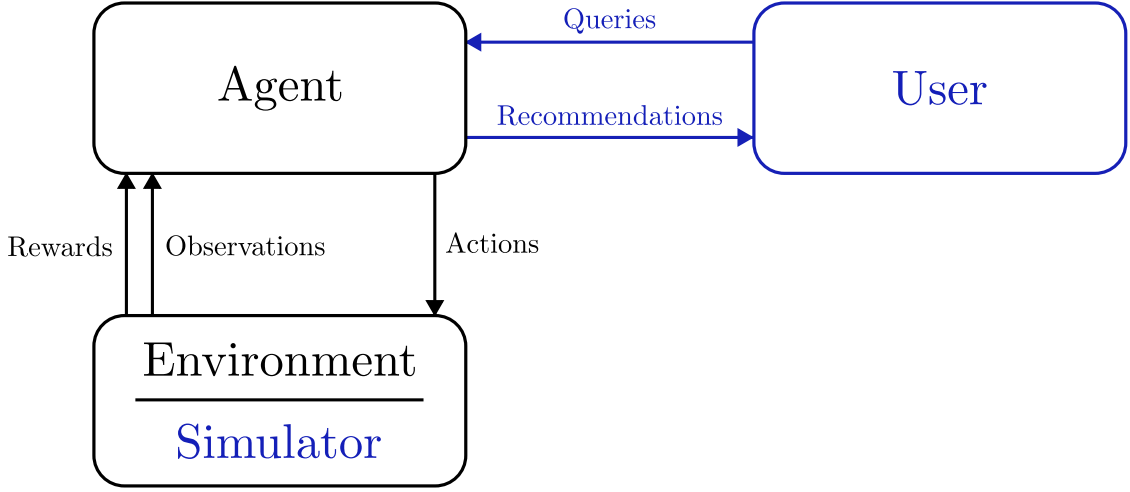


Figure 2.2: An overview of reinforcement learning. Left: depicts the typical scenario where an agent can perform actions in an environment and is given feedback in the form of observations and rewards. Right: shows a similar scenario where the agent instead plans using a simulated environment and is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

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 receives.

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 equipped 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 occasionally 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. 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:*

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

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

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

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:*

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

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

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:

$$J(\pi) = V^\pi(s_0; 0). \quad (2.6)$$

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:

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

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

$$\pi^*(s) = \arg \max_a Q^*(s, a). \quad (2.8)$$

[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](#) :

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

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

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:

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

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

In each iteration of Value Iteration, these values are computed for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $t \in \{0, 1, \dots, H\}$. The Bellman equations can be shown to be *contraction operators* [TODO: refs](#), which can be used to show that $V^k \rightarrow V^*$ as $k \rightarrow \infty$, 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. [4].

Let \mathcal{H} denote the (Shannon) entropy function [11]:

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

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:*

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

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

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

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:*

$$V_{\text{sft}}^*(s; t) = \max_{\pi} V_{\text{sft}}^{\pi}(s; t), \quad (2.16)$$

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

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:*

$$J_{\text{sft}}(\pi) = V_{\text{sft}}^{\pi}(s_0; 0). \quad (2.18)$$

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:*

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

Given V_{sft}^ and Q_{sft}^* the optimal soft police is known to take the form [4]:*

$$\pi_{\text{sft}}^*(a|s; t) = \exp((Q_{\text{sft}}^*(s, a; t) - V_{\text{sft}}^*(s; t)) / \alpha). \quad (2.20)$$

Equations similar to the Bellman equations, aptly named the *Soft Bellman equations*, can be defined for maximum entropy reinforcement learning [4]. 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* [4]:

$$V_{\text{sft}}^*(s; t) = \alpha \log \sum_{a \in \mathcal{A}} \exp(Q_{\text{sft}}^*(s, a; t)/\alpha), \quad (2.21)$$

$$Q_{\text{sft}}^*(s, a; t) = R(s, a) + \mathbb{E}_{s' \sim p(\cdot|s, a)}[V_{\text{sft}}^*(s'; t + 1)]. \quad (2.22)$$

Again, similarly to standard reinforcement learning, *soft Bellman backups* can be defined that admit an analogous algorithm *Soft Value Iteration* [4]:

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

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

2.3 Multi-Armed Bandits

TODO: Haven't started 2nd pass on this section at all, because it has the most work left to do

TODO: Introduce tree search using multi-armed bandits?

TODO: list

- Would like to think a bit about some of the bandits work that sample actions (from adversarial I think), because they were similar to boltzmann search but I hadn't seen details about those works when writing dents
- Also the gradient based MAB stuff in sutton and barto book? Looks relevant? Maybe consider that as update to DENTS paper? Either way, another idea for getting good Go results.

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



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

- 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 $\mathcal{A} = \{1, \dots, K\}$

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

This section introduces the K -armed bandit problem TODO: cite MAB orig, 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 was introduced in the context of clinical trials, where each “arm” corresponds to a treatment option TODO: cite. 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

- For each round $m = 1, 2, \dots$:
 - the agent selects an arm $x^m = \sigma(m) \in \{1, \dots, K\}$ to pull;
 - the agent receives a reward $y^m \sim f_{x^m}$.

Figure 2.4: The multi-armed bandit problem, following a strategy.

best strategy. Conversely, the agent should want to exploit, using the information that it has obtained, so that it makes good decisions. 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 algorithm [TODO: cite](#). Sections 2.3.1 and 2.3.2 considers two variations on the MAB problem that will be relevant in this thesis.

Figure 2.4 shows how the operation of a K -armed bandit problem proceeds. Formally, let f_1, \dots, f_K be the probability distributions for the rewards of each of the K arms, with expected values of μ_1, \dots, μ_K respectively. On the m th round, if the arm $x^m \in \{1, \dots, K\}$ is pulled, then a reward $y^m \sim f_{x^m}$ is received. A

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

$$\text{cum_regr}_{\text{MAB}}(n) = n\mu^* - \sum_{m=1}^n y^m, \quad (2.25)$$

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

To theoretically analyse algorithms for MAB problems, the quantity of expected regret, $\mathbb{E}[\text{cum_regr}_{\text{MAB}}]$ 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 $O(\log N)$.

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

$$N_x^m = \sum_{i=1}^m \mathbb{1}[x^i = x], \quad (2.26)$$

$$\bar{y}_x^m = \frac{1}{N_x^m} \sum_{i=1}^m y^i \mathbb{1}[x^i = x]. \quad (2.27)$$

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

$$\pi_{\text{UCB}}(m) = \arg \max_{j \in \{1, \dots, K\}} \bar{y}_j^m + b_{\text{UCB}} \sqrt{\frac{\log(m)}{N(j)}} \quad (2.28)$$

TODO: define b_{UCB}

TODO: talk about explortaiton-exploitation trade off – see Edwin thesis too because said it nicely

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 slightly. 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 $\{1, \dots, n\}$:
- 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 m th round is:*

$$\text{sim_regr}_{\text{EMAB}}(m) = \mathbb{E}_{i \sim \psi^m}[\mu^* - \mu_i]. \quad (2.29)$$

TODO: talk about the bounds they show in their work

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

2.3.2 Contextual Bandits

TODO: sleepy and rushed this section a bit

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 $\{1, \dots, n\}$:
- agent receives context w^m
- agent selects arm x^m according to policy π
- agent receives 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.

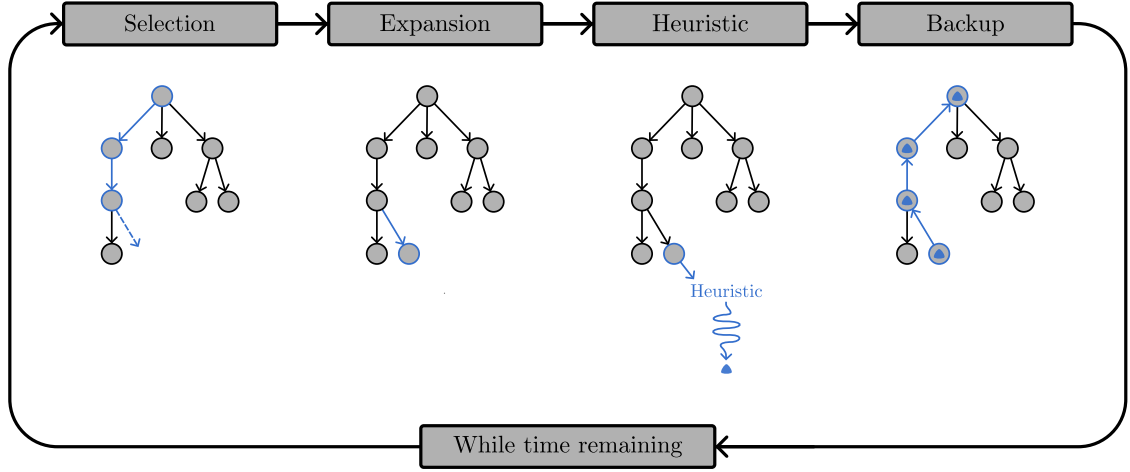


Figure 2.5: Overview of one trial of MCTS, where **TODO: re-describe the four stages similarly to how described inline.**

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

$$\text{ctx_regr}_{\text{CMAB}}(\pi, n) = \sum_{i=1}^n \mu_{w^i}^* - y^i, \quad (2.30)$$

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

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

TODO: Talk about contextual zooming precursor (need to read and add to litrev)

TODO: Write about contextual zooming, using CHMCTS paper, and use opportunity to add to litrev

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 use of 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 [3], which are depicted in Figure 2.5. 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 truncated trajectory is sampled 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) [7] generalises heuristic tree search methods used for planning, such as Trial-Based Real Time Dynamic Programming [2] and LAO* [5]. However, THTS differs slightly from how MCTS-1 is presented above. In a THTS trial, after the expansion/heuristic 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 2.7.

THTS++ [10] is introduced in Section 2.4.2, which is an open-source, parallelised extension of the THTS schema, and was built to facilitate the work in this thesis. Section 2.4.3 covers the Upper Confidence Bound applied to Trees (UCT) algorithm [8, 9], which is a commonly used MCTS algorithm, and Section 2.4.4 presents the Maximum ENtropy Tree Search (MENTS) algorithm [13], 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 correspondence 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.

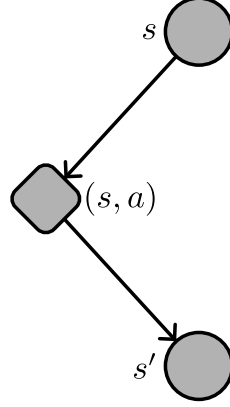
Specifically, this allows the notation for value functions to avoid explicitly writing the timestep parameter, so that $V^\pi(s)$ can be written instead of $V^\pi(s; t)$.

2.4.2 Trial Based Heuristic Tree Search

This subsection introduces THTS++ , which extends the Trial-Based Heuristic Tree Search (THTS) schema [7], 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-1trials described previously to be incorporated. 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 [12]. The remainder of this section will be presented in the context of planning for fully-observable MDPs, but 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 2.6 shows how decision and chance nodes will be depicted 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 2.7. 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);
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

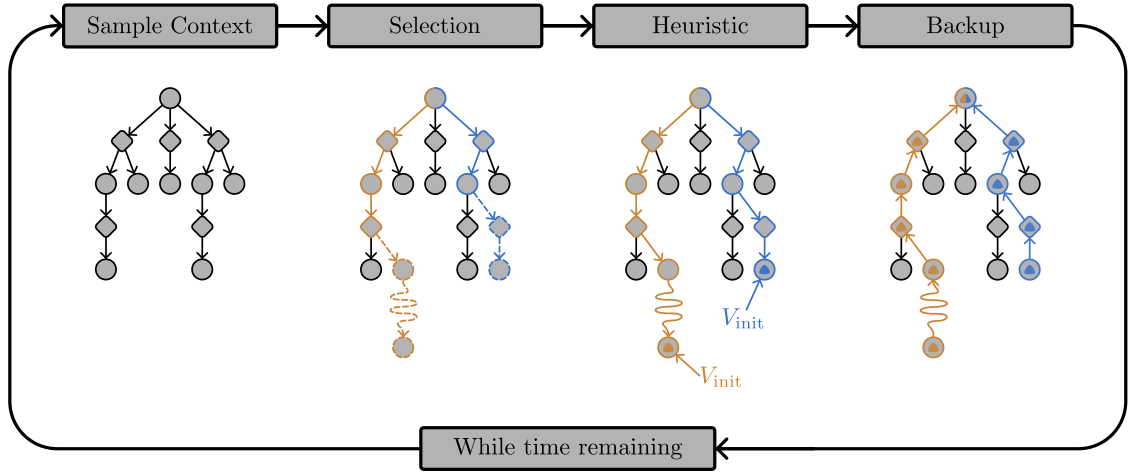


Figure 2.7: Overview of one trial of THTS++, where orange shows an example when `mcts_mode` is True, and blue shows an example when `mcts_mode` is False. From left to right: first a context is sampled, which stores any necessary per-trial state (not depicted) and the search tree at the beginning of the 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}_{init} heuristic function; and finally on the right, shows the backup phase, where the arrows directions are changed to show that information is being propagated back up the tree. **TODO: update for data going back up the tree**

notation $\text{node}(s)$ and $\text{node}(s, a)$ 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 $\text{node}(s)$ and $\text{node}(s, a)$ have been visited, or equivalently, 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. $\text{node}(s).V$ is used to denote the set of values stored at node $\text{node}(s)$, and $\text{node}(s, a).Q$ for the set of value stored at node $\text{node}(s, a)$.

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

TODO: make mctsmode a defn?

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. That is, in `mcts_mode`, a

single decision node is added to the search tree on each trial, similarly to MCTS-1. When not running in `mcts_mode`, the trajectory is sampled until timestep H , the finite horizon of the MDP, similarly to THTS. These two modes are depicted in blue (`mcts_mode` on) and orange (`mcts_mode` off) in Figure 2.7.

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 resources (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.

TODO: make context a defn?

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.

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, `node(s).V` and `node(s, a).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;

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}_{init} used as a heuristic to initialise values for new decision nodes added to the tree;

Heuristic action function: A function \hat{Q}_{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`.

At the beginning of running 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}_{init} as necessary, and `sample_outcome`. If running in `mcts_mode`, then $\tau_{:h}$ is such that $s_t \in \mathcal{T}$ for $t = 0, \dots, h-1$ 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 $\text{node}(s_h).V \leftarrow \hat{V}_{\text{init}}(s_h)$. Finally, for the backup phase, the `backup_q` and `backup_v` functions are used to update the

```

1  def run_trial(search_tree:  $\mathcal{T}$ ,
2              search_policy:  $\pi_{\text{search}}$ ,
3              heuristic_fn:  $\hat{V}_{\text{init}}$ ):
4      # context sampling
5      cntxt = sample_context()
6      # simulation phase
7       $\tau_{:h}$  = sample_trajectory( $\mathcal{T}$ ,  $\pi_{\text{search}}$ , cntxt)
8       $\mathcal{T} \leftarrow \mathcal{T} \cup \tau_{:h}$ 
9      # heuristic phase
10     node( $s_h$ ).V  $\leftarrow \hat{V}_{\text{init}}(s_h, \text{cntxt})$ 
11     # backup phase
12     for i in { $h-1, h-2, \dots, 1, 0$ }:
13         node( $s_i, a_i$ ).backup_q(node( $s_i, a_i$ ).chldrn,  $\tau_{:h}$ ,  $\hat{V}_{\text{init}}(s_h)$ , cntxt)
14         node( $s_i$ ).backup_v(node( $s_i$ ).chldrn,  $\tau_{:h}$ ,  $\hat{V}_{\text{init}}(s_h)$ , cntxt)
15
16 def sample_trajectory(search_tree:  $\mathcal{T}$ ,
17                     search_policy:  $\pi_{\text{search}}$ ,
18                     cntxt):
19     i = 0
20     while (not mcts_mode or  $s_i \in \mathcal{T}$ ) and i < H:
21          $a_i \sim \pi_{\text{search}}(\cdot | s_i, \text{cntxt})$ 
22          $r_i \leftarrow R(s_i, a_i)$ 
23          $s_{i+1} \leftarrow \text{sample_outcome}(s_i, a_i, \text{cntxt})$ 
24         i += 1
25     return ( $s_0, a_0, r_0, s_1, \dots, s_{i-1}, a_{i-1}, r_{i-1}, s_i$ )
26
27 def sample_outcome(s, a, cntxt):
28      $s' \sim p(\cdot | s, a)$ 
29     return s'

```

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

values of $\text{node}(s_t, a_t).Q$ and $\text{node}(s_t).V$ for $t = h - 1, \dots, 0$. Figure 2.7 depicts this process, and psuedocode is given in Listing 2.1.

2.4.3 Upper Confidence Bounds Applied to Trees (UCT)

Upper Confidence Bounds Applied to Trees (UCT) [8, 9] is a commonly used tree search algorithm, which is based on the Upper Confidence Bounds (UCB) [TODO: cite](#) algorithm covered in Section [TODO: ref](#). 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

nodes made. 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}_{UCT} and \bar{Q}_{UCT} is used for a value estimate. The search policy that UCT then follows is given by:

$$\pi_{\text{UCT}}(s) = \arg \max_{a \in \mathcal{A}} \bar{Q}_{\text{UCT}}(s, a) + b_{\text{UCT}} \sqrt{\frac{\log(N(s))}{N(s, a)}}, \quad (2.31)$$

where b_{UCT} is a *bias* parameter that controls the amount of exploration UCT will perform. In Equation (2.30), 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}_{init} is unnecessary for UCT.

There are two common approaches to implementing \hat{V}_{init} in UCT: the first consisting of using a function approximation \tilde{V} and setting $\hat{V}_{\text{init}} = \tilde{V}$ [TODO: cite some papers doing this, rave?](#), where \hat{V} aims to approximate the optimal value function V^* ; 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 $\hat{V}^{\pi_{\text{rollout}}}$ is used to estimate the value function $V^{\pi_{\text{rollout}}}$ and is used for \hat{V}_{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}_{init} . 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 :

$$\hat{V}^{\pi_{\text{rollout}}}(s_h) = \sum_{i=h}^{H-1} r_i. \quad (2.32)$$

If no informative policy is available to be used for π_{rollout} , then uniformly random policy is often used [TODO: cite](#).

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:

$$\bar{Q}_{\text{UCT}}(s_t, a_t) \leftarrow \frac{1}{N(s_t, a_t)} \left((N(s_t, a_t) - 1) \bar{Q}_{\text{UCT}}(s_t, a_t) + \tilde{r} + \sum_{i=t}^{h-1} r_i \right) \quad (2.33)$$

$$\bar{V}_{\text{UCT}}(s_t) \leftarrow \frac{1}{N(s_t)} \left((N(s_t) - 1) \bar{V}_{\text{UCT}}(s_t, a_t) + \tilde{r} + \sum_{i=t}^{h-1} r_i \right) \quad (2.34)$$

TODO: Make a comment about how when use a random rollout, that using `mctsmode OFF` is identical?

2.4.4 Maximum Entropy Tree Search

Maximum ENTropy Tree Search (MENTS) [13], 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}_{MENTS} and \hat{Q}_{MENTS} . The MENTS search policy is given by:

$$\pi_{\text{MENTS}}(a|s) = (1 - \lambda_s) \exp \left(\frac{1}{\alpha_{\text{MENTS}}} \left(\hat{Q}_{\text{MENTS}}(s, a) - \hat{V}_{\text{MENTS}}(s) \right) \right) + \frac{\lambda_s}{|\mathcal{A}|}, \quad (2.35)$$

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 (TODO: ref)).

In MENTS the backups `backup_v` and `backup_q` update the soft value estimates and are updated using soft Bellman backups as follows:

$$\hat{Q}_{\text{MENTS}}(s_t, a_t) \leftarrow R(s_t, a_t) + \sum_{s' \in \text{Succ}(s, a)} \left(\frac{N(s')}{N(s_t, a_t)} \hat{V}_{\text{MENTS}}(s') \right), \quad (2.36)$$

$$\hat{V}_{\text{MENTS}}(s_t) \leftarrow \alpha \log \sum_{a \in \mathcal{A}} \exp \left(\frac{1}{\alpha} \hat{Q}_{\text{MENTS}}(s_t, a) \right). \quad (2.37)$$

In [13], 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 $\hat{Q}_{\text{init}}(s, a) = \log \tilde{\pi}(s|a)$.



Figure 2.8: The decision-support scenario for Multi-Objective Reinforcement Learning. [6]

2.5 Multi-Objective Reinforcement Learning

This thesis follows a utility based approach to Multi-Objective Reinforcement learning similar to the review of [6] [TODO: change this to an et al citation, and work out how to do that \(google natbib cite methods\)](#). This work will specifically consider *linear utility* functions and the *decision support scenario* which is depicted in Figure 2.8. 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 2.1 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. In Section 2.5.1, a multi-objective extension of Value Iteration [TODO: ref?](#) 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 $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, \mathbf{R}, p, H)$, 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 $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^D$, where D is the dimension of the rewards and the *MOMDP*, $p(s'|s, a)$ is a next state transition distribution $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ 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* \mathcal{M} pair. Let the

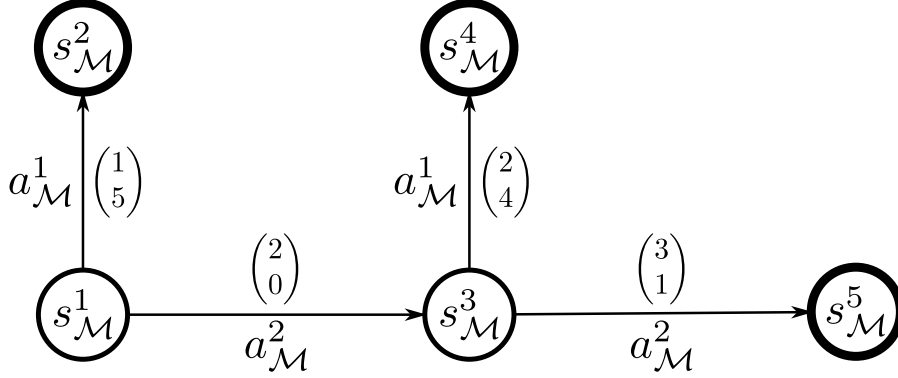


Figure 2.9: An example MOMDP \mathcal{M} , where TODO: description of MDP drawn

trajectory be $\tau = (s_0, a_0, \mathbf{r}_0, s_1, a_1, \mathbf{r}_1, \dots, s_{H-1}, a_{H-1}, \mathbf{r}_{H-1}, s_H)$, 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 2.1.4) 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:

$$\mathbf{V}^\pi(s; t) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{i=t}^{H-1} \mathbf{r}_i \middle| s_t = s \right]. \quad (2.38)$$

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

$$\mathbf{Q}^\pi(s, a; t) = \mathbf{R}(s, a) + \mathbb{E}_{s' \sim p(\cdot | s, a)} [\mathbf{V}^\pi(s'; t + 1)]. \quad (2.39)$$

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”. 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*

$$\Delta^D = \{\mathbf{w} \in \mathbb{R}^D | w_i > 0, \sum_i w_i = 1\}. \quad (2.40)$$

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) $u : \mathbb{R}^D \times \Delta^D \rightarrow \mathbb{R}$ 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:*

$$u_{\text{lin}}(\mathbf{v}; \mathbf{w}) = \mathbf{w}^\top \mathbf{v}. \quad (2.41)$$

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:

$$U(\Pi; u) = \left\{ \pi \in \Pi \mid \exists \mathbf{w} \in \Delta^D. \forall \pi' \in \Pi : u(\mathbf{V}^\pi(s_0; 0); \mathbf{w}) \geq u(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}) \right\}. \quad (2.42)$$

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

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 $\mathbf{Vals}(\Pi') = \{\mathbf{V}^\pi(s_0; 0) \mid \pi \in \Pi'\}$.

Undominated sets often have an infinite cardinality, and as such are infeasible to compute. However, in undominated sets there are often many redundant 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:

$$\forall \mathbf{w} \in \Delta^D. \exists \pi \in CS(\Pi; u). \forall \pi' \in \Pi : u(\mathbf{V}^\pi(s_0; 0); \mathbf{w}) \geq u(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}). \quad (2.43)$$

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 $CCS_{\min}(\Pi) \subseteq CH(\Pi)$ is minimal if the following holds

$$\forall \pi \in CCS_{\min}(\Pi). \exists \mathbf{w} \in \Delta^D. \forall \pi' \in CCS_{\min}(\Pi) - \{\pi\}. u_{\text{lin}}(\mathbf{V}^\pi(s_0; 0); \mathbf{w}) > u_{\text{lin}}(\mathbf{V}^{\pi'}(s_0; 0); \mathbf{w}). \quad (2.44)$$

Now the geometry of convex coverage sets is considered, to see how something like $\mathbf{Vals}(CCS_{\min}(\Pi))$ 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 2.10). As a result, the values of the convex hull, $\mathbf{Vals}(CH(\Pi))$ lie on a *geometric (partial) convex hull* (see (b) in Figure 2.10). Moreover, any points in $\mathbf{Vals}(CH(\Pi))$ 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 2.10).

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 $\mathbf{Vals}(CCS_{\min}(\Pi))$ as the *optimal convex hull value set*.

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 \mathbf{V} it is defined as:

$$\text{cvx_prune}(\mathbf{V}) = \{\mathbf{v} \in \mathbf{V} \mid \exists \mathbf{w} \in \Delta^D. \forall \mathbf{v}' \in \mathbf{V} - \{\mathbf{v}\}. \mathbf{w}^\top \mathbf{v} > \mathbf{w}^\top \mathbf{v}'\}. \quad (2.45)$$

`cvx_prune` can be implemented using *linear programming* [TODO: cite](#), and an example of its operation on a set of vectors can be seen in (b) of Figure 2.10, where `cvx_prune` would prune all points but the blue circles.

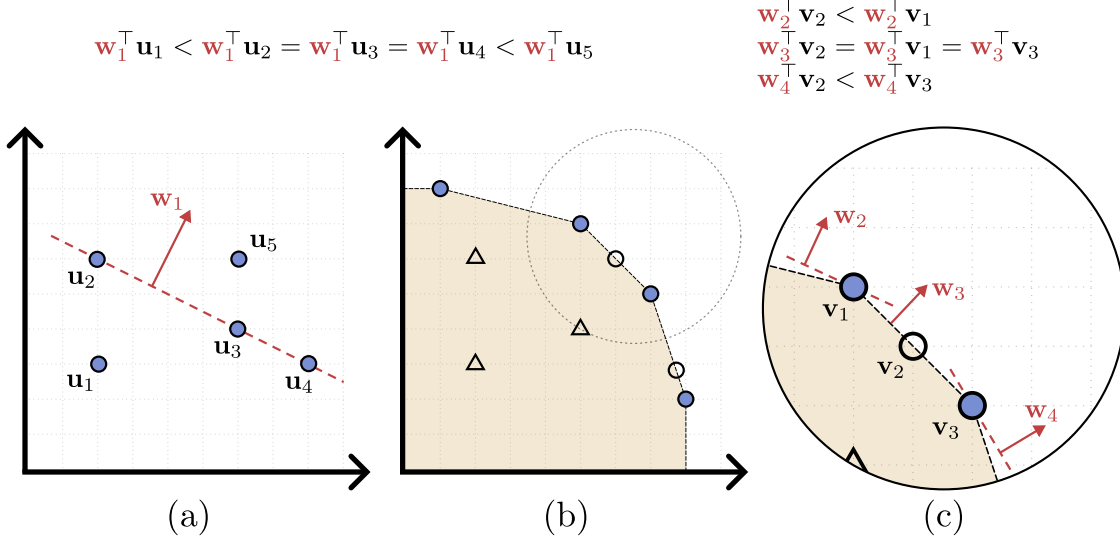


Figure 2.10: The geometry of convex coverage sets, shown with $D = 2$. In each image the points depicted correspond to a value set $\mathbf{Vals}(\Pi')$, for some set of policies Π' . (a) demonstrates that values $(\mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4)$ obtaining the same utility lie on a hyperplane. 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 $\mathbf{Vals}(CH(\Pi'))$, the blue circles form the minimal convex coverage set, $\mathbf{Vals}(CCS_{\min}(\Pi'))$, and triangles are not part of the convex hull. All of the unfilled shapes would be pruned by `cvx_prune`(Equation (2.44)); (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, it 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 $\mathbf{Vals}(\Pi')$

2.5.1 Convex Hull Value Iteration

Convex Hull Value Iteration (CHVI) [1] is a tabular dynamic programming algorithm similar to Value Iteration [TODO: ref](#), 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 $\hat{\mathbf{V}}_{\text{CHVI}}(s; t)$ and $\hat{\mathbf{Q}}_{\text{CHVI}}(s, a; t)$.

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 2.11 Given the vector sets \mathbf{U} and \mathbf{V} , define multiplication by a scalar s ,

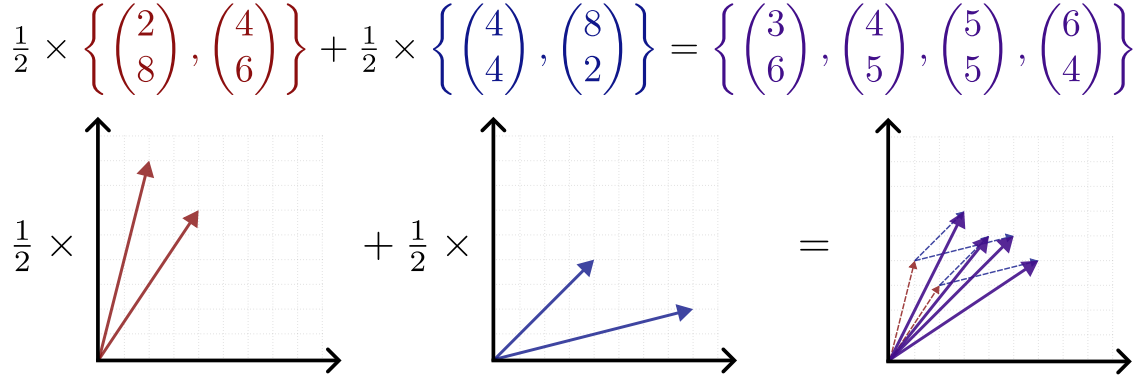


Figure 2.11: An example of arithmetic over vector sets. Two vector sets are shown in red and blue, which are multiplied by $\frac{1}{2}$ using Equation (2.45), and then added using Equation (2.46). On the right the resulting vector set is shown in purple, which are the sum of appropriately scaled red and blue vectors.

addition with a vector \mathbf{x} and addition between sets as follows:

$$\mathbf{x} + s\mathcal{V} = \{\mathbf{x} + s\mathbf{v} | \mathbf{v} \in \mathcal{V}\} \quad (2.46)$$

$$\mathcal{U} + \mathcal{V} = \{\mathbf{u} + \mathbf{v} | \mathbf{u} \in \mathcal{U}, \mathbf{v} \in \mathcal{V}\}. \quad (2.47)$$

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

$$\hat{\mathcal{V}}_{\text{CHVI}}^{k+1}(s; t) = \text{cvx_prune} \left(\bigcup_{a \in \mathcal{A}} \hat{\mathcal{Q}}_{\text{CHVI}}^{k+1}(s, a; t) \right), \quad (2.48)$$

$$\hat{\mathcal{Q}}_{\text{CHVI}}^{k+1}(s, a; t) = \mathbb{E}_{s' \sim p(\cdot | s, a)} [\mathbf{R}(s, a) + \hat{\mathcal{V}}_{\text{CHVI}}^k(s'; t + 1)]. \quad (2.49)$$

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 : $\bigcup_{a \in \mathcal{A}} \hat{\mathcal{Q}}_{\text{CHVI}}^{k+1}(s, a; t)$.

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 [1]:

$$Q_{\mathbf{w}}(s, a; t) = \max_{\mathbf{q} \in \hat{\mathcal{Q}}_{\text{CHVI}}(s, a; t)} \mathbf{w}^\top \mathbf{q}, \quad (2.50)$$

$$\pi_{\text{CHVI}}(s; t, \mathbf{w}) = \arg \max_{a \in \mathcal{A}} Q_{\mathbf{w}}(s, a; t). \quad (2.51)$$

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

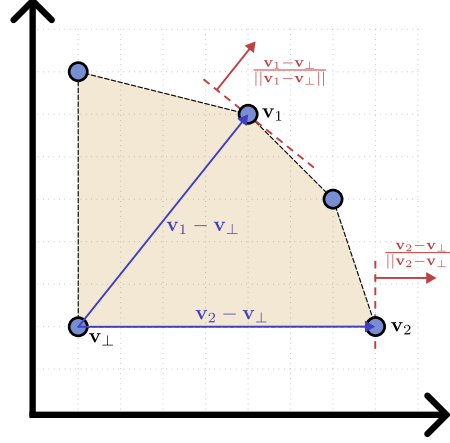


Figure 2.12: A visualisation, reusing the convex hull from Figure 2.10, demonstrating how to compute a weight vector that can be used to extract a specific value from a vector set. The hyperplane running through \mathbf{v}_i with normal $\mathbf{w}_i = \frac{\mathbf{v}_i - \mathbf{v}_\perp}{\|\mathbf{v}_i - \mathbf{v}_\perp\|}$ 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 (2.50) will give the desired value \mathbf{v}_i . **TODO: add the $\|\cdot\|_2$ bit and add the \mathbf{w}_i bit into diagram**

reference point, defined at each index as:

$$(v_\perp)_i = \min\{v'_i | \mathbf{v}' \in \hat{\mathcal{V}}_{\text{CHVI}}(s_0; 0)\}. \quad (2.52)$$

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 $\hat{\mathcal{V}}(s_0; 0)$, as is demonstrated in Figure 2.12. Hence, to extract a policy that achieves the value \mathbf{v} the weight \mathbf{w} should be set to:

$$\mathbf{w} = \frac{\mathbf{v} - \mathbf{v}_\perp}{\|\mathbf{v} - \mathbf{v}_\perp\|_2}, \quad (2.53)$$

and then follow $\pi_{\text{CHVI}}(s; t, \mathbf{w})$ as defined in Equation (2.50).

2.6 Sampling From Catagorical Distributions

TODO: Reference to chapter (TODO: ref ch:4-dents) section where talk about using this with THTS

some of the work in this thesis will involve sampling from catagorical distributions. Let $f : \{C_1, \dots, C_m\} \rightarrow [0, 1]$ be the probability mass function of a catagorical distribution with m categories. Suppose that we want to sample $c \sim f$. A naive

```

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 sampling from a catagorical distribution.

```

1 def sample_from_alias_table(alias_table):
2     index ~ Uniform({1,...,m})
3     (c0,c1,threshold) = alias_table[index]
4     u ~ Uniform(0,1)
5     if (u <= threshold):
6         return c0
7     return c1

```

Listing 2.3: Psuedocode for sampling from an alias table.

method to sample from f will take $O(m)$ time, where a 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* [TODO: cite1, cite2](#) 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 m th of the probability mass of f . Each entry takes the form $(c_0, c_1, \text{threshold})$, where $c_0, c_1 \in \{C_1, \dots, C_m\}$ are categories and $\text{threshold} \in [0, 1]$ 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 **threshold**, from which $c \sim f$ can be returned by looking it up in the table. Listing 2.4 gives psuedocode for sampling from an alias table, and Figure 2.13 visualises the construction and sampling of an alias table.

In [TODO: cite](#) it is shown [TODO: double check that it's proven](#) 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.

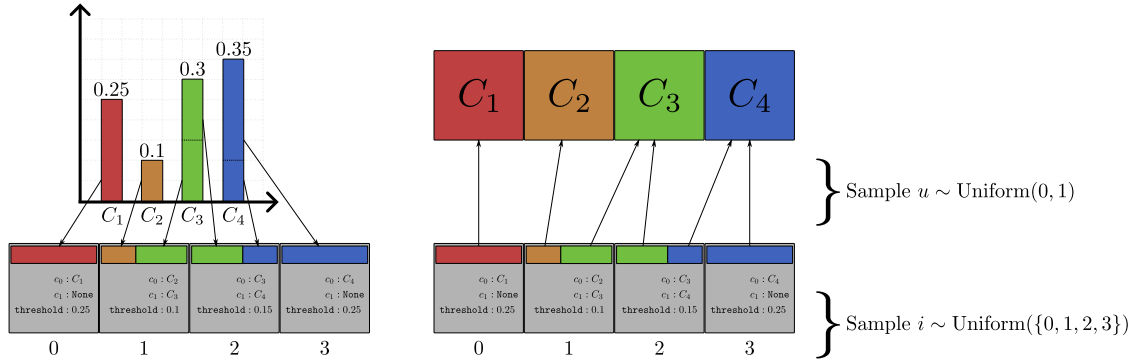


Figure 2.13: TODO: write caption TODO: shorten threshold, maybe to a symbol? So can clearly read what it says. Maybe also get rid of sampling bits, maybe make arrows bigger too

Pseudocode for constructing an alias table is given in Listing ??, following TODO: cite. 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” categories that have at most $\frac{1}{m}$ mass left to be added 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.

```

1  def build_alias_table(f):
2      # lists of categories and mass remaining
3      small = []
4      big = []
5
6      # fill big and small lists
7      for c in f:
8          if f(c) >  $\frac{1}{m}$ :
9              big.append((c, f(c)))
10             else:
11                 small.append((c, f(c)))
12
13     # construct alias table
14     alias_table = []
15     while len(small) > 0:
16         # create entry
17         c0, c0_mass_remaining = small.pop()
18         if mass_remaining ==  $\frac{1}{m}$ :
19             alias_table.append((c0, None, 1.0))
20             continue
21         c1, c1_mass_remaining = big.pop()
22
23         # add entry to table
24         alias_table.append((c0, c1, c0_mass_remaining))
25
26         # put c1 back in lists appropriately
27         c1_mass_remaining -= c0_mass_remaining
28         if c1_mass_remaining >  $\frac{1}{m}$ :
29             big.append((c1, c1_mass_remaining))
30         else:
31             small.append((c1, c1_mass_remaining))
32
33     return alias_table

```

Listing 2.4: Psuedocode for constructing an alias table.

3

Literature Review

Contents

3.1	Reinforcement Learning	41
3.2	Multi-Armed Bandits	42
3.3	Trial-Based Heuristic Tree Search and Monte-Carlo Tree Search	42
3.3.1	Trial Based Heuristic Tree Search	42
3.3.2	Monte-Carlo Tree Search	42
3.3.3	Maximum Entropy Tree Search	43
3.4	Multi-Objective Reinforcement Learning	43
3.5	Multi-Objective Monte Carlo Tree Search	44

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 $O(\log n)$ 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 [6]
- 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 scalarisation function that aren't 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/lizotte13a.html> 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 [Rojers 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)
- TODO: write about & make sure its implemented - its because just updating for 1 is more efficient in deterministic, and say that the additions can be implemented as updating for 1 value when deterministic
- 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	47
4.2	Boltzmann Search	48
4.3	Toy Environments	48
4.4	Theoretical Results	48
4.5	Empirical Results	48
4.6	Full Results	49

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	51
5.2	Contextual Tree Search	51
5.3	Contextual Zooming for Trees	52
5.4	Convex Hull Monte Carlo Tree Search	52
5.5	Results	52

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	53
6.2	Simplex Maps	54
6.3	Simplex Maps in Tree Search	54
6.4	Theoretical Results	54
6.5	Empirical Results	54

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

7.1	Summary of Contributions	55
7.2	Future Work	55

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



List Of Appendices To Consider

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

Bibliography

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] Eric A Hansen and Shlomo Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.
- [6] 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.

- [7] 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.
- [8] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.
- [9] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1:1–22, 2006.
- [10] Michael Painter. THTS++. Available at <https://github.com/MWPainter/thts-plus-plus>.
- [11] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- [12] David Silver and Joel Veness. Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, 23, 2010.
- [13] Chenjun Xiao, Ruitong Huang, Jincheng Mei, Dale Schuurmans, and Martin Müller. Maximum entropy monte-carlo planning. *Advances in Neural Information Processing Systems*, 32, 2019.