Regularization in Tree Search Algorithms

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Julia Grosse

aus Tübingen

Tübingen

2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	20.01.2026
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Philipp Hennig
2. Berichterstatterin:	Prof. Dr. Ulrike von Luxburg

Acknowledgements

Above all, I wish to thank Philipp Hennig for the privilege to conduct research in his group. His insights on computation, inference, and their interplay, as well as the high standards he sets in communicating those, are an invaluable source of inspiration for me. I am deeply grateful for his patience, his steady confidence in my ability to complete this thesis, and the thoughtful, human-centered atmosphere he cultivates within the group. I also want to express my gratitude for the guidance and advice that I received from Cheng Zhang. Her analytical, discerning, and constructive questions and the perspectives she brought to our meetings have been immensely helpful. My sincere thanks go to Agustinus Kristiadi for our collaboration on ULTS and for his insights into efficiently navigating a research project from choosing a topic to writing rebuttals. Thank you as well to my other co-authors Rahel Fischer, Roman Garnett, Ruotian Wu, Ahmad Rashid, and Pascal Poupart for challenging and enjoyable discussions. I am also grateful to Caterina De Bacco and Matthias Hein for serving on my thesis advisory committee, and to Ulrike von Luxburg, Georg Martius and Lena Schlipf for forming my examination committee. I am thankful to Microsoft Research for supporting my work through the EMEA PhD scholarship, and to the International Max Planck Research School for Intelligent Systems for providing a stimulating research environment. Last but not least, I would like to thank the former and current members of the MoML group, whose kindness and sharp humor have made it such a welcoming place. The time I got to spend with you will always hold a special place in my heart.

Abstract

Tree search algorithms form a fundamental class of optimization algorithms for discrete, sequential-decision-making problems and hierarchical approaches to continuous optimization. Their goal is to find a path from the root of a tree to a leaf node that maximizes an objective function over the leaf nodes. However, the number of leaves in a tree typically grows exponentially with the depth of the tree. As a result, it is necessary to balance the exploitation of promising paths with the exploration of new paths to search efficiently. Solving this trade-off between exploitation and exploration requires regularization, i.e., prior assumptions on the objective function or the optimal policy for traversing the tree. The main focus of this thesis is the design of probabilistic models and algorithms that can incorporate and exploit such assumptions, e.g., by having a Gaussian process prior over the objective function as it is common in the field of Bayesian optimization. We begin by formally connecting Bayesian optimization to Optimistic optimization, a class of non-probabilistic tree search algorithms, by mapping the kernel function of a Gaussian process prior to the semi-metric used by Optimistic optimization algorithms for regularizing the search problem. This leads to a runtime-efficient tree search algorithm for the Bayesian optimization setting, suitable for objective functions with low evaluation costs. In scenarios where evaluating the values of leaf nodes is expensive, reducing the overall number of such evaluations gains importance over the runtime efficiency of the decision-making process for which path to follow down the tree. We develop a Bayesian tree search algorithm that exploits correlations between nodes across the search tree to improve the search efficiency along this dimension. While the above approaches assume generic Gaussian priors for the objective function, there are also interesting non-Gaussian settings. An increasingly important example is the decoding process of Large Language Models (LLMs), where paths in the tree represent token sequences and the objective function is their likelihood. We assume a prior for the stepwise, categorical likelihood over the next token and under independence and regularity assumptions, develop an efficient probabilistic decoding method for LLMs. Finally, we shift our focus from probabilistic models for tree search to a particular tree search problem relevant as a subroutine in other probabilistic methods, like Bayesian quadrature. The problem consists of finding sets of k items (or points) that maximize the log-determinant over a Kernel Gram matrix. We consider a regularized, stochastic version of this search problem – which coincides with sampling from k-Determinantal Point Processes (k-DPPs). We suggest a greedy, approximate search strategy and show that it is a $(1 - 1/e)$-approximation to the optimal one. Taken together, this dissertation deepens the understanding of regularization in tree search methods through both theoretical analysis and empirical validation. It also proposes novel techniques that improve the data efficiency and computational performance of tree search algorithms.

Zusammenfassung

Baumsuchalgorithmen bilden eine fundamentale Klasse von Optimierungsverfahren für diskrete Probleme der sequenziellen Entscheidungsfindung sowie für hierarchische Ansätze in der Optimierung auf kontinuierlichen Räumen. Sie zielen darauf ab, denjenigen Pfad von der Wurzel zu einem Blattknoten zu finden, der eine Zielfunktion maximiert. Die Anzahl der Blätter wächst jedoch typischerweise exponentiell mit der Tiefe des Baums. Daher ist es notwendig, eine effiziente Suche durch ein ausgewogenes Verhältnis zwischen dem Weiterverfolgen vielversprechender Pfade (Exploitation) und der Erkundung neuer Pfade (Exploration) zu erreichen. Die Lösung dieses Zielkonflikts zwischen Exploration und Exploitation erfordert Regularisierung, d.h. a-priori-Annahmen über die Zielfunktion oder die Suchstrategie. Der Schwerpunkt dieser Dissertation liegt auf dem Entwurf probabilistischer Modelle und Algorithmen, die solche Annahmen integrieren und ausnutzen können – beispielsweise durch die Verwendung eines Gauß-Prozesses als Prior über der Zielfunktion, wie es im Bereich der Bayesschen Optimierung üblich ist. Wir beginnen damit, Bayessche Optimierung formell mit Optimistischer Optimierung – einer Klasse nicht-probabilistischer Baumsuchalgorithmen – in Beziehung zu setzen, indem wir die Kernfunktion eines Gauß-Prozesses auf die in Optimistischer Optimierung zur Regularisierung der Suchprobleme verwendete Semi-Metrik abbilden. Dies führt zu einem laufzeiteffizienten Baumsuchalgorithmus im Rahmen der Bayesschen Optimierung, der sich besonders für Zielfunktionen mit geringen Evaluationskosten eignet. In Szenarien, in denen die Auswertung der Blattknoten kostspielig ist, gewinnt die Reduktion der Gesamtanzahl solcher Auswertungen gegenüber der Laufzeiteffizienz des Entscheidungsprozesses darüber, welcher Pfad verfolgt wird, an Bedeutung. Zu diesem Zweck entwickeln wir einen Bayesschen Baumsuchalgorithmus, der Korrelationen zwischen Knoten im Suchbaum ausnutzt, um die Suche in dieser Hinsicht effizienter zu gestalten. Während die oben beschriebenen Ansätze von generischen Gaußschen Priors für die Zielfunktion ausgehen, gibt es auch interessante nicht-Gaußsche Szenarien. Ein zunehmend wichtiger Anwendungsfall ist der Dekodierungsprozess großer Sprachmodelle (engl. *Large Language Models*, LLMs), bei dem die Pfade im Baum Token-Sequenzen repräsentieren und die Zielfunktion deren Likelihood ist. Wir nehmen dabei einen Prior für die schrittweise, kategoriale Likelihood über das nächste Token an und entwickeln – unter Unabhängigkeits- und Regularitätsannahmen – eine effiziente probabilistische Dekodierungsmethode für LLMs. Abschließend verlagern wir den Fokus von probabilistischen Modellen für die Baumsuche hin zu einem speziellen Baumsuchproblem, das als Unterroutine in anderen probabilistischen Verfahren wie der Bayesschen Quadratur relevant ist. Das Problem besteht darin, Mengen aus k Elementen (oder Punkten) zu finden, welche die Log-Determinante einer Gramsche Matrix maximieren. Wir betrachten eine regularisierte, stochastische Version dieses Suchproblems – die dem Sampling aus k-Determinantalen Punktprozessen (k-DPPs) entspricht. Wir schlagen eine gierige (engl. *greedy*), approximative Suchstrategie vor und zeigen, dass sie eine $(1 - 1/e)$-Approximation des optimalen Verfahrens darstellt. Zusammenfassend vertieft diese Dissertation das Verständnis der Regularisierung in Baumsuchverfahren durch theoretische Analysen und empirische Validierung. Zudem werden neue Techniken vorgeschlagen, die die Daten- und Recheneffizienz von Baumsuchalgorithmen verbessern.

Contents

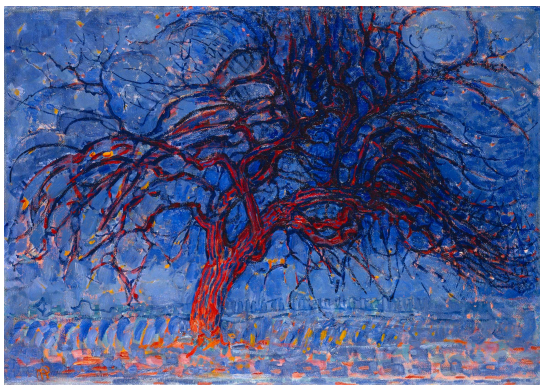
Introduction

Chapter 1

Introduction

The tree structure is a fundamental pattern in nature, art, and science. Figure 1.1 shows two paintings by the Dutch painter Piet Mondrian. In his early years, he painted trees in a figurative style as the one on the left. These early pieces laid the groundwork for his later, fully abstract compositions like the right one, for which he is best known. Mondrian wrote about his work *"In painting a tree, I progressively abstracted the curves: you can understand that very little 'tree' remained. […] Line and color must be composed otherwise than in nature."* (Mondrian et al. (1993), p. 77). However, that little "tree" that remained still displays certain properties such as the number of lines, where crossings are placed, and the size of the resulting cells. For some cells, one observes a color; for others, one does not. Cells with the same color tend to be close, but sometimes disconnected cells far away share the same color, too. We will not attempt to reconstruct the reasons for Mondrian's artistic choices, but we will see how similar properties are mirrored in the design of tree search algorithms.

This class of algorithms is central to Computer Science, and in particular to Machine Learning, where the tree serves as an abstract formalism of sequential decision-making problems. The goal is to learn a strategy for how to traverse the tree from the root to end up at a leaf node that maximizes a problem-specific reward function. Popular examples are game trees where the nodes of the trees represent states of the game boards and the edges between the nodes correspond to admissible moves. A win results in a positive reward, a loss in a negative reward, and a draw might result in zero reward. One of the most publicly visible successes of machine learning in the last decade, the win of AlphaGo (Silver et al., 2016) against Lee Sedol, took place in such a game-playing setting. However, the use of tree search algorithms extends well beyond that, with applications in practically relevant tasks such as chemical synthesis planning (Segler et al., 2018; Wang et al., 2020;



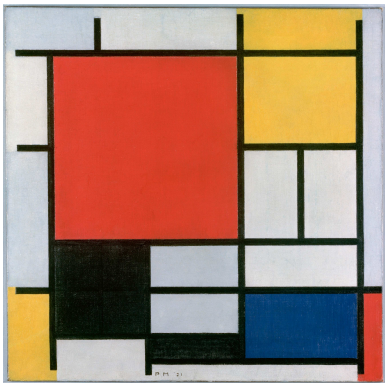


Figure 1.1.: Paintings from Piet Mondrian. Left: Evening; Red Tree (1908-10) **Right:** Composition with Large Red Plane, Yellow, Black, Grey and Blue (1921). Image sources: Public domain. Wikimedia Commons. https://commons.wikimedia.org/wiki/Piet_Mondrian.

Lee et al., 2020; Kreutter and Reymond, 2023) in the Natural Sciences; the scheduling of trains (Wang et al., 2024) and earth-observing satellites (Herrmann and Schaub, 2022) in Engineering; and automatic code generation (Lim and Yoo, 2016) or chip layout optimization (He and Bao, 2020) within Computer Science. In order to equip LLMs with reasoning abilities, tree search is on the rise again, as seen in recent work (Yao et al., 2024; Zhang et al., 2024; DeLorenzo et al., 2024; Hu et al., 2023; Feng et al., 2023; Chen et al., 2024b). The primary challenge for tree search algorithms is managing the combinatorial explosion of the search space. As the number of possible paths grows exponentially with the depth of the tree, these algorithms must balance exploration, by discovering new options, with exploitation, by focusing on the most promising paths. To improve search efficiency, *regularization techniques* are employed, incorporating additional assumptions about the underlying problem structure.

In this work, we will look at this problem through a probabilistic lens, aiming to incorporate such assumptions via a prior distribution over the possible values of the reward function in yet-to-explore parts of the search space. Additionally, probabilistic inference allows for tracking computational uncertainty, which could further help with improving the exploration-exploitation trade-off, as known from other Bayesian optimization methods. We will focus on the relationship between regularization in Bayesian optimization and the regularization found in a group of classical tree search algorithms in the field of optimization. Another aspect of interest is whether search efficiency can be improved by incorporating stronger priors, particularly those that express relationships between disconnected nodes that are potentially far apart in the search tree. Additionally, we will explore a setting where the *number* of potentially promising paths is regularized. However, even determining a sequence of actions that optimally explores a search space – without exploitation on top of that – is a difficult one. k-DPPs offer an elegant formalism for modeling exploration, making them broadly applicable across various machine learning domains (Garnett, 2023; Nava et al., 2022; Kathuria et al., 2016; Wang et al., 2017; Bardenet and Hardy, 2016; Belhadji et al., 2019; Gautier et al., 2019a; Kulesza and Taskar, 2012; Wilhelm et al., 2018; Kang, 2013; Mariet and Sra, 2015; Zhang et al., 2017; Elfeki et al., 2019). We therefore dedicate the last part of the thesis to this class of processes by formulating sampling from a k-DPP as a regularized tree search problem and deriving a runtime-efficient, approximate solution with theoretical guarantees on the approximation quality.

1.1. Outline and Main Contributions

This thesis is divided into two main parts. The first part establishes the essential theoretical background on existing algorithms and regularization techniques, while the second part presents novel tree search methods and theoretical insights on their performance by connecting, generalizing, and analyzing those techniques for a variety of tree search problems.

Part I: Background

Chapter 2 formally introduces the tree search setting considered in this thesis. It also provides an overview of fundamental search strategies on trees – including greedy search, beam search, A* search, and Monte Carlo Tree Search – which not only still remain in use, but also serve as foundational templates for the more advanced algorithms discussed and developed in later chapters.

Chapter 3 explores advanced regularization techniques and key design principles in tree search. We begin with the Optimistic Principle, reviewing its practical application in the widely used *Upper Confidence Bounds applied to trees* (UCT) algorithm for discrete, sequential decision-making, and continue with the Optimistic optimization algorithm class that embodies this principle for optimization over continuous domains. Next, we introduce the framework of Maximum Entropy regularized objectives and their role in decision-making under uncertainty. Finally, we examine Bayesian optimization, summarize its core concepts, and discuss a Bayesian model for tree search that serves as a foundation for some of the contributions presented in the second part of this thesis.

Part II: Contributions

Chapter 4 focuses on the connection between Optimistic optimization and Bayesian optimization. By mapping the kernel to a dissimilarity, we obtain an Optimistic optimization algorithm named **Gaussian-Process Optimistic Optimization** (GP-OO) for the Bayesian optimization setting with a run-time of up to $\mathcal{O}(N \log N)$ where N denotes the number of function evaluations. We examine to which extent the conceptual advantages of Bayesian optimization can be integrated with the computational efficiency of Optimistic optimization.

Disclaimer

Chapter 4 is based on the peer-reviewed journal publication *'Optimistic Optimization of Gaussian Process Samples'* by J. Grosse, C. Zhang, and P. Hennig, published in TMLR (2023), with the following co-author contributions:

	Ideas (%)	Experiments (%)	Analysis (%)	Writing (%)
J. Grosse	80	100	100	80
C. Zhang	5	0	0	10
P. Hennig	15	0	0	10

Section 2 of this publication has been removed from Chapter 4 and has instead been incorporated into Chapter 3, Sections 3.1.3 and 3.3.1, with minor extensions.

Chapter 5 addresses sequential decision-making problems in settings where the underlying state spaces exhibit latent structures that go beyond what tree-based models can

capture. To tackle this, we develop a probabilistic search algorithm, **Probabilistic DAG Search** (PDAGS), which exploits the latent structure of the search space by enabling information sharing across the search tree. The method combines approximate inference in jointly Gaussian models for the explored portion of the tree with an abstraction of the unexplored part, reducing inference complexity through an imposed regularity assumption.

Disclaimer

Chapter 5 is based on the peer-reviewed conference publication *'Probabilistic DAG Search'* by J. Grosse, C. Zhang, and P. Hennig, presented at UAI (2021), with the following co-author contributions:

	Ideas (%)	Experiments (%)	Analysis (%)	Writing (%)
J. Grosse (during M.Sc.)	20	20	20	10
J. Grosse (during Ph.D.)	40	80	70	70
C. Zhang	20	0	5	10
P. Hennig	20	0	5	10

Section 3.2.1 of this publication has been removed from Chapter 5 and has instead been incorporated into Chapter 3, Section 3.3.2 as Box 3.3.1. 3.3.2 in Chapter 3 is also based in parts on Section 3.2 of this publication.

Chapter 6 again takes a probabilistic stance on tree search, albeit on a very different search space structure: The paths are sentences and the objective value is the likelihood of a sentence under an LLM. We define a prior belief over the LLM's output and obtain a posterior belief over the most promising paths in each iteration. Based on these beliefs, we derive a Bayesian optimization-like algorithm named **Uncertainty-Guided Likelihood-Tree Search** (ULTS) that allows for a more efficient exploration scheme than standard LLM decoding algorithms.

Disclaimer

Chapter 6 is based on the preprint "Uncertainty Guided Likelihood Tree Search" by J. Grosse, R. Wu, A. Rashid, C. Zhang, P. Hennig, P. Poupart, and A. Kristiadi, currently under review and available on arXiv at `https://arxiv.org/pdf/2407.03951` (2025), with the following co-author contributions:

	Ideas (%)	Experiments (%)	Analysis (%)	Writing (%)
J. Grosse	35	75	50	55
R. Wu	5	0	5	0
A. Rashid	5	0	5	0
C. Zhang	10	0	0	0
P. Hennig	15	0	5	5
P. Poupart	5	0	15	5
A. Kristiadi	25	25	20	35

Chapter 7 introduces a formulation for *sampling* from a k-DPP as a regularized version of *searching* for sequences that maximize the Gaussian entropy function. We then derive a greedy approximation to the optimal search policy and prove a constant-factor approximation guarantee for it.

Disclaimer

Chapter 7 is based on the peer-reviewed conference publication *'A Greedy Approximation for k-Determinantal Point Processes'* by J. Grosse, R. Fischer, R. Garnett, and P. Hennig, presented at AISTATS (2024), with the following co-author contributions:

	Ideas (%)	Experiments (%)	Analysis (%)	Writing (%)
J. Grosse	40	55	75	50
R. Fischer	10	5	5	0
R. Garnett	25	20	10	25
P. Hennig	25	20	10	25

Chapter 8 concludes this thesis with a summary and a discussion of the overarching themes that govern the individual contributions. In particular, we focus on the role of structural assumptions, concentration properties, robustness, and curvature for regularizing tree search algorithms.

Background

CHAPTER 2

The Roots of Tree Search

2.1. Trees and Graphs

An undirected **graph** $\mathcal{G}$ is defined as an ordered pair $\mathcal{G} = (\mathcal{X}, \mathcal{E})$, where $\mathcal{X} = \{x_0, x_1, \dots, x_n\}$ is a non-empty set of elements called **nodes** and $\mathcal{E} \subseteq \{\{x_i, x_j\} \mid x_i, x_j \in \mathcal{X}\}$ is a set of unordered pairs of nodes, called **edges**. In a **directed graph** the edges are ordered pairs of nodes, i.e. $\mathcal{E} = \{(x_i, x_j) \mid x_i, x_j \in \mathcal{X}\}$. A **path** ω is a sequence of nodes $(x_1, x_2, \dots, x_k)$ where $x_i \in \mathcal{X}$ for all $1 \leq i \leq k$ such that there is an edge between each consecutive pair of nodes in the sequence, i.e. $\{x_i, x_{i+1}\} \in \mathcal{E}$ for all $1 \leq i < k$ in the case of undirected graphs and $(x_i, x_{i+1}) \in \mathcal{E}$ for all $1 \leq i < k$ for directed graphs. If the path between node x_1 and x_k is unique or clear from the context, we will refer to it as $\omega_{1 \to k}$. A **cycle** is a path with $x_1 = x_k$. Graphs without cycles are called **acyclic**. A undirected **tree** is an undirected graph in which any two nodes are connected by exactly one path. A directed tree is a **directed acyclic graph** (DAG) whose underlying undirected graph is a tree. A rooted tree is a tree in which one node has been designated the **root**. For the remainder of the text, we will refer to rooted trees simply as trees and define x_0 to be the root. In a rooted tree, the **parent** of node x_c is the node x_p connected to x_c on the unique path to the root x_0; the root itself does not have a parent. A **child** x_c of a node x_p is a node of which x_p is the parent. Nodes that share a parent are referred to as **siblings**. We use the notation $\text{parent}(x_i), \text{children}(x_i)$ and $\text{siblings}(x_i)$ to refer to the respective nodes for a node x_i. An **ascendant** x_a of a node x_d is any node on the path from x_d to the root. Conversely, a node x_d is considered a **descendant** of x_a if x_a lies on the path connecting x_d to the root. Nodes with no children are called **leaves**, and the remaining nodes are **internal nodes**. We use $\mathcal{L}$ to denote the set of leaves. The **depth** (or **level**) of a node is the number of edges on the path from the root to this node. The **height** of a node is the number of edges on the longest path from the node to a leaf. The depth D and height H of a tree are the maxima over the corresponding quantity over all nodes. The **branching size** (or **branching factor**) b of a node is the number of its children. The **branching size** of a tree typically refers to the maximum or average branching size across all internal nodes.

2.2. Tree Search

Having reviewed the terminology of trees, we now move on to search algorithms on trees[1]. Tree search is a technique in Computer Science for systematically exploring the nodes of a tree data structure. The objective of tree search algorithms is to search through the nodes of the tree to find a particular node or path. Within machine learning and for the

1: Many of the search algorithms presented in this section also extend to general graphs.

scope of this thesis, this will usually be a path that maximizes **rewards** along the nodes of the path. Formally, a reward function $r : \mathcal{X} \to \mathbb{R}$ assigns a reward $r(x_i) = r_i$ to each node x_i in the tree, and the goal is to solve

$$\omega^* = \arg\max_{\omega \in \Omega} \sum_{x_i \in \omega} r_i, \tag{2.1}$$

where $\Omega = \{\omega = (x_0, ..., x_l) \mid x_l \in \mathcal{L}\}$ is the set of all paths leading from the root x_0 to a leaf x_l. We introduce the shortcut $r_{i \to j}$ to denote the cumulative reward $r_{i \to j} = \sum_{x_k \in \omega_{i \to j}} r_k$ along the path $\omega_{i \to j}$.
A crucial concept for solving the Equation 2.1 is that of **optimal values**, which represent the best possible result that can be achieved from a particular node. The optimal value v_i of a node x_i is defined recursively as:

$$v_i = \begin{cases} r_i & \text{if } x_i \in \mathcal{L}, \\ \max\limits_{x_c \in \text{children}(x_i)} \{r_c + v_c\} & \text{otherwise.} \end{cases} \tag{2.2}$$

The above equation is referred to as the **Bellman equation** and can be solved via **Dynamic Programming**: Starting at the leaf nodes, one computes the optimal values for the sub-trees in a *bottom-up* fashion and re-uses the solutions to these sub-problems for the computation of the optimal values in the level above. The optimal path ω^* is then recovered by starting at the root node and choosing the child node with the highest optimal value. Despite operating in a divide-and-conquer fashion, the exponential number of sub-problems usually still makes dynamic programming prohibitively expensive. Tree Search algorithms, therefore, typically operate in a *top-down* manner, starting to explore the search space from the root node. There are two fundamentally opposite ways in which one can traverse a tree – **depth-first** or **breadth-first** – and all search algorithms are located somewhere in the spectrum in between. In depth-first search (Figure 2.1), one explores as far as possible along each branch before backtracking. The idea is to go deep down the tree before visiting sibling nodes. In breadth-first search (Figure 2.2), on the other hand, one explores all nodes at the present depth before moving on to nodes at the next depth. However, to be efficient, it is necessary to prune nodes, i.e. not follow up on certain paths anymore. This is when it becomes important to trade-off between breadth-first and depth-first strategies, or in other words, between exploration and exploitation. How this is done is specified by a **search policy**. A **policy** is a function that maps a node to a probability distribution over nodes, typically the node's children. Formally, a policy can be defined as $\pi : \mathcal{X} \to \mathcal{P}_{\mathcal{X}}$ where $\mathcal{P}_{\mathcal{X}}$ denotes the set of probability distributions over the set of nodes $\mathcal{X}$. A policy is called *deterministic* if π maps all nodes to probability distributions that place all the mass on a single node. Otherwise, the policy is called *stochastic*. Often, one distinguishes between the search policy that guides the search and the **target policy** that is used during the evaluation.

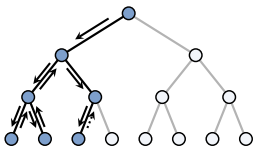


Figure 2.1.: Depth-First Traversal explores as far down a branch as possible before backtracking to explore other branches.

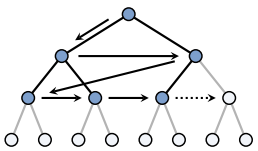


Figure 2.2.: Breadth-First Traversal explores all nodes at the current depth level before moving to the next level.

Remark 1. An elementary distinction between tree search settings lies in the availability of **intermediate rewards**, i.e., rewards assigned to internal nodes $\mathcal{X} \setminus \mathcal{L}$. Formally, the case without intermediate rewards can be treated as a special instance of the general setting by assigning zero reward $r_i = 0$ to all internal nodes $x_i \in \mathcal{X} \setminus \mathcal{L}$. However, this seemingly

minor formal difference has non-trivial algorithmic implications: strategies that rely on local reward signals – such as the heuristic, deterministic search algorithms discussed below – become less straightforward to apply, as they can no longer distinguish effectively among internal nodes during traversal.

Remark 2. In some settings the goal is not to maximize accumulated reward, but to compute the outcome of an optimal strategy against an opponent. These problems are modeled by *min-max trees*, where nodes alternate between decision points for the maximizing and minimizing player. The value function for node x_i is then defined recursively as

$$v_i = \begin{cases} r_i & \text{if } x_i \in \mathcal{L}, \\ \max\limits_{x_c \in \text{children}(x_i)} v_c & \text{if } x_i \text{ is a max node}, \\ \min\limits_{x_c \in \text{children}(x_i)} v_c & \text{if } x_i \text{ is a min node}. \end{cases} \tag{2.3}$$

While min-max trees fall outside the additive reward framework of Equation 2.1, they still fit into the broader view of tree search as recursive value computation and often do not require major modifications to the search principles. We will refer to this variant in one of the experiments in Chapter 5, but otherwise focus on the non-adversarial setting.

Relation to Reinforcement Learning

Tree search can be seen as a special case of reinforcement learning. Consider a **Markov Decision Process** (MDP) defined as a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$, where $\mathcal{S}$ is the set of states and $\mathcal{A}$ is a set of actions. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, which describes the probability of reaching a new state given a current state and action. The discount parameter $\gamma \in [0, 1]$ determines how much future rewards are valued compared to immediate rewards and $\gamma < 1$ ensures that the optimization problem stays well-defined in infinite-horizon settings. A policy $\pi \in \Pi$ is a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that gives the probability $\pi(a \mid s)$ of selecting action a in state s. A given policy π defines a function over state action-pairs (s, a) representing the expected cumulative discounted reward the agent can obtain when it starts by taking action a in state s and follows π thereafter. This function, $Q^\pi(s, a)$, is formally defined as:

$$Q^\pi(s, a) = \mathbb{E}_{\pi,P,R}\left[\sum_{i=0}^{\infty} \gamma^i r_{i+1} \,\middle|\, s_0 = s, a_0 = a \right], \tag{2.4}$$

where r_{i+1} is the reward received after the i-th transition when following policy π beginning at state $s_0 = s$ with action $a_0 = a$. From the **Q-Values**, the so-called **V-Values** can be derived as

$$V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s, a). \tag{2.5}$$

They represent the expected reward when following cumulative discounted reward when following policy π by marginalizing over the actions available in state s. The goal of

solving an MDP is to find the optimal policy π^*, which maximizes the expected cumulative discounted reward for the agent across all states and actions.
In the special case of tree search, the MDP is finite-horizon, deterministic, and acyclic. The state space $\mathcal{S}$ corresponds to the tree's nodes $\mathcal{X}$, and the action space $\mathcal{A}$ determines the available transitions between a parent and its children. The transition function P and reward function R are deterministic, and the discount factor is set to $\gamma = 1$, reflecting the finite planning horizon we assumed for tree search. The reward r_c at node x_c is interpreted as the reward for taking the action $a_{p\to c}$ that leads to a child x_c from a parent node x_p. Due to the determinism, the definition of the V-values and Q-values at a node x_i under the optimal policy π^* simplifies in tree search to (cf. Equation 2.2):

$$V^{\pi^*}(x_p) = \max_{x_c \in \text{children}(i)} \left\{ r_c + V^{\pi^*}(x_c) \right\} = \max_{x_c \in \text{children}(i)} Q^{\pi^*}(x_p, a_{p\to c}), \tag{2.6}$$

with $Q^{\pi^*}(x_p, a_{p\to c}) := r_c + v_c$. Throughout the remaining text, we use the short-cut v_i to denote the **optimal V-value** $V^{\pi^*}(x_i)$ of a node x_i. Several of the results in the literature presented in Section 3.2 that were originally derived for the general reinforcement learning setting are rephrased for the tree search setting for the purpose of this text.

2.2.1. Deterministic Heuristic Search

This section introduces three simple yet influential search strategies – greedy search, beam Search, and A* Search – that form the algorithmic foundation for the methods developed in Chapters 6 and 7. For greedy search, we additionally examine its theoretical properties in the context of submodular set function maximization, which are central to the analysis in Chapter 7. For beam search, we focus on its regularization behavior in a setting where the reward function is a log-likelihood objective as such setting motivates the algorithmic design of ULTS in Chapter 6. A* Search is introduced as a first instance of an optimistic search, also related to ULTS.

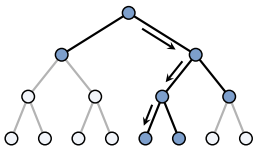


Figure 2.3.: Greedy Search descents exactly one path. At each node, the intermediate rewards of all children are considered, and the one with the highest is selected. In this schematic example, evaluated nodes are colored, and the traversed path is indicated with the arrows.

Greedy Search

Greedy search is an algorithm that makes decisions based on the best immediate, or "greedy", choice at each step (see Algorithm 1, Figure 2.3). Once a decision is made, the algorithm does not reconsider its choices. The search terminates as soon as a leaf node $x_l \in \mathcal{L}$ is reached.
Greedy search does not always find the optimal path, but the structural properties that influence its performance have been studied extensively in combinatorial optimization, beginning with the seminal work by Edmonds (1971) on matroids[2] and in many works thereafter. One of those insights is that the greedy algorithm achieves an approximation ratio of at least $1 - e^{-1}$ for monotonically increasing, submodular set functions (Nemhauser et al., 1978) under a cardinality constraint on the solution set. Let $\mathcal{J}$ be a finite set. A *set function* is a function $f : 2^{\mathcal{J}} \to \mathbb{R}$, which assigns a real value $f(\mathcal{I})$ to each subset $\mathcal{I} \subseteq \mathcal{J}$. Here, $2^{\mathcal{J}}$ denotes the power set of set $\mathcal{J}$. For such a function, the discrete derivative at

2: **Matroids** formalize the concept of independence in a combinatorial setting, capturing structural properties analogous to linear independence in vector spaces, acyclic subgraphs in graph theory and certain entropic structures in information theory. For a technical introduction to matroids and further properties of the greedy algorithm, see Oxley (2006) or Korte and Lovász (1984).

Algorithm 1: Greedy Search

```
GREEDY-SEARCH(r, G):
    x_i = x_0 // Begin at root
    while x_i ∉ L do
        x_i = arg max_{x_c ∈ children(x_i)} r_c // Child with highest immediate reward
    return ω_{0→i} // Return traversed path
```

set $\mathcal{I} \in 2^{\mathcal{J}}$ in the direction of element $j \in \mathcal{J}$ is defined as the marginal gain obtained by adding element j to the set $\mathcal{I}$:

$$\Delta_j(\mathcal{I}) = f(\mathcal{I} \cup \{j\}) - f(\mathcal{I}). \tag{2.7}$$

The set function f is said to be monotonically increasing if

$$\Delta_j(\mathcal{I}) = f(\mathcal{I} \cup \{j\}) - f(\mathcal{I}) \geq 0, \quad \forall \mathcal{I} \subseteq \mathcal{J}, \quad \forall j \in \mathcal{J} \setminus \mathcal{I}. \tag{2.8}$$

and submodular if

$$\mathcal{K} \subseteq \mathcal{I} \subseteq \mathcal{J} \implies \Delta_j(\mathcal{K}) \geq \Delta_j(\mathcal{I}), \quad \forall j \in \mathcal{J} \setminus \mathcal{I}. \tag{2.9}$$

Intuitively, submodularity leads to a *diminishing returns*: The more items a set already contains, the less one gains by adding new items. The bound of Nemhauser et al. (1978) can be tightened by taking the *total curvature* of the set function into account. The total curvature of a set function measures how close the function is to being *modular*[3] by quantifying how much adding an element to a set depends on the elements already in the set. It is defined as:

$$\kappa = \max_{j \in \mathcal{J}^*} \left(1 - \frac{\Delta_j(\mathcal{J} \setminus \{j\})}{\Delta_j(\varnothing)}\right), \tag{2.10}$$

where $\mathcal{J}^* = \{j \in \mathcal{J} \mid \Delta_j(\varnothing) > 0\}$ is the set of elements with positive initial marginal contribution. Conforti and Cornuéjols (1984) show that the algorithm is an $(1 - e^{-\kappa})/\kappa$ approximation for curvature $0 < \kappa < 1$.

3: A set function is said to be modular, if it is additive over the elements:

$$f(\mathcal{I}) = \sum_{i \in \mathcal{I}} w_i,$$

where $w_i \in \mathbb{R}$ for each $i \in \mathcal{I}$

Definition: c-approximation (Williamson and Shmoys (2011), Def. 1.1)

A c-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of c of the value of an optimal solution.

When both modularity and matroid structure are present, the greedy algorithm provably finds an optimal solution. These combinatorial constraints therefore impose implicit structure on the solution space, i.e. are acting as a form of *structural regularization*.

Beam Search

An extension of greedy search is beam search (Reddy (1977), Algorithm 2, Figure 2.4). It builds upon the idea of breadth-first search but limits the number of paths that are explored at each level of the tree, known as the *beam width* or *beam size*. Starting at the root node, one expands the best k options based on the rewards observed so far. Out of the newly added options, one again takes the k best and so on. This process repeats iteratively until a leaf node is reached or a predefined number of iterations is completed. Beam search with $k = 1$ corresponds to greedy search.

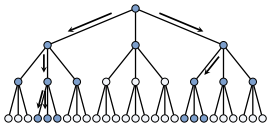


Figure 2.4.: Beam Search expands exactly k (here $k = 2$) nodes per level. The intermediate rewards of all children of these k nodes are examined and out of their union the k best paths ("beams") are followed. Beams can end in a level (e.g., such the right beam on the second level), or split into more than one (e.g., such the left beam on the second level).

Algorithm 2: Beam Search

BEAM-SEARCH$(r, \mathcal{G}, k)$**:**
- $\mathcal{B} = \{x_0\}$ // Set of beams initially only contains root
- **while** $\mathcal{B} \cap \mathcal{L} = \varnothing$ **do**
  - $\mathcal{C} = \bigcup_{x_b \in \mathcal{B}} \text{children}(x_b)$ // Expansion
  - $\mathcal{B} = \underset{x_c \in \mathcal{C}}{\text{top-}k}\, r_{0 \to c}$ // Selection
- // Return beam with highest cumulative reward
- $x_l = \underset{x_b \in \mathcal{B}}{\arg\max}\, r_{0 \to b}$
- **return** $\omega_{0 \to l}$

Meister et al. (2020b) study the regularization properties of beam search when applied to log-likelihood maximization, particularly in the context of Natural Language Processing. Formally, the goal is to find a sequence of nodes ω^* that maximizes a log-likelihood reward function:

$$\omega^* = \underset{\omega \in \Omega}{\arg\max} \sum_{i=1}^{|\omega|} \underbrace{\log p(x_i \mid x_1, ..., x_{i-1})}_{r_i}, \tag{2.11}$$

where $\omega = (x_1, ..., x_T)$ is a path (sequence of nodes) and $|\omega| = T$. Meister et al. (2020b) slightly deviate from this standard objective by returning a set Ω_k of k sequences instead of a single sequence. Since beam search does not guarantee finding the maximum likelihood sequence, they propose augmenting the objective with a regularization term $\mathcal{R}(\Omega_k)$ weighted by regularization strength λ:

$$\Omega_k^* = \underset{\substack{\Omega_k \subseteq \Omega \\ |\Omega_k| = k}}{\arg\max} \left[\sum_{\omega \in \Omega_k} \sum_{i=1}^{|\omega|} \log p(x_i \mid x_1, ..., x_{i-1}) \right] - \lambda \mathcal{R}(\Omega_k). \tag{2.12}$$

The key question is how to define $\mathcal{R}(\Omega_k)$ such that beam search with beam size k exactly optimizes this objective. Meister et al. (2020b) propose the following regularization

term:

$$\mathcal{R}(\Omega_k) = \sum_{t=1}^{T} \left(u_t(\Omega_k^t) - \min_{\substack{\Omega_k' \subseteq \mathcal{B}_t \\ |\Omega_k'|=k}} u_t(\Omega_k') \right)^2, \tag{2.13}$$

where:

- $\Omega_k^t = \{\omega \in \Omega_k \mid |\omega| = t\}$ is the set of all sequences in Ω_k of length t,
- $u_t(\Omega_k^t)$ is the negated likelihood *for step* t (accumulated over all sequence in Ω_k^t):

$$u_t(\Omega_k^t) = - \sum_{\omega=(x_1,\ldots,x_t)\in\Omega_k^t} \log p(x_t \mid x_1, \ldots, x_{t-1}), \tag{2.14}$$

- and $\mathcal{B}_t$ is the set of beam hypotheses at time step t, defined recursively as:

$$\mathcal{B}_0 = \{x_0\}, \tag{2.15}$$

$$\Omega^t = \arg\max_{\substack{\Omega' \subseteq \mathcal{B}_t \\ |\Omega'|=k}} \sum_{\omega\in\Omega'} \sum_{i=1}^{|\omega|} \log p(x_i \mid x_1, \ldots, x_{i-1}), \tag{2.16}$$

$$\mathcal{B}_{t+1} = \{(\omega, x_{t+1}) \mid \omega \in \Omega^t, x_{t+1} \in \mathcal{X}\}. \tag{2.17}$$

With this regularizer, the following result holds:

Theorem 2.2.1 (Meister et al., 2020) The regularized objective in Equation 2.12, using the regularizer in Equation 2.13, is optimally solved by beam search with beam size k in the limit $\lambda \to \infty$.

From a theoretical standpoint, the result stated in Theorem 2.2.1 appears rather limited in strength: the squared difference at depth t in the regularizer (2.13) is minimized *by construction* of the beam search algorithm, and this term solely dominates the objective as $\lambda \to \infty$. Nonetheless, Meister et al. (2020b) offer an interesting interpretation of this regularization scheme by connecting it to the *Uniform Information Density hypothesis*[4], a concept in Cognitive Science. Since $-\log p(x_t \mid x_0, \ldots, x_{t-1})$ formalizes how surprising x_t is, $u_t(\Omega^t)$ quantifies the total surprise in the explored paths up to step t. The regularizer penalizes deviation from the minimal possible surprisal at each time step t.
To the best of our knowledge, as well as according to recent work from Chen et al. (2024a), stronger theoretical explanations of the regularization properties of beam search beyond the above statement have not yet been offered.

4: **Uniform information density hypothesis** (Jaeger, 2010): Speakers tend to favor utterances that spread information more evenly across what they say. When multiple grammatical options are available to express the same message, they generally prefer the version that achieves a smoother, more uniform distribution of information (all else being equal).

A* Search

While beam search does not take future rewards into account at all, one of the first attempts to do so via problem-specific knowledge is the A* search algorithm (Hart et al. (1968b),

Algorithm 3). This algorithm was originally developed to find the shortest path between a start node and a target node in a graph, but the tree search objective to find a path that maximizes a reward function as defined by Equation 2.1 can be easily converted into one by minimizing the negated distance function as well as connecting all leaf nodes to an additionally introduced target node with zero reward. For the A* algorithm to arrive at the optimal solution, rewards in the shortest-path problem have to be positive, so a necessary requirement in the tree search setting is that all rewards are originally negative. A* algorithm relies upon a problem-specific heuristic function $h : \mathcal{X} \to \mathbb{R}$, that estimates the remaining reward for each node. Starting at the root node x_0, A* search in the tree search setting iteratively selects the node with the highest sum of the rewards observed so far and the heuristic for the future rewards. If this node coincides with the target node, the search concludes. Otherwise, the algorithm examines the unexplored children of the selected node. A* search discovers the optimal path on trees and graphs, provided the chosen heuristic function is admissible meaning the heuristic function never overestimates the actual cost or, respectively, underestimates the actual reward. The special case of A* with $h(x_i) = 0 \, \forall x_i \in \mathcal{X}$ is known as the **Dijkstra Algorithm** (Dijkstra, 2022).

Algorithm 3: A Star Search

```
1 A-STAR-SEARCH(r, G, h):
2     B = {x_0} // Start at root
3     while x_i ∉ L do
          // Select based on highest reward observed so far *and*
             estimated remaining reward
4         x_i = arg max_{x_b ∈ B} r_{0→b} + h(x_b)
          // Expansion
5         B = (B \ {x_i}) ∪ children(x_i)
6     return ω_{0→i}
```

2.2.2. Monte-Carlo-Tree-Search

The search strategies above heavily rely on the information from the rewards at internal nodes; however, there are scenarios where only the leaf nodes have non-zero rewards, e.g. think of a game tree where only at the end of the game it can be determined who won or lost. Domain-specific heuristics, as required by A* search, are not always available or costly to design. Abramson (1990) therefore introduced the idea of *estimating* the value of the nodes at the frontier through random rollouts, laying the groundwork for a class of tree search algorithms that would later become known as **Monte Carlo Tree Search** (MCTS, Coulom (2006)) algorithms. MCTS comprises tree search algorithms that iteratively build a *search tree* by repeating the following four steps:

Step 1: Selection. This phase begins at the root node x_0 of the search tree. The search tree is traversed by selecting child nodes x_i based on a policy π_{select}, some examples of

which are introduced in Section 3.1.1 and 3.2.2. The policy π_{select} balances exploration and exploitation, for example, by considering both the average reward of a node and the number of times it has been visited. The selection process continues until a leaf node or a node at the current boundary $\mathcal{B}$ of the search tree is reached.

Step 2: Expansion. Upon reaching such a node x_i, the algorithm adds one or more child nodes from children(x_i) to the search tree.

Step 3: Simulation.: Also known as the **random rollout** phase, this involves simulating a random rollout from the newly expanded node until a terminal state is reached. The simulation follows a default policy $\pi_{rollout}$ to traverse from the current node x_i to a leaf node x_l. Often the rollout is done by descending the tree uniformly at random (unbiased) or by following the guidance of a neural network (biased). The reward r_l of the leaf node x_l is observed.

Step 4: Back-Up. In this phase, the observed reward r_l is propagated back through the tree to the root node. Each node in the path from the expanded node x_i to the root x_0 updates its value based on r_l. This update typically involves incrementing a visit count and adjusting the value estimate of the node to reflect the new information.

See Algorithm 4 for pseudocode and Figure 2.5 for an example iteration of MCTS. Reviews of the many applications of MCTS can be found in (Kemmerling et al., 2024; Mańdziuk, 2018; Świechowski et al., 2023; Browne et al., 2012).

Algorithm 4: MCTS Search

```
MCTS(r, G, T):
    B = {x_0}
    for t ∈ 1, ..., T do
        x_i = x_0
        while x_i ∉ B do
            x_i ~ π_select(x_i) // Selection
        B = B \ {x_i}
        for x_c ∈ children(x_i) do
            B = B ∪ {x_c} // Expansion
            x_l ~ π_rollout(x_c) // Simulation
            while x_c ≠ x_0 do
                update(x_c, r_l) // Back-Up
                x_c = parent(x_c)
```

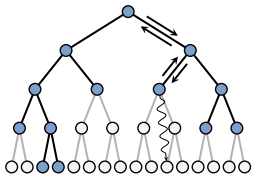


Figure 2.5.: An iteration during Monte Carlo Tree Search.

2.3. Bandits and Regret

An influential idea in combination with MCTS has been to treat the selection steps as **Bandit problems**:

Definition: k-armed Bandit

A k-armed Bandit has K arms defined by distributions $(p_k)_{1 \leq k \leq K}$ with bounded support. At each iteration t, $t = 1, ..., T$, one selects an arm $a_t \in \{1, ..., K\}$ and observes a reward $r_t \sim p_{a_t}$ sampled from the according distribution. Independence from previous rewards is assumed.

The k children nodes form k arms and the observed rewards from the random roll-outs are treated as observations sampled from the distributions of the arms. A possible concept to evaluate the performance of a Bandit or a tree search algorithm in a sequential decision-making task is by measuring its regret. Regret quantifies the difference between the actual rewards obtained by the algorithm and the maximum possible rewards that could have been achieved with perfect hindsight. A formal definition of regret is provided below.

Definition: Regret

Let T denote the total number of decisions in a sequential-decision-making problem with actions $\mathcal{A}$. Let $a_t \in \mathcal{A}$ denote the action chosen at step t. An action results in (possibly) stochastic rewards $r_t \sim p_{a_t}$. We define $\mu_a = \mathbb{E}_{r \sim p_a}[r]$. The optimal action a^* is the one that maximizes the expected reward $a^* = \arg\max_{a \in \mathcal{A}} \mu_a$ and the corresponding optimal expected reward is μ^*.
The **simple regret** at step t is defined as $\text{regret}_t = r_{a^*} - r_{a_t}$.
The **cumulative regret** is the sum of the simple regret over all time steps

$$\text{Regret}_T = \sum_{t=1}^{T} r_{a^*} - r_{a_t}. \tag{2.18}$$

and the **average regret** is given by Regret_T / T.

In the k-armed bandit settings described above, the goal is to identify a policy π that minimizes the cumulative regret Regret_T. In the next chapter, there will be two examples of such a policy. However, for a more comprehensive overview and recent developments in bandit algorithms, see the book from Lattimore and Szepesvári (2020).

CHAPTER 3

Principles and Techniques in Tree Search

In this chapter, we explore the fundamental principles and techniques that have shaped contemporary tree search algorithms, with a particular emphasis on *optimism* in Section 3.1, *softness* in Section 3.2 and *uncertainty* in Section 3.3. The principle of optimism is given prominence due to its foundational role in the widely adopted UCT algorithm, which dominated the field of tree search algorithms for many years. With an introduction to Optimistic optimization, Section 3.1 also contains relevant background for Chapter 4. Softness is discussed both as a technique employed in a line of more recently developed tree search algorithms, and due to its relevance to our setting for the k-DPP sampler in Chapter 7. Lastly, uncertainty is emphasized as a key concept motivating the development of our new tree search algorithms in Chapter 5 and 6. The above principles should not be understood as mutually exclusive. Their exposition is merely grouped by the corresponding branches of literature.

3.1. Optimism

In the context of optimization, the principle of optimism states that, when there is uncertainty about the true condition relevant to a decision, one should assume the most favorable of all conditions consistent with what has been observed so far and then select the next action accordingly.

3.1.1. Upper-Confidence-Bounds for Bandits

An elementary learning algorithm that adheres to the principle of optimism is the *Upper Confidence Bound* (UCB, Auer (2002)) algorithm applied to k-armed bandit problems. The UCB algorithm embodies the optimistic principle by maintaining upper confidence bounds on the expected values of each arm. Let $\hat{\mu}_{k,t}$ represent the empirical mean of the observed rewards from arm k up to time t, and $n_{k,t}$ denote the number of times arm k has been chosen by time t. After playing each arm at least once, at each iteration t, the next arm a_t is selected according to:

$$a_t = \arg\max_k \hat{\mu}_{k,t} + \sqrt{\frac{2\log t}{n_{k,t}}} \tag{3.1}$$

High previously observed rewards push for exploitation through the mean $\hat{\mu}_{k,t}$, and sufficient exploration is ensured by the second term decreasing relatively with the number of times an arm is played. The effectiveness of this strategy is driven by the concentration of the empirical mean around the true mean with an increasing number of plays. The bounds are derived using the Chernoff-Hoeffding concentration inequality for empirical means. As the search progresses (with increasing t), one makes sure that the probability that these bounds hold increases. This requires that every arm has to be chosen occasionally, but it can be shown that suboptimal arms are selected only logarithmically often (see Figure 3.1).

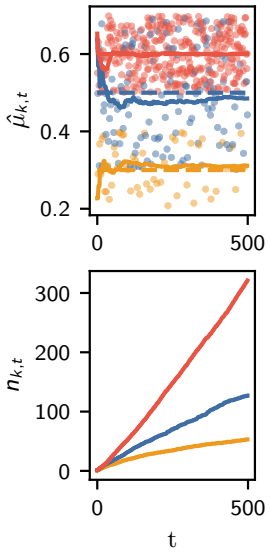


Figure 3.1.: UCB on a 3-armed Bandits problem. The dotted lines in the upper plot mark the true means of the arm's distributions, the dots the observed reward, and the solid lines show the empirical means. The lower plot shows how often the arms were selected over time.

If the number of arms k is very large relative to the available budget T, or even infinite, the UCB strategy becomes impractical, as it requires every arm to be played at least once. In such scenarios, additional assumptions about the reward structure become necessary. Rather than assuming that the rewards for each arm are independent, we must exploit shared structure between arms – for example, by assuming that arms are organized hierarchically, such as the leaves of a tree. This consideration leads to the UCT algorithm reviewed next, which leverages the hierarchical structure to implicitly generalize from observed to unobserved arms.

3.1.2. Upper-Confidence-Bounds for Tree Search

The UCT algorithm (Kocsis and Szepesvári, 2006a) performs an MCTS, where the selection step at each node is treated as a k-armed bandit problem. The selection policy aims to maximize an upper confidence bound,

$$\pi_{select}(x_i) = \underset{x_c \in \text{children}(x_i)}{\arg\max} \hat{\mu}_{c,t} + C\sqrt{\frac{\ln n_{i,t}}{n_{c,t}}} \tag{3.2}$$

where $\hat{\mu}_{c,t}$ is the average reward of node x_c, C is a constant determining the trade-off between exploration and exploitation, $n_{i,t}$ is the total number of times the parent node x_i was visited up until iteration t, and $n_{c,t}$ is the number of times child node x_c has been visited so far. These quantities are updated in each iteration during the back-up step.

While locally following the UCB strategy, globally UCT does not correctly apply the optimistic principle. This is due to the independence assumption between the rewards observed below a node being hurt. Choosing a particular descendant of a node affects the selection decision in future steps and thereby the distribution of future rewards at that node. Compared to the theoretical analysis of UCB, one now has to additionally take a potential drift over time in the mean of a bandit's arm into account. However, Kocsis and Szepesvári (2006a) could still show that if the rewards are scaled to lie in the unit interval, then the bias of the estimated expected reward $\mu_{0,t}$ at the root node is $\mathcal{O}\left(\frac{\log(t)}{t}\right)$. They further show that the failure probability at the root node converges to zero at a polynomial rate as the number of iterations goes to infinity.

Regularization in UCT

Grill et al. (2020) studied the regularization properties of UCT. They show that UCT can be phrased as approximate *regularized policy optimization* (see Box 3.1.1), where a penalty term is placed on the search policy.

Box 3.1.1.: Regularized Policy Optimization

In policy optimization, the policy π_θ is parameterized by a parameter vector θ (e.g., via a neural network) and is generally stochastic. In order to simplify the notation, we omit the dependency of the policy on node x_i and iteration t when clear from the context. Iterative updates to the current policy π_θ are performed to push it to select the child with the highest optimal value estimate $\hat{v}$, but under the constraint of a *penaltly* or *regularization term*, which stabilizes and possibly accelerates the convergence:

$$\pi'_\theta = \arg\max_{\pi \in \mathcal{S}} \mathbb{E}_{x_c \sim \pi}[\hat{v}_c] - \lambda \cdot \mathcal{R}(\pi, \pi_\theta). \tag{3.3}$$

Here, $\mathcal{S}$ is the k-dimensional simplex (where k is the number of actions a.k.a. child nodes) and $\mathcal{R} : \mathcal{S}^2 \to \mathbb{R}$ a convex regularization term with regularization strength λ.

For the analysis of UCT, consider the policy $\bar{\pi}$, defined as

$$\bar{\pi} = \arg\max_{\pi \in \mathcal{S}} \mathbb{E}_{x_c \sim \pi}[\hat{v}_c] - \lambda_N \mathcal{R}(\pi, \pi_{unif}), \tag{3.4}$$

where π_{unif} denotes the uniform distribution and the regularizer is set to the f-divergence $\mathcal{R}(\pi, \pi') = 2 - 2\sum_i \sqrt{\pi_i \cdot \pi'_i}$. The term λ_N is set to $\sqrt{\frac{\log N}{k+N}}$ with $N = \sum_c n_{t,c}$ being the number of times the node at that the current selection happens has been visited. It behaves as $\widetilde{\mathcal{O}}\left(\frac{1}{\sqrt{N}}\right)$ reducing the regularization strength over time. With $\hat{\pi}$ denoting the empirical visit distribution of UCT,

$$\hat{\pi}(x_c|x_i) = \frac{n_{c,t} + 1}{\sum_c n_{c,t} + k}. \tag{3.5}$$

Then one can show that the empirical visit distribution $\hat{\pi}$ tracks the policy $\bar{\pi}$ in the sense of

$$\pi_{select}(x_i) = \arg\max_{x_c \in \text{children}(x_i)} \frac{\partial}{\partial n_{t,c}} \left(\mathbb{E}_{x_c \sim \hat{\pi}}[\hat{v}_c] - \lambda_N \mathcal{R}(\hat{\pi}, \pi_{unif}) \right). \tag{3.6}$$

Note that a greedy update for maximizing the above objective in Equation 3.4 would be in the direction of the partial derivative in Equation 3.6. However, since one is restricted to *discrete* update via the visit counts, Equation 3.6 is as close as one gets to tracking $\bar{\pi}$ with $\hat{\pi}$. Grill et al. (2020) also derived regularization terms for the policy of other UCT variants. Dam et al. (2021) introduce further MCTS algorithms based on other convex regularization terms, like the Tsallis entropy. In Section 3.2, we will come across another example of a regularized MCTS algorithm, where the regularization term is the Shannon

entropy of the policy.

3.1.3. Optimistic Optimization

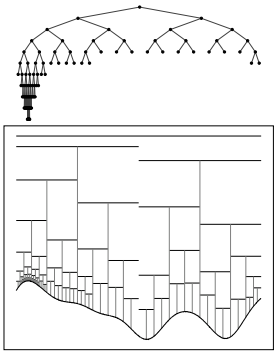


Figure 3.2.: Optimistic optimization applied to a function (here: a sample from a Gaussian Process). The upper bounds are shown as horizontal bars. The vertical lines point to the evaluated locations.

The optimism principle has also been applied to tree-based optimization in continuous domains $\mathcal{X}$ resulting in a family of methods named *Optimistic optimization*. Such algorithms are used to optimize functions $f : \mathcal{X} \to \mathbb{R}$ that are known to fulfill a local Lipschitz assumption with respect to a dissimilarity [1] $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$\forall x \in \mathcal{X} : |f(x^*) - f(x)| \leq \delta(x^*, x). \tag{3.7}$$

Throughout, we denote by f^* the maximum value of the function and by x^* a maximizer. The method revolves around a hierarchical partitioning of the search space $\mathcal{X}$ that can be described by an infinite binary tree. The root node corresponds to the entire search space $\mathcal{X}_{(0,1)} = \mathcal{X}$ and is named $x_{(0,1)}$. Consider a node $x_{(d,i)}$ at depth d. The left child $x_{(d+1,2i-1)}$ and right child $x_{(d+1,2i)}$ represent two subregions $\mathcal{X}_{(d+1,2i-1)}$ and $\mathcal{X}_{(d+1,2i)}$ such that $\mathcal{X}_{(d,i)} = \mathcal{X}_{(d+1,2i-1)} \cup \mathcal{X}_{(d+1,2i)}$, i.e. they cover the entire space. Intuitively, it makes sense to select the cells in such a way that all points in a cell are similar to each other and all similar points are in the same cell. Formally, this can be expressed by the following two conditions:

(a) $\forall\, x, y \in \mathcal{X}_{(d,i)} : \{\delta(x, y) < \Delta(d)\}$

(b) $\exists\, x \in \mathcal{X}_{(d,i)} : \{y \in \mathcal{X} : \delta(x, y) < C\Delta(d)\} \subset \mathcal{X}_{(d,i)}$,

where $\Delta(d)$ is a decreasing sequence of diameters and C is a global constant. During search, the tree is built incrementally by adding the two children of a selected node. When a new node $x_{(d,i)}$ is added, an observation is made at the center* $x_{(d,i)}$ of the region $\mathcal{X}_{(d,i)}$. In each round, the leaf with the highest upper bound $U_{(d,i)} = f(x_{(d,i)}) + \Delta(d)$ is selected for expansion. Since f is assumed to be locally Lipschitz with respect to δ, selecting nodes by $U_{(d,i)}$ is a valid upper bound strategy (assuming noiseless observations). The sequence of $\Delta(d)$ controls exploration. Therefore, the partitioning should be chosen in such a way that the $\Delta(d)$ can be as small as possible. The Optimistic optimization principle is summarized in Algorithm 5 and Figure 3.2. Using a binary heap, the priority queue for the leaf nodes can be realized in $\mathcal{O}(T \log T)$ if the budget T is known in advance.

The Optimistic optimization principle has its origins in the bandit setting. With *Hierarchical Optimistic Optimization* (HOO), Bubeck et al. (2011) apply it in the noisy setting (this requires an additional logarithmic exploration term compared to the exposition above), with *Deterministic Optimistic Optimization* (DOO), Munos (2011) apply it in the noiseless setting and Kleinberg et al. (2008) uses it with a slightly different Lipschitz assumption. The assumptions on the dissimilarity δ vary, e.g. some theoretical analyses require that it additionally be a semi-metric or metric. The popular *Dividing Rectangles* algorithm (DiRect, Jones and Martins (2021)) also belongs to the family partitioning based global optimization algorithms, but does not allow for encoding of prior knowledge on the

1: See Appendix A.1 for the formal requirements for a function to be a *dissimilarity*, *semi-metric*, *pseudo-metric* or *metric*.

* We overload the notation with both the node and the center of the cell $\mathcal{X}_{(d,i)}$ corresponding to the node being denoted with $x_{(d,i)}$.

Algorithm 5: Optimistic Optimization

```
1 OPTIMISTIC OPTIMIZATION(f, 𝒳, Δ):
    // Initialize priority queue with root
2   priority-queue.push([x_(0,1), ∞])
3   for t = 1 to T do
      // Select and remove node x_(d,i) from priority queue
4     x_(d,i) = priority-queue.pop()
      // Calculate child utilities
5     U_(d+1,2i−1) = f(x_(d+1,2i−1)) + Δ(d + 1)
6     U_(d+1,2i) = f(x_(d+1,2i)) + Δ(d + 1)
      // Add children based on their utilities to priority queue
7     priority-queue.push([x_(d+1,2i−1), U_(d+1,2i−1)])
8     priority-queue.push([x_(d+1,2i), U_(d+1,2i)])
```

smoothness of the function. It recursively partitions the search space into hyperrectangles and selects potentially optimal ones based on both function value and size, but the size of the hyperrectangle is typically measured with the Euclidean norm. OO has also been combined with UCT in order to apply UCT in continuous search spaces via the *Hierarchical Optimistic Optimization Algorithm applied to Trees* (HOOT, Mansley et al. (2011)). Instead of discretizing the search space and applying the UCB criterion for node selection at each decision point in UCT, HOOT uses HOO for node selection. Mao et al. (2020) build on this approach by replacing the additional logarithmic exploration term with a polynomial exploration term, which leads to improved convergence when addressing the non-stationary node selection problem in UCT.

The deterministic and noiseless setting in the work from Munos (2011) is closest to our setting in Chapter 4. Regarding the theoretical properties of Optimistic optimization under these specifications, Munos (2011) gives guarantees in terms of the near-optimality dimension[2] m: Let $h(t)$ be the smallest integer h such that $C \sum_{l=0}^{h} \Delta(l) - m \geq t$ and $x(t)$ be the location of the t-th function evaluation. Then the loss of DOO, defined as $r_t = \sup_{x \in \mathcal{X}} f(x) - f(x(t))$, is bounded as $r_t \leq \Delta(h(t))$. In addition, one can show that the loss decreases polynomially if $m > 0$ and exponentially if $m = 0$.
For the special case, where f is a sample path from Brownian motion, Grill et al. (2018) provide a more explicit statement on the sample complexity of deterministic Optimistic optimization. Since the sample path is a random quantity, the near-optimality dimension, too, is a random number. However, one can show that for any ϵ, there exists some particular metric δ_ϵ such that the Brownian motion is δ_ϵ-Lipschitz with probability $1 - \epsilon$, and there exists a constant $C(\epsilon) = \mathcal{O}(\log(1/\epsilon))$ such that $m = 0$.
Munos (2011) introduces a variant of the principle, *Simultaneous Optimistic Optimization* (SOO), for situations in which the dissimilarity function is unknown. Valko et al. (2013) extend this method to the noisy setting.

2: For any $\epsilon > 0$, let $X_\epsilon = \{x \in \mathcal{X} \mid f(x) \geq f^* - \epsilon\}$. and fix $\nu > 0$. The **near-optimality dimension** $m \geq 0$ is the smallest value such that there exists a constant $C > 0$ satisfying $\forall \epsilon > 0$:

$$\mathcal{N}(X_\epsilon, \nu\epsilon) \leq C\epsilon^{-m},$$

where $\mathcal{N}(X_\epsilon, \nu\epsilon)$ denotes the maximal number of disjoint balls with respect to semi-metric δ of radius $\nu\epsilon$ centered in X_ϵ.

3.2. Softness

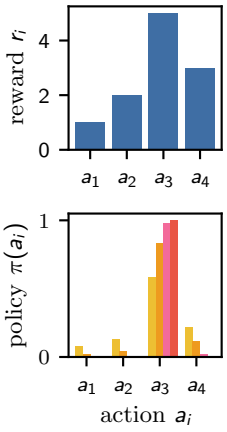


Figure 3.3.: The softargmax of a reward function. Top: Rewards for $k = 4$ arms. **Bottom:** The softargmax distribution over the arms for four different values of the parameter β:
• $\beta = 2$ (—) • $\beta = 1$ (—)
• $\beta = \frac{1}{2}$ (—) • $\beta = \frac{1}{4}$ (—)
Parameter β controls the tightness of the approximation (see Equation 3.8).

Next to optimism, another concept from the optimization literature that has been leveraged in the design of tree search algorithms is *softness* – the idea of searching for a *soft maximum* rather than the exact or "hard" maximum. Intuitively, the maximum operator selects the single best option and completely ignores all others. In contrast, the softmax operator assigns higher probabilities to better options while still giving non-zero weight to suboptimal ones, thus providing a smooth approximation to the maximum (see Figure 3.3).

3.2.1. Softness as Regularization

From the viewpoint of regularization, softness in the final policy can be achieved by adding an entropy regularization term on the policy. The resulting framework is known as Maximum Entropy reinforcement learning and goes back to (Ziebart, 2010; Toussaint, 2009; Todorov, 2008). A comprehensive overview is given in the tutorial from Levine (2018). The special case of tree search is discussed in *Maximum Entropy Tree Search* (MENTS, Xiao et al. (2019)). Given K actions $a_1, ..., a_K$ and the corresponding K-dimensional reward vector $\mathbf{r} = (r_1, ..., r_K) \in \mathbb{R}^K$, the idea is to solve

$$\pi^* = \arg\max_{\pi} \big(\mathbb{E}_{a_k \sim \pi}[r_k] + \beta \mathcal{H}[\pi] \big) \tag{3.8}$$

instead of only maximizing the expected reward $\mathbb{E}_{a_k \sim \pi}[r_k]$ as in the standard reinforcement learning and tree search settings. The solution to Equation 3.8 has a closed-form solution (Haarnoja et al., 2017; Nachum et al., 2017; Xiao et al., 2019), namely the **soft argmax** $\mathbf{f}_\beta(\mathbf{r})$ of $\mathbf{r}$:

$$\mathbf{f}_\beta(\mathbf{r}) = \exp\left((\mathbf{r} - \mathcal{F}_\beta(\mathbf{r}))/\beta\right) \quad \text{with} \quad \mathcal{F}_\beta(\mathbf{r}) = \beta \log \sum_{k=1}^{K} \exp\left(\mathbf{r}_k/\beta\right). \tag{3.9}$$

The term $\mathcal{F}_\beta(\mathbf{r})$ is the corresponding **softmax** and is an upper bound on the maximum value, and the gap closes for $\beta \to 0$. The **soft argmax** $\mathbf{f}_\beta(\mathbf{r})$ maps $\mathbf{r}$ to a **Boltzmann policy** π^*, the solution of the above objective in Equation 3.8. In the following two Subsections 3.2.2 and 3.2.3, we review, for the sake of completeness, how MENTS solves the Maximum Entropy regularized objective via a MCTS. However, in Chapter 7, where we work with such an objective function, the size of the underlying tree renders an MCTS-based approach impractical. Readers primarily interested in Chapter 7 may, therefore, choose to skip the next two subsections.

3.2.2. Stochastic SoftMax Bandits

The first step toward a tree search for the Maximum Entropy regularized objective is again a k-armed bandit setting but with the goal of learning a policy that maximizes Equation 3.8 instead of only the expected cumulative reward as introduced in the definition of a k-armed

bandit in Section 2.3. Similar to UCB, one tracks the empirical means $\hat{\mathbf{r}}_t = \left(\hat{r}_{1,t}, ..., \hat{r}_{k,t}\right)$ of the observed rewards from the k arms up to time t. In iteration t, one selects the next action according to a stochastic policy defined as

$$\pi_t(a) = (1 - \lambda_t)\left(\mathbf{f}_\beta(\hat{\mathbf{r}}_t)\right)_a + \lambda_t \frac{1}{K}, \tag{3.10}$$

where $\lambda_t = \epsilon K / \log(t+1)$ is a decay rate for the exploration and $\epsilon > 0$ an exploration hyperparameter. Assuming σ^2-subgaussian[3] reward distributions for the arms, Xiao et al. (2019) show that this is the optimal sampling strategy for the Stochastic Softmax Bandit problem.

Similar to how UCB is driven by the concentration of the empirical mean around the actual mean, the above strategy relies on a concentration result for the concentration of the counts around the counts under the optimal policy and on the implied concentration of the empirical softargmax around the actual softargmax values. Again one has to make sure, that as the search progresses (with increasing t) the probability that these bounds hold increases. For this to happen, it is necessary that each arm has to be pulled at least $\mathcal{O}(\log t)$ times in t rounds.

3: A random variable X is called σ^2-*sub-Gaussian* if there exists a positive constant σ such that for all $t \geq 0$,

$$P(|X| \geq t) \leq 2\exp\left(-\frac{t^2}{\sigma^2}\right).$$

3.2.3. Maximum Entropy Tree Search

Just as UCT models the selection step at each node as a k-armed bandit and applies the UCB criterion, MENTS models the decision at each node as a stochastic Softmax Bandit and uses the above softmax-based selection policy. At node x_i, the next child $x_c \in \text{children}(x_i)$ is selected according to the policy

$$\pi(x_c \mid x_i) = (1 - \lambda_i)\left(\mathbf{f}_\beta(\hat{\mathbf{v}}^{\text{soft}}_{x_i})\right)_{x_c} + \lambda_{x_i}\frac{1}{K}, \tag{3.11}$$

where $\hat{\mathbf{v}}^{\text{soft}}_{x_i} = \left(\hat{v}^{\text{soft}}_{x_i}(x_{c_1}), \ldots, \hat{v}^{\text{soft}}_{x_i}(x_{c_K})\right)$ is the vector of soft optimal value estimates for the children of node x_i and the coefficient

$$\lambda_{x_i} = \frac{\epsilon K}{\log\left(\sum_{x_c \in \text{children}(x_i)} n_{x_c} + 1\right)} \tag{3.12}$$

controls the exploration–exploitation trade-off, where $\epsilon > 0$ is a hyperparameter and n_{x_c} denotes the number of times child x_c has been visited from node x_i.

The soft optimal value $v^{\text{soft}}_{x_i}(x_c)$ at a node x_c is defined recursively as

$$v^{\text{soft}}_{x_i}(x_c) = \begin{cases} r_c + R & \text{if } x_c \text{ is at boundary,} \\ r_c + \mathcal{F}_\beta\left(\mathbf{v}^{\text{soft}}_{x_c}\right) & \text{otherwise,} \end{cases} \tag{3.13}$$

where r_c is the reward obtained at node x_c and R is a heuristic estimate for the remaining reward beyond the current search depth. The soft optimal value estimates are initialized to zero and updated in each iteration using Equation 3.13, enabling smoothed value propagation through the tree.

3.2.4. Relations to other Frameworks

In what follows, we discuss how the Maximum Entropy regularized objective relates to the standard objective and explore it from the perspective of statistical physics. We review how optimizing such an objective can be cast as probabilistic inference and in which way contemporary search algorithms derived under the softness principle still fall short of being fully uncertainty-aware.

Relation to Standard Reinforcement Learning Objective

The idea behind Maximum Entropy reinforcement learning goes back to (Kalman, 1960), who showed that Kalman filtering can be applied to solve control problems with linear dynamics and quadratic costs in the case of MDPs with continuous state and action spaces [4]. In such problems, the maximum entropy solution results in a linear-Gaussian policy, where the mean exactly matches the optimal deterministic policy of the standard setting. This relationship is often referred to as the Kalman duality (Todorov, 2008). Unfortunately, this duality does not generalize beyond the linear-quadratic setting. Despite not optimizing the original objective, the derived algorithms are said to perform well empirically in reinforcement learning domains even under the standard reinforcement learning objective (Tarbouriech et al., 2024). There is even extensive evidence that humans and animals choose actions with probability proportional to how much reward that action brings, for an overview see (Eysenbach and Levine, 2019) or (Vulkan, 2000). Eysenbach and Levine (2019) point out that the stochasticity of the learned policies leads to two important benefits: Eventually, every path will be tried, and one does not always choose the same path. Intuitively, this is particularly useful in settings where one cannot observe the true reward functions, as well as in adversarial settings, as stochastic policies are harder to exploit for an adversary. The intuition is made formal in an analysis by Eysenbach and Levine (2019) who show that Maximum Entropy reinforcement learning optimally solves MDPs with variability in the reward function as well as a certain reward-robust control objectives in an adversarial two-player zero-sum game. More recently, there has also been effort to modify MENTS to converge to the solution of the standard objective again: With *Boltzmann Tree Search* (BTS), Painter et al. (2024) suggest to keep the Boltzmann selection policy from MENTS but to use backups that optimize for the standard objective. They also introduce *Decaying ENtropy Tree Search* (DENTS) which can interpolate between MENTS and BTS.

4: For a formal exposition of this special case, see Appendix A.3

Relation to Statistical Physics

Rahme and Adams (2019) offer a perspective from statistical physics on the Maximum Entropy reinforcement learning framework. For the special case of tree search, the relation to statistical physics is as follows. Necessary assumptions for their interpretations are that the number of child nodes is uniformly bounded over all nodes by b and that the rewards are bounded from above by R_{max}. By subtracting R_{max} from all rewards, one can assume

without loss of generality that all rewards are non-positive. However, rewards at the leaf nodes are allowed to be positive. For a path $\omega_{i\to l} = (x_i, ..., x_l)$ from a node x_i to a leaf x_l, one can define an energy function $E(\omega_{i\to l})$ according to the negated cumulative reward along the path

$$E(\omega_{i\to l}) = -\sum_{x_j \in \omega_{i\to l}} r_j = -r_{i\to l}, \tag{3.14}$$

as well as a partition function for node x_i

$$Z(x_i, \beta) = \sum_{\omega \in \Omega(x_i)} e^{\beta E(\omega) + \mu|\omega|}, \tag{3.15}$$

where $\Omega(x_i)$ is the set of paths starting at node x_i and leading to a leaf node, β is a inverse temperature parameter and $\mu \leq 0$ a penalty parameter for the sequence length. The partition function at node x_i is related to the partition functions of its children through a Bellman-like recursion:

$$Z(x_i, \beta) = \sum_{x_c \in \text{children}(x_i)} e^{\beta r_c + \mu} + Z(x_c, \beta). \tag{3.16}$$

The Boltzmann distribution induced by the energy function is given by

$$P(\omega \mid \beta, \mu, x_i) = \frac{\mathbf{1}_{\Omega(x_i)}(\omega)}{Z(x_i, \beta)} \exp\left(-\beta E(\omega) + \mu|\omega|\right) \tag{3.17}$$

and can be used to calculate the average energy for a node x_i either as an explicit expectation or as the partial derivative of the log partition function with respect to the inverse temperature β:

$$\langle E \rangle = \sum_{\omega \in \Omega(x_i)} \frac{1}{Z(x_i, \beta)} \exp\left(-\beta E(\omega) + \mu|\omega| E(\omega)\right) = -\frac{\delta}{\delta \beta} \log Z(x_i, \beta). \tag{3.18}$$

The negative of the average energy gives the optimal value v_i of node x_i under temperature β.

The length penalty parameter μ in this framework can be interpreted as a chemical potential and plays a crucial role in the approximation guarantee for our greedy k-DPP sampler in Chapter 7. For further discussion of this parameter, see Box 3.2.1.

Box 3.2.1.: Length Penalty and Discount

Rahme and Adams (2019) point out that the length penalty μ in the above soft reinforcement learning formulation plays a similar role to the discount parameter γ used in standard reinforcement learning. Assuming unconstrained sequence lengths, the parameter μ ensures that infinite series converge. Moreover, the parameters μ and the discount parameter are necessary to ensure that the Bellman operators are contractions in their respective frameworks[5]. However, they also point out an important difference

5: Appendix A.5 contains a more formal version of this statement.

between the two parameters: When using the discount rate γ, the order of rewards in a sequence does matter and can thereby influence the learned policy. But this is not the case for the length penalty μ.

More recent work by Rathnam et al. (2024) sheds light on the regularizing role of the discount parameter γ in reinforcement learning *under the standard objective*. They focus on **discount regularization** – a setting in which the agent plans using a shorter effective horizon than the true horizon of the environment, by deliberately using a smaller discount factor γ. While their results rely on the transitions being stochastic and the horizon of the MDP being infinite (i.e. not the tree search setting), they nevertheless provide context for our theoretical analysis in Chapter 7 and are therefore summarized in Appendix A.2 in more detail for the interested reader. In short: reducing γ is equivalent to imposing a Dirichlet prior over the transition dynamics, where the concentration strength increases with decreasing γ. The framework also admits an interpretation as an implicit regularization of the Q-values: reducing the discount yields the same optimal policy as a modified Q-learning update in which Q-values are updated via a weighted average between the learned estimate and a fixed prior. In this form, discounting acts as a *shrinkage penalty* on the Q-function, encouraging smoother value estimates.

Relation to Probabilistic Inference

Probabilistic inference (see Box 3.2.2) formalizes reasoning under uncertainty by treating unknown quantities as random variables governed by probability distributions. Since probabilistic inference techniques play a key role in the algorithms presented in later chapters, we now examine how Maximum Entropy reinforcement learning can be formulated as a probabilistic inference problem. Our exposition follows Levine (2018), adapted to the specific case of tree search in reinforcement learning.

For each node x_i and its child $x_c \in \text{children}(x_i)$, introduce a binary optimality variable o_i indicating whether the selection of x_c at x_i is optimal. The conditional probability of optimality is chosen as:

$$p(o_i = 1 \mid x_i, x_c) = \exp(r_c), \tag{3.19}$$

where r_c is the immediate reward received upon visiting node x_c. Note that this implies $r_c \leq 0$ as one has to ensure proper normalization. Further, assume a uniform prior over child selection, $p(x_c \mid x_i) = \frac{1}{|\text{children}(x_i)|} = \frac{1}{K}$. Conditioned on the optimality $\mathbf{o}_{0:T-1} = (o_0, ...o_{T-1})$ of all choices along a path $\omega = (x_0, x_1, \dots, x_T)$, the posterior

distribution over paths is proportional to the exponentiated cumulative reward:

$$p(\omega \mid \mathbf{o}_{0:T-1} = \mathbf{1}) \propto p(\omega, \mathbf{o}_{0:T-1} = \mathbf{1}) \tag{3.20}$$

$$= p(x_0) \prod_{i=0}^{T-1} p(o_i = 1 \mid x_i, x_{i+1}) \cdot p(x_{i+1} \mid x_i) \tag{3.21}$$

$$\propto \prod_{i=0}^{T-1} \exp(r_{i+1}) = \exp\left(\sum_{i=0}^{T-1} r_{i+1}\right). \tag{3.22}$$

The optimal policy can be derived via backward message passing[6] starting from the leaves. Define messages the message

6: For a summary of how message passing works, see Appendix A.7.

$$m_i(x_i) = p(\mathbf{o}_{i:T-1} = \mathbf{1} \mid x_i) \quad \text{and} \quad m_i(x_c, x_i) = p(\mathbf{o}_{i:T-1} = \mathbf{1} \mid x_i, x_c). \tag{3.23}$$

These satisfy the recursive relations

$$m_i(x_i) = \sum_{x_c \in \text{children}(x_i)} m_i(x_c, x_i)\, p(x_c \mid x_i), \tag{3.24}$$

$$m_i(x_c, x_i) = p(o_i = 1 \mid x_i, x_c)\, m_{i+1}(x_c), \tag{3.25}$$

with the base case $m_T(x_T) = 1$ at terminal nodes. The resulting optimal policy is the posterior over child selection given optimality:

$$\pi^*(x_c \mid x_i) = p(x_c \mid x_i, \mathbf{o}_{i:T-1} = \mathbf{1}) = \frac{m_i(x_c, x_i)}{m_i(x_i)}. \tag{3.26}$$

Taking the logarithm of $m_i(x_i)$, one obtains the soft optimal value $v^{soft}(x_i) = \log m_i(x_i)$ of node x_i – up to a constant shift of $\log K$ from the uniform prior, which does not affect the resulting policy:

$$v^{\text{soft}}(x_i) = \log\left(\sum_{x_c \in \text{children}(x_i)} \exp(r_c + v^{\text{soft}}(x_c))\right) - \log K. \tag{3.27}$$

$$\tag{3.28}$$

Box 3.2.2.: Probabilistic Inference

At the heart of probabilistic inference is Bayes' theorem, which describes how to update one's beliefs in light of new information. It arises from the following two fundamental rules of probability theory:

Sum rule. For a variable X that can take values x in a set of possible outcomes, the probability of an event A (where A is related to X in some way) can be computed by adding all the results in A:

$$p(A) = \sum_{x \in A} p(X = x)$$

The sum rule is often used for *marginalization*, a process to obtain the probability of a subset of variables by adding the other variables.

Product rule. The product rule provides a way to express the joint probability of two events, A and B, in terms of **conditional probabilities**. For two variables X and Y, the joint probability $p(X, Y)$ can be factored as

$$p(X, Y) = p(Y \mid X)p(X).$$

Two variables are said to be **independent** if $p(X, Y) = p(X) \cdot p(Y)$.

Together, these two rules lead to **Bayes theorem**, which calculates the **posterior probability** $p(\theta \mid D)$ of a parameter θ given observed data D:

$$p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{p(D)}$$

Here, $p(D \mid \theta)$ is the **probability** of observing the data given the parameter θ, and $p(\theta)$ is the **prior** belief about θ. The denominator $p(D)$ serves as a normalization constant, ensuring that the posterior distribution sums to one, and is called the **evidence**.

A case for epistemic uncertainty

O'Donoghue et al. (2020) and Tarbouriech et al. (2024) criticize the softness framework for ignoring the *epistemic uncertainty* over the optimal variables and conclude that this shortcoming leads to invalid and arbitrarily poor posteriors for optimal actions under the standard reinforcement learning objective. O'Donoghue et al. (2020) demonstrate this with the help of a toy bandit problem:

Problem 1 (O'Donoghue et al., 2020)

Fix $K \in \mathbb{N} \geq 3, 1 >> \epsilon > 0$ and define two reward functions f_1 and f_2 as follows:

$$f_1(a_1) = 1,\ f_1(a_2) = +2,\ f_1(a_i) = 1 - \epsilon \text{ for } i = 3, ..., K$$
$$f_2(a_1) = 1,\ f_2(a_2) = -2,\ f_2(a_i) = 1 - \epsilon \text{ for } i = 3, ..., K$$

The two reward functions differ only for action a_2, so there is only one unknown action in this problem.

For $N > 3$ iterations the optimal strategy for Problem 1 is to first choose x_2 and observe r_2. If $r_2 = +2$, then one knows that the reward function is f_1 and can keep picking a_2 for the remaining iterations with total regret $R = 0$. If $r_2 = -2$, one learns that the reward function is f_2 and from then on one picks action a_1 with a total regret $R = 3$. For K large and without prior information, a Boltzmann policy as the one above is very unlikely to select action a_2. Even for an informed prior with equal probability for both reward functions,

one would have $\mathbb{E}[r_2] = 0$ and it would still be unlikely that a_2 would be sampled under a Boltzmann policy as $\mathbb{E}[r_1] = 1$ and $\mathbb{E}[r_i] = 1 - \epsilon$ for $i \geq 3$. There is a proposal from O'Donoghue et al. (2020), which is discussed further in Tarbouriech et al. (2024), for how to incorporate epistemic uncertainty into the softness framework. However, they focus on uncertainty over the transitions, as well as uncertainty over the rewards in the form of independent priors with additive σ-sub-Gaussian noise. For the tree search setting, where the transitions are known, the first source of uncertainty is irrelevant and the way the uncertainty over the rewards is modeled and leveraged does more closely resemble an upper-bound based approach like UCT rather than a full Bayesian treatment with explicit distributional information. Their approach is therefore less suitable as the basis for the tree search methods in this thesis, but the interested reader finds a summary in Appendix A.4 for comparison.

3.3. Uncertainty

Motivated by the need for uncertainty awareness, we now turn to the Bayesian optimization framework, which uses Bayesian inference to explicitly model uncertainty over the unobserved parts of the objective function – offering a more principled perspective on the exploration–exploitation trade-off. This chapter assumes familiarity with Gaussian processes, roughly on the level of Chapters 2 to 4 in the book from Hennig et al. (2022) or Chapters 2 and 3 in the one from Garnett (2023). Some basic definitions are also included in Appendix A.8 for reference.

3.3.1. Bayesian Optimization

We consider the problem of maximizing a black-box function $f : \mathcal{X} \to \mathbb{R}$. Bayesian optimization addresses this problem by modeling f with a probabilistic surrogate, such as a Gaussian process, and using an *acquisition function* to decide where to query next. This acquisition function encodes a trade-off between exploring uncertain regions and exploiting areas likely to yield high function values. By iteratively optimizing the acquisition function, which is typically a much cheaper subproblem, Bayesian optimization transforms the global optimization task into a sequence of informed queries. In continuous domains, this often involves gradient-based optimization, leveraging the smoothness of both the surrogate and the acquisition function.

In our setup, we assume that f is a sample from a Gaussian Process $\mathcal{GP}(\mu, k)$ with mean function μ and kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. For simplicity, we assume the Gaussian process is centered, i.e., $\mu = 0$. Observations are noisy: we access $y_i \sim f(x_i) + \mathcal{N}(0, \lambda)$. The noiseless case corresponds to taking $\lambda \to 0$.

After each new observation, the posterior over the function is updated as follows:

$$\mu_n(x) = \mathbf{K}_n(x)^T(\mathbf{K}_n + \lambda I)^{-1}\mathbf{y}_n, \tag{3.29}$$

$$k_n(x, x) = k(x, x) - \mathbf{K}_n(x)^T(\mathbf{K}_n + \lambda I)^{-1}\mathbf{K}_n(x), \tag{3.30}$$

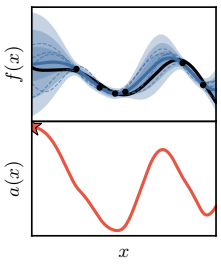


Figure 3.4.: GP-UCB. Top: Objective function (—) with six noisy function evaluations (●). The posterior mean and variance (═) and posterior samples (┄) of the Gaussian process are also shown. **Bottom:** GP-UCB acquisition function (—). The star (★) marks the next evaluation point.

where $\mathbf{y}_n = [y_1, ..., y_n]^T$ is the vector of observed values, $\mathbf{K}_n(x) = [k(x_1, x), ..., k(x_n, x)]^T$, and $\mathbf{K}_n$ is the kernel matrix with $(\mathbf{K}_n)_{ij} = k(x_i, x_j)$.

The posterior mean and variance are used to define an acquisition function $a_n(x)$, which is optimized to select the next evaluation point. A widely used acquisition function, which is especially relevant for the following chapters, is the Upper Confidence Bound (UCB):

$$a_n(x) = \mu_n(x) + \beta_n^{1/2}\sqrt{k_n(x, x)}, \tag{3.31}$$

where $\beta_n^{1/2}$ is an appropriately chosen constant that controls the exploration–exploitation trade-off. The corresponding method is known as GP-UCB (Srinivas et al., 2009), illustrated in Figure 3.4 on page 33. Alternative acquisition functions include expected improvement (Jones et al., 1998), probability of improvement (Lizotte, 2008), and information-theoretic approaches such as maximizing information gain about the function's maximum (Wang and Jegelka, 2017) or its location (Hennig and Schuler, 2012). For a comprehensive overview, see Frazier (2018); Garnett (2023).

Many acquisition functions can be understood as maximizing an *expected utility*, where the utility function encodes preferences over possible outcomes (e.g., improvement, information gain). Formally, the acquisition function $a_n(x)$ at iteration n can be written as an expectation,

$$a_n(x) = \mathbb{E}_{f\sim p_n(f)}\big[u(x, f)\big], \tag{3.32}$$

where $p_n(f)$ is the current posterior distribution over functions and $u(x, f)$ is a utility function evaluating the benefit of sampling at x.

The role Gaussian processes play for regularizing regression problems and thereby indirectly also for regularizing Bayesian optimization and tree search problems can be made more explicit by the following theorem:

Theorem 3.3.1 (Kernel Ridge Regression as Posterior Mean) Consider the Bayesian regression model

$$\begin{aligned} f &\sim \mathcal{GP}(0, k), \\ y_i \mid f &\sim \mathcal{N}(f(x_i), \lambda), \quad i = 1, \dots, n, \end{aligned}$$

where $\mathbf{y} = [y_1, \dots, y_n]^\top$ are noisy observations at input locations $\mathbf{x} = [x_1, \dots, x_n]$.

Then, the posterior mean of the Gaussian process at a test point $x \in \mathcal{X}$ is given by

$$\mu_n(x) = \mathbf{K}_n(x)^\top(\mathbf{K}_n + \lambda I)^{-1}\mathbf{y},$$

where $\mathbf{K}_n(x) = [k(x_1, x), \dots, k(x_n, x)]^\top$ and $\mathbf{K}_n \in \mathbb{R}^{n\times n}$ is the Gram matrix with $(\mathbf{K}_n)_{ij} = k(x_i, x_j)$.

Moreover, $\mu_n(x) \in \mathcal{H}_k$, the Reproducing Kernel Hilbert Space (RKHS) associated with the kernel k, and it is the unique minimizer of the regularized empirical risk

$$f^* = \arg\min_{f \in \mathcal{H}_k} \left\{ \frac{1}{\lambda} \sum_{i=1}^{n} (f(x_i) - y_i)^2 + \|f\|^2_{\mathcal{H}_k} \right\}.$$

A proof of this theorem, as well as many other useful properties of Gaussian process regression, can be found in (Kanagawa et al., 2018).

Notice that here the regularization term is applied to the reward function f. This contrasts with the approaches discussed in the previous sections, where the regularization term was instead placed on the policy π. Introducing a prior on the function f effectively induces a prior on the argmax of that function and, consequently, on the optimal policy π that selects this argmax. However, as far as we are aware, this relationship is complex and usually analytically intractable. There is a Bayesian optimization method, Entropy Search (Hennig and Schuler, 2012), where this relationship is made explicit by deriving a belief over the argmax from the belief over the reward function, but doing this requires advanced approximate inference techniques or expensive sampling. A potential advantage of regularizing the reward function f instead of directly regularizing the policy π is that it might be easier to phrase prior knowledge about f by incorporating semantic similarities in the search space via a kernel function than to phrase prior knowledge about π by choosing between different convex regularization terms such as the ones UCT and MENTS rely on.

Another motivation for developing Bayesian optimization methods – and ultimately Bayesian tree search – is the ability to hierarchically infer the hyperparameters θ of the prior. In the Gaussian process setting, this is achieved by maximizing the marginal likelihood of the observations $\mathbf{y}_n$, given inputs $\mathbf{x}_n = [x_1, \dots, x_n]$, with respect to θ. The marginal likelihood under a Gaussian process prior $\mathcal{GP}(0, k^\theta)$ with Gaussian noise variance λ is given by:

$$\hat{\theta} = \arg\max_{\theta} \ \log p(\mathbf{y}_n \mid \mathbf{x}_n, \theta) \tag{3.33}$$

$$= \arg\min_{\theta} \ \frac{1}{2} \mathbf{y}_n^\top \left(\mathbf{K}_n^\theta + \lambda I \right)^{-1} \mathbf{y}_n + \frac{1}{2} \log\det \left(\mathbf{K}_n^\theta + \lambda I \right) + \frac{n}{2} \log 2\pi, \tag{3.34}$$

where $\mathbf{K}_n^\theta$ is the kernel Gram matrix with entries $\left(\mathbf{K}_n^\theta\right)_{ij} = k^\theta(x_i, x_j)$. From this hierarchical perspective, the log-determinant term can be interpreted as a regularizer that penalizes overly complex models. Together with the constant term $\frac{n}{2} \log 2\pi$, it corresponds to the negative entropy of the Gaussian predictive distribution.

A recent survey and taxonomy of Bayesian optimization for discrete domains is provided by (González-Duque et al., 2024). Though tree search is a notable subclass of discrete optimization problems, it receives relatively little attention in this branch of literature. Nevertheless, there are existing efforts to bridge ideas from Bayesian optimization to tree search algorithms (Tesauro et al., 2012; Baum and Smith, 1997; Stern et al., 2007; Hennig et al., 2010). The approach most closely aligned with the ideas in this work is that of Hennig et al. (2010), which we will review in the remainder of this chapter.

3.3.2. Probabilistic Tree Search

As described above, the core principle of Bayesian optimization is to reduce an expensive black-box optimization problem into a sequence of cheaper subproblems by iteratively optimizing an acquisition function. However, this becomes challenging in discrete, tree-structured domains, where the acquisition function is non-differentiable. As a result, gradient-based optimization methods are no longer applicable. On the other hand, the hierarchical structure of the domain offers an opportunity: instead of optimizing over the entire space at once, one can define and optimize acquisition functions locally at each level of the tree. This strategy can scale linearly with the depth of the tree, offering significant efficiency gains. Since the acquisition function in Bayesian optimization requires a probabilistic model for the objective function, the main challenge is to efficiently derive such a model for the intermediate levels of the tree, where the optimal value function takes on the role of the objective function in standard Bayesian optimization.

Generative Scores and Optimal Values

A key idea in the approach from Hennig et al. (2010) for making inference over the optimal value of a node *tractable* is to *decompose* this value into two components: a *generative score* that reflects the utility of the node under a *random policy*, and an *increment* Δ_i that quantifies the expected gain from selecting subsequent actions *optimally*, rather than at random. Formally, the proposal is to model the *optimal value* v_i of a node x_i as the sum of its generative score g_i and an increment Δ_i:

$$v_i = g_i + \Delta_i \tag{3.35}$$

For a leaf node x_l the generative scores correspond to the optimal value, thus $\Delta_l = 0$, because if no decisions are left there is of course also no difference between making random and optimal decisions. At internal nodes, the Δ_i terms obey the following recurrence relation:

$$\Delta_i = \max_{j \in \text{children}(i)} \{\Delta_j + (g_j - g_i)\}. \tag{3.36}$$

Smoothness

The probabilistic model for the generative scores imposes a *smoothness* assumption: nodes that are close in graph distance tend to have similar values. This is captured by modeling the generative score g_j of a child node x_j as a Brownian motion step from its parent x_i:

$$p(g_j \mid g_i) = \mathcal{N}(g_j; g_i, c) \tag{3.37}$$

Each score increment is Gaussian (with scale c) and independent given its parent, inducing a Wiener process prior across the tree. This correlation structure extends to the reward function at the leaf nodes: Nearby leafs have strongly correlated values, while distant ones

differ more significantly. While the joint distribution is difficult to visualize due to the high dimensionality of the tree, Figure 3.5 shows the marginal priors over the generative scores and the implied distribution of the optimal values.

Efficient Inference on the Increments

Having chosen a Gaussian prior for the generative scores, one can derive a Gaussian approximation for the Δ terms. For leaf nodes x_l one has $\Delta_l = 0$. For the Δ_i of nodes further up, one obtains recursively (cf. Equation 3.36):

$$\Delta_i = \max_{j \in \text{children}(i)} \{\Delta_j + \xi_j\}, \tag{3.38}$$

where $\xi_j \sim \mathcal{N}(0, c)$ is the Brownian step from parent to child node. The maximum involved in this equation is approximated with a Gaussian via the approximation scheme in Box 3.3.1.

Assuming a constant branching factor, at least per level, the computation of the beliefs over the Δ can be simplified: Under a constant branching factor, the approximation for the Δ is the same for all nodes in the same level, i.e. it is sufficient to calculate the approximation for a single Δ per level and use copies for the sibling nodes for the subsequent step. The calculation effort for the Δ's is therefore not exponential, as it seems in their recursive definition, but linear in the depth of the tree[7].

7: The probabilistic model can be easily extended to Min-Max-Trees by replacing the maximum operators in Equation 3.38 by minimum operators for the Min levels of the Min-Max-Tree. In fact, it was originally defined for Min-Max-Trees in (Hennig et al., 2010).

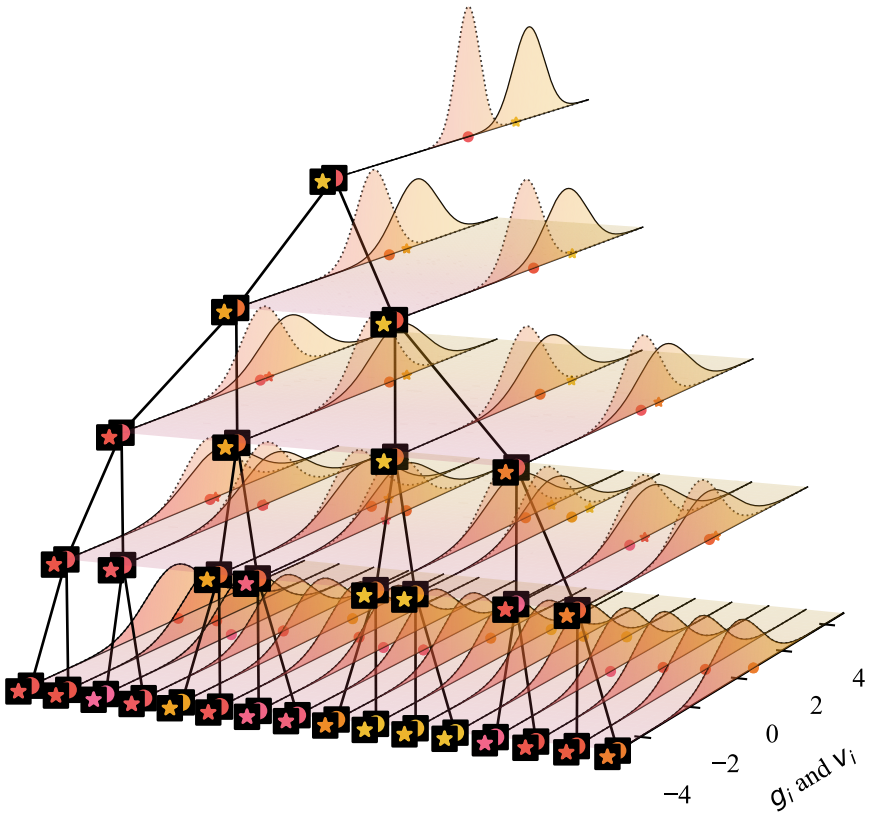


Figure 3.5.: The prior model for probabilistic tree search. Marginals from the priors over the generative scores (—) and optimal values (---), as well as a sample from the model. Samples for generative scores are shown as dots (●) and samples of the optimal values are shown as stars (★). Yellow indicates a high value, and red a low value. On the lowest level the optimal values are the same as the generative scores.

Inference and Search

Similarly to UCT and MENTS, the probabilistic tree search algorithm proposed by Hennig et al. (2010) adheres to the canonical iterative MCTS scheme with the selection, expansion, roll-out, and backup steps. During selection, child nodes are chosen according to an upper-confidence bound policy π, derived from current posterior beliefs over their optimal values. Specifically, the policy selects the child x_c that maximizes a confidence-adjusted value estimate:

$$\pi(x_c \mid x_i) = \arg\max_c \left(\mu_{v_c} + \beta\, \sigma_{v_c}\right), \tag{3.39}$$

where μ_{v_c} and σ_{v_c} are respectively the mean and standard deviation of the current Gaussian belief over the optimal value of node c, and β is an exploration parameter. Once a leaf node x_i on the current search boundary is selected, it is expanded, and a random roll-out is performed from that node.

The outcome of this roll-out provides new data for updating the Gaussian posterior over the generative score, g_i.[8] Due to the assumption of a factorized Gaussian prior over generative scores in the latent-value model, message passing enables efficient posterior updates for the generative scores linear in the depth of the current tree, avoiding the need to revisit the entire tree. During backup, at boundary nodes, the beliefs $p(v_i)$ are computed by summing the independent Gaussian components g_i and Δ_i. For internal nodes, posterior beliefs $p(v_i)$ are obtained by recursively applying the Gaussian approximation scheme described in Box 3.3.1, as marginals $\{p(v_{c_i})\}_{i=1,..,k}$ for the children $c_1, ..., c_k$ of node x_i are available.

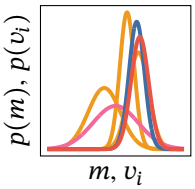


Figure 3.6.: Gaussian Approximation of the maximum of Gaussian distributed variables. The blue and red lines show Gaussian approximations to the maximum m of three independent Gaussian variables v_1, v_2, v_3 plotted in yellow (—). The approximation plotted in blue (—) includes a Gaussian prior (—) for the maximum. The red one (—) is obtained if no prior information on the maximum is included.

8: In the original model from Hennig et al. (2010) – which was developed for game trees with binary rewards – the likelihood for the observed rewards is a Bernoulli likelihood, but the obtaining the posterior for the standard Bayesian optimization setting described above – where the likelihood function is Gaussian too –simplifies the posterior inference.

Box 3.3.1.: Approximating the Maximum of Gaussian Distributed Variables

Consider the problem of determining the distribution of the maximum of a finite set of jointly Gaussian distributed variables, as required, inter alia, for the Δ. Suppose, for the moment, one is interested in the maximum m of only two variables v_1 and v_2. The knowledge $\mathcal{I}_x$ on v_1 and v_2 is expressed by a joint Gaussian distribution $p(v_1, v_2 \mid \mathcal{I}_x) = \mathcal{N}(x; \mu, \Sigma)$. In some cases, there might be prior information $\mathcal{I}_0$ on the maximum in the form of a Gaussian belief $p(m \mid \mathcal{I}_0) = \mathcal{N}(m; \mu_0, \sigma_0^2)$. Both sets of information are combined into a posterior-like distribution of m:

$$p(m \mid \mathcal{I}_v, \mathcal{I}_0) = Z^{-1} p(m \mid \mathcal{I}_0) \cdot \int\int p(v_1, v_2) p(v_1, v_2 \mid \mathcal{I}_v) dv_1 dv_2, \tag{3.40}$$

where Z is a normalization constant. The solution to this problem requires computing the probability for either of them to be the maximum, and the value of the maximum itself:

$$
\begin{aligned}
&p(m \mid \mathcal{I}_v, \mathcal{I}_0) \\
&\quad = Z^{-1} \cdot \underbrace{p(m \mid \mathcal{I}_0) \int_{-\infty}^{\infty} \delta(v_1 - m) \int_{-\infty}^{v_1} p(v_1, v_2 \mid \mathcal{I}_g) dv_2 dv_1}_{m \text{ is distributed like } v_1 \text{ if } v_1 > v_2} \\
&\quad + Z^{-1} \cdot \underbrace{p(m \mid \mathcal{I}_0) \int_{-\infty}^{\infty} \delta(v_2 - m) \int_{-\infty}^{v_2} p(v_1, v_2 \mid \mathcal{I}_g) dv_1 dv_2}_{m \text{ is distributed like } v_2 \text{ if } v_2 > v_1}, \qquad (3.41)
\end{aligned}
$$

where δ denotes the Dirac-delta function. Solving the above integrals amounts to computing generalized error functions, which are intractable in the multivariate case (Genz and Bretz, 2009). The resulting distribution is also not Gaussian. Hennig (2009) derives the moments of $p(m \mid \mathcal{I}_v, \mathcal{I}_0)$ which can be used for moment matching this distribution with a Gaussian. The resulting expressions are listed in Appendix C. The Gaussian approximation for the maximum of two variables can be easily extended to an iterative approximation scheme for the maximum of a finite set with more variables as follows. At first, the maximum over two of the variables is approximated. In each subsequent step, a further variable is added, and the maximum of it and the previous maximum is approximated. An approximation for the minimum can be obtained in the same way since $\min_i\{v_i\} = -\max_i\{-v_i\}$. The computational complexity of approximating the extremal value of b variables is $O(b^2)$ if the variables are correlated and reduces to $O(b)$ if they can be assumed to be independent. Figure 3.6 shows two approximations of the maximum of three independent Gaussian variables, one with the standard Gauss as prior and another one without prior information (i.e. $\sigma_0 \to \infty$).

Contributions

Chapter 4

Optimistic Optimization of Gaussian Process Samples

4.1. Introduction

Bayesian optimization (Shahriari et al., 2015) is a popular and successful framework for global optimization. As reviewed in Section 3.3, the foundation of most Bayesian optimization frameworks is a Gaussian process regression method, whose kernel encodes prior knowledge about the objective function. This Gaussian process regressor provides a posterior over the objective, a probabilistic surrogate on which one can then reason about the extremum, and its location.

The computational cost of Bayesian optimization itself is significant, and arises from at least two sources: First, exact Gaussian process inference has cost $\mathcal{O}(N^3)$, where N is the number of observed function values ("samples"). Second, finding the next evaluation location requires a *continuous, numerical* optimization of the acquisition function. Since this utility function is generally multimodal, optimization should, at least in principle, be carried out on an increasingly fine grid, which contributes up to $\mathcal{O}(N^{2m})$ costs (m is the domain's dimension) (Salgia et al., 2021). A third source of cost can be the construction of the acquisition function itself, but there are some popular choices, like GP-UCB, for which this step is essentially a trivial sum of the mean and marginal standard-deviation constructed during inference, of negligible overhead. If the cost of individual function evaluations is very high, then the overhead of Bayesian optimization is irrelevant. But there are scenarios in which the cost of Bayesian optimization *is* a concern, namely when individual function evaluations are of intermediate cost, and (perhaps as a direct consequence), the total number N of evaluations is sufficiently large to make the cubic cost of Gaussian process inference noticeable. An example are settings involving computer simulations runs, e.g. in robotics, biology, chemistry and human-computer interaction design.

Considering this "middle ground" between sample and computational efficiency, we study a competing framework for global optimization, *Optimistic optimization*, which has drastically lower computational overhead. Optimistic optimization does not require computing an explicit global posterior on the objective. However, Optimistic optimization can nevertheless leverage at least certain kinds of prior knowledge, captured in the form of a Lipschitz condition with respect to a pseudo-metric, or more generally a dissimilarity function, as we have seen in Section 3.1. Functions from a Gaussian process prior fulfill a similar condition with respect to the canonical pseudo-metric of the Gaussian process. This in effect produces a map from a Gaussian process prior one might otherwise use in Bayesian

optimization to a (much more time-efficient) Optimistic optimization model, so that prior knowledge encoded in the Gaussian process can be leveraged without the intermediate step of (cubically expensive) Gaussian process regression. For noiseless observations, the resulting Optimistic optimization algorithm achieves $\mathcal{O}(N \log N)$ computational cost. Some forms of prior information can not be leveraged in this way, so there are settings in which Bayesian optimization is indeed preferrable, even in terms of raw overall speed.

In summary, we provide a practical map between Bayesian optimization to Optimistic optimization for global optimization (Section 4.2), resulting in a hybrid method we call GP-OO. We then contribute a formal analysis of this method in terms of regret (Section 4.4), also pointing out limitations in the process. Experiments (Section 4.5) corroborate that the proposed GP-OO can be significantly more time-efficient than Bayesian optimization in settings with low function evaluation costs.

Section 2 from the original publication with background information on Bayesian optimization and Optimistic optimization was removed since these topics are already fully covered by Chapter 3 of this text.

4.2. Connection between Bayesian and Optimistic optimization

In Section 3.1, we have seen that the policy of Optimistic optimization is based on measuring (dis)similarity in the input domain in terms of a pseudo-metric. It turns out that Gaussian process models – or, more precisely, kernels – can be used immediately to define such a pseudo-metric (Section 4.2.1). We further show how the pseudo-metric can be used to obtain upper bounds on the supremum of the cell (Section 4.2.2), that allow for the application of the Optimistic optimization principle in the Bayesian optimization setting. Section 4.2.3 is concerned with how to choose the cells and in Section 4.2.4 we illustrate the derived concept (GP-OO) on concrete examples.

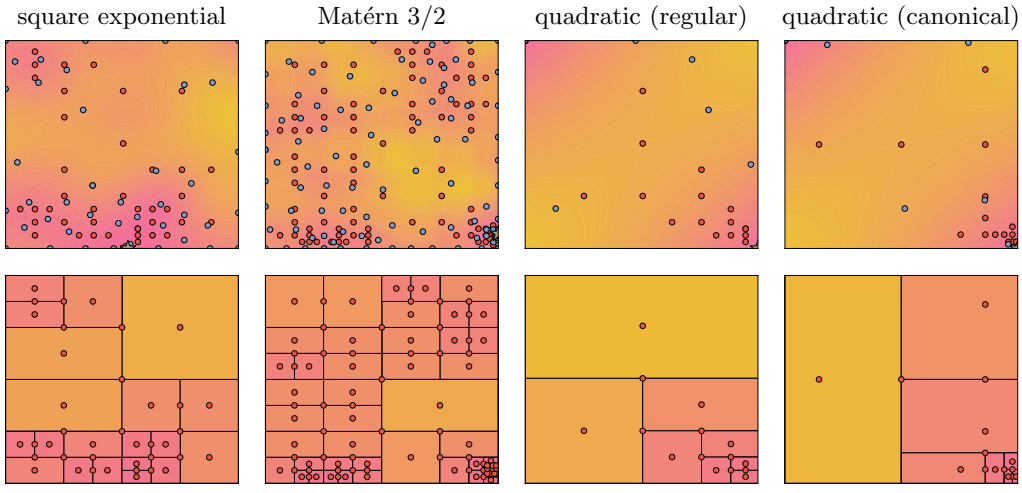


Figure 4.1.: Top: Sample from a Gaussian process and evaluation locations of GP-OO ● and GP-UCB ●. More pink colors indicate higher function values. ★ marks the optimal function value. On the examples here they all happen to lie on the boundaries of the domain. **Bottom:** Cells and diameters Δ. More pink colors indicate smaller diameters.

4.2.1. Canonical pseudo-metric

The canonical pseudo-metric $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for a centered Gaussian process is defined as

$$\delta(x, y) = \mathbb{E}_{f \sim \mathcal{GP}(0,k)}[(f(x) - f(y))^2] \tag{4.1}$$
$$= \sqrt{k(x, x) + k(y, y) - 2k(x, y)}. \tag{4.2}$$

The relevancy of the canonical pseudo-metric for optimization arises from the following deviation inequality (Pisier (1999), Theorem 4.7):

$$\forall u > 0, \mathbb{P}(|f(x) - f(y)| \geq u) \leq 2 \exp\left(-\frac{u^2}{2\delta(x, y)^2}\right) \tag{4.3}$$

The intriguing aspect of this inequality is that it relates distances $|f(x) - f(y)|$ "along the ordinate" with distances $\delta(x, y)$ "along the abscissa". This suggests that the inequality is informative for balancing exploitation ("progress along the ordinate") with exploration ("progress along the abscissa"). From a conceptual point of view, the main motivation of this work consists in deriving an adaptive optimization algorithm, i.e. an optimization algorithm that is able to trade off exploration with exploitation based on observed function values, relying on the above inequality instead of Gaussian process regression. The Optimistic optimization principle is well suited for this attempt due to its upper-bound based acquisition strategy with a simple additive structure of function observations and exploration terms.

4.2.2. Upper bound on the supremum of a cell

The first challenge in applying the Optimistic optimization principle on samples of a Gaussian process consists in the probabilistic nature of the deviation inequality. To obtain a valid upper bound for the maximal deviation in a cell $\sup_{x \in \mathcal{X}_n} |f(x) - f(x_n)|$, the deviation for *all* points in the cell has to be bounded. We approximate such an upper bound by introducing two simplifications: We discretize the search space $\mathcal{X}$ into a finite number of points $\widehat{\mathcal{X}}$ and we introduce an independence assumption between the $|f(x) - f(x_n)|$. Then we take a union bound approach:

$$\mathbb{P}(\sup_{x \in \widehat{\mathcal{X}}_n} |f(x) - f(x_n)| \geq u) \tag{4.4}$$
$$\leq \sum_{x \in \widehat{\mathcal{X}}_n} \mathbb{P}(|f(x) - f(x_n)| \geq u) \tag{4.5}$$
$$\leq 2 \sum_{x \in \widehat{\mathcal{X}}_n} \exp\left(-\frac{u^2}{2d(x, y)^2}\right) \tag{4.6}$$
$$\leq 2|\widehat{\mathcal{X}}_n| \exp(-\frac{u^2}{2\Delta(\mathcal{X}_n)^2}) \tag{4.7}$$
$$\text{where } \Delta(\mathcal{X}_n) = \max_{x \in \mathcal{X}_n} \delta(x_n, x). \tag{4.8}$$

The bounds have to hold at each step n, so we additionally take a union bound over the number of steps. This implies the following statement, that holds with high probability:

$$\forall n : \sup_{x \in \widehat{\mathcal{X}}_n} |f(x) - f(x_n)| \leq \beta_n^{1/2} \Delta(\mathcal{X}_n) \tag{4.9}$$

where β_n are appropriate constants specified in Appendix B. The union bound approximation will be good if the $|f(x) - f(x_n)|$ are (nearly) uncorrelated, or the size of the discretization $|\widehat{\mathcal{X}}|$ is chosen sufficiently small. Otherwise, the bounds are loose, which leads to over-exploration. Though approximate, this idea of neglecting correlations to simplify the calculation of the expected supremum of dependent Gaussian variables is, for example, also done in Maximum Value Entropy Search (Wang and Jegelka, 2017) for Bayesian optimization and in related settings (Grosse et al., 2021). The other extreme, a greedy approach with $\beta_n = 1$ has also been taken in recent work (Rando et al., 2022). For stationary kernels, we experimented with heuristical choices based on the lengthscale, see Appendix B. With generic chaining[1](Talagrand, 1996) it is possible to improve over the union bound approach. However, to the best of our knowledge, state-of-the-art algorithms (Borst et al., 2020) to optimize for tighter bounds require polynomial run-time in the number of points per cell for arbitrary kernel functions. For special cases, like a Wiener kernel (Talagrand, 2021) or Matérn kernel functions (Shekhar and Javidi, 2018) analytical attempts to derive chaining based upper bounds exist.

1: **Chaining** is a technique from empirical process theory used to bound the supremum of a stochastic process. It constructs a sequence of increasingly fine coverings of the input space and bounds the process incrementally over these scales, and in this way leads to tighter bounds than naïve union bounds.

4.2.3. Choosing the partitioning

The second main step in applying the Optimistic optimization principle is to choose the cells and location of the centers in such a way that the diameters of the cells are as small as possible. where $\Delta(d)$ is a decreasing sequence of diameters and C is a global constant. For k children nodes, this is a metric k-center problem – one of the classical NP-hard problems (Gonzalez, 1985). A greedy approximation consists in iteratively picking the k centers with the largest distance to the previously picked centers, and requires $\mathcal{O}(|\widehat{\mathcal{X}}|k)$ time. The greedy procedure is guaranteed to result in a 2-approximation, and there is no polynomial time algorithm doing better (unless P=NP). Working with a greedy instead of the optimal partitioning scheme thereby leads to an additional factor of 2 in the below regret bound, but is not harmful in the sense that the search gets stuck in a local optimum. NP-hardness also appears in the context of Bayesian optimization, e.g. the exploration term used in GP-UCB. And the acquisition functions of information-theoretic Bayesian optimization methods (Hennig and Schuler, 2012; Wang and Jegelka, 2017) are related to the maximization of information gain, which is also an NP-hard problem.
From an implementation perspective, it is desirable to constrain the partitioning to axis-parallel boxes. For some kernel functions, e.g. the polynomial kernel, requirement (b) of the Optimistic optimization principle[2] cannot be fulfilled with axis parallel boxes. One can nevertheless run the algorithm, but it will clearly be less information-efficient. One may even consider a randomized choice of centers, e.g. as done in Monte Carlo Tree Search (Chaslot et al., 2008). However, it still remains to calculate the maximal distance from a point in the cell $\mathcal{X}$ to the center x_n. The computational complexity of this

2: **Reminder:** Formally, one has the following two requirements for the selecting the cells in Optimistic optimization:

(a) $\forall x, y \in \mathcal{X}_{(d,i)} : \{\delta(x, y) < \Delta(d)\}$

(b) $\exists x \in \mathcal{X}_{(d,i)} : \{y \in \mathcal{X} : \delta(x, y) < C\Delta(d)\} \subset \mathcal{X}_{(d,i)}$,

where $\Delta(d)$ is a decreasing sequence of diameters and C is a global constant.

part is comparable to the numerical optimization of the acquisition function in Bayesian optimization. An advantage is that the domain over which one optimizes shrinks in each step. A disadvantage, though, is that if the numerical optimization is suboptimal and the maximal distance within a cell is underestimated, cells containing the optimum might get irreversibly pruned.
An important observation is that the partitioning scheme itself does not depend on the objective f, but only the kernel/distance function (how the search tree grows, however, *does* depend on f). This opens up the possibility of finding a good partitioning and the corresponding maximal distances analytically and prior to the search. An important class of kernel-induced metrics is formed by monotonic transformations of the Euclidean metric, i.e. $\delta(x, y) = g(\|x - y\|_2)$ where $g : \mathbb{R} \mapsto \mathbb{R}$ is monotonically increasing. Many kernels used in practice are in this class, e.g. the square-exponential kernel, the Matérn class of kernels, the rational-quadratic kernels and the Wiener kernel as well as sums and products thereof. We refer to this class of kernels as $\mathcal{K}$. For distances derived from kernels in $\mathcal{K}$, one can apply the following *regular* partitioning scheme: At each step, cut along the longest dimension in order to obtain the two children cells with respect to the Euclidean metric. Use the Euclidean centers as centers. A point that maximizes the distance to the center will always be one of the 2^m corner points in m dimensions. However, one only has to calculate the distance from the center to one of them. Thus, for this type of kernels, the costs reduce to $\mathcal{O}(N2^m)$ for the partitioning, or $\mathcal{O}(N \log N + N2^m) = \mathcal{O}(N \log N)$ in total. Kernels not in $\mathcal{K}$ are e.g. polynomial or periodic kernels.

4.2.4. GP-OO

Motivated by the analysis above, we propose a variant of Optimistic optimization, which we call GP-OO. It consists of running Algorithm 5 (see page 25) with the utility $\mathcal{U}_{(n)} = f(x_{(n)}) + \beta_n^{1/2} \Delta(\mathcal{X}_n)$ in lines 5 and 6, where Δ is as defined in Equation 4.8. Figure 4.1 on page 44 shows the algorithm running on Gaussian process samples from a square-exponential and a Matérn kernel with $\nu = 2/3$ on the domain $[0, 5]^2$, as well as from a quadratic kernel on the domain $[-1, 1]^2$. In regions with higher function values, the partitions are finer. The Matérn kernel yields higher distances than the square exponential, leading to more exploration, reflecting that the samples are less smooth. By optimizing the partitions and centers of the cells with respect to the canonical pseudo-metric, larger parts of the search space can be covered while keeping the cell diameters constant, as shown by the two examples with the quadratic kernel. However, as GP-OO is restricted to evaluations on a grid, it sometimes requires more steps than GP-UCB to reach the optimum, even if the grid is refined in the right regions.

4.3. Related Work

4.3.1. Work at the intersection of Bayesian optimization and Optimistic Optimization

With the exception of (Grill et al., 2018), the fundamental difference to all the work described below is that we do not keep track of a Gaussian process-posterior, thus saving significant computational costs.

Work without the canonical pseudo-metric

Bayesian optimization methods have been combined with SOO (Munos (2011)), the version of Optimistic Optimization with unknown dissimilarity. In BamSOO, Wang et al. (2014) use GP-UCB to reduce the number of evaluations required when running SOO alone. By using SOO, they can in return reduce the optimization costs of the acquisition function. Gupta et al. (2021) improve upon the basic version of BamSOO by a more elaborate partitioning scheme: Instead of dividing a cell into k children along the longest side of the cell, they divide along the b longest dimensions into a cells, where $b^a = k$. Salgia et al. (2021) use a random walk based strategy on a tree to improve over the grid-based optimization of the GP-UCB acquisition function. For the Matérn and Squared Exponential kernel, they achieve order optimal regret, but the computational complexity is $\mathcal{O}(N^4)$.

Work with the canonical pseudo-metric

Shekhar and Javidi (2018) use the Gaussian process's canonical pseudo-metric to improve the numerical optimization by pruning the search regions. They additionally keep track of the posterior to only evaluate at locations where posterior uncertainty exceeds the cell's upper bound. Rando et al. (2022) follow this approach and additionally introduce a Nyström approximation, which allows for approximate inference in $\mathcal{O}(N^2 M_{\text{eff}}^2)$, where M_{eff} is the effective dimension of the search space. Contal et al. (2015) replace the GP-UCB bounds with bounds derived from the pseudo-metric, but do not use a hierarchical approach, i.e. they construct bounds for individual points. They update the posterior and the exploration terms after every new observation.
In Section 3.1, we saw that Grill et al. (2018) applied the optimistic optimization principle to a one-dimensional Brownian walk. A minor difference is that they evaluate a cell at the corners of an interval and not in the center. There are cases where this is advantageous, e.g. think of samples from a Gaussian process with a polynomial or linear kernel. However, the number of corners scales exponentially with the dimension.

4.3.2. Work on scalable Bayesian optimization

TurBO (Eriksson et al., 2019) uses independent local Gaussian process models for a number of trust regions. Trust regions are shrunk or expanded based on heuristics capturing how much progress was made in the previous steps. A global Bandit strategy is used to decide in which of the trust regions to continue the search. Due to the heuristics involved, it is less amenable for theoretical analysis. Other approaches to speed up Bayesian optimization rely on Bayesian neural networks (Snoek et al., 2015), lower dimensional embeddings (Wang et al., 2016), approximations to the Gaussian process (e.g. Jimenez and Katzfuss (2022)) or additive model assumptions (Han et al., 2021; Mutny and Krause, 2018).

4.4. Regret

While computational and not sample efficiency is our main motivation to apply the Optimistic optimization principle in the Bayesian optimization setting, we show that the resulting method nevertheless leads to non-trivial regret. In particular, it is asymptotically regret-free in the limit $\lim_{N\to\infty} \text{Regret}_N/N$. Here, Regret_N denotes the cumulative regret defined as $\text{Regret}_N = \sum_{n=0}^{N} f(x^*) - f(x_n)$.
Building upon arguments from Munos (2011) and Shekhar and Javidi (2018), we obtain the following guarantee for the cumulative regret:

Proposition 4.4.1 (Regret of GP-OO) Let $\mathcal{X}$ be finite, $\epsilon \in (0,1)$, and define

$$\beta_n = 2\log\left(2N|\mathcal{X}_n|/\epsilon\right).$$

Running GP-OO with parameters β_n for a function f sampled from a Gaussian process with zero mean and covariance function $k(x,x)$, the following regret bound holds with probability at least $1-\epsilon$:

$$\text{Regret}_N \leq \sqrt{\beta}\sum_{n=1}^{N}\Delta(\log n),$$

where $\beta = \max\{\beta_1,\dots,\beta_N\}$, and $\Delta(d)$ denotes the diameter of a cell evaluated at depth d.

A full proof is provided in the Appendix B. The high-level idea is that either the explored cell $\mathcal{X}_n$ contains the optimum x^*, then the simple regret $|f(x^*) - f(x)|$ is trivially upper bounded by the maximal deviation $\beta^{1/2}\Delta(\mathcal{X}_n)$. Or the cell does not contain the optimum, but then then its utility was higher than the one of a node containing the optimum in its region, and thereby higher than the optimum itself. For the cumulative regret we assume the worst-case of a uniformly growing tree. For a broad class of kernels, the bound in Proposition 4.4.1 can be further specified:

Proposition 4.4.2 (Regret of GP-OO) Assume the Gaussian process's canonical pseudo-metric δ satisfies $\delta(x,y) \leq C\|x-y\|_2^\alpha$, where $C > 0, \alpha > 0, m/\alpha > 1$. Running GP-OO on a finite domain $\mathcal{X} \subset [0,1]^m$ with regular partitions, one has a worst-case cumulative regret Regret_N of

$$\widetilde{\mathcal{O}}(N^{1-\alpha/m}(\log N)^{\alpha/m})$$

with high probability The $\widetilde{\mathcal{O}}(\cdot)$ notation supresses poly-logarithmic factors.

In particular, for the squared exponential kernel and Matérn kernels with half-integer values $\nu \geq 3/2$, one has $\alpha = 1$ and $\text{Regret}_N \in \widetilde{\mathcal{O}}(N^{1-1/m}(\log N)^{1/m})$. For comparison, the regret in GP-UCB grows as $\widetilde{\mathcal{O}}(\sqrt{N}\log(N)^{\frac{m+1}{2}})$ for the squared exponential kernels and thereby scales better to higher dimensions. For the Matérn class, GP-UCB is guaranteed to have regret at most $\widetilde{\mathcal{O}}\left(N^{\frac{\nu+m(m+1)}{2\nu+m(m+1)}}\right)$ for $\nu \geq 1$. Our bound is tighter, e.g., for the values $\nu = 3/2$ or $\nu = 5/2$ often used in practice.

Shekhar and Javidi (2018) establish regret bounds for their Bayesian optimization method mentioned in Section 4.3 in terms of the near-optimality dimension $\widetilde{m}$. This measure is commonly used in optimistic optimization to characterize the size of the set of ϵ-optimal points in terms of packing numbers. The near-optimality dimension does not only depend on the underlying metric, but also on the function f itself and is thereby a random variable. Smaller values are associated with deeper growing trees, whereas larger values lead to more balanced, uniformly growing trees. Assuming the worst case of $\widetilde{m} = m$, the bounds in Shekhar and Javidi (2018) become $\widetilde{\mathcal{O}}(N^{1-\alpha/m})$ for the Matérn class and match our worst-case bound. However, we restricted the analysis to finite domains $\mathcal{X}$, and $|\mathcal{X}|$ enters our bound logarithmically. Grill et al. (2018) showed that in the specific case of Brownian motion, the tree built during optimistic optimization does not grow with the worst-case uniform rate.

This remark, including Figure 4.2, is not part of the original publication this chapter is based on and was added in the aftermath.

Remark. In Figure 4.2, we illustrate different classes of tree-structured search problems by sampling functions from Gaussian processes whose canonical pseudo-metrics satisfy the condition in Proposition 4.4.2. Each row corresponds to a different exponent α in the kernel $k_\alpha(x,y) = -\|x-y\|_2^\alpha + C$, which induces the pseudo-metric $d_\alpha(x,y) = \sqrt{2\|x-y\|_2^\alpha}$. C is an appropriately chosen constant based on the discretization size in order to ensure positive semi-definiteness of k_α. Smaller values of α result in rougher sample paths, while larger values yield smoother functions. This reflects how the choice of α controls the spatial correlation structure in the Gaussian process, and hence the difficulty of the associated search problem.

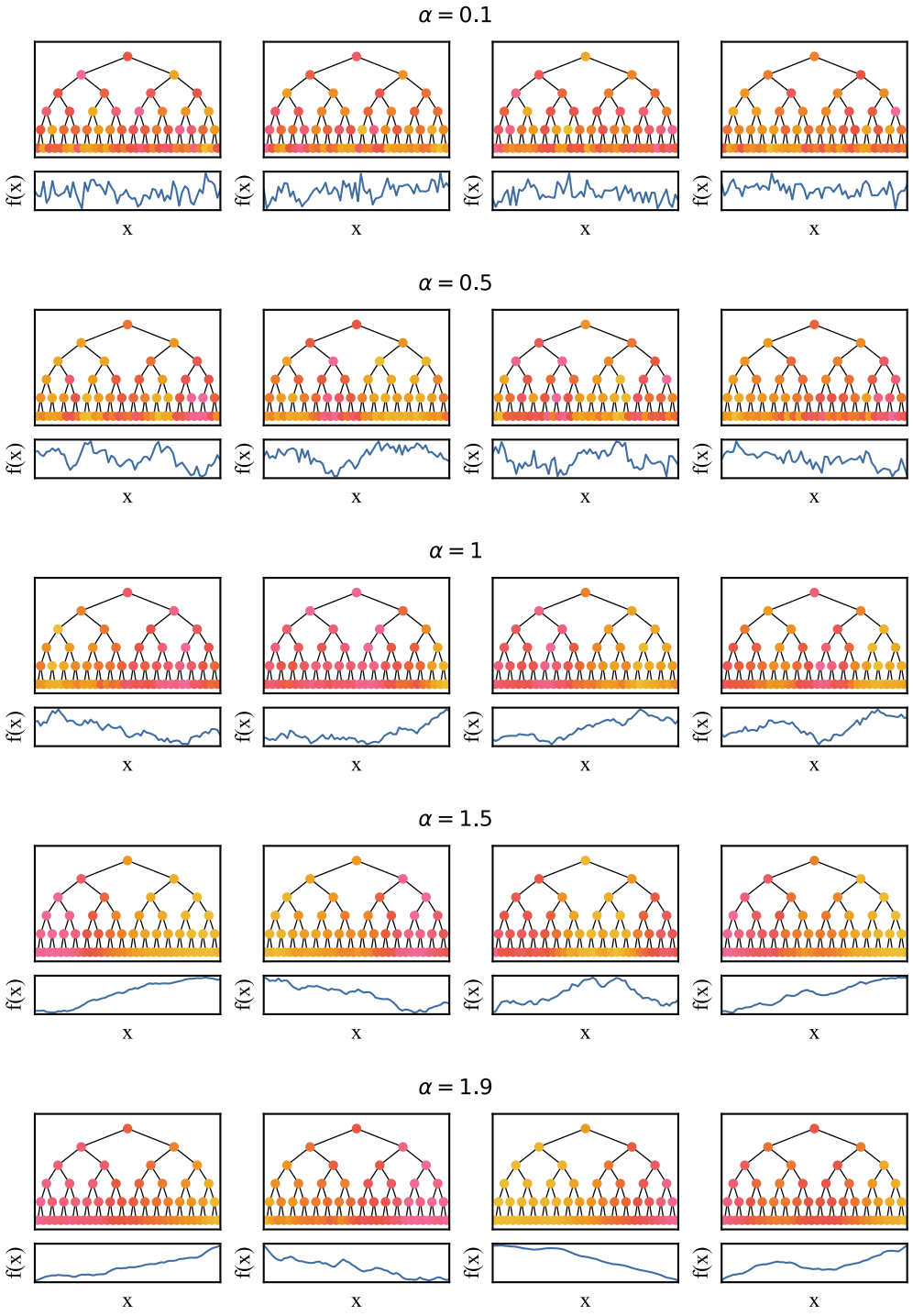


Figure 4.2.: Samples from Gaussian processes which pseudo-metrics satisfy the condition in Proposition 4.4.2 for different levels of α. Each row shows four samples from a Gaussian process with kernel $k_\alpha(x, y) = -\|x - y\|_2^\alpha + C$. The trees correspond to regular partitions and the nodes are colored based on the corresponding function values (yellow values indicate higher function values.)

4.5. Experiments

We empirically compare GP-OO to GP-UCB, EI, TurBO and DiRect in terms of regret and time, on synthetic, three-dimensional samples from Gaussian processes, and Benchmark functions from (Surjanovic and Bingham, 2022). All experiments were performed on a desktop machine. Since computational efficiency is central to our analysis, we also report wall-clock runtimes. While these are implementation-dependent, they are crucial to understanding the interplay of sample efficiency and time efficiency. For GP-UCB and EI, we use implementations from Emukit (Paleyes et al., 2019; Gardner et al., 2018). For TurBO, we build upon the published code.* However, we turn off the optimization of the lengthscale and noise scale and instead provide it with ground-truth values, for a fair comparison with the other methods who also have this information. Also, we changed TurBO's default batch size from 10 to 2 to make it the same as GP-OO's. Other than that, we use its default settings. For DiRect, we use the python implementation available at https://github.com/swyoon/DiRect/blob/master/LICENSE.

Experiments on synthetic functions

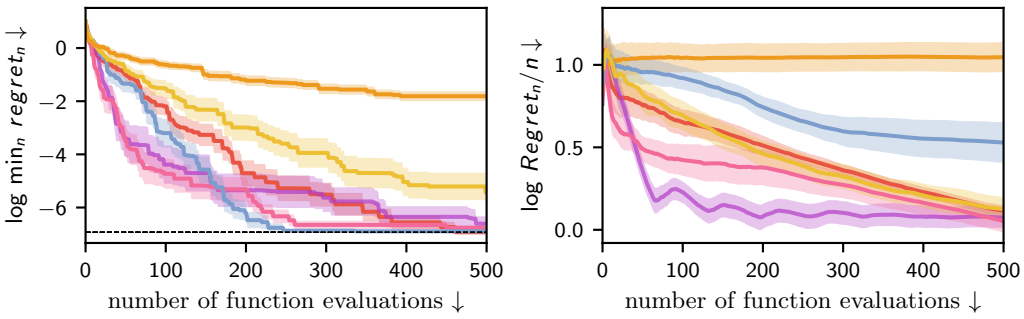


Figure 4.3.: Best simple regret (left) and cumulative regret (right) on synthetic functions sampled from a Gaussian process. GP-OO is shown in ▬, GP-UCB in ▬, EI in ▬, DiRect in ▬, TURBO in ▬ and random selection in ▬. Averages and standard error of the mean from 20 runs.

* https://github.com/uber-research/TuRBO

Regular partitions. We begin with on-model experiments for a common setting in Bayesian optimization, where GP-OO can showcase its speed advantages: 20 samples from a Gaussian process with squared exponential kernel (lengthscale $l = 0.2$) on the unit cube $\mathcal{X} = [0,1]^3$. For all experiments, the noise level for GP-UCB is set to a very small constant $\sigma = 0.0005$ since we assume noiseless observations. For GP-OO, we use $\beta_n = 2\log(2|\widehat{\mathcal{X}}_n|/\epsilon)$ with a discretization of $1/l$ points along each dimension. For additional heuristical choices of β as well as constant choices of βn 0.1, 1, 10, 100, see Appendix B. For GP-UCB, we use $\beta_n = 2\log(2|\widehat{\mathcal{X}}|n^2\pi^2/6\epsilon)$ according to theory with a grid search over the size of the discretization $|\widehat{\mathcal{X}}|$ with values in $\{1, 10, 100, 1000\}$, as well as constant values β in $\{1, 10, 100, 1000\}$. For EI, we did a grid search over the jitter parameter with values in $\{0.0001, 0.001, 0.01, 0.1, 1\}$. GP-OO and GP-UCB require the specification of a confidence level ϵ that is set to 0.05 for both methods. Figure 4.3 shows the simple regret $\min_n \text{regret}_n$ and the average cumulative regret Regret_n/n in terms of number of function evaluations n. Figure 4.4 and 4.5 display the runtime of GP-UCB, GP-OO and TurBO. In terms of number of function evaluations there is no improvement over the state-of-the art, however in terms of runtime GP-OO is orders of magnitude faster than classical Bayesian optimization approaches and also improves over TurBO's runtime. In terms of sample-efficiency, GP-OO improved over DiRect. This indicates that GP-OO can exploit the smoothness encoded in the prior. For a more detailed comparison of DiRect and GP-OO, see also Appendix B.

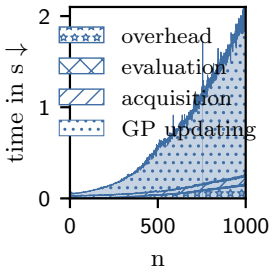


Figure 4.4.: Runtime per iteration for different components of GP-UCB for on model optimization of Gaussian process samples.

Non-regular partitions. For demonstration, we also explore a setting with non-regular optimal partitions, where GP-OO can not perform as well. We sample 100 functions from a Gaussian process with quadratic kernel $k(x,y) = (x^T y)^2$ with bias 0 on the domain $[-1,1]^3$. We consider GP-UCB and GP-OO, once with Euclidean partitions and once with partitions optimized with respect to the canonical metric. The exploration constant was set to $\beta = 1$. The results shown (Figure 4.6) show that it is advantageous to optimize the partitioning scheme with respect to the canonical metric. However, we restrict ourselves to partitions with axis-aligned cells, i.e. the perfectly correlated corner points with $k(x,y) = 1$ and $\delta(x,y) = 0$ end up in different cells. At this point, GP-UCB has a clear advantage, as information can be shared across the search space between the corner points. Also, the run-time advantage decreases due to the numerical optimization of the partitions.

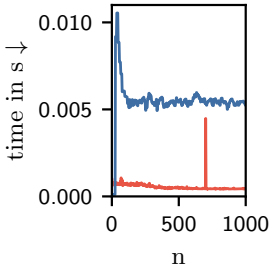


Figure 4.5.: Total time per iteration (averaged over batch size) for GP-OO — and TurBO — .

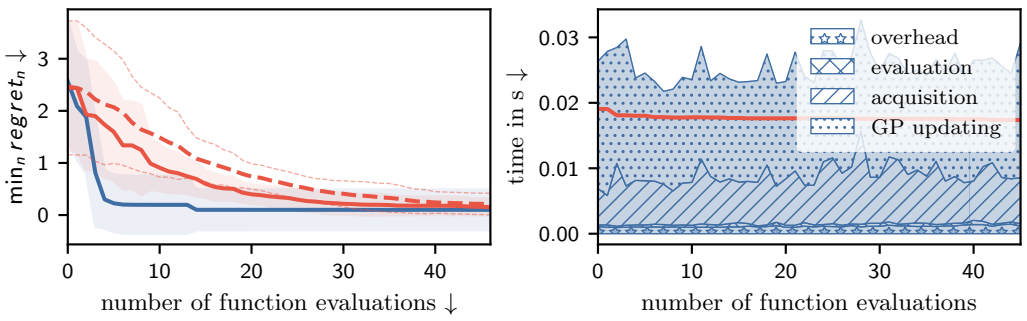


Figure 4.6.: Experiment with a quadratic kernel. Left: Performance in terms of the minimal obtained simple regret $|f(x^*) - f(x)|$ for GP-UCB —, for GP-OO with the canonical metric — and GP-OO with the Euclidean metric ---. **Right:** Wall-clock time with GP-UCB for different algorithmic components per iteration. Wall-clock time with GP-OO in — .

Experiments on benchmarks

We consider the minimization of common optimization benchmark functions, on domains with dimensions ranging from 2 to 10. Since the functions are deterministic, and GP-OO is deterministic up to random tie breaks, we randomly sample 10 domains $\mathcal{X}$ to obtain variation (see Supplements). Since evaluations of the benchmark functions are very cheap, we artificially simulated higher function evaluation costs by adding post-hoc costs from 0.001 to 1 seconds to the evaluation times. We use a Matérn kernel with $\nu = 3/2$ and length-scales as recommended by Rando et al. (2022), who obtained them with a grid search. For the exploration constant β we did a grid search over $\{0.1, 1, 10, 100\}$ for GP-UCB and GP-OO. For EI, we did a grid search for the jitter parameter with values as above. We use a confidence level $\epsilon = 0.05$. Figure 4.7 shows the minimal simple regret over the number of iterations and Figure 4.8 shows the minimal simple regret over time. The Supplements contain additional figures for function evaluation costs of 0.01 and 0.1 seconds, as well as a figure showing cumulative run-times.
GP-OO is competitive with the Bayesian optimization based approaches on 7 of the 12 benchmarks (Dixon-Price, Bohachevsky A, B and C, Ackley, Hartmann and Shekel). This even holds in the number of function evaluations and not only wall-clock time. However, we do not find a consistent advantage over DiRect and in some cases GP-OO is outperformed by DiRect. Since the benchmark functions are typically not within the RKHS, it cannot be guaranteed that GP-OO converges to the global optima. In particular, on Rosenbrock and Beale, it happened that GP-OO got stuck in very bad local optima. For DiRect this happened less, indicating that it was harmful to rely on the smoothness assumptions, when they are not fully met. TurBO is also less prone to get stuck in local optima as it has an in-built criterion for random restarts of trust regions once they converged to a (local) optimum.

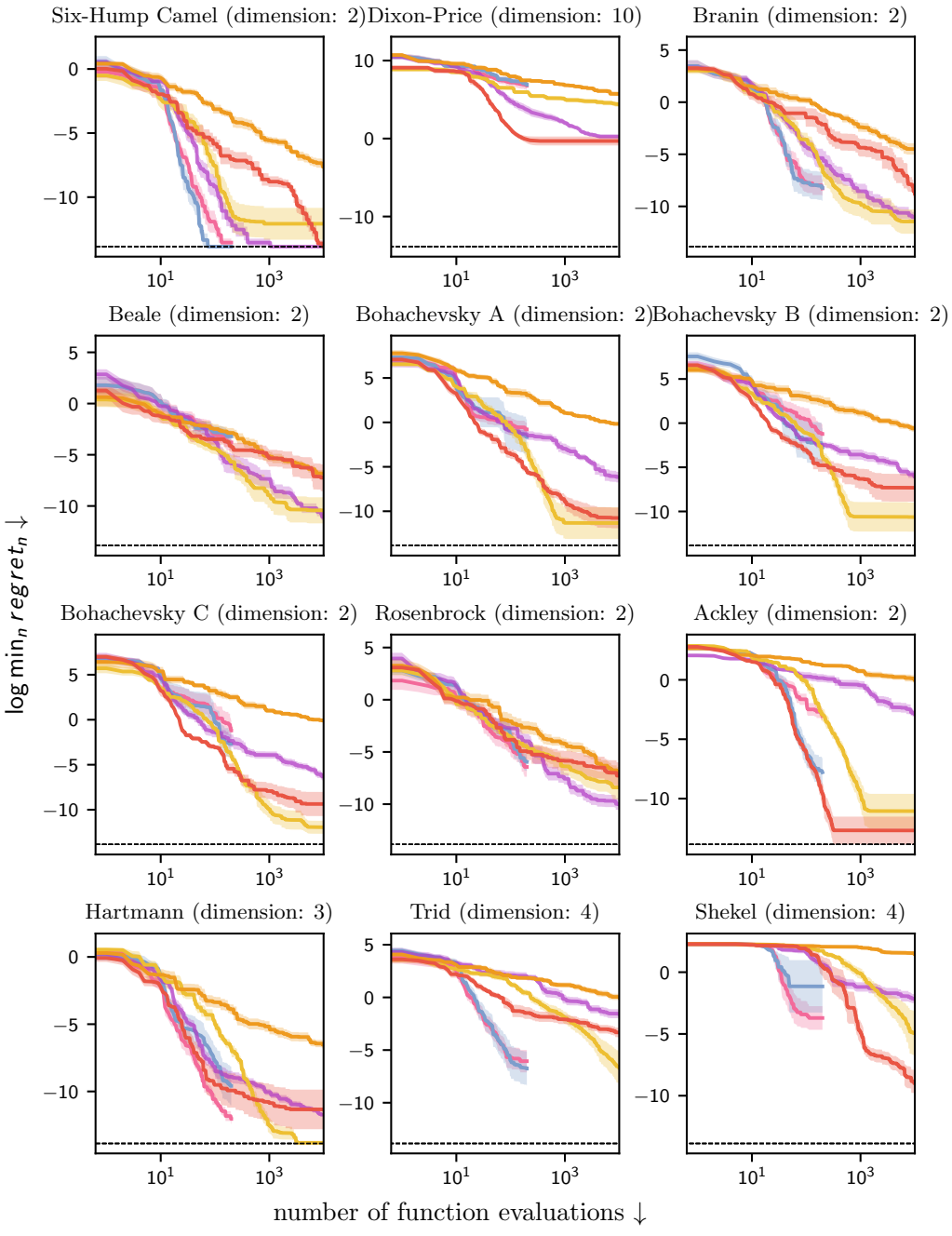


Figure 4.7.: Optimization performance (minimal simple regret) in terms of number of iterations on common benchmarks with GP-OO (—), GO-UCB (—), EI (—), TurBO (—), DiRect (—), and random selection (—) of evaluations. Averages and standard error of the mean from 10 runs.

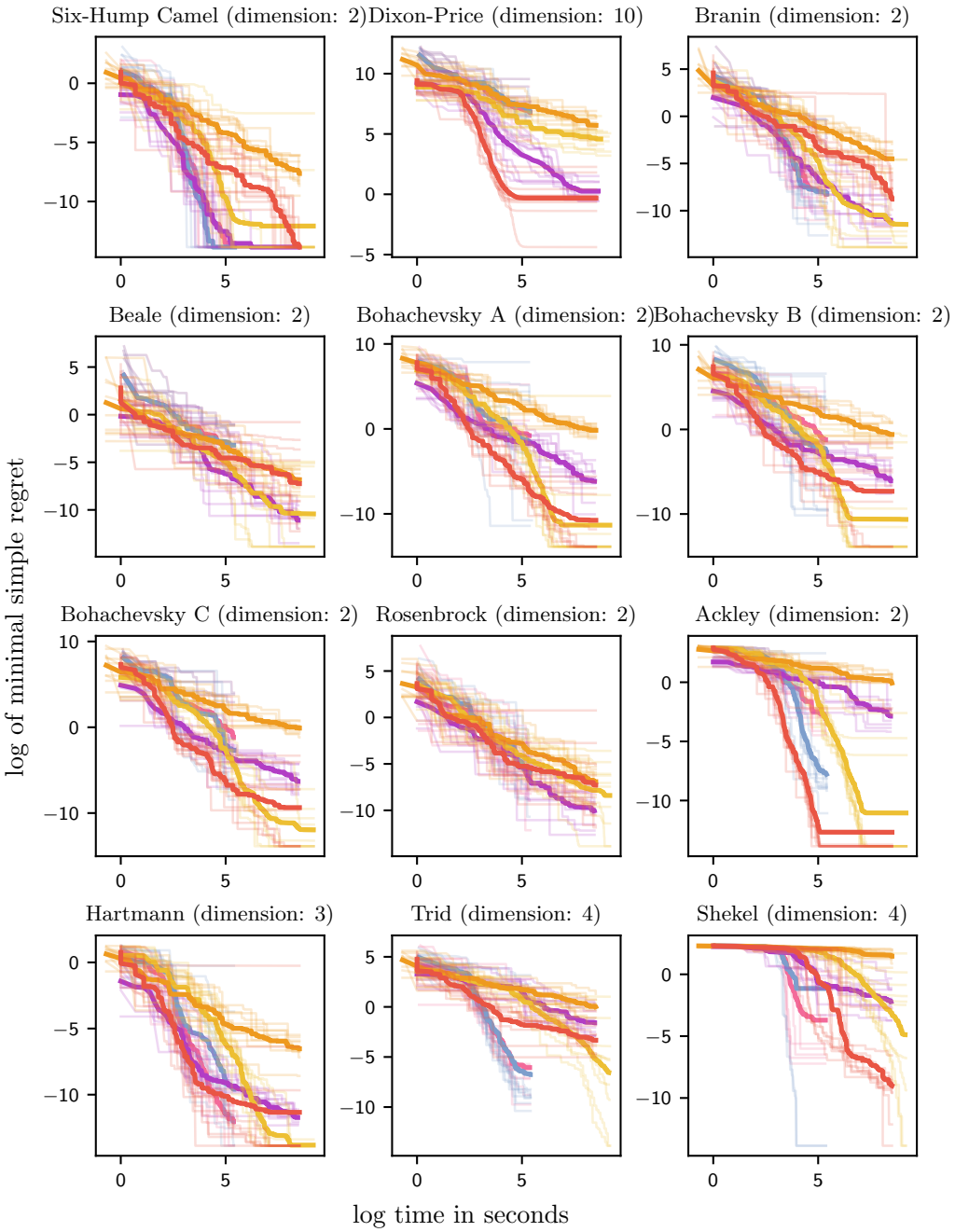


Figure 4.8.: Optimization performance (minimal simple regret in terms of wall clock time for simulated function evalution times of 1 Second common benchmarks with GP-OO (), GO-UCB (), EI (), TurBO (), DiRect (), and random selection () of evaluations. Averags in bold.

4.6. Conclusion

Bayesian optimization and Optimistic optimization are closely connected through a mapping from the kernel-function to the canonical pseudo-metric. We showed that, for *some* kernels, this connection can be exploited to derive a computationally efficient method for the Bayesian optimization setting that captures prior information. For common choices of kernels, like the Matérn class, we outperform classical Bayesian optimization approaches, like GP-UCB and EI, if the objective function has low evaluation cost. In many cases, this simple approach achieves performance similar to or better than scalable Bayesian optimization approaches highly optimized for runtime. On the other hand, the strong reliance on the prior assumptions without being able to adapt the hyperparameters of the prior on the fly, is currently still a limitation in practical settings. A setting, where one quickly wants to find the maximum of synthetic Gaussian process sample, might already be interesting application cases of GP-OO: Grill et al. (2018) suggest to use Optimistic optimization as a subroutine for Thompson sampling. Instead of sampling the function values from the posterior over an entire grid and then taking its maximum, one runs optimistic optimization and only samples the function values at the evaluation locations found with optimistic optimization. In this setting, the hyperparameters of the posterior are known. The same goes for Bayesian optimization algorithms like Maximum Entropy Search or Maximum Value Entropy Search, where, one has to generate distributions of the (location of the) maximum from ”on model ”samples from the posterior. Nonetheless, our work shows that the ability to use prior information in Bayesian optimization does not *have* to be expensive per se. Of particular conceptual importance is the insight that optimization can be performed explicitly tracking a posterior.

CHAPTER 5

Probabilistic DAG Search

5.1. Introduction

In this chapter, we address the fundamental challenge of data efficiency arising from the exponential complexity of the search tree in many planning and decision-making problems. This complexity can often be reduced by collapsing nodes that represent identical problem states into a single node. We build on this idea by proposing a method that also leverages similarity relations—generalizations of the identity relation—between nodes to further improve search efficiency.

As a didactic example, we will consider the search tree of the popular game Tic-Tac-Toe (Figure 5.1). The nodes and paths represent positions and sequences of decisions made by the two players, respectively. Different sequences of decisions sometimes lead to the same position—the paths overlap. While the tree has 58524 nodes, there are only 765 different board states in Tic-Tac-Toe. In the context of games, this is known as *move transposition*, but the same phenomenon also occurs in other decision-making problems beyond the scope of games, e.g. in feature selection. Apart from representing the search space as a DAG, our method also includes relations between board states with overlapping signs, such as node A and B in Figure 5.1(b). Observing a win by playing the game starting from state A also increases the estimated score of node B.

As a second, more realistic example (on account of its scale) we will study the problem of selecting a limited subset of pixels from MNIST images (Deng, 2012) to maximize classification accuracy of the images. The greedy approach, considering the pixels in isolation and picking the most informative ones one after another, is evidently suboptimal: For example, neighbouring pixels could be selected that are informative in themselves, but together provide redundant information. However, testing all possible combinations is, well, *combinatorially* hard. To find the best subset of $k = 10$ pixels from the $N = 28 \times 28 = 784$ pixels of MNIST images, one has to search $\binom{N}{k} \approx 2.28 \cdot 10^{22}$ possible combinations. One way to do so is to build a search tree starting with the empty subset as the root and iteratively adding the remaining features in each level. This tree has $\sum_{i=0}^{k} \frac{N!}{(N-i)!} \approx 8.29 \cdot 10^{28}$ states. For a feature selection problem, though, the order of features does not matter, so the search tree can be reduced to a DAG, which has only $\sum_{i=0}^{k} \binom{N}{i} \approx 2.31 \cdot 10^{22}$ nodes. This is still a large number, so further domain knowledge must be used to leverage *"smoothness"*: For example, the values of nodes representing combinations of pixels, that have some pixels in common, can be assumed to be similar to

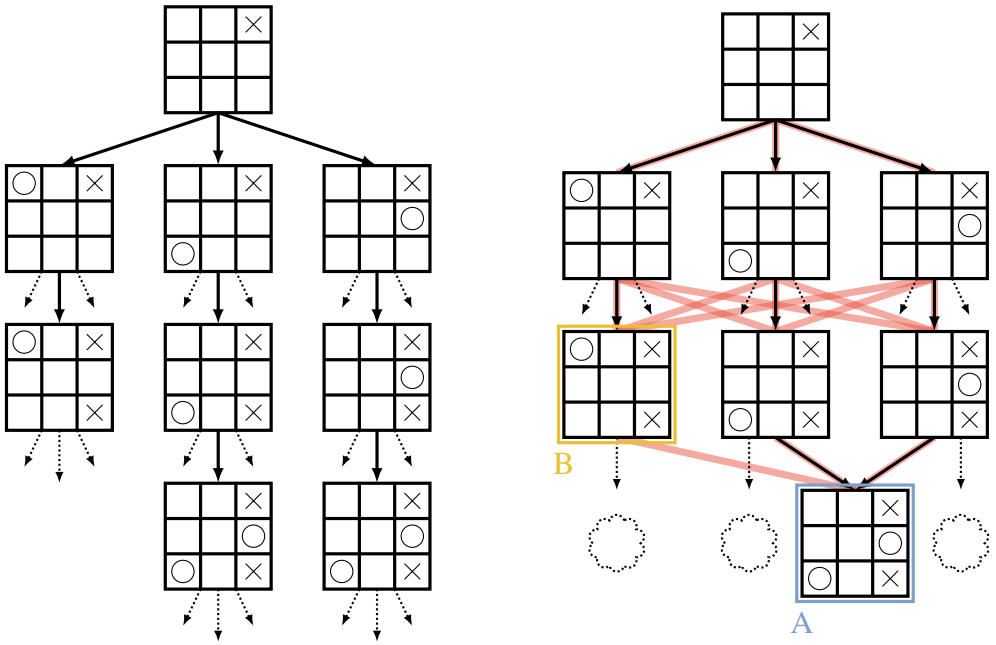


Figure 5.1.: The advantage of treating a search space as a DAG instead of a tree. Left: In a tree, identical states are represented by different nodes if reached by different move sequences. **Right:** In a DAG, such states are collapsed into one node. In the proposed model states with overlapping signs $(\times, \circ)$ will be related to each other even if they are not (yet) connected by a path, as indicated by the red lines. We assign a priori beliefs to the unexplored part of the DAG, illustrated by the clouds.

each other. The information collected to estimate the value of one of the nodes is thereby also useful to learn something about the values of the others.
We develop a probabilistic inference scheme for search problems on (smooth) DAGs, that allows information sharing across the search space, by extending an earlier approximate probabilistic inference framework for the optimal values in (adversarial game) trees (Hennig et al., 2010).

5.1.1. Problem Definition and Notation

We deal with the problem of finding an optimal sequence of decisions in a discrete space. The different sequences form the search space that is represented as a DAG $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ with nodes $\mathcal{X}$ and directed edges $\mathcal{E}$. Each node $x_i \in \mathcal{X}$ corresponds to a state s_i of the search space. The *root* of the DAG corresponds to the initial state. The *children* of a node correspond to the states that can be reached from the *parent's* state by making a decision. A path from the root to a *leaf* (terminal node) describes a complete sequence of decisions. Leaf nodes are associated with a *reward*, the value/utility of this sequence of decisions. We assume that there are no intermediate rewards at internal nodes available. The goal is to find an *optimal path* through the DAG — a path leading from the root to a leaf with maximal reward. We distinguish between two scenarios: in *adversarial settings*, every second decision belongs to an opponent that tries to minimize the reward. Nodes at which the opponent takes the decision are therefore called MIN nodes. Levels in the tree with MIN nodes alternate with levels with MAX nodes. In *non-adversarial settings*, there are only MAX nodes (or MIN nodes). During search *roll-outs* from a node can be requested. For a roll-out, a path from this node to a leaf is descended, guided by a roll-out policy. We use a uniform random policy in this work. The information available from a roll-out consists of the reached leaf x_l and its reward r_l. Nodes have two kinds of values attached:

a *generative score* g and an *optimal value* v. The generative score of a node models the obtained reward if a random path from this node to a leaf is followed. For the optimal value, it is instead assumed that an optimal decision is made at each node of such a path.

Reminder: MCTS and Probabilistic Tree Search

The most popular contemporary approach to tree search is the family of MCTS algorithms. MCTS involves building a growing *search tree* by adding a new node in each iteration. Such an iteration begins with a descent from the root through the search tree, ending at a node that has not been visited before. From there begins a roll-out through the unexplored part of the tree. The reward observed at its end is backpropagated up to the root, improving a value estimate of the existing nodes, which in turn allows a more informative descent in the next step. A key implicit assumption underlying MCTS is that the search space is "smooth": The rewards at leaf nodes are not independent of each other, but those of nodes close to each other in the tree (i.e. nodes that share a larger part of their ancestry) are more similar to each other (Pearl, 1984).

Common forms of MCTS, such as the UCT algorithm are based on PAC (probably approximately correct) point estimates (Auer et al., 1995). The main argument for PAC estimates is that they require little in the way of prior assumptions. This property is a double-edged sword, however, because it also *prevents* the use of patently available prior information. The alternative is to construct an explicit probabilistic model for the tree and perform inference on it. The main problem with this latter approach is that it requires approximations, because inference on the maximum of correlated random variables is notoriously challenging. An early work on such approximate methods for probabilistic tree search by Hennig et al. (2010) constructs a joint generative model for the values of nodes in a game tree both under random and under optimal play. Their construction begins with imposing a Gaussian generative model (based on a Brownian random walk) for the values of board positions *under random play*. The motivation for this model is the intuition that each decision on the way from the root to a leaf changes the value of the position, the probability to win or lose, by a small amount, and this amount is iid. (and assumed Gaussian for convenience) if both players act randomly. From this relatively simple Gaussian model, the op.cit. then derive an inference scheme for the values under *optimal play* as a recursive application of expectation propagation (Minka, 2001a,b).

5.2. Related Work

We integrate three aspects of tree search research: A probabilistic formulation of the search; search on DAGs; and generalization between related states. In addition to the aforementioned probabilistic tree search by Hennig et al. (2010), on which our methods are based, there are others described in Tesauro et al. (2012), Baum and Smith (1997) and Stern et al. (2007). In contrast to the method described above, the latter methods all

model the optimal values of sibling nodes as independent of each other. There have been other (non-probabilistic) attempts to search on DAGs (Section 5.2.1), and to generalize across states (Section 5.2.2).

5.2.1. Search Space as DAG

The idea of treating the search space as a DAG instead of a tree proved valuable in chess (e.g. Plaat et al. (1996); Warnock and Wendroff (1988)) and some other games (e.g. Veness and Blair (2007); Rasmussen et al. (2007)). Because the predominant search algorithm of this era was alpha-beta search, much of the literature is concerned with this depth-first method. For contemporary MCTS, Saffidine et al. (2012) suggest three ways how to incorporate data from roll-outs in the DAG setting, in which the ancestors traversed during the descent from the root to the currently explored node are a *subset* of all the ancestors of the node. The simplest approach is to update only the ancestors in this subset with the usual UCT update rule. A second option is to update all ancestors. As a compromise between the first two options, one could also update the visited ancestors and non-visited ancestors who do not exceed a certain distance from the explored node. There are difficulties with all three options: In the first option, information is wasted. For the second option, Saffidine et al. (2012) provide a counterexample, where the search is likely not to converge, and in the third option, it is unknown how one should choose the distance. Pélissier et al. (2019) describes another idea for MCTS on DAGs. Their work differs from ours in that they recursively update confidence intervals — in contrast to distributions — for the optimal values of interior nodes. Perhaps, the more interesting difference is how the confidence interval/distribution for the optimal value of a node at the boundary is constructed. The confidence interval is based solely on observations from this node. Whereas in our work, the optimal values are based on the generative scores, which allows us to share information from observations made elsewhere.

5.2.2. Generalization between Related States

From the DAG perspective, nodes that represent identical states are collapsed into a single node. A generalization of this discrete form of similarity is the continuous-valued notion of a kernel / covariance. Above, we use this idea to construct a joint Gaussian model with a custom covariance function over the state space. This allows sharing observations over different parts of the DAG. A related, but less general idea is the so-called Rapid Action Value Estimation (RAVE), originally introduced for Go by Gelly and Silver (2007), where RAVE assigns a global score to each of the $19 \times 19 = 361$ available board locations, which is updated whenever in the tree the move is selected. As illustrated above for the pixel selection scenario, the value of a decision often depends on the decisions made before. In order to take this aspect into account, one can also define *local* RAVE-scores for each decision conditioned on others. Nodes are then selected based on a combination of the original value estimate and the RAVE scores. These scores have proven valuable in Go,

but they are not straightforward to extend to other settings, where the set of available actions depends on the context of the state.

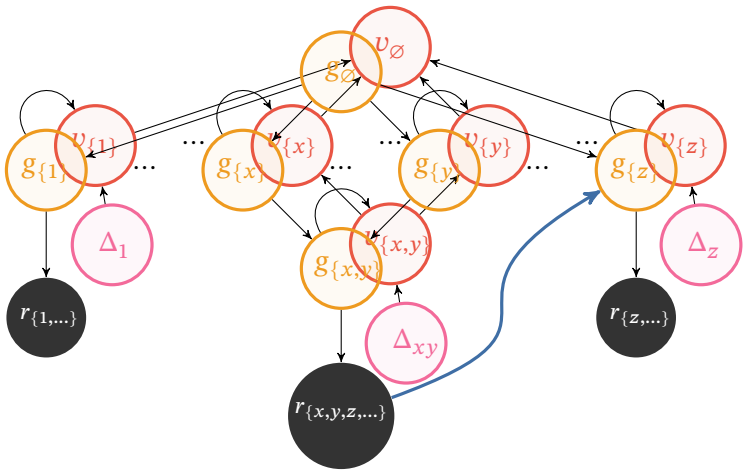


Figure 5.2.: Generative model for feature selection. Figure 2: Generative model for feature selection. Each node in the DAG represents a feature bag. Three variables belong to each node: generative scores g in yellow, optimal values v in red, and increments Δ in pink. Note that even the sub-graph for g's is not a tree, and that not all states are included in this search-graph. Roll-outs from a node result in the observation of a reward r, shown in black. The information from roll-outs is used to update the generative scores. It is shared across the search space, e.g. the observed reward in the roll-out from node $x_{\{x,y\}}$ influences the generative score of node $x_{\{z\}}$ (indicated by the blue line) if the terminal node also contains feature z. For nodes at the boundary, v is inferred from g and Δ and for inner nodes, it is derived from the children's v.

5.3. Method

To construct a probabilistic method for search on DAGs, we extend the algorithmic structure of MCTS outlined above: The observed part of the DAG is traversed from the root to a node at the current boundary of the tracked DAG using a policy based on the beliefs over the optimal values. Once a node at the border is reached, this node is *expanded*: The node's children are added to the DAG—if not already part of the DAG. A roll-out from the node is then performed and the revealed information is used to update the beliefs over the generative and optimal values in the DAG. Doing so in a probabilistic fashion requires a generative model (Section 5.3.1), and an approximate algorithm for inference in it (Section 5.3.3). Following the approach of Hennig et al. (2010), we define the *value*, the quantity to be optimized, in a recursive fashion by leveraging an easier-to-define Gaussian *generative score* describing the evolution under random paths through the graph. Figure 5.2 provides an overview of the generative model in the context of a feature selection application. In this example, the generative score of a node, say node $x_{\{x,y\}}$, models the classification accuracy based on features x and y and $k-2$ additional, randomly selected features. For the node's optimal score $v_{\{x,y\}}$, the *best* $k-2$ of the

remaining features are selected instead.
A crucial part of the proposed probabilistic DAG search is that the information from a roll-out is not only used to update the estimates for the node and its ancestors itself, but also for nodes in other parts of the search space, that are related to the leaf node reached by the roll-out. In Figure 5.2, the roll-out from node $x_{\{x,y\}}$ terminates in a leaf node, that also contains feature z. The reward $r_{\{x,y,z,..\}}$ collected there influences the generative score of node $x_{\{z\}}$. This is realised through a positive correlation of the modeled g-scores of nodes with shared features.

5.3.1. Generative Model

Definition. The joint prior distribution over the *generative scores* g_i over nodes $x_i \in \mathcal{X}$ in the DAG $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ is given by a multivariate Gaussian distribution $p(g) = \mathcal{N}(g; \mu, c \cdot \mathbf{K})$ with arbitrary $\mu \in \mathbb{R}^{|\mathcal{X}|}$ and symmetric positive definite $\mathbf{K} \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$.
The covariance $\mathbf{K}_{ij}$ of the generative scores g_i and g_j should be defined such that it captures the problem-specific relationship of the states represented by nodes x_i and x_j. For example, in Tic-Tac-Toe, a plausible choice is the number of overlapping $\times$'s and $\circ$'s. In our second example, feature selection for MNIST images, we define the covariance as the shared number of selected pixels in the feature bag represented by the two nodes (+1 on the diagonals to ensure positive semi-definiteness).
The scaling factor c in the prior variance captures the range of possible rewards and depth of the tree. In our feature selection experiments, we standardize the rewards at leaf nodes such that they have unit variance and zero mean. In this case, c can be set to the inverse depth of the DAG, and the prior mean μ to zero.

Definition. The likelihood of an observed reward η at a leaf node x_l is defined as $p(\eta_l \mid g_l) = \mathcal{N}(\eta; g_l, \sigma)$ where $\sigma \in \mathbb{R}$ is a small constant modeling noise in the observations.

For example, in game scenarios an observation η consists of the outcome of the game encoded numerically, and in the feature selection scenario an observation could be the classification accuracy.

Definition. The *optimal value* v_l of a node is defined recursively: For a leaf node x_l, set $v_l = g_l$. For interior nodes in adversarial situations (where choices are made in alternating fashion between a MAX and a MIN player), the optimal value is the maximum or minimum of the children's optimal values:

$$v_i = \begin{cases} \max\{v_j \mid x_j \in \text{children}(x_i)\} & \text{if } x_i \text{ is MAX} \\ \min\{v_j \mid x_j \in \text{children}(x_i)\} & \text{if } x_i \text{ is MIN} \end{cases} \tag{5.1}$$

(for non-adversarial settings, every node is MAX).

5.3.2. Inference

The goal of our algorithm is to find a path through the DAG that leads to a leaf x_l with globally maximal v_l. The end of the recursion in the definition of the optimal values are the leaf nodes. Since the DAG is only observed up to the current boundary and we only keep track of beliefs of observed nodes, we have to approximate the optimal values of the nodes at the boundary. This is where the generative scores come into play. We assume that the optimal value v_i of a node x_i at the boundary consists of its generative score g_i and some increment Δ_i:

$$v_i = g_i + \Delta_i \tag{5.2}$$

The increment Δ_i of a node indicates how much we can improve the score of the node if the remaining steps are selected in an optimal manner in contrast to random selection. As reviewed in Section 3.3, Hennig et al. (2010) show how to derive beliefs for these increments, prior to data acquisition. We take over their derivations, i.e. we treat the *unexplored* part of the search space as a tree, and assume that the generative score g_j of a child node x_j can be obtained from the parent's generative score g_i by adding a (scaled) Brownian step $\xi_j \sim \mathcal{N}(0, c)$ to the parent's generative score g_i:

$$p(g_j \mid g_i) = \mathcal{N}(g_j; g_i, c) \tag{5.3}$$

The Brownian steps from a parent node to its children, on which the definition of the Δ is based, are independent of each other in the tree model, i.e. the Δ of sibling nodes are not correlated. If the search space is actually a general DAG, the effective number of options is generally smaller than in a tree, because some paths reunite in later steps. In other words, the optimal increments Δ of sibling nodes are correlated. For example, think of the extreme case where all paths end in the same leaf node and the Δ would be perfectly correlated.

5.3.3. Inference on Generative Scores

During search, observations from roll-outs are used to iteratively update the distribution over the generative scores, which then also informs the optimal values. This is where the smoothness of the DAG is explicitly used. Since both prior and likelihood for g are Gaussian, this task is straightforward: Given observations $O = \{(x_l, r_l) : l = 1, ..., t\}$ from the first t roll-outs, the posterior mean and covariance are obtained with the usual formulas for Gaussian regression:

$$\mu_i' = \mu_i + \mathbf{K}_{iO}(\mathbf{K}_{OO} + \sigma^2)^{-1}(\mathbf{r} - \boldsymbol{\mu}_O) \text{ and} \tag{5.4}$$

$$\mathbf{K}_{ij}' = \mathbf{K}_{ij} - \mathbf{K}_{iO}(\mathbf{K}_{OO} + \sigma^2)^{-1}\mathbf{K}_{Oi} \tag{5.5}$$

The matrix $\mathbf{K}_{OO} \in \mathbb{R}^{t \times t}$ denotes the covariance between the g scores of the observed leaf nodes $\{x_l : l = 1, ..., t\}$ and the vector $\mathbf{r} \in \mathbb{R}^t$ contains the corresponding observed rewards $\{r_l : l = 1, ..., t\}$. To simplify exposition we consider this generic joint Gaussian

model in this paper and do not assume further simplifying structure to $\mathbf{K}_{OO}$. This means that inference is of cubic cost in the size of the state-space in memory (not the full, potentially exponentially large problem!). We note in passing that a naive implementation would have cost $\mathcal{O}(T^4)$ because of repeated matrix inversions, but this can be reduced to $\mathcal{O}(T^3)$ through careful use of the matrix-inversion lemma. Of course the various means of speeding up Gaussian inference to quadratic and linear time complexity developed in the machine learning community over recent years can be applied more or less directly to our framework to alleviate this polynomial complexity issue (Snelson and Ghahramani, 2006; Titsias, 2009; Hensman et al., 2013).

5.3.4. Inference on Optimal Values

While inference on the generative scores g is easy, inference on the extremal values v is non-analytic anywhere other than at the boundary of the graph: For boundary nodes, the optimal value is the sum of the generative score and Δ (Equation 2). Since the beliefs over the generative scores and Δ's are Gaussian, their sum is also a Gaussian random variable:

$$p(v_i) = \mathcal{N}(\mu_{g_i} + \mu_{\Delta_i}, k_{g_i} + k_{\Delta_i}) \tag{5.6}$$

Yet, for the interior nodes, v values are defined as the maximum or minimum of their children's v values. We recursively use the approximation scheme from Section 3.3.2 to approximate the v values of interior nodes based on the approximations of their children's v values. The above scheme can actually account for correlations between v-values if they are known, but in our experiments we assumed independence between the optimal values. This is of course a strong, and formally incorrect simplification, but it is currently unclear how to correctly compute the correlation coefficients. Such assumption has the effect that maximas are overestimated and minimas are undererstimated, respectively. A simplistic, but in our experiments effective way to counteract this is to introduce a (Gaussian) prior for the maximum that acts as a regularizer. The selection of the hyperparameters of this prior can be motivated by the range of the rewards. Assuming standardized rewards, we use a standard normal distribution as prior for the optimal value.

Inference on a node's optimal value requires beliefs over the optimal values of all children. This is why we suggest to add all children of a node to the DAG as soon as it is expanded. However, this effectively increases the size of the search DAG by the branching factor of the graph. For problems with a high branching size (such as the MNIST pixel-selection example below), this can be a computation and memory bottleneck. As a simple, lightweight approximation, we then calculate the optimal values by considering the optimal values of just the observed children and a single summary variable for the maximum of all the currently *unobserved* children. We approximate the value of this unexplored option by the parent's generative score and the Δ one would get at the parents level assuming a branching factor equal to the number of the remaining unexplored children.

5.3.5. Simplified Estimation of Optimal Values

A further simplification of the inference on the optimal value v_j at a parent's node can be achieved by replacing the inference scheme in Box 3.3.1 and instead approximating v_j more ad-hoc, as a weighted sum of the children's values $v_j := \sum_{i=1}^{B} w_i v_i$ with higher weights w_i for children with higher estimated v_i value. Using the softmax function as a soft version of the maximum function we propose to set the weights w_i to

$$w_i = \frac{e^{-(\mu_{max}-\mu_i)/k_{v_i}^{1/2}}}{\sum_{b=1}^{B} w_b} \tag{5.7}$$

with $\mu_{max} = \max_k \mu_{v_k}$. The weight vector can be interpreted as a categorical distribution for each of the children nodes to be the best choice (the argmax). Assuming independence between the children and constant weights, one obtains for the parent's mean μ_{v_j} and variance k_{v_j}:

$$\mu_{v_j} = \sum_i w_i \mu_{v_i} \text{ and } k_{v_j} = \sum_i w_i^2 k_{v_i} \tag{5.8}$$

The experiments below suggest that this new update rule for the optimal values, although ad hoc, performs only slightly worse than full inference while making the implementation considerably easier.

This remark is not part of the original publication this chapter is based on and was added in the aftermath.

Remark. The simplified estimation of the optimal value we introduced in Section 5.3.5 is similar to the softness framework discussed in Chapter 3.2. However, a key distinction is that the softargmax in equation 5.7 accounts for the uncertainties i in the optimal value of each child v_i. Since these uncertainties vary across child nodes, this approach corresponds to Maximum Entropy regularization with individual temperature parameters. Incorporating these uncertainties breaks the scale invariance of the Boltzmann distribution in 5.7: adding a constant to all children's optimal values no longer leaves the softargmax unchanged. To mitigate this undesirable effect, we standardized the softargmax by shifting each child's optimal value so that the maximum is always zero across all nodes in the tree. It is also important to note that though the above approximation is also based on the softargmax, the back-up step is different from the one used in MENTS in Section 3.2.3: We do not propagate the softmax – the sum of the expected value plus the entropy – up the tree, but instead the expected value and the scale as in Equation 5.8. In this way, information about the scale of the optimal value is not lost. However, using the entropy allows Xiao et al. (2019) to show that MENTS actually operates with an upper bound on the maximum, whereas there is no such guarantee for the upper bound strategy we introduced in Section 5.3.6. While the focus of this chapter is on the impact of exploiting node similarities across the search tree, an in-depth comparison between the different back-up operators poses an interesting next step towards a better understanding of the role of uncertainty quantification in tree search algorithms.

5.3.6. Policy

The procedure outlined above yields local, approximately Gaussian distributions over the optimal values v, which can be used to guide the search, as well as during evaluation. For exploratory *search* we use a standard upper confidence bound (Auer et al., 1995) policy π to select nodes (for MAX nodes, and mutatis mutandis for MIN nodes):

$$\pi(x_j) = \arg \max_{x_i \in \text{children}(j)} \mu_{v_i} + \beta \sqrt{\log(n_j) k_{v_i}} \tag{5.9}$$

The variable n_j counts how often node x_j was visited during the search (cf.Section 5.4) on how the exploration parameter β was set in the experiments).

5.3.7. Summary

To wrap up, an iteration in probabilistic DAG search consists of the following steps: A new node is added to the explored DAG and roll-out is carried out. The reward from the roll-out is used to update the generative scores of all nodes on the boundary via a Gaussian regression. Marginals for the optimal values of boundary nodes are inferred by adding up the marginals for the generative scores g and the pre-computed optimal increments . The optimal values of all of the remaining nodes in the explored part of the DAG are updated using the Gaussian approximation scheme.

5.4. Experiments

We test our method in three different settings: The first experiment is on a synthetically generated DAG/tree, where the covariance structure of the ground truth is in full accordance with the generative model assumed by the probabilistic DAG search. The second experiment, Tic-Tac-Toe, is an example of an adversarial search problem. The last one is a high-dimensional feature selection experiment on MNIST with a branching factor $b = 784$, higher than (but not as deep as) that of the Go game tree. See Figure 2 for how to formulate feature selection as DAG search. We compare our algorithm both to the probabilistic tree search of Hennig et al. (2010), as well as classic UCT and UCT on DAGs (UCD, Saffidine et al. (2012)), as discussed in Section 2.2. We use the simplest version of UCD, where only the traversed ancestors are updated.

5.4.1. Synthetic search problem

In this first problem, the underlying search space is a DAG $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ with nodes $\mathcal{X} = \{x \subset F \mid |x| \leq m\}$ and edges $E = \{(x_1, x_2) \mid x_2 = x_1 \cup f \in F\}$, where $F = \{1, ..., N\}$. We impose a covariance $\mathbf{K} \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ on the generative scores of nodes x_i and x_j defined as $\mathbf{K}_{ij} = (|x_1 \cap x_2| + 1)/(m + 1)$. The ground-truth rewards at the leaf nodes are sampled from $\mathcal{N}(0, \mathbf{K})$. The ground truth covariance structure is

known to the probabilistic versions. $N = 15$ and $m = 5$, i.e. the DAG has $|\mathcal{X}| = 4944$ nodes of which 3003 are leaf nodes. σ was set to 10^{-4} for the probabilistic versions. $\beta \in \{0.01, 0.1, 1, \sqrt{2}, 10\}$ for all versions and Figure 6.3 shows the results from the best choice, respectively. It is clearly advantageous to use the covariance, whereas the reduction in the number of nodes from tree to DAG has only a negligible effect on performance. We also note that, here, the simplified inference rule achieves nearly the same performance as the fully probabilistic version.

The outcome of Tic-Tac-Toe is either a win ($r = 1$), a draw ($r = 0$) or a loss ($r = -1$). All search versions are evaluated after each step against an optimally playing MIN player. In this case, the best outcome that the MAX player can reach is a draw. During evaluation, the MAX player selects its move based on the maximum-a-posterior estimates of the v values or selects its move randomly if it reaches a previously unexplored node. We used the hyperparameters $c = 0.5, \sigma = 0.1$ for the probabilistic DAG and tree version and $\sigma = 0.001$ for the softmax-based simplification (we did not perform an extensive hyperparameter search, as these are relatively natural choices). The exploration constant is set to $\beta = 1$ for all search versions. Figure 5.4 shows the average reward for the MAX player with respect to the number of search steps. The probabilistic search on the DAG is able to use its similarity metric to outperform the other algorithms by a few hundred steps. Another factor for the better performance of the probabilistic DAG search could be that several nodes (all children) are added to the DAG in each search step. We also found that UCD worked better than UCT, supporting the expected gain from the reduced overall number of states. A comparison of the probabilistic DAG version using the fully probabilistic updates and the softmax-based simplification suggests that the lead of the probabilistic DAG version is not only due to the covariance structure, but also due to the probabilistic treatment of the optimal value estimation.

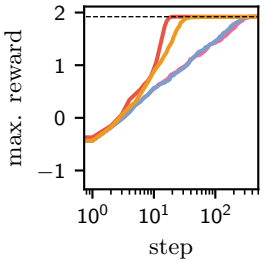


Figure 5.3.: Performance on synthetic searchproblems of PDAGS (—) and PDAGS with softmax values (—) with UCD (—) and UCT (—). All methods obtain the optimal reward (---) in the end. Shaded regions show standard deviations from 500 runs each.

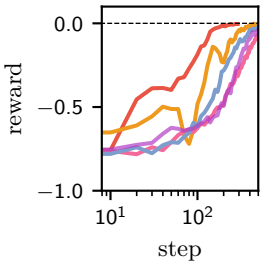


Figure 5.4.: Performance on Tic-Tac-Toe of PDAGS (—) and PDAGS with softmax values (—) with UCD (—) and UCT (—) and PTS (—). All methods obtain the optimal reward (---) in the end. Average reward from 20 search runs. Standard deviations over 25 such repetitions (of 20 runs) as shaded regions.

5.4.2. Pixel Selection for MNIST Images

Our final experimental task is to select a subset of 10 pixels from the MNIST images based on which the images are classified.

As a classifier for this task, we use a Convnet (cf. Appendix C). The classifier is trained with images where all but 10 randomly selected pixels are masked. 1000 randomly selected images of the MNIST train set are held back as validation set. During the search, the classifier is evaluated on this set to obtain estimates of the accuracy. Before the search, 20 random roll-outs are performed and the standard deviation and mean of the observed accuracies are used to standardize the rewards. We use $\beta = 0.5$ as exploration constant for all search versions and, since the classification accuracy can be evaluated to high precision, $\sigma = 10^{-4}$ as noise parameter for the likelihood in the probabilistic methods. For the approximation of maxima and Δ we use a standard Gaussian prior. For this high-dimensional feature selection problem, UCT explores the search space almost uniformly, due to the high branching factor. Gaudel and Sebag (2010) propose a heuristic based on RAVE scores in order to make UCT applicable for large scale feature selection problems. A detailed description of this heuristic can be found in Appendix C. Figure 5.5 shows the

Figure 5.6.: Final set of selected pixels (in blue), show on randomly selected images from the MNIST test set.

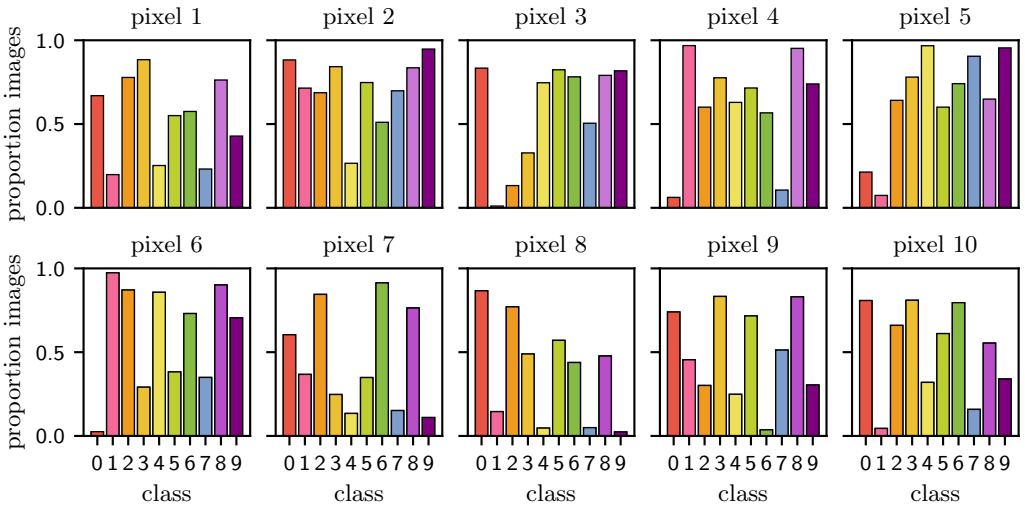


Figure 5.7.: Proportion of images per class with positive value for the 10 selected pixels in Figure 5.6 (labelled accordingly).

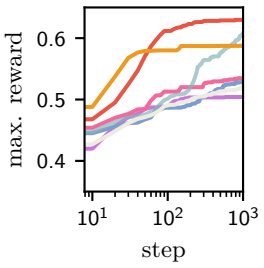


Figure 5.5.: Performance on the MNIST pixel selection task of (—) and PDAGS with softmax values (—) with PTS (—), UCD (—), UCT (—), UCT-RAVE (—) and random search (—). Average reward from 10 rounds. Shaded regions indicate the standard deviations.

highest validation accuracy obtained by previous evaluations of leaf nodes. We observe a difference in performance between the search versions that generalize across the state – probabilistic DAG, probabilistic DAG (simplified), UCT-RAVE – and the remaining ones. This indicates that the exploitation of similarity relations among states was more important than just a reduction in the number of states based on identity relations and even essential to learn anything on this problem. We also find a headstart of the probabilistic DAG version(s) as compared to UCT-RAVE and an advantage of the full-probabilistic update over the simplified, softmax-based updates.

Figure 5.6 shows 10 pixels selected by our method after 1200 steps. Figure 5.7 shows, for each class and each pixel, the proportion of images in the test dataset for which the respective pixel has a positive value. They give an intuition for their discriminative power. For example, pixel 3 separates classes 0 and 7 well from classes 1 and 8.

5.5. Conclusion

We have presented a probabilistic search algorithm for problems whose state-spaces can be represented by DAGs. A benefit of this formulation over tree search, beyond the obvious reduction in the size of the state-space itself, is that it allows for an exploitation

of similarities between states. The key ingredients of our method are an approximate inference scheme for extremal values, which we constructed as an instance of expectation propagation, and an ad-hoc prior for the optimal value of unexplored parts of the search space, for which we extended earlier work on tree-structured problems. Our experiments in the adversarial setting of Tic-Tac-Toe and the non-adversarial case of feature selection demonstrate not just good performance but also scalability. In addition, we examined a simplified version for the inference of optimal values as opposed to the one in Hennig et al. (2010), that is based on the softmax function as a relaxation of the argmax function. While it is relatively straightforward to include correlations between child nodes in the approximate inference scheme, it is challenging to construct a model to identify such correlations across the state space, a task that we reserve for future work.

Chapter 6

Uncertainty-Guided Likelihood-Tree Search

6.1. Introduction

In this chapter, we will once again focus on the aspect of *epistemic* or *computational uncertainties* (Hennig et al., 2022) involved in tree search: An uncertainty that *could be* fully resolved if enough compute was available to examine all paths, but in practice is present due to the limited resources. Standard search algorithms for large-scale search trees, such as beam search (Koehn et al., 2003), completely ignore this uncertainty. Meanwhile, uncertainty-aware algorithms such as Monte Carlo tree search (MCTS, Kocsis and Szepesvári, 2006b) and its Bayesian counterparts (Hennig et al., 2010; Tesauro et al., 2012; Mern et al., 2021) require repeated (costly) evaluation of nodes, additional expensive models, or complicated approximate inference.

To maximize efficiency and cost-effectiveness, we are thus interested in an online tree-search algorithm that can exploit computational uncertainty without rollouts, expensive models, and complicated Bayesian inference. We posit that even a *simple* way of quantifying the aforementioned uncertainty can enhance exploration within the search tree; ultimately reducing the number of node expansions while maintaining the same or achieving a better reward. To this end, we incorporate computational uncertainty into the search process to guide it in a non-myopic fashion—i.e., accounting for our *belief* about the values of future nodes, taking inspiration from how humans might plan (Baumeister et al., 2016)—and in a data-efficient manner, akin to Bayesian optimization (Kushner, 1964; Močkus, 1975).

Bayesian optimization is recognized for its efficiency not merely because it quantifies uncertainty, but also because it exploits the structural characteristics of the problem. E.g., in continuous optimization, prior knowledge, such as the smoothness of a function, is often available through Gaussian processes (Rasmussen and Williams, 2005) or neural networks (Hernández-Lobato et al., 2017; Kristiadi et al., 2023). In our setting, we have the structural assumption that the immediate reward at each node of the search tree is a component of a Categorial distribution, i.e. normalized in $[0, 1]$. Intuitively, the reward of each node is the *likelihood* of the node being in a given root-to-leaf path in the tree, and the goal is to find a path with the highest total likelihood (cumulative reward). This setting is of great interest since it occurs in problems such as autoregressive generation processes like in large language models.

In our setting, the *a priori* characteristic property we aim to exploit is the *concentration strength*: whether the Categorical distributions are all highly concentrated at a few realistic

```
tokenizer = AutoTokenizer.from_pretrained("your-model")
model = AutoModelForCausalLM.from_pretrained(
  "your-model", torch_dtype=torch.bfloat16
)
model.eval()
model_inputs = tokenizer("input prompt", return_tensors="pt")
-output = model.generate(**model_inputs, num_beams=5, max_new_tokens=40)
+import ults
+output = ults.generate(model=model, model_inputs=model_inputs, max_tokens=40)
generated_text = tokenizer.decode(output.sequences[0])
```

Figure 6.1.: Code example of applying ULTS to LLM decoding instead of beam search.

options, or if some of them are nearly uniform, making any individual options drastically less likely to be the optimal one. Intuitively, one would expect this to have a strong influence on the number of paths that need to be considered. For instance, when the distribution is concentrated, it is less likely under our belief that other paths will overtake later on and one can be more greedy. Meanwhile, when the distribution has high entropy, the uncertainty of our belief about "which next child node is best" is higher and thereby requires more exploration and computational budget.

In this work, we propose a probabilistic framework that captures this aspect of the search space, which can help to decide which paths should be pursued and which can be ignored based on the posterior belief. We do so by using a non-myopic acquisition function based on the samples of such a belief, which can be seen as a generalization of Thompson sampling (Thompson, 1933). Crucially, these samples are *cheap* to obtain, unlike other empirical sample-based tree-search methods like MCTS.

Ultimately, building upon elements from beam search, MCTS, and Bayesian tree-search methods, we propose an uncertainty-aware heuristic search called ***Uncertainty-guided Likelihood-Tree Search (ULTS)***. This method only adds a small runtime overhead relative to the likelihood/reward evaluation in scenarios where such a likelihood is obtained through a costly model evaluation since neither rollouts, additional expensive models, nor elaborate Bayesian techniques are present in ULTS. Nevertheless, in experiments involving large language models, we show how ULTS can generally achieve a favorable cost-performance tradeoff compared to standard baselines. In sum:

i. We propose ULTS: a probabilistic heuristic search for non-myopic sequential decision-making with step-wise likelihood rewards. We leverage samples from a prior over the reward and show how to easily sample from the implied posteriors over future nodes' values. These samples are useful to efficiently expand the search tree.

ii. We demonstrate the efficiency and extensibility of ULTS in on- and off-model experiments. In the latter, we assume the likelihood of the paths is obtained from querying large language models.

iii. We open-source a Python implementation of ULTS, which is compatible with `transformers` (Wolf et al., 2020) for large language model applications (see Fig. 6.1).

6.2. Setting

Formally, our setting can be viewed as a specific instance of a MDP, characterized by a tree-structured state space with deterministic transitions and a constraint on the also deterministic reward function. Let $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ be a MDP,* where $\mathcal{A} = \{a_1, \dots, a_B\}$ is a finite set of actions and $\mathcal{S} = \bigcup_{d=0}^{D} \mathcal{A}^d$ the set of states containing all sequences of actions up to length D, the function $P(s' \mid s, a)$ given by

$$P(s' \mid s, a) = \begin{cases} 1 & \text{if } s' = \text{concat}(s, a) \\ 0 & \text{otherwise,} \end{cases}$$

is the transition probability, and $R(s, a)$ is the reward function, giving the immediate reward received for taking action a in state s. We assume that at each non-terminal state s_i, there is a Categorical distribution $\boldsymbol{c}_i : \mathcal{A} \to [0, 1]$ available that models the *myopic* probability for the next action to be the best choice. We have the constraint $\boldsymbol{c}_i(a) \geq 0$ and $\sum_{a \in \mathcal{A}} \boldsymbol{c}_i(a) = 1$. In order to obtain the usual additive structure of the rewards, we optimize in log space, i.e. $R(s_i, a) = \log \boldsymbol{c}_i(a)$. Thus, the R in this setting can be interpreted as (log) likelihood, and we call the search space a *likelihood tree*.

The goal is to find a path $s_0 \to s_D := (s_0, ..., s_D)$ from the root node s_0 to a terminal state $s_D \in \mathcal{A}^D$ that maximizes the cumulative rewards $c_{s_0 \to s_D} := \exp \sum_{d=1}^{D} R(s_d, a_d)$, i.e. the (log) *likelihood* of the path. Specifically, we assume the setting where evaluating $R(s, a)$ is *expensive*, and the depth D and branching factor B of the tree are *large*.

Lastly, we define recursively the optimal value v_i of a node x_i as the sum of the logarithm of the transition probabilities $c_{s_0 \to s_i}$ from the root node s_0 to s_i and a remaining term which we refer to as $\log \Delta_i$, i.e., we have $v_{s_i} = \log c_{s_0 \to s_i} + \log \Delta_i$.[1] Intuitively, the term Δ_i, quantifies the likelihood that we get in the remaining steps from s_i to a leaf node when we take all remaining decisions optimally. It can be defined by the following recurrence relation:

$$\Delta_i = \begin{cases} 1 & \text{if } s_i \text{ is a leaf,} \\ \max\limits_{s_j \in \text{children}(s_i)} \{\boldsymbol{c}_i(a_j) \cdot \Delta_j\} & \text{otherwise.} \end{cases} \tag{6.1}$$

1: This notation slightly deviates from the standard reinforcement learning definition of an optimal value, which typically excludes past rewards. However, our definition aligns with standard practice again once we interpret the reward as being payed off only upon reaching a terminal state.

Example: Decoding Large Language Models Here, the set of actions $\mathcal{A}$ is a vocabulary consisting of (natural) language tokens. The root of the search tree is given by the context, e.g., a question in a question-answering system. The reward function R is determined by a forward pass through an LLM, i.e. a large-scale neural network that utilizes the attention mechanism (Vaswani et al., 2017) to predict the probability of the next token appearing in a sequence. This process is done recursively until termination; either when a specified depth D has been reached or when a special token "⟨EOS⟩" is selected. Considering the vocabulary size B ranges from around 32k to 256k (Radford et al., 2019; Chowdhery et al., 2023) and the reward evaluation amounts to a costly forward

* We assume a discount factor $\gamma = 1$ for simplicity, but the discussion can easily be extended to include other values as well.

pass, even when the sequence length D is small, computing the optimal sequence using standard tree-search algorithms like MCTS is infeasible. Thus, in practice, one is limited to using heuristic algorithms that are close to greedy, e.g. beam search and its variants (Vijayakumar et al., 2018; Meister et al., 2021; Freitag and Al-Onaizan, 2017).

Beyond LLM decoding, several MDPs fit within our problem formulation, where stepwise likelihood serves as a reward. In general, ULTS is applicable to any autoregressive model, not just in the language domain, but for example also in chemistry tasks (Christofidellis et al., 2023; Gao et al., 2024). Further specific example applications include optimizing process parameters for machines (such as paper drying machines in (Chen et al., 2025)), feature selection (Luo et al., 2019), various neural combinatorial optimization problems by decoding pointer networks (Vinyals et al., 2015), structure learning tasks (Kok and Domingos, 2005), and finding Maximum A Posteriori sequences as in (Pellom and Hansen, 2002).

6.3. Method

Here, we introduce **Uncertainty-guided Likelihood-Tree Search (ULTS)** which places a prior belief over the log-likelihood reward function and computes the implied posterior samples over nodes' values. We introduce and discuss the modeling assumptions in Section 6.3.1 and show how to derive approximate beliefs over the optimal values in the search tree in Section 6.3.1. The samples of the posterior beliefs are then used to make decisions about which subtree to expand next and when to stop the search (Section 6.3.2).

6.3.1. Probabilistic model

Prior beliefs over immediate rewards

A straightforward belief one can consider is the Dirichlet distribution. For tractability, we assume that the probabilities are iid. draws from a symmetric Dirichlet distribution with parameter $\alpha > 0$, i.e., $p(\boldsymbol{c}_s) = \mathrm{Dir}(\alpha)$. Thus, α controls how concentrated the sampled probability vectors are. For small α, one would typically strongly favor a few children, whereas for large α the discrete distribution would closely resemble a uniform distribution over the children. The symmetry of the prior implies in our context that we do not have a preference for particular children *a priori*, but it is also possible to replace the symmetric prior with an asymmetric one with no additional costs.

In settings where the Categorical distributions are predicted by a trained sequence model f, an alternative option is to define an empirical prior over $\boldsymbol{c}_i$ based on predictions $\boldsymbol{c}_i = f(s_i)$ on a subset of the training/validation data. Let $\{s_{0,n}\}_{n=1}^N$ be N samples of root nodes from this subset. We can obtain a set $\{s_{0,n}, s_{1,n}, \dots, s_{D,n}\}_{n=1}^N$ of D-step completions of $s_{0,n}$'s with f (e.g. through a greedy search). We can then collect samples of the Categorical distributions from this generation process: $\mathcal{C} = \{\boldsymbol{c}_{0,n}, \dots, \boldsymbol{c}_{D-1,n}\}_{n=1}^N$. Then, instead of

sampling from $\mathrm{Dir}(\alpha)$, we can sample from $p(\boldsymbol{c}_i) = \mathrm{Unif}(\mathcal{C})$. This prior is more flexible than the Dirichlet prior since no symmetry nor unimodality assumptions are made at the cost of more overhead. In any case, all these priors are *precomputed*, so they incur a fixed $\mathcal{O}(1)$ cost.

Prior beliefs over values

Having picked a prior for the categorical distributions, we can derive the implied priors over the optimal values v_{s_i} for each node s_i in the tree (Hennig et al., 2010). As introduced in Section 6.2, we write the optimal value of a node as $v_{s_i} = \log c_{s_0 \to s_i} + \log \Delta_i$. Due to the iid. assumption above (Section 6.3.1), we have the joint distribution $p(c_{s_0 \to s_i}, \Delta_i) = p(c_{s_0 \to s_i})\, p(\Delta_i)$. While all these quantities are not analytically available, we shall show that sampling from the *posterior* belief $p(v_{s_i} \mid c_{s_0 \to s_i})$ is easy if we are able to sample from $p(\Delta_i)$. Let us, therefore, derive an approximate sampling scheme for $p(\Delta_i)$. It follows Hennig et al. (2010); Grosse et al. (2021) who used Gaussian priors. We recursively approximate the prior distribution of the Δ_i's at level l with Beta distributions $\mathrm{Beta}_l(\Delta_i)$. In a bottom-up approach, we generate N samples $\{\max_j \boldsymbol{c}_n(a_j) \mid \boldsymbol{c}_n \sim p(\boldsymbol{c})\}_{n=1}^N$ for a Δ_i starting at level $l = D - 1$. Using these samples, we empirically fit the parameters of the Beta distribution $\mathcal{B}_{D-1}(\Delta_i)$ via maximum likelihood. The distributions of the Δ_i are the same for all nodes on the same level due to the iid. assumption, so this has to be done only once. Note that we need this approximation since the distribution of the maximum $\max_j \boldsymbol{c}_i(a_j)$ has no known analytic solution. We then continue by recursively sampling sets of the form (one per level) $\{\max_j \boldsymbol{c}_n(a_j) \cdot \Delta_j \mid \boldsymbol{c}_n \sim p(\boldsymbol{c}), \Delta_j \sim \mathcal{B}_{l+1}(\Delta)\}_{n=1}^N$ for a Δ_i of the level l and using it to fit the parameters of $\mathcal{B}_l(\Delta_i)$. The time complexity is $\mathcal{O}(DBN)$ for computing the approximations, i.e., it is linear in the depth and width of the tree. We emphasize that they can be pre-computed before the search and reused across different decoding runs. Alg. 6 shows the pseudocode.

Posterior beliefs on frontier nodes

Notice that whenever a new node s_i is added to the search tree, the likelihood associated with the path from the root to that node $c_{s_0 \to s_i}$ is fully observed—its distribution is simply a Dirac delta. This means the joint distribution becomes $p(\boldsymbol{c}_{s_0 \to s_i}, \Delta_i) = \delta(c_{s_0 \to s_i})\, p(\Delta_i)$. Therefore, to sample v_{s_i} given that we have observed $c_{s_0 \to s_i}$—i.e., sampling from the posterior $p(v_{s_i} \mid c_{s_0 \to s_i})$—it is sufficient to sample from $p(\Delta_i)$ and then simply scale all samples by $c_{s_0 \to s_i}$. During the search, these samples are backed up the tree and leveraged to make decisions in exploring the tree. Notice that these samples are cheap to get, unlike empirical samples based on simulations/rollouts.

6.3.2. Algorithm

At each iteration, we use the posterior samples by following the steps of Monte Carlo tree search, but without the expensive rollout step. Pseudocode in Appendix D.4.

Algorithm 6: Prior belief over Δ

```
1  BUILD-PRIOR(D, B, p(c), N)
       // Input: Depth of the tree D, branching size of tree B,
          prior p(c) over categorical rewards, number of samples N
       // Output: Table of parameters params_l for Beta
          distributions B_l for each level l of the tree
2      for l ← D − 1 to 1 do
3          for n ← 1 to N do
4              Sample c_n ~ p(c);
5              for j ← 1 to B do
6                  if l = D − 1 then
7                      Δ_nj ← 1;
8                  else
9                      Sample Δ_nj ~ B_{l+1}(Δ | params_{l+1});
10             Δ_n ← max_{j=1,...,B} (c_nj · Δ_nj);
11         params_l ← beta-MLE({Δ_n}_{n=1}^N);
```

Step 1: Selection Starting from the root node s_0, we recursively pick a child based on an acquisition function until an unexpanded node is selected. From a decision-theoretic perspective, this is done by choosing a utility function that encodes our preferences about the outcome (e.g. high likelihood) and then integrating out the unknown variables influencing this outcome. Let $\boldsymbol{v}_i$ contain the (unknown) optimal values of the descendants of s_i and let $u(s_c, \boldsymbol{v}_i)$ be a utility function. The idea is to select the child node s_c that maximizes the expected utility:

$$\int u(s_c, \boldsymbol{v}_i)\, p(\boldsymbol{v}_i \mid c_{v_{s_0 \to s_i}})\, d\boldsymbol{v}_i, \tag{6.2}$$

For ULTS, we use an utility function that encodes our preference for finding a descendant with high optimal value: $u(s_c, \boldsymbol{v}_i) := \mathbb{I}[v_{\hat{s}_c} = \max_{s_j \in \text{children}(s_i)} v_{\hat{s}_j}]$. Different realizations of $\hat{s}_c$ and $\hat{s}_j$ can be used: The most straightforward is to use the children's optimal values themselves. This corresponds to simply setting $\hat{s}_j = s_j$ and $\hat{s}_c = s_c$.

Another strategy is to use the optimal values of each child's best descendant—intuitively, it ignores other paths in the tree that might have higher values, leading to a more pessimistic belief. This can be beneficial to bias the search towards more exploration (for further discussion, see Appendix D.3.6). In this case, $\hat{s}_j$ is defined as (similarly for $\hat{s}_c$):

$$\hat{s}_j = \begin{cases} s_j & \text{if } s_j \text{ is a leaf,} \\ \underset{s_c \in \text{children}(s_j)}{\arg\max}\ a_j(s_c) & \text{otherwise.} \end{cases} \tag{6.3}$$

Note that they have the same costs since beliefs over the optimal values for both strategies are readily available due to the BACKUP step below, i.e., we do not actually perform the

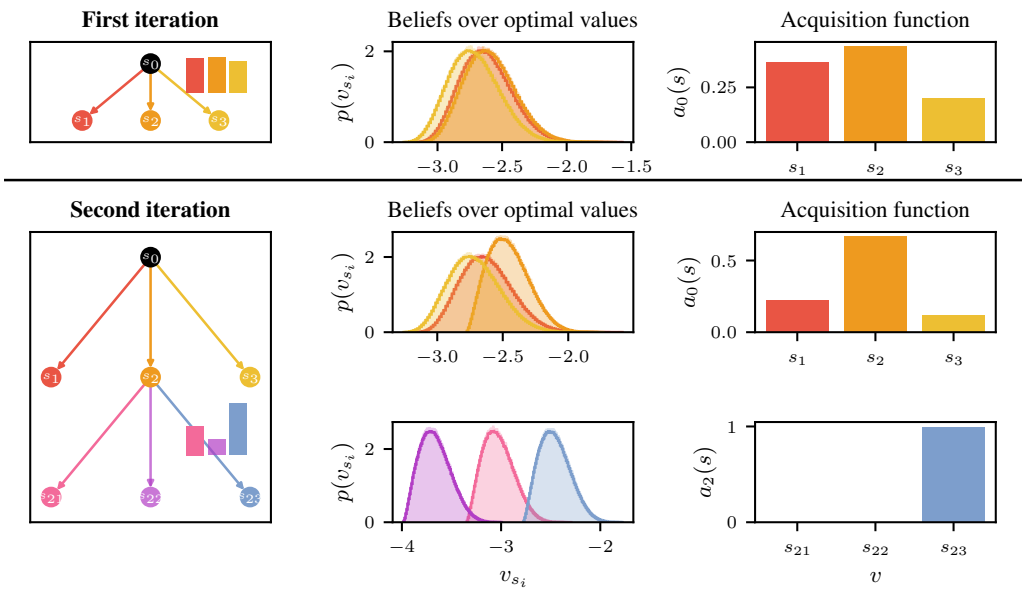


Figure 6.2.: An example of two iterations with ULTS. The upper row show the observed categorical distirbution (left), the implied posterior over the optimal values in log space (center) and the resulting acquisition function over the children in the first level of the tree (right). The lower two rows show the corresponding quantities for the first and second level of the tree after the second iteration.

recursion (6.3) in this step.

The acquisition function is then derived as a sample-based approximation to the expectation in (6.2):

$$a_i(s_c) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\Big[v_{\hat{s}_c,n} = \max_{s_j \in \text{children}(s_i)} v_{\hat{s}_j,n}\Big]. \tag{6.4}$$

Note that we can replace the above utility function with *any* other utility function and then use the posterior samples to derive the acquisition function (Wilson et al., 2018). For example, we can use a utility function that encodes a preference for sequences with fewer repetition, see Appendix D.3.8.

Step 2: Expansion Given an unexpanded node s_i, we evaluate its children's rewards. These newly obtained rewards are new observations that can be combined with the prior samples to obtain N posterior samples $\{v_{s_c,n}\}_{n=1}^{N}$ of each child s_c. In turn, they will affect the acquisition function in the next iteration.

This step is often the most expensive part of any tree-search algorithm since we assume costly evaluation of the rewards. Fewer node expansions is thus preferable.

Step 3: Backup We recursively propagate the newly obtained posterior samples $\{v_{s_c,n}\}_{n=1}^{N}$ back up the tree until the root via the path selected in the previous steps. This is done to update the posterior samples contained in each node of the path. They will then influence the selection process in the next iteration, updating the exploration-exploitation tradeoff.

Different update strategies can be used depending on the choice of the acquisition function (6.4). When posterior samples of the children node s_c are used to compute $a_i(s_c)$, then we propagate up the posterior samples of the newly expanded nodes as in when computing the prior by recursively taking their maximum, i.e. the n-th sample for the optimal value of parent node s_p is given by the maximum over the n-th sample of the children's optimal

values:

$$v_{s_p,n} = \max_{s_c \in \text{children}(s_p)} v_{s_c,n}. \tag{6.5}$$

When the posterior samples of the best descendant of s_c are used in $a_i(s_c)$, we simply propagate up the posterior samples of the best child among the newly expanded nodes, without taking further maximums along the path:

$$v_{s_p,n} = v_{s_c*,n}, \quad \text{where } s_c^* = \arg\max_{s_c \in \text{children}(s_p)} a_p(s_c). \tag{6.6}$$

Figure 6.2 shows examples for the first two iterations of ULTS under the former acquisition function and backup scheme. See Appendix D.3.6 for the corresponding illustration for the latter.

Step 4: Termination The posterior samples over the optimal values can not only be used for the selection of new nodes but also to monitor the progress of the optimization. For instance, one can compute the following empirical probability $\hat{\mathbb{P}}(c^* < v_{s_0}) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\left[c^* \leq v_{s_0,n}\right]$, which corresponds to the probability of the current best likelihood c^* among all leaves ULTS has visited so far is lower than the best value v_{s_0} at the root, according to our posterior belief.

Note that the computation of such a probability is done as in the acquisition function $a(x)$, i.e., using the posterior samples of the root node s_0 or the posterior samples of the best descendant. Then, one can decide to stop the search once this probability is below some confidence level $\varepsilon > 0$.

6.3.3. Remarks

Practical considerations In order to put a tractable upper bound on the runtime of ULTS, we introduce a hyperparameter $k_{\max}$ on the maximum number of nodes that can be expanded per level, similar to beam search. Moreover, we stop the search as soon as a set of $k_{\max}$ leaves is attained or the termination probability exceeds below $1 - \varepsilon$. We shall see in Section 6.5 that even under these further constraints, we still obtain good results while being efficient.

Limitations In this work we focus on the uncertainty arising from the exponentially growing search space under limited compute resources. However, there might be other sources of uncertainty, e.g. uncertainty about the likelihood/reward function R. In LLM decoding, there is evidence that higher likelihood is not always correlated with human preferences (Holtzman et al., 2019; Wiher et al., 2022; Stahlberg and Byrne, 2019; Eikema and Aziz, 2020; Zhang et al., 2020). Appendix D.3 contains additional evaluations of ULTS in terms of other metrics than the likelihood, where there is indeed no perfect correlation. There is an ongoing line of research on how to counteract miscalibration in LLMs that is complementary to our work Yang et al. (2023); Kapoor et al. (2024); Onal et al. (2024); Tonolini et al. (2024), but combining and evaluating such approaches with ULTS goes beyond the scope of this paper. Moreover, the iid. prior assumption might not

fully hold, but in our inference scheme it is essential to obtain tractability. This is akin to how priors in Bayesian neural networks (Wilson and Izmailov, 2020) are usually chosen, which are just iid. Gaussians.

6.4. Related Work

Probabilistic tree search methods have been explored for various applications. Hennig et al. (2010); Tesauro et al. (2012) introduced a method for game trees like Go, using a Gaussian process prior and expectation propagation (Minka, 2001b) to model the beliefs. Grosse et al. (2021) extended this to directed acyclic graphs. Mern et al. (2021) also used Gaussian processes for general planning problems. Unlike these methods, we focus on trees with a Categorical likelihood reward function. Additionally, the above methods rely on MCTS-based roll-outs, which do not scale efficiently. ULTS avoids costly rollout statistics by leveraging precomputed belief samples for decision-making.

Our approach employs a best-first search strategy, akin to $A^\star$ (Hart et al., 1968a). An $A^\star$-like beam search algorithm, under the name of best-first beam search, has also been studied in Meister et al. (2020a). Unlike the non-probabilistic $A^\star$ and best-first beam search, we approach optimization-on-tree problems via the lens of decision-making under uncertainty—putting a prior belief about the unknown, updating it based on the observations, and making decision based on the posterior belief.

MCTS is a staple uncertainty-aware search method and it has also been proposed for various LLM applications (Leblond et al., 2021; Liu et al., 2023; Hao et al., 2023; Feng et al., 2023; Zhang et al., 2023; Zhou et al., 2023; DeLorenzo et al., 2024). However, unlike ULTS, MCTS is non-probabilistic and generally costly since it requires rollouts. Moreover, it is often used for cases where rewards are only observable at leaf nodes (Zhang et al., 2023, etc.) or when external value estimators is available (Feng et al., 2023, etc.). The latter introduces a substantial overhead in evaluating a value network, which is often another large-scale model. ULTS, meanwhile, uses a simple model which introduces negligible overhead (see Figure D.4).

Minimum Bayes risk methods (Kumar and Byrne, 2004; Eikema and Aziz, 2021; Bertsch et al., 2023) also rely on the maximization of an expected utility function. However, they do not employ probabilistic models and do not aim to make decisions in a sequential manner. Rather, they generate a set of full-sequence samples and make a decision on which sequence in the set to be selected.

6.5. Experiments

In this section, we evaluate ULTS in both on-model and off-model problems. In the former, we assume that the rewards are sampled from a Dirichlet distribution. In the latter, we use black-box LLMs as reward functions. Unless specified otherwise, we use the acquisition function in Equation 6.3 for the selection and backup steps. See Appendix D.3

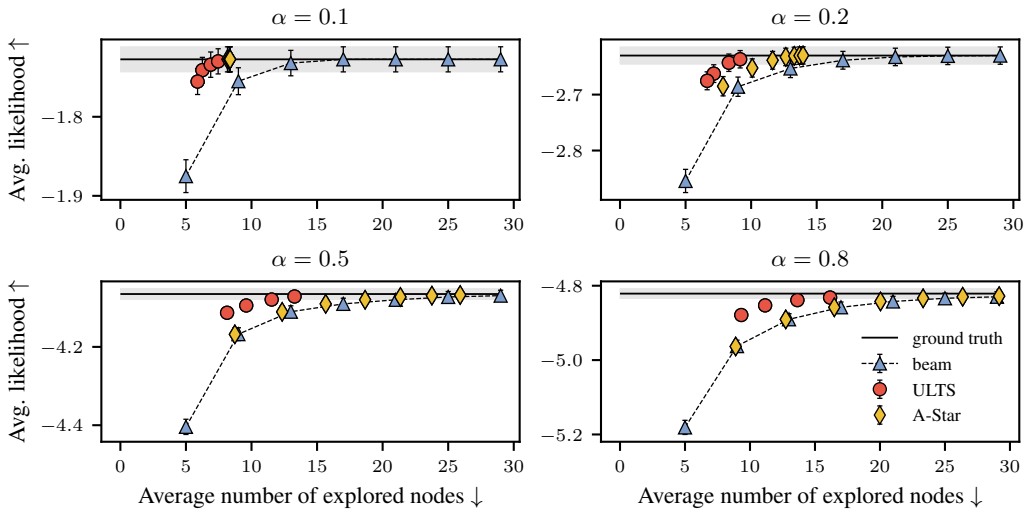


Figure 6.3.: Comparison of ULTS with beam search and A* on synthetic rewards sampled from a Dirichlet distributions with varying α. ULTS finds the optimum efficiently. Errorbars indicate one times the standard error of the mean.

for results with the other strategy and further dataset/setting. All experiments are done on a single NVIDIA GeForce RTX 2080 Ti and NVIDIA A40 48GB GPUs for GPT-2 and Llama-2-7b, respectively. Errorbars reported in the plots and tables indicate the standard error of the mean as calculated with `scipy.stats.sem`.

6.5.1. On-model experiments

We compare ULTS to beam search on artificially generated search problems from Dirichlet priors. The trees have branching factor $B = 8$ and depth $D = 5$. Since these trees are so small we optimized the acquisition function in (6.4) over the entire frontier, i.e. non-recursively with $k_{\max} = \infty$ like standard A*. The transition probabilities at each node are sampled from a Dirichlet prior with fixed $\alpha \in \{0.1, 0.2, 0.5, 0.8\}$. The comparison is on-model, i.e., ULTS is run with the ground truth parameter of α. We repeat the experiment with different values for the confidence parameter ε of ULTS from $\{0.05, 0.1, 0.3, 0.5\}$. Beam search is run with beam sizes ranging from 1 to 7. Since the Δ-Terms of ULTS play the same role as the heuristic terms in A*, we also add A* with uninformative, optimistic heuristic (i.e. estimating the probability of the remaining sequence parts with 1) for an additional layer of ablation. The results in Figure 6.3 show that ULTS dominates across the entire range of hyper-parameters. This suggests that knowledge about concentration strength helps reduce the number of search steps.

6.5.2. Off-model experiments with LLMs

Setup We set ULTS' $k_{\max} \in \{2, 3, 4, 5, 10, 20\}$ and set ε to a default value of 0.1. The error bars indicate ± 1 standard error of the mean based on all test sentences. We compare ULTS against beam search, as well as other recent baselines commonly used for LLM decoding: multinomial beam search (Multinomial BS; Kool et al., 2019), contrastive search (Su et al., 2022) nucleus sampling (Holtzman et al., 2019), best-of-k sampling (Stiennon et al., 2020), speculative decoding (Leviathan et al., 2023), and DoLA (Chuang

et al., 2024). For beam search, multinomial beam search, and best-of-k baselines, we evaluate beam sizes and numbers of samples $k \in \{1, 2, 3, 4, 5, 10, 20\}$, respectively. We set any other hyperparameters of all baselines as suggested by the original papers or by the `transformers` library. We use Huggingface's implementation of the baselines (except for A* which is not part of Huggingface's repertoire). We use GPT-2 (Radford et al., 2019) and Llama-2-7b (Touvron et al., 2023) for text generation on articles from the Wikipedia (See et al., 2017), CNN Daily Mail (Hermann et al., 2015), and Reddit TL;DR (Völske et al., 2017) datasets.[†] Since many of the text samples in the Wikipedia dataset end with e.g., references instead of full sentences, we filter for text samples with at least 500 tokens, resulting in a test set with 332 token sequences (out of a random subset of originally 1000 sequences). We use 200 tokens as input and predict 20 tokens. We do the same for the CNN Daily Mail dataset, where we end up with 790 token sequences. We use a context length of 300 and generate 60 tokens. We also include a summarization task, where the goal is to generate a 40-token long summary of the input sequence. For this, we use 1000 random samples from the TL;DR dataset with variable-length contexts. We run ULTS with both the Dirichlet prior with $\alpha = 10^{-4}$ and the empirical prior. For more details regarding the choice of prior, see Appendix D.2.2.

Results Figure 6.4 on page 85 shows the results. Baselines that are underperforming are shown instead in Figure D.9 in the Appendix. No matter the choice of $k_{\max}$ (and k), ULTS yields sequences with the same or a higher log-likelihood while expanding fewer nodes. This is the case for both choices of priors, with the empirical prior encouraging exploration more. The histogram in Figure D.1 (Appendix D.2.2) suggests that the distribution $\max_i c_{ji}$ is bimodal and not fully captured by the Dirichlet prior. In particular, our choice of Dirichlet prior is slightly too pessimistic. As a result, the search may stop too early and the available budget may not be fully utilized. However, the budget that *is* used, is used efficiently. Our recommendation is thus as follows. When efficiency is the main goal, a Dirichlet prior with low concentration is preferable—it performs similarly to beam search with smaller beam widths, while being more efficient. If the search performance is important and an additional tree exploration can be afforded (still more efficient than beam search), then the empirical prior is the best choice—it is also hyperparameter-free. Despite expanding fewer nodes than beam search, ULTS is currently slower in settings where different nodes expansion in beam search can be batched. However, note that batching is not always possible, e.g. in memory-constrained settings (the memory resources depend on the model size, sequence length, as well as batch size). For more details on the runtime see Appendix D.3.5.

6.6. Conclusion

We have discussed ULTS, a probabilistic decision-making algorithm for efficiently finding high-likelihood paths on large, expensive search trees, such as those induced by LLMs. Our method quantifies and leverages computational uncertainty over node values and exploits the structure of log-likelihood rewards by placing a prior over them, using posterior

[†] Additional results on machine translation tasks can be found in Appendix D.3.9.

samples to guide non-myopic exploration-exploitation. ULTS combines advantages of heuristic methods like beam search, Monte Carlo tree search, and Bayesian tree search, while crucially supporting backtracking without requiring rollouts, additional models, or complex inference. It efficiently finds high-likelihood sequences while expanding few nodes and maintaining low latency.

Future work One can explore batched acquisition strategies, similar to batch Bayesian optimization (González et al., 2016; Wu and Frazier, 2016), or incorporate uncertainty over rewards to account for miscalibration. Finally, our connection between probabilistic inference and LLMs paves the way for probabilistic reasoning in the tree of thoughts framework (Yao et al., 2024).

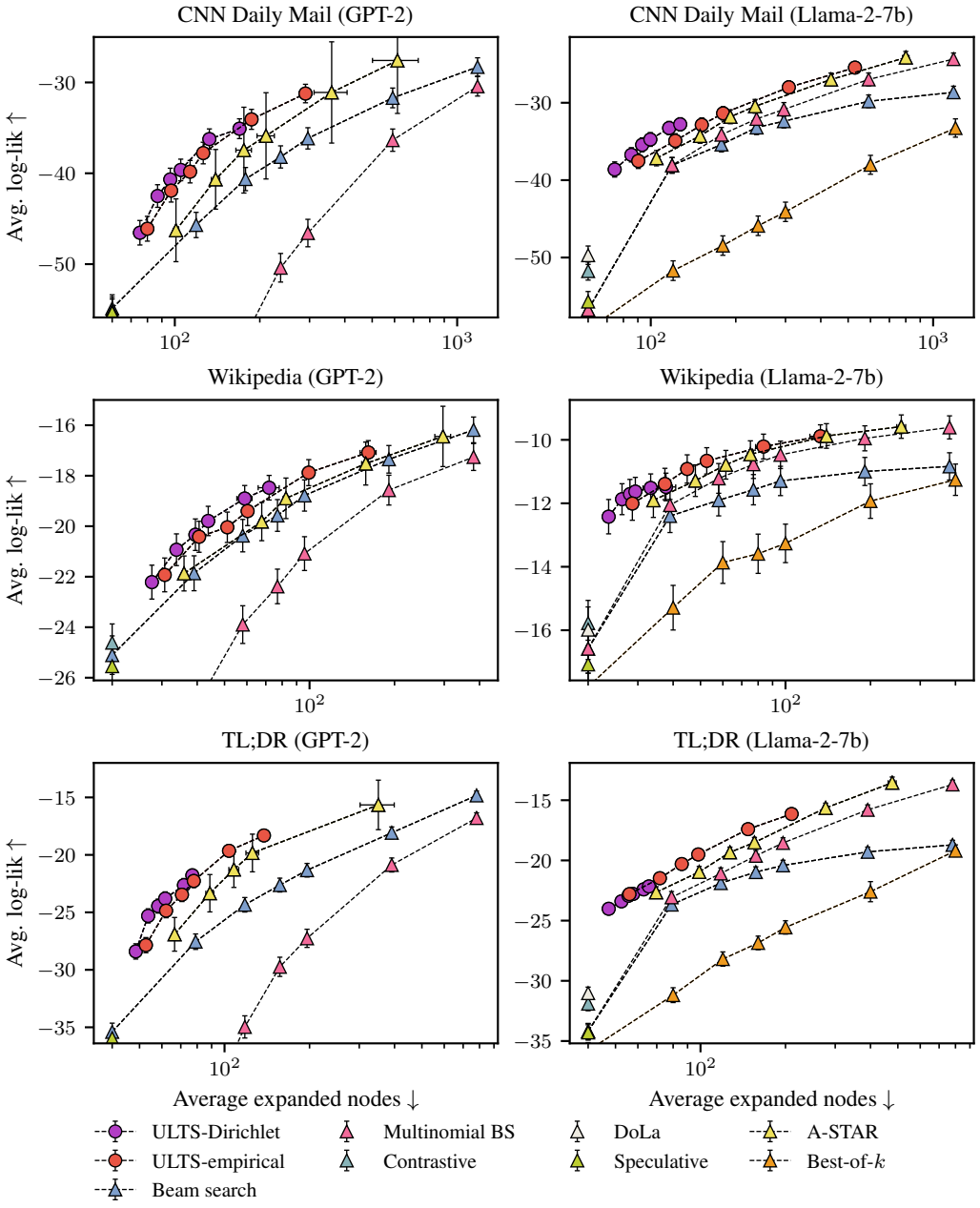


Figure 6.4.: Decoding experiments with Llama-2-7b and GPT-2 for text generation on CNN Daily Mail and Wikipedia articles and text summarization for the TL;DR dataset. The methods are evaluated for different computational budgets, i.e. different values of k and k_{max}. ULTS dominates the baselines in low-budget settings. Errorbars indicate two times the standard error of the mean.

Chapter 7

A Greedy Approximation for k-Determinantal Point Processes

7.1. Introduction

DPPs, introduced by Macchi (1975), are point processes whose joint intensity is proportional to the determinant of a positive definite kernel Gram matrix. Intuitively, this introduces a dependence between points in samples from such processes that gives them a repulsive property—points drawn from DPPs cover a space more regularly than uniform random samples; see Figure 7.1 (left vs. middle).

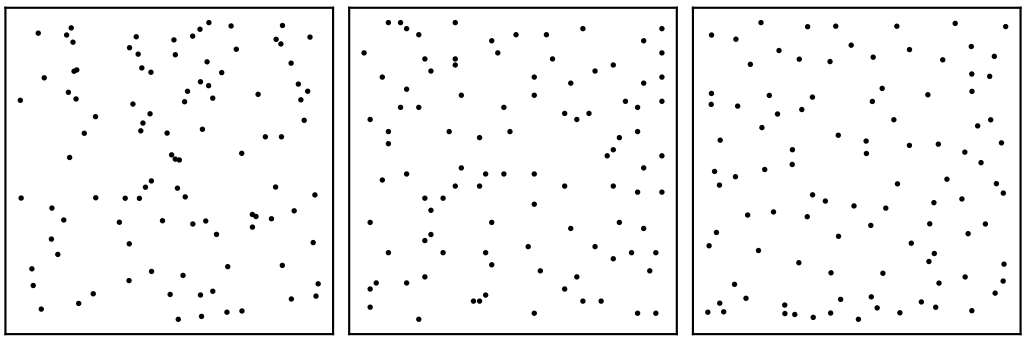


Figure 7.1.: 100 random points on the interval $[0, 1]^2$ sampled uniformly at random **(left)**, sampled from a k-DPP with square-exponential kernel with a length-scale of 0.1 **(middle)**, and sampled from its greedy approximation that is the subject of this paper **(right)**.

DPPs initially arose in the study of fermionic gases in physics and have since found application in other areas, such as random matrix theory (Mehta, 1991). A review of their statistical properties is provided by Soshnikov (2000). Meanwhile, DPPs have also received attention in machine learning and statistics. A reason for the growing interest in them is that they provide an elegant theoretical view on the notion of *exploration* that is of relevance across machine learning. In areas like active and reinforcement learning, as well as in numerical tasks like marginalization in graphical models, the basic challenge is that the algorithm should in some sense "probe" an input domain in a maximally informative way. The repulsive property of DPPs can help automatically guide such a procedure.

As we will review below, DPPs have a direct connection to the entropy of Gaussian process models, which closely ties them to many basic probabilistic algorithms in machine learning, e.g. in Bayesian optimization (Garnett, 2023; Nava et al., 2022; Kathuria et al., 2016; Wang et al., 2017), Bayesian quadrature (Bardenet and Hardy, 2016), kernel quadrature (Belhadji et al., 2019) and Monte Carlo integration (Gautier et al., 2019a). DPPs have been used as diversity-inducing priors (e.g., Kulesza and Taskar, 2012) and found applications in several other areas, such as recommender systems (Wilhelm et al., 2018), clustering (Kang, 2013), neural network compression (Mariet and Sra, 2015), batch stochastic gradient

descent (Zhang et al., 2017) or learning diverse generative models (Elfeki et al., 2019). In these applications, the number of points desired k is usually fixed by the users in advance, whereas in a DPP, this size is a random variable. To this end, we may consider a so-called k-DPP, which is a DPP conditioned to have fixed size. In machine learning, the process of sampling from a k-DPP is often used as a subroutine rather than the end goal in itself. Therefore, it can then be desirable to trade-off accuracy in the sampling process for speed. This is the motivation for our contributions here.

After reviewing the definition of a k-DPP in Section 7.2, we start by characterizing the k-DPP as a point processes maximizing a natural score function. This score function consists of two components: One that ensures that the elements of a single point set sampled from the process are expected to be diverse. And another one that rewards a high entropy of the point processes itself and thereby ensures that multiple point sets sampled from the process are expected to be diverse themselves.

We then introduce the greedy strategy for approximate maximization of this score function. Our motivation to choose a greedy approach for the approximation is due to its success in the analogous non-stochastic setting. It is common practice in tasks that involve the (non-stochastic) exploration of a function or a domain, e.g. by maximizing information gain (Srinivas et al., 2009; Hennig and Schuler, 2012; Ma et al., 2018) or entropy (Sharma et al., 2015). Hence, we suspect it to be useful for our stochastic *sampling* setting, too. We continue with a theoretical analysis of the approximation quality, where we use tools from submodular optimization to show that the point process defined by our greedy sampling procedure achieves a near-optimal value of the introduced score function.

While a large amount of computational costs is saved by sampling greedily, the implementation costs of a single greedy decision can still be significant over large domains or continuous domains. However, we will show that the kernels typically encountered in machine learning are amenable to an efficient implementation. In Section 7.5, we present an analytic sampling scheme, by way of example on the popular square-exponential kernel, which we generalize to other common kernels, including the Matérn class. We conclude with an empirical study of the approximation quality in an applied setting in Section 7.6 and a runtime comparison to exact and MCMC sampling algorithms in Section 7.7.

7.2. Determinantal Point Processes

Let $\ell: \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ be a symmetric positive semi-definite kernel over some compact Euclidean space $\mathbb{X}$. Given two sets $\mathrm{A} := [a_1, \dots, a_I], \mathrm{B} := [b_1, \dots, b_J] \subseteq \mathbb{X}$, the symbol $\mathbf{L}_{\mathrm{AB}} \in \mathbb{R}^{I \times J}$ is a matrix containing the elements $[\mathbf{L}_{\mathrm{AB}}]_{ij} = \ell(a_i, b_j)$. For our purposes, a k-DPP is a stochastic process, such that a sample of cardinality k, $\mathrm{X} := [x_1, \dots, x_k] \subset \mathbb{X}$ from the process has joint probability proportional to the determinant of the corresponding Gram matrix $\mathbf{L}_{\mathrm{XX}}$:

$$p_{k\text{-DPP}}(\mathbf{X} = \mathrm{X}) = Z \det \mathbf{L}_{\mathrm{XX}}. \tag{7.1}$$

Here, Z is a normalization constant, the existance of which can be shown via a general argument (Hough et al., 2006; Kulesza and Taskar, 2011). More precise definitions of DPPs and k-DPPs can be found in (Soshnikov, 2000; Hough et al., 2009; Bardenet and Hardy, 2016), and Kulesza and Taskar (2011). They require a discussion of base measures and other properties of point processes, which unnecessarily complicate the exposition in our context. A short summary of basic properties of DPPs is given in Appendix A.6. Kulesza and Taskar (2012) also provide a relatively complete introduction to discrete DPPs, where $\mathbb{X}$ is restricted to be a discrete space. In this work, we consider a finite discretization of a continuous space. In order to simplify the exposition, we assume that $\mathbb{X}$ is a unit cube $[0,1]^D$ discretized into an equally spaced grid of size N. For a more general box constraint $\tilde{x}_d \in [a_d, b_d]$ for each $d = 1, \dots, D$, one can apply the linear transformations $x_d = {}^{(\tilde{x}_d - a_d)}/_{(b_d - a_d)}$.

7.3. Greedy Approximation

We begin by outlining a connection between the k-DPP and the softargmax of the Gaussian differential entropy in Section 7.3. This relation serves as the motivation behind our choice of the score function. Then, we greedily maximize this score function, resulting in our greedy approximation of the k-DPP in Section 7.3. Finally, we examine the theoretical properties of the approximation in Section 7.4.

Motivation

Consider an algorithm aiming to learn the function $f\colon \mathbb{X} \to \mathbb{R}$ by choosing k evaluation points ("designs") $\mathrm{X}_{1:k}$, using a Gaussian process prior $p(f) = \mathcal{GP}(\mu, \ell)$ with arbitrary mean function $\mu\colon \mathbb{X} \to \mathbb{R}$ and kernel function ℓ as above. The slicing notation $\mathrm{X}_{i:j}$ denotes the elements selected in steps $i, \dots, j$ (for $j < i$, $\mathrm{X}_{i:j} = \varnothing$). We allow for a stochastic and sequential policy π, defined over a product probability space $(\mathbb{X}^k, 2^{\mathbb{X}^k}, \pi)$:

$$\pi(\mathbf{X}_{1:k} = \mathrm{X}_{1:k}) = \prod_{i=1}^{k} \pi\big(\mathbf{X}_i = x_i \mid \mathbf{X}_{1:i-1} = [x_1, \dots, x_{i-1}]\big). \tag{7.2}$$

In many cases, the evaluation order does not matter, that is, one is only interested in the distribution over unordered sets of points $\mathrm{X} \in \mathcal{X} = \{\mathrm{X} \subseteq \mathbb{X} \mid |\mathrm{X}| = k\}$. Every sequential policy π induces a random variable $\mathbf{X}$ over the discrete probability space $(\mathcal{X}, 2^{\mathcal{X}}, p_\pi)$, where p_π is obtained by summing over all permutations $\mathrm{perm}(\mathrm{X})$ of the elements in a set X:

$$p_\pi(\mathbf{X} = \mathrm{X}) = \sum_{\mathrm{X}' \in \mathrm{perm}(\mathrm{X})} \pi(\mathbf{X}_{1:k} = \mathrm{X}')\,. \tag{7.3}$$

Aiming to collect informative observations, assume the algorithm may randomly place

evaluations $\mathrm{X}_{1:k}$ such that the differential entropy

$$h_{\mathrm{diff}}(f_{\mathrm{X}_{1:k}}) = \frac{1}{2}\log(2e\pi)^k \det \mathbf{L}_{\mathrm{X}_{1:k}\mathrm{X}_{1:k}} \tag{7.4}$$

of the corresponding multivariate Gaussian $f_{\mathrm{X}_{1:k}} \sim \mathcal{N}(\mu_{\mathrm{X}_{1:k}}, \mathbf{L}_{\mathrm{X}_{1:k}\mathrm{X}_{1:k}})$ has a high value. To simplify notation, we drop constants and use

$$h(\mathrm{X}_{1:k}) = \log\det \mathbf{L}_{\mathrm{X}_{1:k}\mathrm{X}_{1:k}} \tag{7.5}$$

in the following. To be more precise, we require samples to be draws from the *softargmax* of h:

$$p_\beta(x_1, \dots, x_N) = Z\exp\big(\beta \cdot h(\mathrm{X}_{1:N})\big)\,, \tag{7.6}$$

where $\beta > 0$ is a constant. A policy π for sampling from p_β maximizes the following score function

$$\pi = \arg\max_\pi \; \beta\mathbb{E}_{\mathrm{X}\sim p_\pi} h(\mathrm{X}) + \mathcal{H}(p_\pi)\,, \tag{7.7}$$

where $\mathcal{H}(p_\pi) := -\sum_{\mathrm{X}\in\mathcal{X}} p_\pi(\mathrm{X})\log p_\pi(X)$ is the Shannon entropy and $\mathbb{E}_{\mathrm{X}\sim p_\pi} h(\mathrm{X})$ is the expected value of the objective value h of the sampled subsets. To see this, consider the Kullback-Leibler divergence between p_π and p_β

$$D_{\mathrm{KL}}(p_\pi \| p_\beta) = -\mathcal{H}(p_\pi) - \beta\mathbb{E}_{\mathrm{X}\sim p_\pi} h(\mathrm{X}) + \log Z; \tag{7.8}$$

$$Z := \sum_{\mathrm{X}\in\mathcal{X}} \exp\beta h(\mathrm{X}), \tag{7.9}$$

and note that π achieves the minimum $D_{\mathrm{KL}}(p_\pi \parallel p_\beta) = 0$ per definition of optimality. For $\beta = 1$, one obtains the k-DPP associated with ℓ. This form reveals that a k-DPP is a smooth approximation to the argmax of the differential entropy in the sense that for $\beta \to \infty$, one recovers the exact argmax (and the mode of the k-DPP). The expression $H_\beta(\pi) := \beta\mathbb{E}_{\mathrm{X}\sim p_\pi} h(\mathrm{X}) + \mathcal{H}(p_\pi)$ can be interpreted as a *softmax* corresponding to the *softargmax*. Intuitively, this score function rewards high diversity within samples, as well as high diversity between samples. For active learning, the resulting samples are useful, for example, in so far as the resulting empirical estimator $\mathbb{E}_N[f]$ for expectations of f (even if f is *not* a true sample from $\mathcal{GP}(\mu, \ell)$, or even an element of the RKHS associated with ℓ) converges at a rate dominating that of the Monte Carlo estimator as shown in Bardenet and Hardy (2016).

Method

Finding the exact argmax ($\beta \to \infty$) of the entropy h is known to be NP-hard, but a greedy approximation typically shows good practical performance and is also theoretically well understood (Sharma et al., 2015). The greedy approach to finding the set of points $\mathrm{X}_{1:k}$ with the highest entropy h consists in iteratively selecting the next point x_i by maximizing the marginal gain $\Delta_h(x \mid \mathrm{X}_{1:i-1}) := h(\mathrm{X}_{1:i-1} \cup \{x\}) - h(\mathrm{X}_{1:i-1})$ in each step, i.e.

$$x_i = \operatorname*{argmax}_x \Delta_h(x \mid \mathrm{X}_{1:i-1})\,. \tag{7.10}$$

By taking the Schur complement [1] of $\mathbf{L}_{\mathrm{X}_{1:i},\mathrm{X}_{1:i}}$, one obtains

$$\det(\mathbf{L}_{\mathrm{X}_{1:i},\mathrm{X}_{1:i}}) = \det(\mathbf{L}_{\mathrm{X}_{1:i-1},\mathrm{X}_{1:i-1}}) \cdot \det(\mathbf{L}_{x_i x_i} - \mathbf{L}_{x_i \mathrm{X}_{1:i-1}} \mathbf{L}^{-1}_{\mathrm{X}_{1:i-1}\mathrm{X}_{1:i-1}} \mathbf{L}_{\mathrm{X}_{1:i-1} x_i}), \quad (7.11)$$

and thereby the marginal gain simplifies to

$$\Delta_h(x \mid \mathrm{X}_{1:i-1}) = \log\Big(\mathbf{L}_{xx} - \mathbf{L}_{x\mathrm{X}_{1:i-1}} \mathbf{L}^{-1}_{\mathrm{X}_{1:i-1}\mathrm{X}_{1:i-1}} \mathbf{L}_{\mathrm{X}_{1:i-1} x}\Big). \quad (7.12)$$

The term inside the logarithm will be known to readers experienced with Gaussian processes as the posterior variance of a Gaussian process regression model conditioned on function values at $\mathrm{X}_{1:i-1}$. We will refer to it as $\mathrm{Var}_i(x)$ and in preparation for the derivations in Section7.5, we introduce the shorthand $\mathbf{L}_{(i)} := \mathbf{L}_{\mathrm{X}_{1:i-1},\mathrm{X}_{1:i-1}}$ and re-formulate the posterior variance more explicitly as

$$\mathrm{Var}_i(x) = \ell(x,x) - \sum_{a,b=1}^{i-1} \ell(x,x_a)\,\ell(x,x_b)[\mathbf{L}^{-1}_{(i)}]_{ab}\,. \quad (7.13)$$

In analogy to greedy optimization, we define the greedy approximation to the k-DPP by sampling iteratively from the softargmax of the marginal gain $\Delta_h(x \mid \mathrm{X}_{1:i-1})$:

$$\pi_{\mathrm{greedy}}(x \,|\, \mathrm{X}_{1:i-1}) = \frac{\exp \Delta_h(x \mid \mathrm{X}_{1:i-1})}{Z_{\mathrm{X}_{1:i-1}}} \propto \mathrm{Var}_i(x). \quad (7.14)$$

$Z_{\mathrm{X}_{1:i-1}}$ denotes the normalizing constant, that depends on the previously selected points $\mathrm{X}_{1:i-1}$. Alternatively, one can view the greedy approximation to sampling from a k-DPP as the greedy approximation to the optimum of the score function $H(\pi) := \mathbb{E}_{\mathrm{X}_{1:k}\sim\pi} h(\mathrm{X}_{1:k}) + \mathcal{H}(\pi)$ since

$$\pi_{\mathrm{greedy}}(x \mid \mathrm{X}_{1:i-1}) = \operatorname*{argmax}_{\pi(x|\mathrm{X}_{1:i-1})} \mathbb{E}_{\pi(x|\mathrm{X}_{1:i-1})}\big[\Delta_h(x \mid \mathrm{X}_{1:i-1})\big] + \mathcal{H}(\mathbf{X}_i \mid \mathbf{X}_{1:i-1} = \mathrm{X}_{1:i-1}) \quad (7.15)$$

is equivalent to Equation (7.14) by the same argument based on the Kullback–Leibler divergence as in the previous section. Here, $\mathcal{H}(\mathbf{X}_i \mid \mathbf{X}_{1:i-1} = \mathrm{X}_{1:i-1})$ denotes the conditional Shannon entropy.

7.4. Approximation Guarantee

In this section we derive the approximation guarantee for the greedy optimization of the above score function H. It is based on a classic result from combinatorial optimization giving a $(1 - 1/e)$-approximation ratio between greedy and optimal maximization of monotone submodular set functions (Nemhauser et al., 1978). We extend this guarantee to a stochastic setting and show that it holds for our greedy k-DPP sampling algorithm. For technical reasons, we introduce a free parameter α during this analysis that we will later fix to a convenient value, leaving us with an approximation ratio that depends on k and the spectral properties of the kernel of the k-DPP.

1: Let $\mathbf{L}$ be a block matrix:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{bmatrix}$$

where $\mathbf{L}_{11}$ is invertible. The **Schur complement** of $\mathbf{L}_{11}$ in $\mathbf{L}$ is:

$$S = \mathbf{L}_{22} - \mathbf{L}_{21}\mathbf{L}_{11}^{-1}\mathbf{L}_{12}$$

Determinant identity: If $\mathbf{L}_{11}$ is invertible, then

$$\det(\mathbf{L}) = \det(\mathbf{L}_{11}) \cdot \det(S)$$

where S is the Schur complement of $\mathbf{L}_{11}$.

The function $h(\mathrm{X}) = \log\det \mathbf{L}_{\mathrm{XX}}$ has two helpful characteristics, as pointed out by Krause et al. (2008) and Sharma et al. (2015). It is submodular, i.e. $\forall\, \mathrm{X}_1 \subseteq \mathrm{X}_2$ and $i \notin \mathrm{X}_2$, $h(\mathrm{X}_1 \cup \{i\}) - h(\mathrm{X}_1) \geq h(\mathrm{X}_2 \cup \{i\}) - h(\mathrm{X}_2)$. If additionally, the smallest eigenvalue $\lambda_{min}(\mathbf{L}) \geq 1$, the function h is also monotone, i.e. $\forall\, \mathrm{X}_1, \mathrm{X}_2$ with $\mathrm{X}_1 \subseteq \mathrm{X}_2 \subseteq \mathrm{X}$, $h(\mathrm{X}_1) \leq h(\mathrm{X}_2)$. In combination, monotonicity and submodularity bound the future change in the objective value h in subsequent steps by the previous change. As we have seen in Chapter 2, Nemhauser et al. (1978) used this property to give an upper bound on the optimal objective value $h(\mathrm{O})$ based on $(1 - 1/e)^{-1}$ times the objective value $h(\mathrm{G})$ found with the greedy algorithm:

Theorem 7.4.1 (Nemhauser et al. (1978)) Given a monotone submodular function h, let G be the solution found with the greedy algorithm as defined in Equation (7.10) and O be the optimal solution. It holds:

$$(1 - 1/e)h(\mathrm{O}) \leq h(\mathrm{G})\,.$$

In our case, where we treat the k-DPP as the softargmaximum of h, we derive an analogous statement in terms of the softmaximum instead of the maximum:

Theorem 7.4.2 (Constant factor Approximation) Let h be a submodular set function with $h(\varnothing) = 0$ and $\Delta_h(x \,|\, \mathrm{X}) > (1/k)\log k!$ for all $\mathrm{X} \subset \mathbb{X}$, $x \in \mathbb{X} \setminus \mathrm{X}$. Assume $\mathbb{X}$ is finite. It holds

$$(1 - 1/e)H(p_{\pi_{\text{opt}}}) \leq H(p_{\pi_{\text{greedy}}})\,,$$

with $H(p_\pi) = \mathbb{E}_{X \sim p_\pi}[h(X)] + \mathcal{H}(p_\pi)$ and π_{opt} being an optimal policy.

A full proof is included in Appendix E.4. The idea is to first prove the desired results for the sequential, order-dependent policies, i.e. $(1 - 1/e)H(\pi_{\text{opt}}) \leq H(\pi_{\text{greedy}})$. This can be done by following the series of arguments in Nemhauser et al. (1978) with a replacement of sets by (ordered) set-valued random variables and additional care of the Shannon entropy terms. Then, we transfer the result to the final distribution over unordered sets by exploiting that the order-dependent bound holds for all order-dependent optimal policies, including the ("worst case") one with uniform distribution over all permutations.

In order for the latter step to work out, we introduce the additional requirement $\Delta_h(x \,|\, \mathrm{X}) > (1/k)\log k!$ for all $\mathrm{X} \subset \mathbb{X}$, $x \in \mathbb{X} \setminus \mathrm{X}$ regarding the slope of h. However, note that altering the slope of h can be done easily by scaling the kernel function with a constant value. To make this dependence explicit, we use $h_\alpha(\mathrm{X}) = \log\det \alpha \mathbf{L}_{\mathrm{XX}}$ instead of h for the following analysis. In optimization, as well as in the sampling case, the greedy strategy is invariant with respect to this change because the scale α cancels out when sampling proportionally to the posterior variance. The distribution of samples from a random-sized DPP, though, changes in general. In particular, increasing α increases the expected cardinality of the samples. But for k-DPPs, the scaling again does not matter. For them, we can therefore give the following approximation guarantee:

Corollary 7.4.3 (Greedy k-DPP Approximation) Running the algorithm

$$\pi_{\text{greedy}}(x \,|\, X_{1:i-1}) \propto \text{Var}_i(x)$$

as introduced in Section 7.5 for k iterations on a finite grid is a $(1 - 1/e)$ approximation to the exact distribution $p_{k\text{-DPP}}$ of the corresponding k-DPP, in the sense of

$$(1 - 1/e)H_\alpha(p_{k\text{-DPP}}) \leq H_\alpha(p_{\pi_{\text{greedy}}}),$$

with $H_\alpha(p) = \mathbb{E}_{X\sim p}[h_\alpha(X)] + \mathcal{H}(p)$, $h_\alpha(X) = \log\det(\alpha \mathbf{L}_{XX})$ and $\alpha > k!^{1/k}/\lambda_{min}$, where λ_{min} is the smallest eigenvalue of the Kernel Gram matrix over the grid.

This result follows directly from Theorem 7.4.2. For details, see Appendix E.4. By plugging in $\alpha = k!^{1/k}/\lambda_{min}$, rearranging the terms and additionally applying Stringling's approximation for $k!$, the above inequality reads

$$\begin{aligned}(1 - 1/e)H_1(p_{k\text{-DPP}}) - H_1(p_{\pi_{\text{greedy}}}) \\ \leq (1/e)k\log(\alpha) \\ \leq (1/e)\big(k\log(k) - k + \mathcal{O}(\log k) + k \cdot \log\lambda_{\text{min}}^{-1}\big)\end{aligned}$$

This form reveals that the tightness of the bound increases for a larger smallest eigenvalue $\lambda_{\text{min}}^{-1}$ and a smaller number of points k. In the case of a DPP with random k, larger eigenvalues lead to a higher expected number of sampled points. Therefore, one possible intuition for this result is that in settings in which the sample space volume is not "very tightly filled" (i.e., if k is much less than the expected number of sampled points under the DPP), the problem of placing k self-avoiding points might become "easier" and the greedy approximation can be very close to the exact algorithm. The subtle differences between greedy and exact only matter if the volume is very "packed" (note that this does *not* imply that exact and approximate methods are similar to iid. samples in such "loose" cases. They still self-avoid).

A quantity closely related to the differential entropy h is the information gain. Since it is also known to be submodular and monotone, a $(1 - 1/e)$-approximation guarantee for greedy sampling from the softargmax of the information gain holds as well. Please refer to Appendix E.4 for the corresponding statement. It is restricted to order-dependent sampling and requires a slight modification of the sampling scheme presented in Section 7.5 to take the noise term σ^2 into account. The modification does not impede the efficiency of the method and for $\sigma \to 0$, it corresponds to the algorithm introduced above.

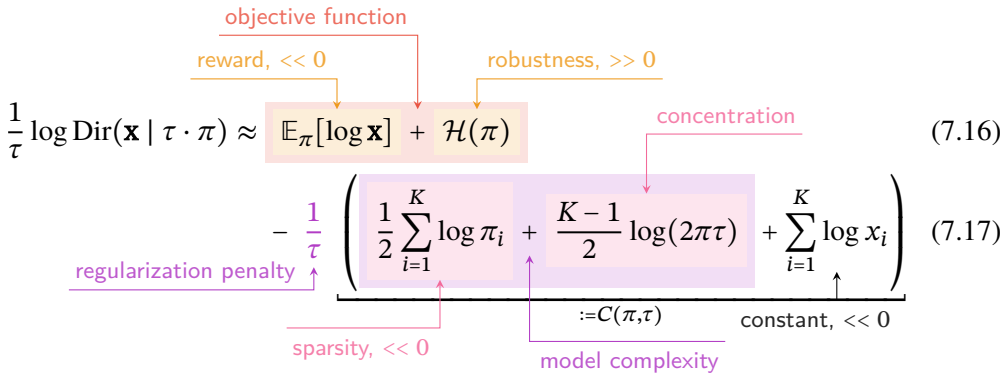
Besides the $(1 - 1/e)$ approximation guarantees on monotone submodular set functions, it is also known that the greedy approach is optimal on matroidal structures, and Lyons (2003) pointed out the close relationship between matroids and orthogonal projection DPPs. This special kind of DPPs is charaterized by all eigenvalues of the correlation kernel matrix $\mathbf{K} = \mathbf{L}(\mathbf{I} + \mathbf{L})^{-1}$ being zero or one. The cardinality of the samples is then deterministic. For orthogonal projection DPPs, following the greedy approach and

sampling iteratively from the posterior variance is known to result in the exact distribution (Hough et al., 2006).

It is also worth noting that the above statements claim nothing about the *similarity of the distributions* itself (e.g. in the sense of a total variation distance of the probability masses). Instead, they state that the two sampling distributions achieve a *similar performance in the task of generating diverse sets* – as quantified by the two entropy terms. While the latter is typically what one cares about in practical applications of DPP sampling, the former can be interesting future work to gain more theoretical insight into the elegant nature of DPPs.

This remark is not part of the original publication this chapter is based on and was added in the aftermath.

Remark In Appendix E.5, we show that *under a logarithmic-order Stirling approximation* the following relation between our Maximum-Entropy regularized objective function and a Dirichlet distribution with mean parameter π and concentration parameter τ holds:



$$\frac{1}{\tau}\log \mathrm{Dir}(\mathbf{x} \mid \tau \cdot \pi) \approx \mathbb{E}_{\pi}[\log \mathbf{x}] + \mathcal{H}(\pi) \tag{7.16}$$

$$- \frac{1}{\tau}\left(\frac{1}{2}\sum_{i=1}^{K} \log \pi_i + \frac{K-1}{2}\log(2\pi\tau) + \sum_{i=1}^{K} \log x_i \right) \tag{7.17}$$

The effect of decreasing the concentration τ is a stronger preference for less uniform policies π. For large τ the effect of the regularization vanishes. In our setting in Section 7.4, the rewards $h_\alpha(X_i) = \log\det(\alpha \cdot \mathbf{L}_{X_i,X_i})$ take the role of the log-probabilities $\log x_i$. Thus, although the policies are scale-invariant in α, the probabilistic interpretation is not: we cannot freely choose α if we wish to interpret the objective function as arising from the above Dirichlet log-likelihood. Instead, α has to be chosen such that reward function is normalized, that is $\sum_i \det(\alpha \cdot \mathbf{L}_{X_i,X_i}) = 1$. In order for our $(1-1/e)$-approximation guarantee to hold, the marginal gains of the reward must be positive, which is in conflict with the above normalization constraint. Assuming a normalized reward function by choosing the appropriate scale α' for the kernel function, we are thus left with a (potentially large) positive term on top of the objective function:

$$\begin{aligned} H_\alpha(\pi) &= \mathbb{E}_{X\sim\pi}[\log\det(\alpha \cdot \mathbf{L}_{X,X})] + \mathcal{H}(\pi) && (7.18)\\ &= \mathbb{E}_{X\sim\pi}[\log\det(\alpha \cdot (\alpha'/\alpha) \cdot \mathbf{L}_{X,X})] + \mathcal{H}(\pi) && (7.19)\\ &= \mathbb{E}_{X\sim\pi}[\log\det(\alpha' \cdot \mathbf{L}_{X,X})] + \mathcal{H}(\pi) + k\log\big(\alpha/\alpha'\big) && (7.20)\\ &= H_{\alpha'}(\pi) + k\log\big(\alpha/\alpha'\big) && (7.21) \end{aligned}$$

A caveat is that for small τ, the Stirling approximation becomes inaccurate, thereby weakening the connection between the above regularized objective and a Dirichlet log-likelihood in the small τ regime.

For $H_\alpha(\pi)$ with $\alpha = \alpha' \exp(-C(\pi,\tau)/(k\cdot\tau))$, we recover the regularized objective function from Equation 7.16. The constraint $\alpha > k!^{1/k}\lambda_{\min}$ can be achieved by lowering τ as much as needed. Note that the choice of α depends on π and τ. In our $(1-1/e)$-

approximation guarantee, we compare the greedy π_{greedy} and the optimal policy $\pi_{k\text{-DPP}}$ assuming the same scaling α. Since the sparsity penalty for both policies cannot expected to be the same in general, this implies that we are indirectly comparing the two strategies under different regularization penalties τ_{greedy} and $\tau_{optimal}$ – however, under the same total amount of regularization $\alpha = \alpha' \exp(-C(\pi_{greedy}, \tau_{greedy})/(k \cdot \tau_{greedy})) = \alpha' \exp(-C(\pi_{k\text{-DPP}}, \tau_{k\text{-DPP}})/(k \cdot \tau_{k\text{-DPP}}))$. It is further important to note that the greedy strategy was not derived from the regularized objective and the k-DPP is not the maximizer of the regularized objective but of the unregularized one with $\tau \to \infty$. Comparing their performance on the regularized objective, therefore, is another limitation of our theoretical analysis above.

7.5. Efficient Implementation

From a computational perspective, sampling proportionally to the posterior variance $\mathrm{Var}_i(x)$ in Equation (7.14) poses two challenges. First, for general kernels ℓ, there is usually no analytic cumulative density function for them which is required to efficiently sample the next point. However, this problem is much less severe in machine learning, because our community enjoys freedom in the design of models and can thus choose kernels with convenient analytic properties. Choosing such a kernel and exploiting these properties directly yields an efficient approximation for the generation of samples from k-DPPs, even in high-dimensional domains.

Second, even if the kernel is analytically convenient, the calculation of the posterior variance $\mathrm{Var}_i(x)$ involves the matrix inverse of $\mathrm{L}_{(i)}$. Given the inverse of $\mathrm{L}_{(i-1)}$ from the preceding step in the iterative sampling scheme, this inverse can be computed with complexity $\mathcal{O}\big((i-1)^2\big)$, using the matrix inversion lemma. Even so, the cost of drawing a sample of cardinality k remains cubic in k. This issue is directly connected to inference in Gaussian process regression models, and many approximations have been proposed over the past decade. Furthermore, in use cases like Bayesian optimization and quadrature, the number k of function evaluations is often low, and a decomposition of the Gram matrix is computed anyway. In such cases, the cubic cost can be unproblematic, and scaling to larger domains (high N and D) may be more important.

We provide an efficient implementation of the greedy principle for k-DPP sampling if the kernel ℓ is analytically integrable. To ease intuition, the derivations will be by way of example, using the popular square-exponential kernel $\ell_{\mathrm{SE}}\colon \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ over the real vector space

$$l_{\mathrm{SE}}(x_a, x_b) = \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_a - x_b)_d^2}{\lambda_d^2}\right). \tag{7.22}$$

To simplify things even further, we initially consider the univariate problem, $D = 1$, then generalize to arbitrary dimensionality. The resulting algorithm draws a k-sized sample at cost $\mathcal{O}(Dk^3 \log(N))$. A general form of the algorithm is summarized in Algorithm 1 in Appendix 2. There, we also provide a more detailed runtime analysis and describe how to extend the scheme to other popular kernels, like the Matérn class.

7.5.1. Sampling in One Dimension

We may draw samples using the classic form of computing a non-normalized cumulative density

$$\mathbb{V}_i(x) = \int_0^x \mathrm{Var}_i(\tilde{x})\,\mathrm{d}\tilde{x}\,, \tag{7.23}$$

and transforming standard uniform random variables $u \sim \mathbb{U}\big[0, \mathbb{V}_i(1)\big]$, produced by a pseudo-random number generator, into exact samples from Var_i (cf. Figure 7.2), by setting

$$x = \mathbb{V}_i^{-1}(u) = \{x \,|\, \mathbb{V}_i(x) = u\}\,. \tag{7.24}$$

For the univariate square-exponential kernel Equation(7.22), (7.13) can be re-written, using standard properties of the Gaussian function, as

$$\mathrm{Var}_i(x) \;=\; 1 \;-\; \sum_{a,b=1}^{i-1} \exp\left(-\frac{(x-m_{ab})^2}{\lambda^2}\right) \cdot \underbrace{\exp\left(-\frac{(x_a-x_b)^2}{4\lambda^2}\right)}_{=:M_{(i),ab}} [\mathrm{L}_{(i)}^{-1}]_{ab}\,. \tag{7.25}$$

where $m_{ab} := {}^1\!/_2(x_a + x_b)$, and we have defined a matrix $\mathbf{M}_{(i)} \in \mathbb{R}^{(i-1)\times(i-1)}$. The variables $m, \mathbf{M}, \mathbf{L}^{-1}$ provide the statistics of the sample needed to draw the subsequent point. After x_i has been drawn, these three variables can be updated in $\mathcal{O}(k^2)$—we use the matrix inversion lemma to update $\mathbf{L}_{(i+1)}^{-1}$; the other two variables can be updated in $\mathcal{O}(k)$. The cumulative density is then

$$\mathbb{V}_i(x) \;=\; x \;-\; \frac{\sqrt{\pi}\lambda}{2} \sum_{a,b=1}^{i-1} \left[\mathrm{erf}\left(\frac{x-m_{ab}}{\lambda}\right) + \mathrm{erf}\left(\frac{m_{ab}}{\lambda}\right)\right] [\mathbf{M}_{(i)} \odot \mathbf{L}_{(i)}^{-1}]_{ab}\,. \tag{7.26}$$

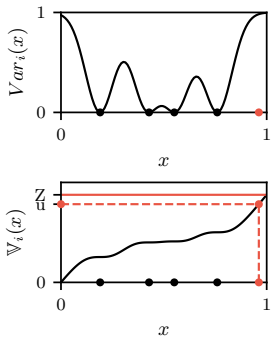


Figure 7.2.: Sketch illustrating analytic sampling from a k-DPP in one dimension, using the square-exponential kernel (Equation7.22). **Top:** Posterior variance $\mathrm{Var}_i(x)$ after the 4th iteration ($i = 5$). **Bottom:** Inverse transform sampling from $\mathrm{Var}_i(x)$, by computing the cumulative density $\mathbb{V}_i$ (black line), drawing a scaled uniform random sample u and finding the point x_i such that $\mathbb{V}_i(x_5) = u$, by interval bisection.

Here, $\odot$ is the Hadamard (element-wise) product, and we have used $\mathrm{erf}(x) = -\mathrm{erf}(-x)$. Given a uniform random draw u, all that is left to do is to find x such that $\mathbb{V}_i(x) = u$. A straightforward, numerically robust, albeit not particularly elegant way to do so is by interval bisection, which takes $\frac{1}{D}\log(N)$ steps of costs $\mathcal{O}(k^2)$. A more elegant search strategy could be constructed using grid refinement methods similar to the popular Ziggurat algorithm of Marsaglia and Tsang (2000).

7.5.2. Multivariate Samples

For square-exponential exponential kernel k-DPPs in dimension $D > 1$, $V_i(x)$ retains much of its structure. Equation (7.25) simply turns into (defining the elements of a new matrix $\mathbf{M} \in \mathbb{R}^{(i-1)\times(i-1)}$ analogous to $\mathbf{M}$ in Equation (7.25)):

$$\mathrm{Var}_i(x) = 1 - \sum_{a,b=1}^{i-1} \exp\left(-\sum_{d=1}^{D} \frac{(x-m_{ab})_d^2}{\lambda_d^2}\right) \cdot \underbrace{\exp\left(-\sum_{d=1}^{D} \frac{(x_a-x_b)_d^2}{4\lambda_d^2}\right)}_{=:\mathbf{M}_{(i),ab}} [\mathbf{L}_{(i)}^{-1}]_{ab}\,. \tag{7.27}$$

The additional challenge in this multivariate case is to construct a parametrization of the cumulative density $\mathbb{V}_i$. This step, too, can be performed in an iterative fashion, drawing one coordinate of the sample point x_i after another (cf. Figure 7.3). Given that the first $d-1$ elements of x_i are given by $x_{i,1:d-1}$, the cumulative density associated with the d-th dimension is given by the sum rule:

$$\mathbb{V}_i(x_{i,d}|x_{i,1:d-1}) = \int_0^{x_{i,d}} \int_0^1 \cdots \int_0^1 \mathrm{Var}_i\big([x_{i,1:d-1}, \tilde{x}_{i,d}, \tilde{x}_{i,d+1:D}\big)\, \mathrm{d}\tilde{x}_{i,d} \prod_{\tilde{d}=d+1}^{D} \mathrm{d}\tilde{x}_{i,\tilde{d}}\,. \quad (7.28)$$

For the square-exponential kernel, this works out to

$$\begin{aligned}\mathbb{V}_i(x_{i,d} = x \,|\, x_{i,1:d-1}) = \\ x - \sum_{a,b=1}^{i-1} \Bigg\{ \exp\left(-\sum_{r=1}^{d-1} \frac{(x - m_{ab})_r^2}{\lambda_r^2}\right) [\mathbf{M}_{(i)} \odot \mathbf{L}_{(i)}^{-1}]_{ab} \\ \cdot \left(\mathrm{erf}\left(\frac{[x_i - m_{ab}]_d}{\lambda_d}\right) + \mathrm{erf}\left(\frac{[m_{ab}]_d}{\lambda_d}\right)\right) \frac{\sqrt{\pi}\lambda_l}{2} \\ \cdot \left(\prod_{j=d+1}^{D} \left(\mathrm{erf}\left(\frac{[1 - m_{ab}]_j}{\lambda_j}\right) + \mathrm{erf}\left(\frac{[m_{ab}]_j}{\lambda_j}\right)\right) \frac{\sqrt{\pi}\lambda_j}{2}\right)\Bigg\}. \quad (7.29)\end{aligned}$$

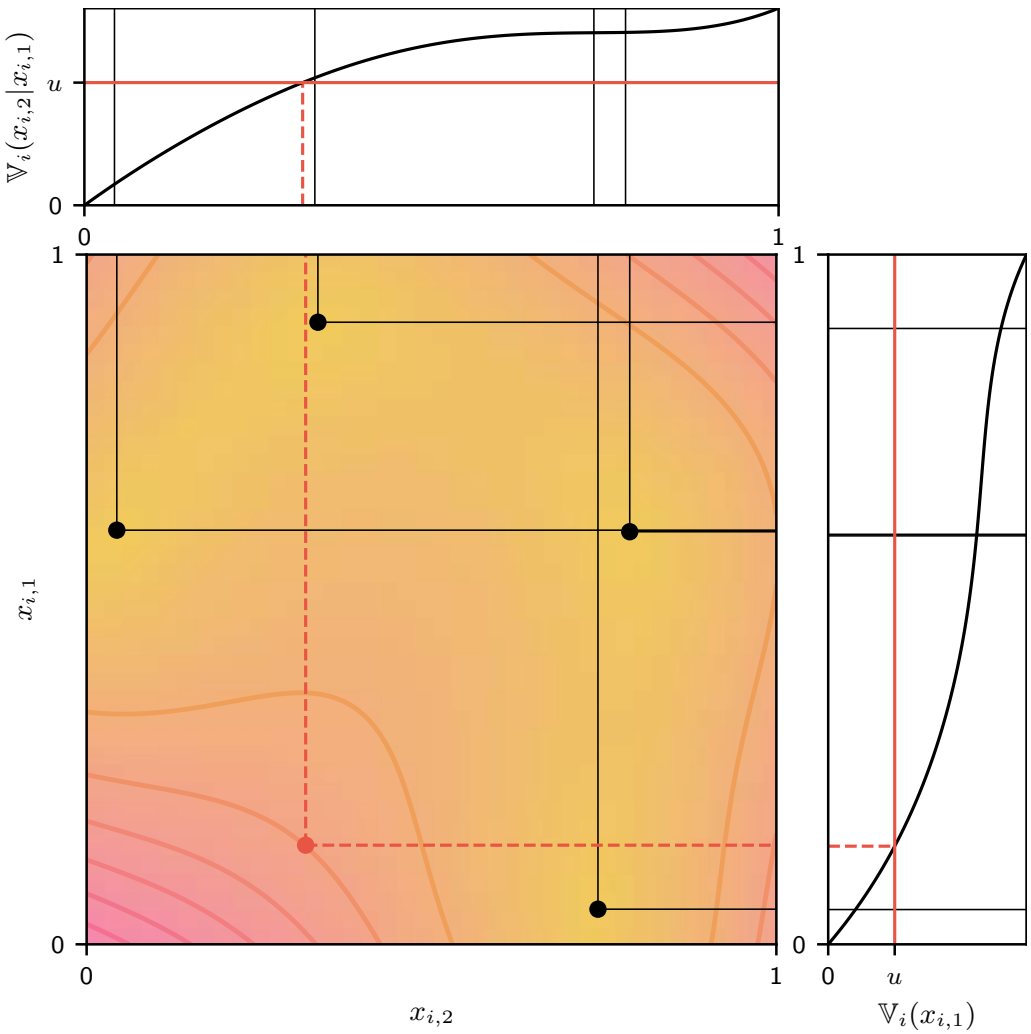


Figure 7.3.: Drawing approximate k-DPP samples in two dimensions. Bottom left: Contour plot of the multivariate probability density $\mathrm{Var}_i(x)$ with $(i = 5)$; preceding four samples as black points. **Right:** The first coordinate of the fifth sample is drawn first, from the marginal density along this dimension. **Top:** The second coordinate is then drawn by computing a cumulative density *conditioned* on the value of the first coordinate.

7.6. Experiments

We conduct an empirical analysis of the approximation quality in an applied setting. If greedily sampled locations cover the domain almost as well as the exactly sampled ones, one would expect to learn a similar amount of information about a function (modeled with a Gaussian process) evaluated at these locations. This should lead to little performance decrease in follow-up tasks, such as the integration of that function. To investigate this, we perform Bayesian Quadrature (BQ) of several benchmark functions with evaluation locations sampled from exact and approximate k-DPPs, as well as uniformly chosen locations. For the BQ we rely on the vanilla version from `emukit` * (Paleyes et al., 2023). As integrands we use the benchmark functions from `The Virtual Library of Simulation Experiments` † (Surjanovic and Bingham). We use a square-exponential kernel with a default lengthscale of 0.2 for all methods and benchmarks. All functions are integrated over the domain $\mathbb{X} = [0, 1] \times [0, 1]$. Evaluation locations are sampled from a regular 50×50 grid. For more details on the methods and hyperparameters,

* https://emukit.github.io/bayesian-quadrature/
† https://www.sfu.ca/~ssurjano/integration.html

see Appendix E.6. The code for all experiments in this paper is available at https://github.com/JuliaGrosse/GreedykDPPSampling.
Figure 7.4 shows the mean error between the true value $F := \int_{x \in \mathbb{X}} f(x)dx$ and the value $\hat{F}$ estimated with BQ for the twelve benchmark functions. On the benchmarks, where there was a significant advantage of the DPP over the uniform distribution, the greedy version performed equally well to the DPP. This indicates that sampling from the exact k-DPP might not be very crucial in an application like this, as long as some repulsiveness is still present.

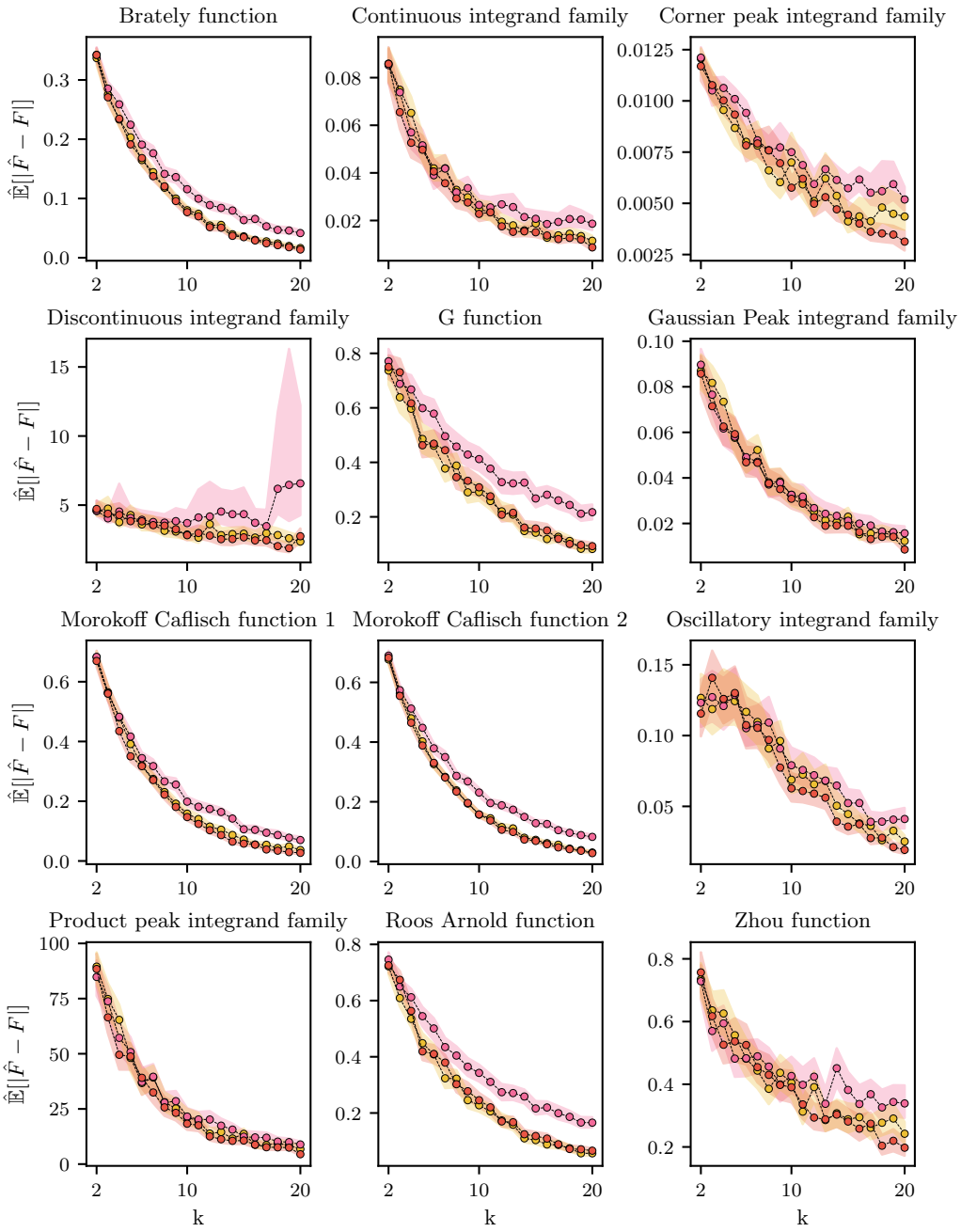


Figure 7.4.: Results from BQ with evaluation locations sampled from an exact k-DPP ( ), the greedy approximation ( ) and uniformly sampled locations ( ). The plots show the mean error over 100 samples and 95% confidence intervals for the mean error as shaded areas.

7.7. Related Work

The maximum entropy formulation we use as starting point for our derivations is prominent in reinforcement learning. For an introduction to *Maximum Entropy reinforcement learning* see, e.g. Section 3.2 or Levine (2018). These methods are used to learn a policy π that satisfies the objective in equation 7.7 for an arbitrary function h. Our method differs from this line of work in that we do not learn π, but instead give an analytic approximation for π from scratch for the specific choice of $h(X)$ being the Gaussian entropy itself (Equation 7.5).

Regarding the theoretical analysis, the work from Djolonga et al. (2018) is closest to ours. They also prove a $(1 - 1/e)$ guarantee of a greedy algorithm on a log partition function. Instead of monotonicity and submodularity, they assume that h is a sum of $M^\natural$-concave functions[2]. In addition, their greedy algorithm is a variational approximation and not analytically derived as in our case. The paper by (Hough et al., 2006, Prop. 19) contains a special case of our greedy algorithm for orthogonal projection DPPs (where it is exact). A greedy approximation for maximum a posteriori inference in DPPs is suggested in Chen et al. (2018).

Chen et al. (2022) sample proportionally to the posterior variance of a Gaussian process – without the inverse transform sampling – in the context of quadrature and low-rank matrix approximations. They provide bounds on the expected trace $\mathbb{E}[\mathrm{tr}(\mathbf{A} - \mathbf{A}_k)]$, where $\mathbf{A}_k$ is the sampled low-rank approximation of the matrix $\mathbf{A}$. Epperly and Moreno (2023) further analysed the greedy sampling scheme – specifically for quadrature – and bound the approximation error on the integrand itself. Similarly, Huszár and Duvenaud (2012) and Adachi et al. (2022) studied the convergence properties of the method in the quadrature setting. However, our analysis in terms of the entropies draws a novel connection to k-DPPs, providing a theoretical justification for why it can be seen as an approximation of them.

Due to the large amount of recent literature on DPPs in general, we restrict the remainder of this section to an overview of exact and approximate sampling from k-DPPs only. Originally, exact samplers for generic k-DPPs were based on eigendecompositions of the entire $N \times N$ Kernel Gram matrix and thereby required $\mathcal{O}(N^3)$ time (Kulesza and Taskar, 2011). Derezinski et al. (2019) introduced DPP-VFX, an intermediate sampling method for k-DPPs. They first sample intermediate points from the marginal distributions of a random-sized DPP, and then repeatedly sample from a DPP restricted to the intermediate points until a set of size k is sampled. The algorithm has time complexity in $\mathcal{O}(N \cdot k^{10} + k^{15})$. The linear costs in N can be reduced to less than linear costs by another intermediate sampler named α-DPP (Calandriello et al., 2020), that does not require the computation of all marginals and additionally uses a more efficient reduction method from the DPP to the k-DPP. Their sampling method adaptively builds a sufficiently large uniform sample of the entire N points that is then used to efficiently generate a smaller set of k points, while ensuring that this set is drawn exactly from the target distribution defined on all N points.

2: Let $\mathcal{E}$ be a finite set, and let $f : 2^{\mathcal{E}} \to \mathbb{R} \cup \{-\infty\}$ be a set function. The function f is said to be $M^\natural$-*concave* if for all subsets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{E}$ and all $i \in \mathcal{A} \setminus \mathcal{B}$, at least one of the following conditions holds:

1. There exists $j \in \mathcal{B} \setminus \mathcal{A}$ such that
$$f(\mathcal{A}) + f(\mathcal{B}) \leq f(\mathcal{A} \setminus \{i\} \cup \{j\}) + f(\mathcal{B} \setminus \{j\} \cup \{i\}),$$
2. Or,
$$f(\mathcal{A}) + f(\mathcal{B}) \leq f(\mathcal{A} \setminus \{i\}) + f(\mathcal{B} \cup \{i\}).$$

The resulting complexity is reported to be in $\mathcal{O}((\beta N \cdot k^6 + k^9)\sqrt{k})$, making α-DPP to the best of our knowledge the currently fastest exact sampler for k-DPPs. The constant $\beta \leq 1$ depends on the effective dimension d_{eff} of the matrix $\mathbf{L}$, the sample size k, as well as the largest entry κ^2 in $\mathbf{L}$ and can be specified further to $\beta \leq \min\{k^2\kappa^2/d_{\text{eff}}(L), 1\}$.

Regarding approximate sampling, Markov-Chain-Monte-Carlo (MCMC) methods are popular. Anari et al. (2016) introduced one that runs in $\mathcal{O}(\text{poly}(k)\, N \log(N/\epsilon))$, where k is the number of items to sample from a set of cardinality N. Here, ϵ is a small constant determining the approximation quality of the MCMC samples. The approximation guarantees for MCMC only hold after the mixing time of $\mathcal{O}(N \cdot \text{poly}(k))$. Transition steps in the Markov Chain take time polynomial in k. Rezaei and Gharan (2019) developed a k-DPP sampler for continuous domains. As such its runtime does not depend on N, however, it involves rejection sampling from the conditionals of the k-DPP, which can become expensive in k. If $k \leq e^{D^{1-c}}$ for some constant $0 \leq c \leq 1$, one can show that the time complexity is in $\mathcal{O}(D \log(1/\epsilon)) \cdot k^{\mathcal{O}(1/c)}$. Based on the asymptotic runtimes, the greedy algorithm in this paper compares favorably if exactness is not absolutely crucial and the domain is large or high-dimensional.

Runtime Comparison

We compare the runtime of the greedy algorithm with those of several state-of-the-art exact and approximate samplers (Derezinski et al., 2019; Calandriello et al., 2020; Anari et al., 2016) described above (Figure D.4). For the baselines, we use the Python implementations available in DPPy[‡] (Gautier et al., 2019b) with default parameters. We draw 100 samples with $k = 10$ and $k = 100$ points from a k-DPP with square-exponential kernel with lengthscale 0.01, respectively 0.001, on the interval $[0, 1]$. We run all methods for discretization size $N \in \{10^2, 10^3, 10^4, 5 \cdot 10^4, 6 \cdot 10^4, 7 \cdot 10^4, 10^5\}$. Runs taking longer than 100 seconds for $k = 10$ or 360 seconds for $k = 100$ were stopped, i.e. they do not appear in the figure. For $k = 10$, the experiment was repeated 5 times and for $k = 100$, 3 times. Additional results from a repetition of the experiment on $[0, 1]^3$ with lower discretization sizes are included in Appendix E.6.The results agree with those from the asymptotic runtime analyses.

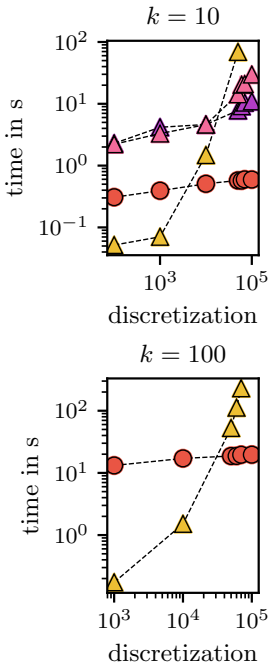


Figure 7.5.: Runtime comparison of the greedy (●)algorithm with state-of-the-art baselines MCMC (▲), DPP-VFX (▲) and α-DPP (▲). Discussion in text.

7.8. Conclusion

We introduced a greedy approximation for sampling from k-DPPs theoretically grounded in an analogy to greedy optimization of the Gaussian differential entropy. We showed that approximation guarantees for greedy optimization of the entropy have a resembling interpretation in the sampling setting and tested the approximation in a BQ application, where we found the approximation error to be empirically negligible. We provided an efficient implementation for continuous domains – logarithmic in the size of the discretization N – for common kernels like the Matérn class. The algorithm described

[‡] https://github.com/guilgautier/DPPy

herein thus offers itself as a low-level routine for all applications that require such diverse points sets as part of a surrounding experimental design loop.

Discussion

Chapter 8

Discussion

This chapter summarizes the methods and results presented in the previous chapters, and then continues with a discussion of the overarching attributes of probabilistic models for tree search. These attributes are not entirely separate, but rather represent interrelated perspectives on tree search and its regularization.

Summary. In this thesis, we explored various approaches to regularizing tree search problems, primarily from a probabilistic perspective through the inclusion of priors. We began by establishing a connection between Bayesian optimization and traditional tree search algorithms in the field of Optimistic optimization. This was done by mapping the prior's kernel function to the dissimilarity function used in these algorithms, leading to the GP-OO algorithm. Subsequently, we introduced two novel probabilistic inference schemes for tree search: PDAGS and ULTS. PDAGS incorporates similarity information between nodes across the tree via a Gaussian prior, while ULTS regularizes tree search for finding a maximum log likelihood sequence by assuming either a Dirichlet prior or an empirically derived prior from similar tree instances. In both cases, we found that these approaches improved the data efficiency of the algorithms compared to their respective baselines, thereby enhancing their exploration–exploitation trade-offs. Finally, we considered a pure exploration scenario. We formulated sampling from k-DPPs as a regularized version of Gaussian entropy maximization and proposed a runtime-efficient approximation for this task.

Graph Structure. With PDAGS and GP-OO, we extended methods (Hennig et al., 2010; Grill et al., 2018) that were originally developed under the assumption that the reward functions exhibit a Wiener kernel structure to more general kernel classes. For a Wiener kernel, the canonical pseudo-metric $\delta(x, y)$ of the Gaussian process is closely connected to the underlying distance function $d(x, y)$ – e.g., graph distance in the discrete search space or Euclidean distance as in continuous search domains – as $\delta(x, y)^2 = d(x, y)$. PDAGS and GP-OO relax this assumption to allow for more expressive kernels. However, this generality comes at a cost: in PDAGS, richer kernels can yield improved data efficiency at the cost of higher runtime; in GP-OO, the runtime efficiency of the Optimistic optimization family of algorithms is preserved, but Bayesian optimization methods often show stronger data efficiency with the same kernels. To establish regret bounds for GP-OO, we imposed structural constraints on the kernel-induced pseudo-metric, requiring that $\delta(x, y) \leq C\|x - y\|_2^\alpha$. This condition ensures sublinear regret and enables a quantifiable trade-off between smoothness (via α) and learning efficiency. While this analysis currently applies only to a restricted class of kernels and is based on worst-case assumptions about tree growth, it provides a first step toward understanding the interplay between kernel structure and search complexity, beyond Brownian models.

Regularity. Both probabilistic tree search methods, PDAGS and ULTS, rely on the approximate inference scheme introduced by Hennig et al. (2010) to capture regularity in tree search problems. The regularity assumption here lies in assuming that the changes in the generative scores, or the intermediate rewards in the case of ULTS, follow the same law across all nodes of the tree. Together with an independence assumption, this enabled polynomial-time predictions for the optimal values of exponentially growing subtrees. As discussed in the corresponding chapters, this assumption is not satisfied for PDAGS and likely not satisfied for ULTS as well. Nevertheless, due to both methods showing good empirical performance, it might be a good starting point for the development of future probabilistic tree search methods as well.

Nevertheless, given the above challenges introduced by the deviation from the regular tree structure, one might ask whether functions with self-similar structure (like samples from a Wiener Process) represent the sweet spot for tree search algorithms. In this light, an interesting class of probabilistic models for potentially very efficient tree searches is the class of Lévy processes (Bertoin, 1996). Like the Wiener process, they exhibit infinite self-similarity. However, their increments are not limited to Gaussian distributions but can instead originate from a more general Lévy distribution. The heavier tails of the Lévy distribution would enable modeling reward functions with larger jumps. While not widespread in machine learning, Lévy processes often appear in financial applications (Schoutens, 2003; Raible, 2000; Barndorff-Nielsen and Shephard, 2001; Rachev et al., 2011).

Another stochastic process that displays self-similar behavior is the Mondrian process (Roy and Teh, 2008) – named after the painting shown in Figure 1.1 on page 3. This process iteratively splits a space (e.g., a hyperrectangle) into smaller regions using axis-aligned cuts. The splits are guided by a Poisson process, where the next cut occurs after an exponentially distributed waiting time. Since the splits are generated iteratively, their positions can be assigned the nodes of a tree, similar to the partition processes in GP-OO. The Mondrian process is self-similar up to a certain scale, the time the tree dies. One can derive a kernel from a Mondrian partition by setting $k(x, y) = 1$ if x and y are in the same cell and $k(x, y) = 0$ otherwise. Taking the expectation over Mondrian kernels from iid. Mondrian processes results in a Laplace kernel with the lengthscale matching the lifetime of the process a.k.a the depth of the tree (Balog et al., 2016). Extensions to a wider class of kernels exist (O'Reilly and Tran, 2020) by allowing for non-axis aligned cuts. Using such kernel structures for probabilistic tree search methods could be an interesting object of study as they lie in the middleground between the generic models used by PDAGS and GP-OO, and the Wiener-kernel based models from Hennig et al. (2010) and Grill et al. (2018). Due to the generation processes of the kernels being tree-based, the sampled functions clearly have a tree structure, but one that might not perfectly align with the underlying (tree) structure of the search domain due to a mismatch in the lifetime and tree depth.

Tree depth and horizon. In settings where the tree depth is not fixed or bounded, it is more appropriate to interpret it as a *horizon*. For instance, in MDPs, the horizon is

theoretically infinite but practically limited by the discount factor. In Box 3.2.1 of Chapter 6.2, we reviewed how a mismatch between the planning discount factor and the true discount factor can act as a form of regularization: the mismatch implicitly induces a Dirichlet prior over transitions. Although this result does not directly apply to our tree search setting – where transitions are deterministic – it motivates a deeper look into how regularization mechanisms relate to the tree depth in our scenarios.
In Section 7.4, we discussed a potentially related observation in the context of our theoretical guarantees for greedy k-DPP sampling. Here, the scaling parameter α controls the effective volume that the k-DPP tries to cover with k self-repelling points. Increasing α boosts the eigenvalues of the kernel matrix, which raises the expected number of sampled points in the unconstrained DPP setting. When k is small relative to this expected number, the space is sparsely populated, making the task of selecting k diverse points less constrained. However, when k approaches the expected number of samples under the DPP, the space becomes more tightly packed, and the difference between greedy and exact selections might become more significant. This mismatch between k and the expected number of sampled points under the unconstrained DPP appears to function as a form of implicit regularization. A deeper understanding of this intuition – analogous to existing results on discount-based regularization discussed before – could be an interesting direction for future investigation.

Given the significant role that tree depth plays for regularization, it may be beneficial to refine how depth is handled in PDAGS and ULTS. Both currently assume a fixed, uniform tree depth. While this assumption was reasonable in the feature selection setting for PDAGS, it was already a simplification in game-playing settings we looked at. In the context of LLM decoding – a primary application case of ULTS – the assumption is also too restrictive, as the decoding process naturally involves variable-length sequences. While early termination (as we did in e.g. the machine translation experiment in Appendix D) is always an option, a more principled approach may be to incorporate variable depth directly into the probabilistic model. This would also bring ULTS in line with competing non-probabilistic methods, such as beam search variants, which already support flexible sequence lengths. Since shorter sequences typically have a higher likelihood, the most straightforward way to do this is to normalize the sequences by simply dividing the likelihood by the sequence's length $|\mathbf{x}|$. Variations of this approach, e.g. dividing by $|\mathbf{x}|^\rho$ with $\rho \in \mathbb{R}_{\geq 0}$, have shown better empirical performance (Wu et al., 2016). They have the disadvantage though, that with the *length penalty* ρ they introduce another hyperparameter to the decoding. Similar to how ULTS uses the concentration strength (learned from the training data) to allow more flexibility in the beam size alias the tree width, it would be interesting to see if a learned distribution over the sequence length could be exploited to further improve the decoding process by allowing for more flexibility in the sequence length alias the tree depth.

Concentration properties. A recurring theme throughout this thesis has been the role of concentration – both in the form of concentration inequalities and as a structural component in probabilistic models. Classic methods such as UCT and MENTS from the literature rely on concentration inequalities to balance exploration and exploitation.

Our GP-OO algorithm similarly evolves around a sub-Gaussian concentration inequality for $|f(x) - f(y)|$ *, and ULTS leverages knowledge about the concentration strength via, e.g., a Dirichlet prior. In the "infinite version" of the Dirichlet distribution – the Dirichlet process – the concentration parameter α influences the number of sampled points. We hypothesize that the scaling parameter α in our analysis of k-DPP sampling plays a similar role to a concentration parameter, as it also affects the number of sampled points in the "infinite version" of the k-DPP, the unconstrained DPP. In the probabilistic interpretation of the objective function H, which we discussed in the remark in Section 7.4, we showed that – under a Stirling approximation – H approximates the log-likelihood of a Dirichlet distribution with mean π and concentration τ, up to a correction term that depends on π and τ. This interpretation allowed us to see α as implicitly encoding a form of regularization derived from the Dirichlet distribution. However, a subtle limitation arises here: we are ultimately only interested in comparing the greedy distribution and the exact k-DPP, i.e. only the means of the Dirichlets. From this perspective, the presence of an additional term in H that involved further entropy or sparsity penalties is somewhat unsatisfactory since it reflects complexity or uncertainty aspects not directly relevant to the mean comparison.

Robustness. Another aspect that must be considered when designing tree search algorithms is that epistemic uncertainty is not necessarily the only kind of uncertainty present. In the k-DPP setting and the entire Maximum Entropy reinforcement learning framework – with an unresolvable stochasticity in the final policies – clearly a different kind of uncertainty also plays a role for tree search. As discussed in Section 3.2, this additional uncertainty contributes to greater robustness in the resulting policies, particularly under adversarial conditions. As mentioned in Chapter 6 this is a feature that many would also like to see in the decoding for LLMs, as the likelihood – the reward function – is known to sometimes exhibit anomalies that in practice can be overcome with stochastic policies like Nucleus sampling Holtzman et al. (2019). Recent theoretical analysis in (Chen et al., 2024a) also confirms that Nucleus sampling optimally solves a myopic approximation to a robust and thereby regularized version of likelihood maximization. While ULTS focuses on handling the computational burden arising from being non-myopic, it might be promising to combine these two approaches together. Ideas about probabilistic models and acquisition functions in robust Bayesian optimization methods (Weichert and Kister, 2021; Bogunovic et al., 2018) could result in fruitful future work at the intersection of Bayesian optimization and LLMs.

Curvature. Another way to view parameter α in the context of DPPs is through the lens of information geometry. In general, DPPs do not form an exponential family. However, it has recently been shown that DPPs form a so-called *curved* exponential family, meaning that they are a subset of an exponential family where the natural parameters are constrained to lie on a lower-dimensional manifold within the parameter space of an exponential family (Hino and Yano, 2024). For each parameter one can calculate an embedding curvature. The embedding curvature can be seen as a measure for how close a curved exponential

* We derived GP-OO from the *deviation inequality* in Pisier (1999) for $|f(x)-f(y)|$. Under our assumption of a centered Gaussian process and with the substitution $X := f(x) - f(y)$, this inequality has the form of a standard sub-Gaussian *concentration inequality* bounding the deviation of X from its expectation

family is to an exponential family in the sense that in an exponential family all parameters have embedding curvature equal to zero. In the DPP family, there are some parameters that have flat curvature. By scaling a DPP with α one increases the values of exactly these parameters, thereby moving the DPP closer to an exponential family. In our case – though referring to k-DPPs and not DPPs – scaling the k-DPP with α increased the approximation ratio between the greedy and optimal strategy. This points towards our greedy strategy not being able to handle curvature correctly.

We introduced soft objective functions such as the one maximized by a k-DPP under the Maximum Entropy reinforcement learning framework. However, there is an alternative optimization framework for this class of objective functions under a concept known as the Bayesian Learning rule. In the style of Variational Inference, this approach involves parameterizing the approximating policy and applying natural gradient descent to optimize the parameters.[1] While the Bayesian Learning was originally derived for finding approximations in minimal exponential families, there have recently been suggested efficient parameter update schemes for cases where the approximating distribution is from a particular curved exponential family (Lin et al., 2021) or from a mixture of minimal exponential families (Lin et al., 2019)[2], as well as to a family of distributions parameterized via Lie-groups (Kiral et al., 2023). While it is not obvious if such approaches would directly apply to k-DPPs, and equally importantly, if one can still sample efficiently from the approximating distribution, it might be worthwhile to think about how to also modify tree-based (approximate) sampling schemes such that they also respect the curvature.

1: Example: The related work from Djolonga et al. (2018) discussed in Section 7.7, which derived a $(1 - 1/e)$-approximation for the soft version of $M^\natural$-concave functions, achieved this using variational inference.

2: k-DPPs are mixtures of projection DPPs (Kulesza and Taskar, 2011)

Closing remark. Looking back at where we started, with Mondrian and his conception that in order to capture the essence of a tree one has to "progressively abstrac[t] the curves" and that "Line and color must be composed otherwise than in nature," we see a reflection of some of the conceptual challenges encountered in this thesis. Confronted with function spaces composed otherwise than search spaces, varying levels of smoothness, the notion of curvature, and the tension between expressiveness and tractability, we could only catch a glimpse of the aspects Mondrian had to address while abstracting the tree structure.

APPENDIX

Appendix A

Additional Material for Chapter 3

A.1. Metrics, Pseudo-Metrics, Semi-Metrics and Dissimilarities

Dissimilarity:
A function $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is called a *dissimilarity* if it satisfies:

- $\delta(x, x) = 0$ for all $x \in \mathcal{X}$ (reflexivity),
- $\delta(x, y) \geq 0$ for all $x, y \in \mathcal{X}$ (non-negativity).

It need not be symmetric or satisfy the triangle inequality.

Semi-Metric:
A function $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a *semi-metric* if it satisfies:

- $\delta(x, x) = 0$ for all $x \in \mathcal{X}$ (identity),
- $\delta(x, y) = \delta(y, x)$ for all $x, y \in \mathcal{X}$ (symmetry),
- $\delta(x, y) \geq 0$ for all $x, y \in \mathcal{X}$ (non-negativity).

A semi-metric does not necessarily satisfy the triangle inequality.

Pseudo-Metric:
A function $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a *pseudo-metric* if it satisfies all properties of a semi-metric and, additionally:

- $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$ for all $x, y, z \in \mathcal{X}$ (triangle inequality),
- $\delta(x, y) = 0$ does not necessarily imply $x = y$.

Metric:
A function $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a *metric* if it satisfies all properties of a pseudo-metric and, additionally:

- $\delta(x, y) = 0$ implies $x = y$ (identity of indiscernibles).

A.2. Discount Regularization

Discount regularization refers to an reinforcement learning scenario where the agent finds a policy using a shorter horizon γ_p during planning than the environment's true horizon γ. Rathnam et al. (2024) analyse this scenario for the standard MDP setting with infinite horizon and stochastic transitions. The stochastic transitions are estimated from data during the training. They can show that using discount regularization implies a Dirichlet prior on the transitions:

Let $\langle c_{i,j,1}, \dots, c_{i,j,K_s} \rangle$ be the transition count data observed from state s_i to states 1 through K_s under action a_j. Using a Dirichlet prior $\mathrm{Dir}(\alpha_{i,j,1}, \dots, \alpha_{i,j,K_s})$ for the state-action pair (s_i, a_j), $T_{\text{prior}}(s_i, a_j, \cdot) \sim \mathrm{Dir}(\alpha_{i,j,1}, \dots, \alpha_{i,j,K_s})$, the posterior mean of the transition matrix, $\hat{T}_{\text{post}}$, is equal to a weighted average of the maximum likelihood estimated transition matrix and the mean of the prior:

$$\hat{T}_{\text{post}}(s_i, a_j, \cdot) = (1 - \varepsilon_{i,j})\hat{T}_{\text{MLE}}(s_i, a_j, \cdot) + \varepsilon_{i,j} T_{\text{prior}}(s_i, a_j, \cdot),$$

where

$$\varepsilon_{i,j} = \frac{\sum_{k=1}^{K_s} \alpha_{i,j,k}}{\sum_{k=1}^{K_s} c_{i,j,k} + \sum_{k=1}^{K_s} \alpha_{i,j,k}}.$$

Assuming a symmetric Dirichlet prior with

$$\alpha_{i,j,k} = \left(\frac{\gamma - \gamma_p}{\gamma_p} \right) \sum_{k=1}^{K_s} c_{i,j,k},$$

the policy found with discount regularization matches the one in the original MDP with this Dirichlet prior.

Rathnam et al. (2024) further prove that the above observation is equivalent to penalizing the Q-function (see Theorem A.2.1). They also establish equivalences with partial termination and with an approximation to a truncated λ-return, another regularization technique in reinforcement learning.

Theorem A.2.1 (Discount Regularization as Penalization on Q-Value Function, Rathnam et al. (2024)) Let M_1 and M_2 be finite-state, infinite horizon MDPs with identical state space, action space, and reward function. Let discount factor $0 \leq \gamma < 1$, regularization parameter $0 < \varepsilon \leq 1$, and let $Q_{\text{reg}}(s, a)$ be a function used to regularize the Q-function that is constant in (s, a), i.e., $Q_{\text{reg}}(s, a) = Q_{\text{reg}}$.

If M_1 uses discount rate $\gamma_1 = (1 - \varepsilon)\gamma$ and state-action value function

$$Q^*(s, a) = R(s, a) + \gamma_1 \mathbb{E}_{s' \sim T(S,A,\cdot)} \left[\max_{a'} Q^*(s', a') \mid S = s, A = a \right]$$

in planning, and M_2 uses discount rate γ and state-action value function

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim T(S,A,\cdot)} \left[\max_{a'} \left[(1-\varepsilon) Q^*(s',a') + \varepsilon Q_{\text{reg}}(s,a) \right] \mid S = s, A = a \right]$$

in planning, then M_1 and M_2 have the same optimal policy.

A.3. Kalman Duality in Maximum Entropy Reinforcement Learning

Assume a MDP with linear and deterministic system dynamics, i.e. $s_{t+1} = \mathbf{C}s_t + \mathbf{B}a_t$ for matrices $\mathbf{C}$ and $\mathbf{B}$. The objective function under Maximum Entropy Regularization, that one aims to maximize is

$$\mathcal{F}(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \left(s_t^\top \mathbf{Q} s_t + a_t^\top \mathbf{R} a_t \right) \right] - \beta \mathbb{E}_\pi \left[\sum_{t=0}^{T} \mathcal{H}(\pi(a_t|s_t)) \right] \tag{A.1}$$

where $\mathcal{H}(\pi(a_t|s_t)) = -\int \pi(a_t|s_t) \log \pi(a_t|s_t)\, da_t$ is the entropy of the policy, and β is a temperature parameter.

The Value function and Q-function can be written down as

$$V_t(s_t) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \left(s_k^\top \mathbf{Q} s_k + a_k^\top \mathbf{R} a_k \right) - \beta \sum_{k=t}^{T} \mathcal{H}(\pi(a_k|s_k)) \middle| s_t \right] \tag{A.2}$$

$$Q_t(s_t, a_t) = s_t^\top \mathbf{Q} s_t + a_t^\top \mathbf{R} a_t + \mathbb{E}_\pi \left[V_{t+1}(s_{t+1}) \right] \tag{A.3}$$

It turns out that the optimal policy $\pi^*(a_t|s_t)$ is a Boltzmann policy over the Q-values:

$$\pi^*(a_t|s_t) \propto \exp\left(-\frac{1}{\beta} Q_t(s_t, a_t) \right) \tag{A.4}$$

Since $Q_t(s_t, a_t)$ is quadratic in a_t, this exponential form results in a Gaussian distribution. This Gaussian can be derived to be

$$\pi^*(a_t|s_t) = \mathcal{N}(a_t|\mu_t, \boldsymbol{\Sigma}_t) \tag{A.5}$$

where:

- $\mu_t = -\mathbf{K}_t s_t$
- $\boldsymbol{\Sigma}_t = \beta \mathbf{R}^{-1}$
- $\mathbf{K} = (\mathbf{B}^\top \mathbf{P} \mathbf{B} + \mathbf{R})^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{C}$
- $\mathbf{P} = \mathbf{Q} + \mathbf{C}^\top \mathbf{P} \mathbf{C} - \mathbf{C}^\top \mathbf{P} \mathbf{B} (\mathbf{B}^\top \mathbf{P} \mathbf{B} + \mathbf{R})^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{C}$

Note that the mean is independent of temperature β and therefore also provides the exact solution to the standard reinforcement learning problem with $\beta \to 0$. Although not in the context of reinforcement learning, this result was first shown by Kalman (1960) and later generalized to nonlinear stochastic systems, discrete stochastic systems, and deterministic systems by Todorov (2008).

A.4. K-Learning

As pointed out in Section 3.2, K-Learning assumes independent Dirichlet priors on the transition functions, as well an additive σ-sub-Gaussian noise on the rewards. Together, this leads to uncertainty over the Q-value $Q(s,a)$ for a state-action pair (s,a), which is therefore a random variable, following a distribution $p(Q(s,a))$. The idea is to approximate the conditional optimality probability at (s,a) in iteration t as:

$$p(o_t = x_c | Q_t^*(s,a)) \propto \exp(\beta Q_t^*(s,a))$$

for some temperature parameter β. In order to properly handle the uncertainty over the Q-value, the next step is to marginalize over the possible values it can take:

$$p(o_t = x_c | Q_t^*(s,a)) = p(o_t = x_c | Q_t^*(s,a)) dp(Q(s,a)) \propto \exp(G_Q(s,a,\beta)), \tag{A.6}$$

where $G_Q(s,a,\cdot)$ denotes the cumulant generating function of the random variable $Q(s,a)$. One can show that the cumulant generating functions at β can be upper bounded by the so called *K-values*:

$$K_t(s,a) = \mathbb{E}\mu(s,a) + \frac{\sigma^2 \beta_t}{2n^t(s,a)} + \sum_{s'} \mathbb{E}p(s',s,a) V_{t+1} K(s') \tag{A.7}$$

$$V_t^K(s) = \beta_t^{-1} \log \sum_a \exp(\beta_l) K_t(s,a) \tag{A.8}$$

$$\beta_t = \beta\sqrt{t} \tag{A.9}$$

The K-values are the fixed points of an optimistic Bellman operator. The second term in the definition of the K-values represents the influence from the prior over the transitions and rewards and as expected it decreases with increasing number of observations $n(s,a)$. Furthermore, this term reminds of the additive term in discount regularization discussed in section 3.2. Following a Boltzmann policy, over the K-values and choosing an appropriate schedule for increasing the temperature parameter, O'Donoghue et al. (2020) can show that their method will eventually take the optimal action.

A.5. Bellmann Operator

The Bellmann operator $\mathcal{T}$ is an operator that takes a value function V and returns a new value function $\mathcal{T}^\pi V$. For a given policy π, the Bellmann operator is defined as:

$$(\mathcal{T}^{\pi}V)(s) = \sum_{a} \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \right] \tag{A.10}$$

Intuitively, this operator updates the estimate $V(s)$ for how good each state s is based on the current estimate $V(s')$of the successor states s'.

For the optimal value function, the *Bellman optimality operator* is given by:

$$(\mathcal{T}^{*}V)(s) = \max_{a} \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \right] \tag{A.11}$$

A key property of this operator is that it is a contraction mapping. Formally, let (X, d) be a metric space. A function $T : X \to X$ is a contraction if there exists a constant $0 \leq \gamma < 1$ such that:

$$d(T(x), T(y)) \leq \gamma \cdot d(x, y) \quad \text{for all } x, y \in X \tag{A.12}$$

Note that γ is required to be less than 1, meaning that each application of the operator pulls values closer together. In the case of a Bellman operator $\mathcal{T}$, one defines distance between two value functions V and U using the sup-norm:

$$\|V - U\|_{\infty} = \max_{s} |V(s) - U(s)|. \tag{A.13}$$

Then one can show that the Bellman optimality operator $\mathcal{T}^{*}$ satisfies:

$$\|\mathcal{T}^{*}V - \mathcal{T}^{*}U\|_{\infty} \leq \gamma \cdot \|V - U\|_{\infty} \tag{A.14}$$

for $\gamma \in [0, 1)$. Because $\mathcal{T}^{*}$ is a contraction, Banach's Fixed Point Theorem applies and $\mathcal{T}^{*}$ has a unique fixed point — the optimal value function: $V^{*} = \mathcal{T}^{*}V^{*}$. Repeatedly applying the Bellman operator is therefore is guaranteed to converge to V^{*}. This happens at a geometric rate, i.e. after k steps, the error shrinks at rate γ^{k}.

A.6. Determinantal Point Processes

Section 7.2 introduced the definition of k-DPPs. Here, we also provide the definition of the unconstrained DPP, which is mentioned briefly in Section 7.4 and Chapter 8, and give an overview of some useful properties of DPPs based on the exposition by Gautier et al. (2019b).

There are two alternative ways in which a (finite) DPP can be defined. One way is to define a DPP via a *correlation kernel*: Let $\mathcal{Y} = \{1, ..., N\}$ be the ground set. For a positive semidefinite matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$, $0 \preceq \mathbf{K} \preceq I$, it holds for any subset $A \subseteq \mathcal{Y}$

$$\mathbb{P}[A \subseteq Y] = \det \mathbf{K}_A, \tag{A.15}$$

where $\mathbf{K}_A$ is the sub-matrix indexed by A. Hence the 1st-order inclusion probabilities are $p_i = \mathbb{P}[i \in Y] = \mathbf{K}_{ii}$. The other way is to define a DPP via a *likelihood kernel*: For a positive semidefinite matrix $\mathbf{L} \succeq 0$, the DPP is defined by

$$\mathbb{P}[Y = A] = \frac{\det \mathbf{L}_A}{\det(\mathbf{L} + I)}, \tag{A.16}$$

for every $A \subseteq \mathcal{Y}$.The two parametrisations are equivalent through

$$\mathbf{K} = \mathbf{L}(\mathbf{L} + I)^{-1}, \qquad \mathbf{L} = \mathbf{K}(I - \mathbf{K})^{-1}. \tag{A.17}$$

The number of points in a set sampled from a DPP is random and follows a sum independent Bernoulli distributions Bernoulli(λ_j) with the eigenvalues λ_j as the mean parameter. For the expected number of sampled points this means by linearity of expectation:

$$\mathbb{E}[|Y|] = \operatorname{tr} \mathbf{K} = \sum_{i=1}^{N} \mathbf{K}_{ii}. \tag{A.18}$$

The *repulsion* or *diversity* property of sampled sets is seen more easily when working with the correlation kernel K. For any $i, j \in \mathcal{Y}$, the following identities hold:

$$\mathbb{P}[i \in Y] = \mathbf{K}_{ii}, \tag{A.19}$$

$$\mathbb{P}[i, j \in Y] = \mathbf{K}_{ii}\mathbf{K}_{jj} - \mathbf{K}_{ij}^2. \tag{A.20}$$

The first equation states that the diagonal entries of $\mathbf{K}$ give the marginal probabilities of inclusion for individual points. The second equation shows that the off-diagonal entries control the pairwise interaction: specifically, the probability that both i and j are included is reduced by $\mathbf{K}_{ij}^2$, which quantifies the *negative correlation* between i and j. Thus, if $\mathbf{K}_{ij}$ is large in magnitude, then i and j are unlikely to co-occur in a sample.

A *projection DPP* is a DPP with a kernel $K \in \mathbb{R}^{N \times N}$ such that:

$$\mathbf{K} = \mathbf{K}^\top = \mathbf{K}^2. \tag{A.21}$$

That is, K is a symmetric idempotent matrix, i.e., an orthogonal projection. The projection

DPP satisfies the inclusion probability condition:

$$\mathbb{P}[A \subseteq Y] = \det(\mathbf{K}_A), \quad \forall A \subseteq \mathcal{Y}, \tag{A.22}$$

where $\mathbf{K}_A$ is the submatrix of $\mathbf{K}$ indexed by A. Moreover, since the eigenvalues of $\mathbf{K}$ are in $\{0, 1\}$, the process selects exactly $k = \operatorname{rank}(\mathbf{K})$ elements with probability one:

$$\mathbb{P}[|Y| = k] = 1. \tag{A.23}$$

A.7. Message Passing

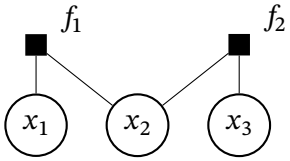


Figure A.1.: Example of a factor graph representing the joint distribution $p(x_1, x_2, x_3 = \frac{1}{Z} \cdot f_1(x_1, x_2) \cdot f_2(x_2, x_3)$. Factor nodes are drawn as black squares.

Message passing is an inference scheme to efficiently compute marginal distributions in a *factor graph*. A factor graph represents the factorization of a joint distribution $p(\mathbf{x})$ over variables $\mathbf{x} = (x_1, ..., x_n)$ and is made up of *variable nodes* $\mathcal{V}$ and *factor nodes* $\mathcal{F}$. Let $\mathcal{V} = \{x_1, \dots, x_n\}$ and $\mathcal{F} = \{\psi_a : a \subseteq \mathcal{V}\}$ denote the corresponding sets of nodes. There is an edge between a variable node x_i and factor node ψ_a is $x_i \in a$. The implied joint distribution factorises as

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{a \in \mathcal{F}} \psi_a(\mathbf{x}_a) \tag{A.24}$$

where $\mathbf{x}_a$ is the tuple of variable nodes neighbouring factor node ψ_a and Z is the partition function. Figure A.1 shows an example. Inference in a factor model is performed by passing messages between factor nodes and variable nodes. The messages for an edge between a variable node x_i and factor node a have the following form:

$$m_{i \to a}(x_i) = \prod_{b \in \mathcal{F}(i) \setminus \{a\}} m_{b \to i}(x_i) \tag{A.25}$$

$$m_{a \to i}(x_i) = \sum_{\mathbf{x}_a \setminus x_i} \psi_a(\mathbf{x}_a) \prod_{j \in \mathcal{V}(a) \setminus \{i\}} m_{j \to a}(x_j), \tag{A.26}$$

where $\mathcal{F}(i)$ denotes the neighbouring factor nodes of variable node x_i and $\mathcal{V}(a)$ denotes the neighbouring variable nodes of factor node ψ_a, respectively. Due to the form of $m_{a \to i}(x_i)$ the algorithm is also known as the sum-product-algorithm.

The marginal distribution $p_i(x_i)$ over variable x_i is obtained by multiplying and normalizing all incoming messages for variable node x_i:

$$p_i(x_i) = \frac{1}{Z_i} \prod_{a \in \mathcal{F}(i)} m_{a \to i}(x_i), \tag{A.27}$$

For a derivation of the message and marginal distribution, see e.g. Chapter 8 in (Bishop and Nasrabadi, 2006). On a tree-structured factor graph the algorithm converges in a finite number of iterations equal to the tree diameter and computes the exact marginals. In general graphs, the computed marginals are only an approximation.

A.8. Gaussian Processes

This section provides definitions of the Gaussian distribution and Gaussian processes, introduces RKHSs, and presents examples of kernel functions in conjunction with Gaussian process, which we referred to throughout the main text.

Definition A.8.1 (Multivariate Gaussian distribution) *A random vector $\mathbf{x} \in \mathbb{R}^d$ follows a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$, written as:*

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{A.28}$$

if its probability density function (pdf) is given by:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{A.29}$$

where:

- $\mathbf{x} \in \mathbb{R}^d$ *is the random vector,*
- $\boldsymbol{\mu} \in \mathbb{R}^d$ *is the mean vector,*
- $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ *is the covariance matrix (must be symmetric and positive semi-definite),*

Definition A.8.2 (Gaussian process) *A Gaussian process is fully specified by:*

- *a mean function $m(\mathbf{x})$, and*
- *a covariance function (or kernel) $k(\mathbf{x}, \mathbf{x}')$,*

and is written as:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \tag{A.30}$$

where:

- $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ *are input vectors,*
- $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ *is the mean function,*
- $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$ *is the covariance function.*

A Gaussian process defines a distribution over functions:

For any finite set of input points $\mathbf{x}_1, \ldots, \mathbf{x}_n$*, the outputs* $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ *are jointly Gaussian:*

$$[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^\top \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{A.31}$$

where $\mu_i = m(\mathbf{x}_i)$ *and* $\Sigma_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

Definition A.8.3 (Reproducing Kernel Hilbert Space) *A Hilbert space* $\mathcal{H}$ *functions* $f : \mathcal{X} \to \mathbb{R}$ *is called an RKHS if there exists a function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *such that 1. for every* $x \in \mathcal{X}$*, the function* $f(x') = k(x, x')$ *is in* $\mathcal{H}$*, and 2.* k *has the reproducing property that the values of every* $f \in \mathcal{H}$ *at any* $x \in \mathcal{X}$ *can be written as* $f(x) = \langle f(\cdot), k(\cdot, x) \rangle$ *(where* $\langle \cdot, \cdot \rangle$ *is the inner product of* $\mathcal{H}$*).*

Some common kernel functions are as follows:

- **Squared Exponential (RBF / Gaussian) Kernel:**

$$k_{\text{SE}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\ell^2}\right)$$

- **Laplace Kernel:**

$$k_{\text{Laplace}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|}{\ell}\right)$$

- **Matérn Kernel (with smoothness parameter ν):**

$$k_{\text{Matérn}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|\boldsymbol{x} - \boldsymbol{x}'\|}{\ell}\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\|\boldsymbol{x} - \boldsymbol{x}'\|}{\ell}\right)$$

  where K_ν is the modified Bessel function of the second kind.

- **Wiener Kernel:**

$$k_{\text{Wiener}}(x, x') = \sigma^2 \min(x, x')$$

  (defined for scalar inputs $x, x' \geq 0$)

- **Polynomial Kernel (degree d):**

$$k_{\text{Poly}}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$$

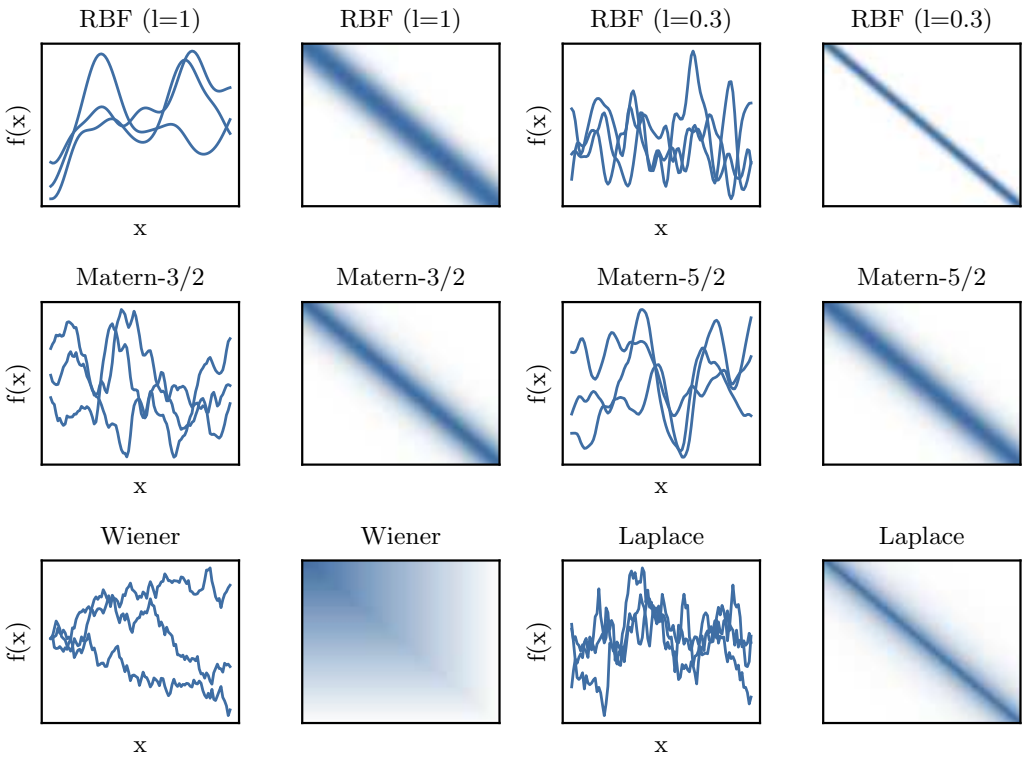


Figure A.2.: Samples from a Gaussian Process for different choices of the kernel matrix (even-numbered columns). The corresponding kernel Gram matrices are shown in the adjacent odd-numbered columns. Blue indicates higher values of the kernel Gram matrix.

where $c \geq 0$ is a constant offset.

Figure A.2 shows samples from a Gaussian process as well as the corresponding covariance matrices for different choices of kernels.

Appendix B

Additional Material for Chapter 4

B.1. Theoretical Analysis

B.1.1. Background

The following three facts will be used during the analysis:

1. **Deviation inequality** For a sample $f \sim \mathcal{G}(0, k)$ from a centered Gaussian process with kernel function k, it holds

$$\forall u, \mathbb{P}(|f(x) - f(y)| \geq u) \leq 2 \exp\left(-\frac{u^2}{2d(x,y)^2}\right), \tag{B.1}$$

   where $d(x, y) = \sqrt{k(x, x) + k(y, y) - 2k(x, y)}$.

2. **Hölder's inequality** Let $a_1, ..., a_N, b_1, ..., b_N$ be real numbers.

$$\sum_{n=1}^{N} |a_n b_n| \leq \left(\sum_{n=1}^{N} |a_n|^p\right)^{1/p} \left(\sum_{n=1}^{N} |b_k|^q\right)^{1/q} \tag{B.2}$$

   with $p, q \geq 1$ and $\frac{1}{p} + \frac{1}{q} = 1$

3. **Growth of Harmonic numbers** The N-th Harmonic number $H_N = \sum_{n=1}^{N} \frac{1}{n}$ grows logarithmically in N:

$$H_N \in \Theta(\log N) \tag{B.3}$$

B.1.2. Upper bounds on the supremum of a cell

Lemma B.1.1 (Bound on supremum) Assume $\mathcal{X}$ is finite and $f \sim \mathcal{GP}(0, k)$. Pick $\epsilon \in (0, 1)$ and set $\beta_n = 2 \log(2|\mathcal{X}_n|N/\epsilon)$. Then

$$\forall n \sup_{x \in \mathcal{X}_n} |f(x) - f(x_n)| \leq \beta_n^{1/2} \Delta(\mathcal{X}_n)$$

holds with probability $\geq 1 - \epsilon$. $\mathcal{X}_n$ is the cell visited at step n with center x_n and $\Delta(\mathcal{X}_n) = \sup_{x \in \mathcal{X}_n} d(x, x_n)$.

Proof of Lemma B.1.1 Fix n. Applying a union bound and B.1, one obtains for all u_n

$$\mathbb{P}(sup_{x\in\mathcal{X}_n}|f(x)-f(x_n)|\geq u_n) \tag{B.4}$$
$$\leq \sum_{x\in\mathcal{X}_n}\mathbb{P}(|f(x)-f(x_n)|\geq u_n) \tag{B.5}$$
$$\leq 2\sum_{x\in\mathcal{X}_n}\exp\left(-\frac{u_n^2}{2d(x,x_n)^2}\right) \tag{B.6}$$
$$\leq 2|\mathcal{X}_n|\exp\left(-\frac{u_n^2}{2\Delta(\mathcal{X}_n)^2}\right) \tag{B.7}$$

For $u_n = \beta_n^{1/2}\Delta(\mathcal{X}_n)$, it holds with probability $1-\epsilon/N$, that $sup_{x\in\mathcal{X}_n}|f(x)-f(x_n))|\leq \beta_n^{1/2}\Delta(\mathcal{X}_n)$. Taking another union bound over N the statement holds.

B.1.3. Upper bound on the regret

Lemma B.1.2 (Bound on regret) Running GP-OO with β_n as specified in Lemma B.1.1 and the canonical pseudo-metric d, the simple regret $r_n = f^* - f_n$ is bounded by $\beta_n^{1/2}\Delta(\mathcal{X}_n)$ for all n with probability $1-\epsilon$.

Proof of Lemma B.1.2 This statement can be shown with typical arguments from the literature on optimistic optimization. We consider the cases $x^*\in\mathcal{X}_n$ and $x^*\notin\mathcal{X}_n$ separately.

Case 1: $x^*\in\mathcal{X}_n$

$$r_n = f^* - f_n = sup_{x\in\mathcal{X}_n}|f(x)-f(x_n)|\leq \beta_n^{1/n}\Delta(\mathcal{X}_n) \tag{B.8}$$

by Lemma B.1.1.

Case 2: $x^*\notin\mathcal{X}_n$.
Because x_n was explored nevertheless, there is a node $x_{n'}$ on the optimal path with $x^*\in\mathcal{X}_{n'}$, that was explored at step $n'<n$, such that

$$f(x_n)+\beta_n^{1/2}\Delta(\mathcal{X}_n)\geq f(x_{n'})+\beta_{n'}^{1/2}\Delta(\mathcal{X}_{n'}) \tag{B.9}$$

Then, $f(x_{n'})-f(x_n)\leq\beta_n^{1/2}\Delta(\mathcal{X}_n)-\beta_{n'}^{1/2}\Delta(\mathcal{X}_{n'})$. For the regret one obtains in combination with Lemma B.1.1:

$$r_n = f^* - f_n \tag{B.10}$$
$$\leq \left[f(x^*)-f(x_{n'})\right]+\left[f(x_{n'})-f(x_n)\right] \tag{B.11}$$
$$\leq \left[\beta_{n'}^{1/2}\Delta(\mathcal{X}_n')\right]+\left[\beta_n^{1/2}\Delta(\mathcal{X}_n)-\beta_{n'}^{1/2}\Delta(\mathcal{X}_{n'})\right] \tag{B.12}$$
$$= \beta_n^{1/2}\Delta(\mathcal{X}_n) \tag{B.13}$$

Proposition B.1.3 (Bound on regret) Assume $\mathcal{X}$ is finite and $f \sim \mathcal{GP}(0, k)$. Pick $\epsilon \in (0, 1)$ and set $\beta_n = 2\log(2|\mathcal{X}_n|N/\epsilon)$. For GP-OO with $k = 2$ children nodes, we obtain the following bound on the cumulative regret

$$\mathbb{P}(R_N \leq \beta^{1/2} \sum_{n=1}^{N} \Delta(\lfloor \log n \rfloor)) \geq 1 - \epsilon \tag{B.14}$$

where $\Delta(h)$ is the radius of a cell at depth h and $\beta = \max\{\beta_1, ..., \beta_N\}$.

Proof Proposition ?? Simple consequence from Lemma B.1.2, where we assume the worst-case of a uniformly growing tree. The depth of a node after n steps is at least $\lfloor \log(n) \rfloor$ in a uniformly growing tree.

B.1.4. Bounds for common kernels

The following analysis is restricted to Gaussian process's, where the canonical pseudo-metric d satisfies:

Assumption 1. There exist $C > 0, \alpha > 0$ such that $d(x, y) \leq C\|x - y\|_2^\alpha$, where $\|\cdot\|_2$ is the Euclidean norm. We additionally require $m/\alpha > 1$, where m is the dimension of the domain.

According to Shekhar and Javidi (2018) the first part of Assumption 1 holds for the squared exponential kernel with $C = \sqrt{2/l}, \alpha = 1$ and the Matern kernels with half integer values. For $\nu = 1/2$, one has $\alpha = 1/2$ and for all other half-integer values $\alpha = 1$.

Lemma B.1.4 (Bubeck et al. (2011)) Assume that $\mathcal{X}$ is a m-dimensional hypercube $[0, 1]^m$ and consider the dissimilarity $d(x, y) = C\|x - y\|_2^\alpha$, where $C > 0, \alpha > 0$. Define the partitioning by recursively splitting the hypercube in the middle along its longest side (ties broken arbitrarily). One has

$$\Delta(\mathcal{X}_n) \leq diam(\mathcal{X}_h) \leq C(2\sqrt{m})^\alpha \left(\frac{1}{2^{\alpha/m}}\right)^h$$

for the cell $\mathcal{X}_h$ of a node at depth h.

Proof of Lemma B.1.4. See Example 1 in Bubeck et al. (2011).

Proposition B.1.5 (Regret for common kernels) *Assume the Gaussian process's canonical pseudo-metric d satisfies $d(x, y) = C\|x - y\|_2^\alpha$, where $C > 0, \alpha > 0, m/\alpha > 1$. Running GP-OO on a finite domain $\mathcal{X} \subset [0, 1]^m$ with regular partitions, one has a worst-case cumulative regret R_N of*

$$\tilde{\mathcal{O}}(N^{1-\alpha/m}(\log N)^{\alpha/m})$$

with high probability. The $\tilde{\mathcal{O}}(\cdot)$ notation supresses poly-logarithmic factors.

Proof of Proposition B.1.5 It follows from Proposition B.1.3 that $R_N \in \tilde{\mathcal{O}}(\sum_{n=1}^{N} \Delta(\lfloor \log n \rfloor))$ with probability $1-\epsilon$. It remains to bound $\sum_{n=1}^{N} \Delta(\lfloor \log n \rfloor)$. For Equation (B.15), we use Lemma B.1.4 and for Equation (B.20), we apply Hölder's inequality (B.2) with $q = m/\alpha$ and $p = \frac{1}{1-\alpha/m}$.

$$\sum_{n=1}^{N} \Delta(\lfloor \log n \rfloor) \leq C\big(2\sqrt{m}\big)^{\alpha} \sum_{n=1}^{N} \left(\frac{1}{2^{\alpha/m}}\right)^{\lfloor \log n \rfloor} \tag{B.15}$$

$$\leq C(2\sqrt{m})^{\alpha} \sum_{n=1}^{N} \left(\frac{1}{2^{\alpha/m}}\right)^{\log n - 1} \tag{B.16}$$

$$= C(2\sqrt{m})^{\alpha} 2^{\alpha/m} \sum_{n=1}^{N} \left(\frac{1}{2^{\alpha/m}}\right)^{\log n} \tag{B.17}$$

$$= C_1 \sum_{n=1}^{N} \left(\frac{1}{2^{\alpha/m}}\right)^{\log n} \tag{B.18}$$

$$= C_1 \sum_{n=1}^{N} \left(\frac{1}{n^{\alpha/m}}\right) \tag{B.19}$$

$$\leq C_1 \left(\sum_{n=1}^{N} 1\right)^{1-\alpha/m} \left(\sum_{n=1}^{N} \left(\frac{1}{n^{\alpha/m}}\right)^{m/\alpha}\right)^{\alpha/m} \tag{B.20}$$

$$= C_1 N^{1-\alpha/m} H_N^{\alpha/m} \tag{B.21}$$

where $C_1 = C(2\sqrt{m})^{\alpha} 2^{\alpha/m}$ and H_N being the N-th harmonic number. Together with (B.3), this implies:

$$\sum_{n=1}^{N} \Delta(\lfloor \log n \rfloor) \in \mathcal{O}(N^{1-\alpha/m} (\log N)^{\alpha/m}) \tag{B.22}$$

B.2. Additional Experimental Details

The experiments were implemented in Python 3.9.1. and run on a machine with macOS 12.3.1, a 4 GHz Quad-Core Intel Core i7 CPU and 32 GB RAM or on a machine with macOS 12.2.1, a 2,7 GHz Dual-Core Intel Core i5 CPU and 16 GB RAM in the case of the Experiment in Section 6.1.

B.2.1. Experiment with benchmark functions

Table E.1 lists the domains and hyperparameters used in the experiment with the benchmark functions. We ran TurBO in its default configuration with 5 trust regions and a batch size of 10. For each run, we sampled a sub-domain by choosing uniformly at random new lower and upper boundaries for the intervals along each dimension, such that the new

Benchmark	Domain	Lengthscale l	β (GP-UCB)	β (GP-OO)	jitter (EI)
Branin	$[-15, 15]^2$	0.5	1	100	0.001
Six-Hump-Camel	$[-2, 2]^2$	0.5	1	10	0.0001
Beale	$[-4.5, 4.5]^2$	1	0.1	100	0.0001
Bohachevsky a	$[-100, 100]^2$	1.7	0.1	10	1
Bohachevsky b	$[-100, 100]^2$	1.7	1	10	0.1
Bohachevsky c	$[-100, 100]^2$	1.7	0.1	10	0.0001
Rosenbrock	$[-3, 3]^2$	0.7	1	100	0.0001
Ackley	$[-35, 35]^2$	3.5	1	10	0.001
Hartmann	$[0, 1]^3$	0.3	1	0.1	0.0001
Trid	$[-16, 16]^4$	10.75	1	100	0.0001
Shekel	$[0, 10]^4$	1.75	1	10	0.001
Dixonprice	$[-10, 10]^{10}$	2	10	0.1	0.1

Table B.1.: Hyperparameters for experiment with benchmark functions

boundaries are in between the previous ones and the location of the minimum. In this way, the location of the minimum stays the same as in the original domain.
Figure B.2 shows the minimal function value found for the benchmark functions and Figure B.3 shows the cumulative runtime. While in terms of the number of function evaluations, GP-OO does not improve over existing methods, it is the fastest.

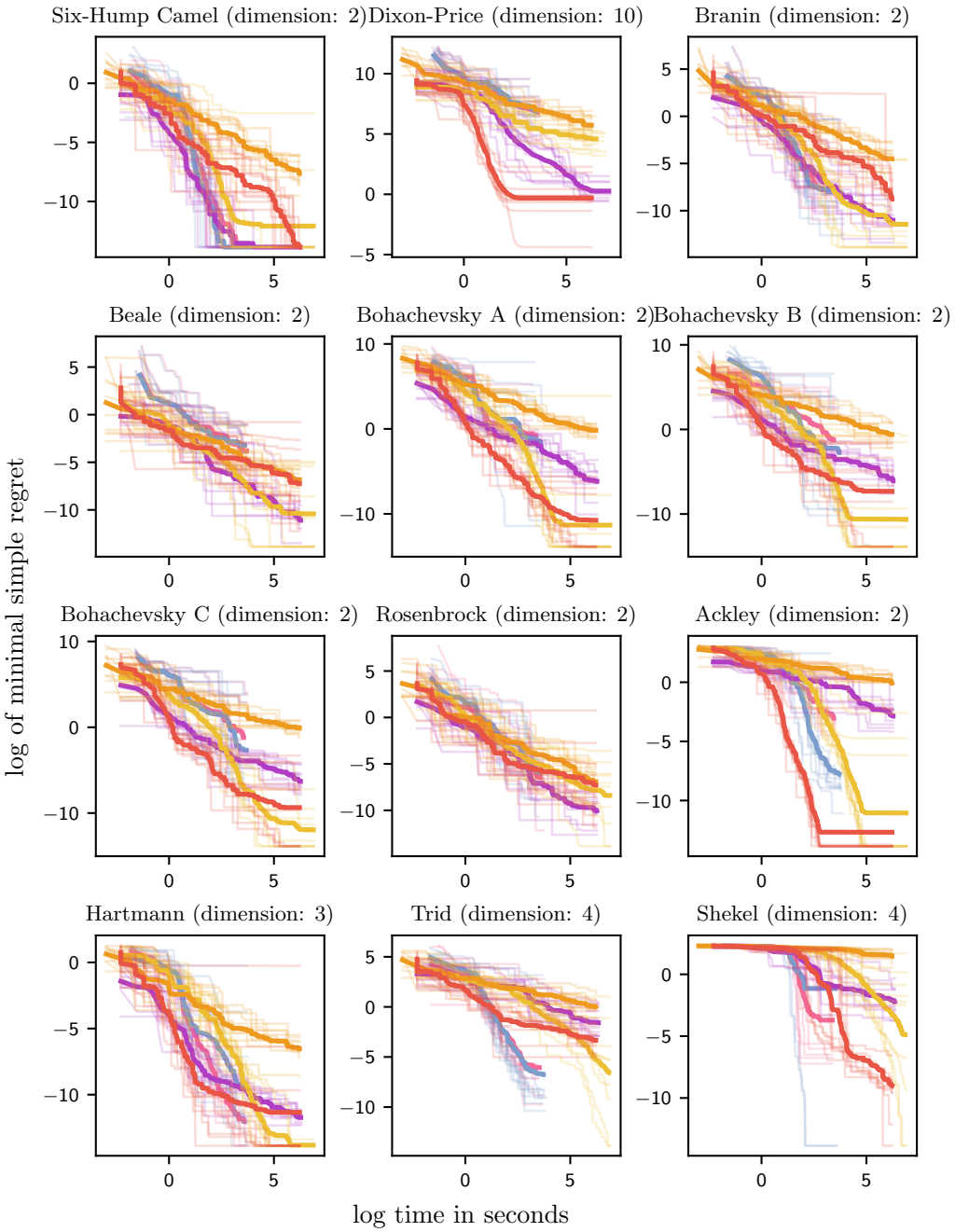


Figure B.1.: Minimal simple regret obtained on common benchmark datasets for simulated function evaluation costs of 0.1 Second. GP-OO is shown in —, GP-UCB in —, EI in —, DIRECT in —, TURBO in — and random selection in —. Averages in bold.

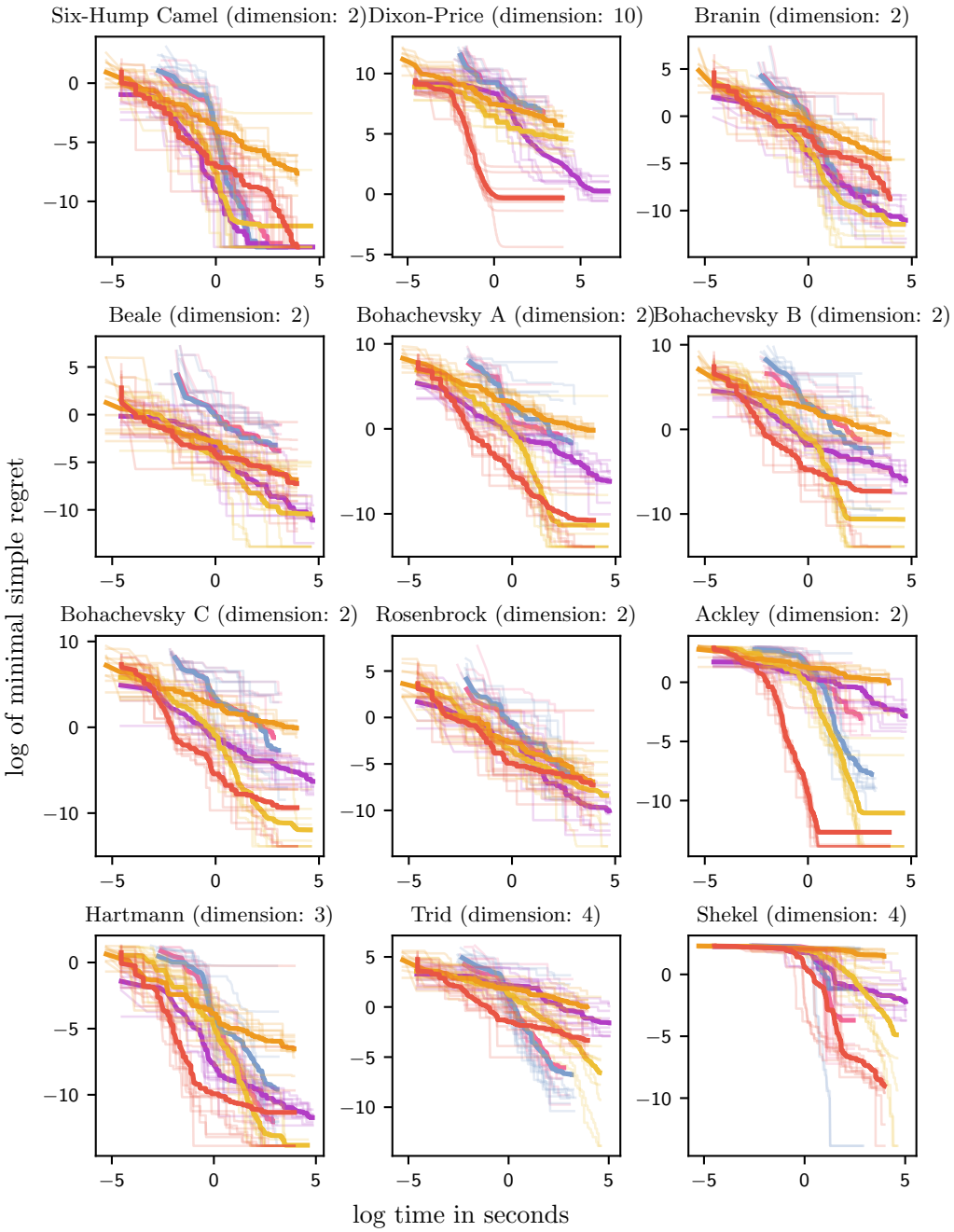


Figure B.2.: Minimal simple regret obtained on common benchmark datasets for simulated function evaluation costs of 0.01 Second. GP-OO is shown in ▬, GP-UCB in ▬, EI in ▬, DIRECT in ▬, TURBO in ▬ and random selection in ▬. Averages in bold.

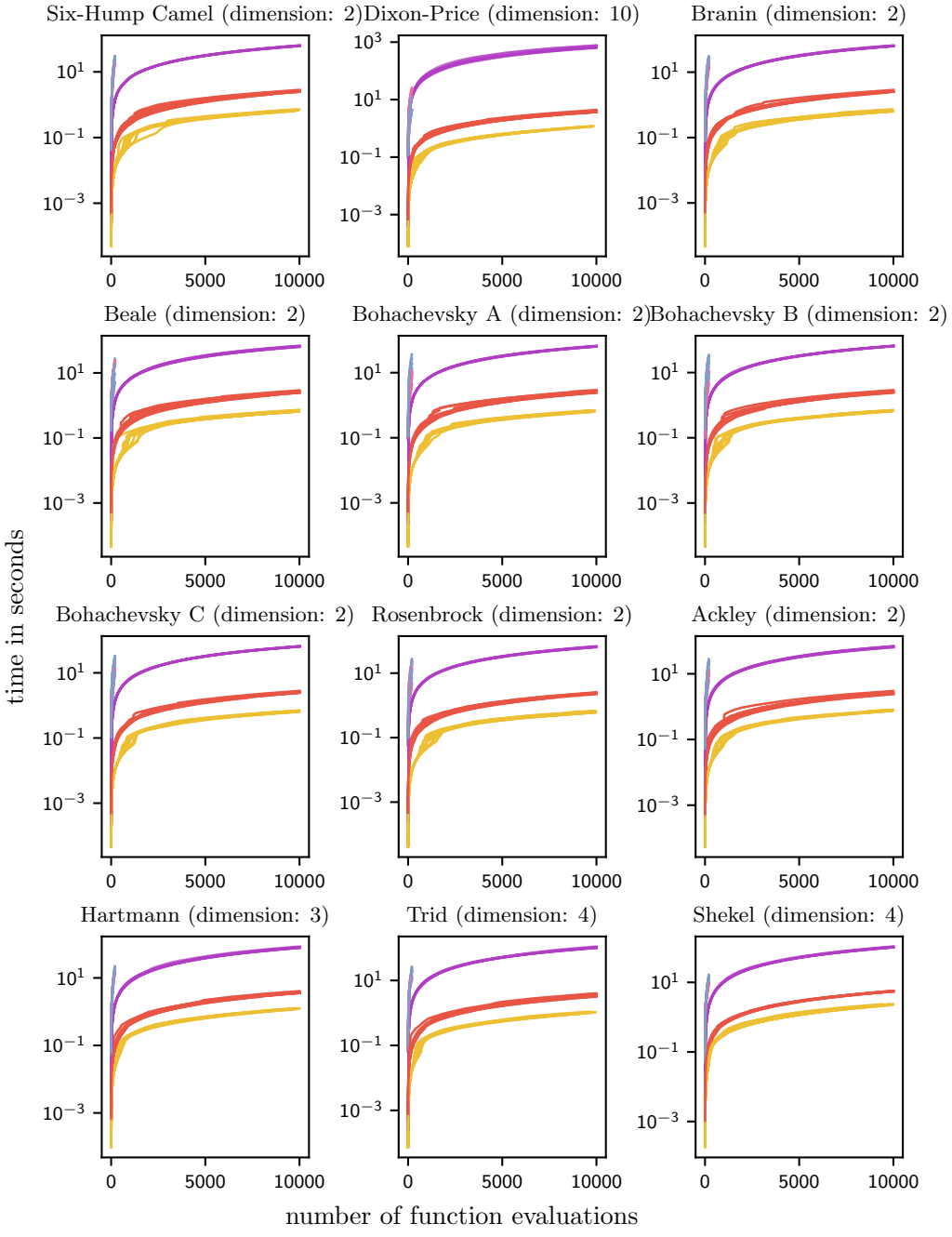


Figure B.3.: Cumulative runtimes for common benchmark datasets. GP-OO is shown in ▬, GP-UCB in ▬, EI in ▬, DIRECT in ▬, TURBO in ▬ and random selection in ▬. Averages in bold.

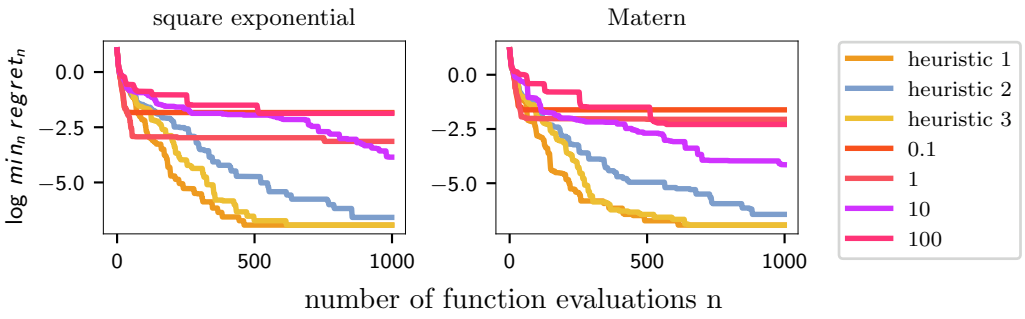


Figure B.4.: Performance of GP-OO for different choices of β_n on 20 samples from a Gaussian process.

	square exponential	Matérn
heuristic 1	15%	0%
heuristic 2	83%	94%
heuristic 3	14%	7%

Table B.2.: Estimated probability that all upper bounds in a tree with 10 levels hold for samples from a Gaussian process.

Heuristical choice of β

Figure B.4 shows GP-OO's performance in the experiment on synthetic samples from a Gaussian process as described in Section 6.1 for three heuristical choices of β_n and constant values of $\beta_n \in \{0.1, 1, 100, 1000\}$. We performed the same analysis for a Matérn kernel with $\nu = 3/2$. According to theory, the upper bounds hold for $\beta_n = 2\log(2|\mathcal{X}_n|N/\epsilon)$ with probability $1 - \epsilon$, but typically a full union bound over both the size of the domain and the total number of observations is too loose. The heuristics are inspired by the observation that the number of "effectively independent" points in the domain is $1/l$ along each dimension. In the following, l denotes the lengthscale and d the dimension of the search domain. C is a constant that we set to 1 for the square exponential kernel and $3/2$ for the Matérn kernel.

- Heuristic 1: $N = 1$ and $\hat{\mathcal{X}}_n = (1/l)^d$
- Heuristic 2: $N = (C/l)^d$ and $\hat{\mathcal{X}}_n = (C/l)^d$
- Heuristic 3: $N = 1000$ (as we run the search for a total of 1000 iterations) and $\hat{\mathcal{X}}_n = (1/l)^d$

In addition, we empirically estimated the probability with which our heuristical choices for β result in valid upper bounds. To do so, we sampled 100 functions from a Gaussian process and for each sample built a fully balanced search tree up to depth 10. For each sampled function and each node, we checked if the upper-bound holds for the corresponding cell. The following table shows the percentage of samples for which *all* upper-bounds from depth 0 to depth 10 did hold. All functions were sampled on a $25 \times 25 \times 25$ grid with a square exponential and Matérn kernel with lengthscale 0.2. ϵ is set 0.05, so one would expect 95% of the bounds to hold under a union bound approach. For the results, see Table B.2. Higher percentages indicate the β is too conservative and lower percentages indicate that the optimization might be too greedy. We find that, even though the latter is the case for our heuristics, this does not immediately lead to decreased search performance as shown by Figure B.4.

Table B.3.: Discretization

dimension/lengthscale	3/1	3/0.1	3/0.05	2/0.5	2/0.05	2/0.005	1/0.5	1/0.05	1/0.005
discretization per dimension	40	40	40	100	200	500	1000	1000	2000

B.2.2. Ablation: Does encoding the smoothness help?

We compare GP-OO to DiRect (Jones and Martins, 2021), a global optimization algorithm that is also based on iterative refinement of axis-aligned rectangles. DiRect assumes that the function fulfills a Lipschitz condition as well, however it does not allow specifying it. With the following experiment, we investigate whether the Lipschitz assumption encoded in the prior allows GP-OO to improve over the performance of DiRect. To do so, we compare both methods on 20 samples from a Gaussian process prior with a Matérn kernel (Figure B.5) or square exponential kernel (Figure B.6) in different dimensions for different lengthscales. For GP-OO we set the exploration constant β to according to heuristic 1 described in the previous section. The confidence parameter ϵ is set to 0.01. The domain is $[0, 1]^d$ for dimension d. The functions sampled from the Gaussian processes are sampled over grids of different discretizations as specified in Table B.3.
We find that GP-OO can exploit the smoothness in the prior, in particular on domains with smaller lengthscales, i.e. more local optima. DiRect is known to converge slowly in the presence of many local optima as it sometimes spends a large number of function evaluations refining the grid around a local optimum before reaching the basin of a global optimum (Jones and Martins, 2021). Knowing the smoothness of the function might help GP-OO to reach a better trade-off here. For large lengthscales the advantage seems to disappear or even turns around on some samples. A potential reason could be a slight misspecification of the exploration constant beta. However, for this work the settings with small lengthscales are more relevant anyway, since in settings with large lengthscale Bayesian optimization is preferable as one does not need many evaluations there and Bayesian optimization then scales sufficiently well.

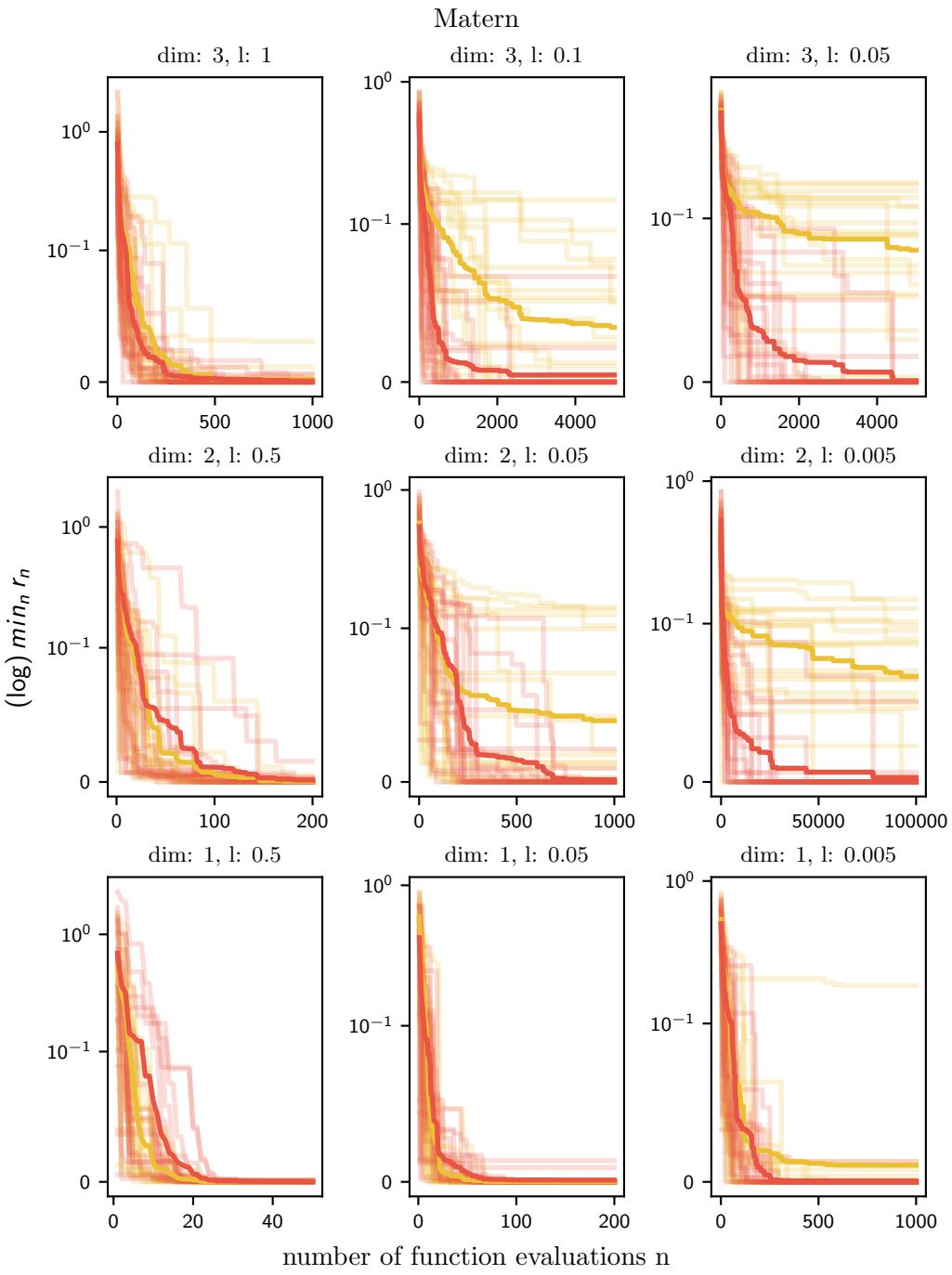


Figure B.5.: Performance of GP-OO (—) and DiRect (—) on 20 samples from a Gaussian process with a with Matérn kernel with different lengthscales and dimensions. The regret is normalized to $[0, 1]$ and the y-axis is semi-logarithmic with a linear threshold at 0.1.

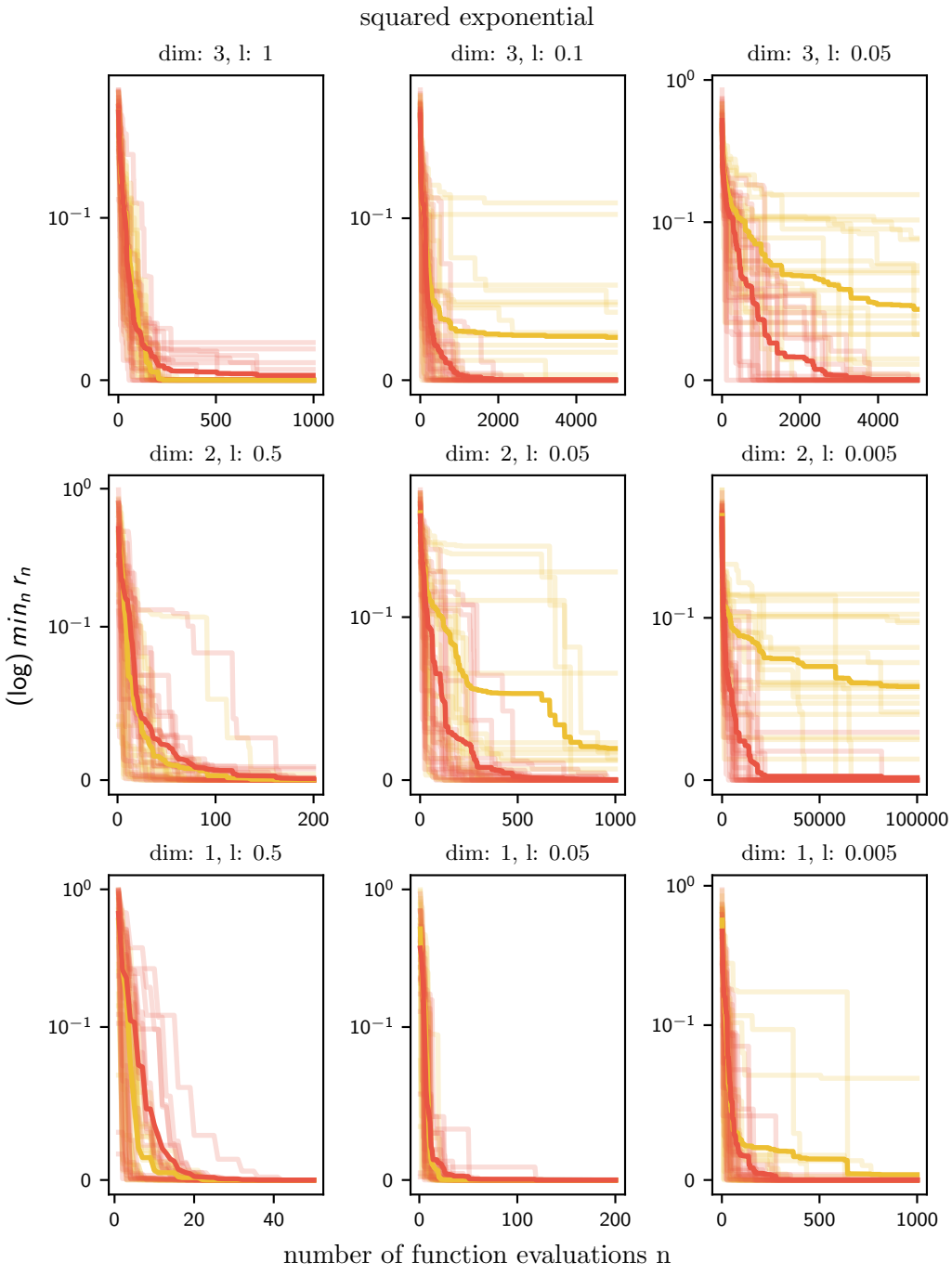


Figure B.6.: Performance of GP-OO (—) and DiRect (—) on 20 samples from a Gaussian process with a with square exponential kernel with different lengthscales and dimensions. The regret is normalized to $[0, 1]$ and the y-axis is semi-logarithmic with a linear threshold at 0.1.

Appendix C

Additional Material for Chapter 5

C.1. A Gaussian Approximation of the Maximum of Gaussian Distributed Variables

Here we give the moments used for the approximations in Section 2 of the paper for the sake of completeness. Their derivations and an extended analysis of the approximation quality of the moment matching can be found in Hennig (2009).

Assume two jointly Gaussian distributed variables x_1 and x_2 with mean $\mu = \begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}$ and covariance $\Sigma = \begin{pmatrix}\sigma_1^2 & \varrho\sigma_1\sigma_2\\ \varrho\sigma_1\sigma_2 & \sigma_2^2\end{pmatrix}$, where ϱ is their correlation. Given prior information $\mathcal{N}(\mu_0, \sigma_0^2)$ on their maximum m, the moments μ_m, σ_m^2 of the distribution of m are:

$$\mu_m = w_1\left[\mu_{c1} + \sigma_{c1}\frac{b_1}{a_1}\frac{\phi(k_1)}{\Phi(k_1)}\right] + w_2\left[\mu_{c2} + \sigma_{c2}\frac{b_2}{a_2}\frac{\phi(k_2)}{\Phi(k_2)}\right] \tag{C.1}$$

$$\begin{aligned}\sigma_m^2 = {} & w_1\left\{[2\mu_{c1}\sigma_{c1}] + \left[2\mu_{c1}\sigma_{c1}\frac{b_1}{a_1} - k_1\sigma_{c1}^2\frac{b_1^2}{a_1^2}\right]\frac{\phi(k_1)}{\Phi(k_1)}\right\}\\ & + w_2\left\{[\mu_{c2}^2 + \sigma_{c2}^2] + \left[2\mu_{c2}\sigma_{c2}\frac{b_2}{a_2} - k_2\sigma_{c2}^2\frac{b_2^2}{a_2^2}\frac{\phi(k_2)}{\Phi(k_2)}\right]\frac{\phi(k_2)}{\Phi(k_2)}\right\},\\ & - \mu_m^2\end{aligned} \tag{C.2}$$

where ϕ is the standard normal probability density function and Φ is the standard normal cumulative distribution function. The remaining terms are defined by:

$$w_1 = Z^{-1}\mathcal{N}(\mu_0;\mu_1,\sigma_0^2+\sigma_1^2)\Phi(k_1) \tag{C.3}$$
$$w_2 = Z^{-1}\mathcal{N}(\mu_0;\mu_2,\sigma_0^2+\sigma_2^2)\Phi(k_2) \tag{C.4}$$
$$a_1 = [\sigma_1^2\sigma_2^2(1-\varrho^2)+(\sigma_1-\varrho\sigma_2)^2\sigma_{c1}^2]^{1/2} \tag{C.5}$$
$$a_2 = [\sigma_1^2\sigma_2^2(1-\varrho^2)+(\sigma_2-\varrho\sigma 1)^2\sigma_{c2}^2]^{1/2} \tag{C.6}$$
$$b_1 = \sigma_{c1}(\sigma_1-\varrho\sigma_2) \tag{C.7}$$
$$b_2 = \sigma_{c2}(\sigma_2-\varrho\sigma_1) \tag{C.8}$$
$$\sigma_{c1}^2 = \frac{\sigma_1^2\sigma_0^2}{\sigma_{c1}^2+\sigma_0^2} \tag{C.9}$$
$$\mu_{c1} = \left(\frac{\mu_1}{\sigma_1^2}+\frac{\mu_0}{\sigma_0^2}\right)\sigma_{c1}^2 \tag{C.10}$$
$$Z = \mathcal{N}(\mu_0;\mu_1,\sigma_0^2+\sigma_1^2)\Phi(k_1)+\mathcal{N}(\mu_0;\mu_2,\sigma_0^2+\sigma_2^2)\Phi(k_2) \tag{C.11}$$
$$k_1 = \frac{(\sigma_1-\varrho\sigma_2)\mu_{c1}-\sigma_1\mu_{c1}-\sigma_1\mu_2+\varrho\sigma_2\mu_1}{[\sigma_1^2\sigma_2^2(1-\varrho^2)+(\sigma_1-\varrho\sigma_2)^2\sigma_{c1}^2]^{1/2}} \tag{C.12}$$
$$k_2 = \frac{(\sigma_2-\varrho\sigma_1)\mu_{c2}-\sigma_2\mu_1+\varrho\sigma_1\mu_2}{[\sigma_1^2\sigma_2^2(1-\varrho^2)+(\sigma_2-\varrho\sigma_1)^2\sigma_{c2}^2]^{1/2}} \tag{C.13}$$

C.2. Additional Experimental Details

This section provides additional experimental details for the MNIST pixel selection experiment. The tree/dag search methods can be combined with arbitrary classifiers to evaluate the reward at terminal nodes as long as it can handle partial input. We use a simple, fast Convnet (see Section C.2.1). The heuristic used for the UCT in MNIST experiment is given in Section C.2.2.

C.2.1. Convnet for MNIST Experiment

The Convnet is built from two convolutional layers of size $32 \times 3 \times 3$ and $64 \times 3 \times 3$ with stride 1 and zero padding. The second convolutional layer is followed by 2×2 max pooling and two fully connected layers of size 9216×128 and 128×10. The inner layers use ReLUs and the softmax function is applied to the output layer. The first convolutional layer is followed by a 0.25 dropout layer and the first fully connected layer by a 0.5 dropout layer. Images are normalized during training and evaluation such that the pixel values are zero mean and unit standard deviation. During training, all but randomly selected pixels are turned off, i.e. set to zero. We use stochastic gradient descent with the adaptive learning rate method (ADADELTA) and parameters $\epsilon = 10^{-6}$, $\rho = 0.9$. The learning rate is set to $\lambda = 1$ in the beginning and is reduced by a factor of $\gamma = 0.7$ after each epoch. The net is trained for 30 epochs with a batch size of 64.

C.2.2. RAVE heuristic for UCT

UCT in its original form selects new nodes based on the UCB formula. At a MAX node x_j, the child x_i that maximizes

$$\mu_j + \beta\sqrt{\frac{ln(n_i)}{n_j}} \tag{C.14}$$

is chosen, where μ_j is the empirical mean of rewards from roll-outs passing through x_j. The variables n_j and n_i count the visits of node x_i and x_j and are used together with hyperparameter β to control the exploration. The RAVE-heuristic, orginally proposed by Gelly and Silver (2007) and adapted to feature selection settings by Gaudel and Sebag (2010), combines the above selection policy with local and global RAVE scores. The global RAVE score of pixel p $g-\text{RAVE}_p$ indicates the global relevance of p and is calculated as the average of the observed rewards at terminal nodes containing p:

$$g-\text{RAVE}_p = \frac{1}{|T_p|}\sum_{i\in T_p} r_i \tag{C.15}$$

where T_p denotes the set of observed terminal nodes that contain pixel p. The local RAVE-score represents the importance of pixel p at a node x_i, conditioned on the other pixels from node x_i. Here, the average is taken only from roll-outs that passed through node x_i.

$$l-\text{RAVE}_{i,p} = \frac{1}{|\{j \in T_p | i \rightsquigarrow j\}|}\sum_{j\in T_p, i\rightsquigarrow j} r_j \tag{C.16}$$

The modified score trades-off the original UCB term, the global RAVE score and the local one. It further includes a restriction of the exploration term in order to handle problems with high branching factor. The final score of a child x_j reached from parent x_i by adding pixel p is defined as:

$$\begin{aligned}(1-\alpha)\cdot\mu_j + \alpha\cdot((1-\beta)\cdot l-\text{RAVE}_{i,p} + \beta\cdot g-\text{RAVE}_p)\\ +\sqrt{\frac{c_1\ln(n_i)}{n_j}\min\left(\frac{1}{4}, \sigma_j^2 + \sqrt{\frac{2\ln(n_i)}{n_j}}\right)}\end{aligned} \tag{C.17}$$

σ_j^2 denotes the empirical variance of rewards observed from roll-outs passing through node x_j. The trade-off is controlled by parameters α and β:

$$\alpha = \frac{c_2}{c_2 + n_j} \tag{C.18}$$

$$\beta = \frac{c_3}{c_3 + n_l} \tag{C.19}$$

n_l denotes the number of observations that are taken into account for the calculation of the l-RAVE score. In the beginning of the search, the RAVE scores that are biased but calculated from more trials count more. In the course of time, their impact decreases and

the mean estimate μ_j is valued more as it is unbiased, but requires more trials to be reliable. The heuristic involves three hyperparameters c_1, c_2, c_3 that were set to $c_1 = 10^{-4}, c_2 = 10^4$ and $c_3 = 10^4$, respectively.

Appendix D

Additional Material for Chapter 6

D.1. Further Discussions

D.1.1. Prior assumptions

The symmetry assumption for the Dirichlet prior on the LLM's softmax outputs likely does not entirely hold—some words occur more often in natural language than others (Kingsley Zipf, 1932). However, ULTS only requires access to the distribution over the *maximum* of the unexplored part of the search space and not over the *arg max*. The former is not affected by permutations of the entries in the categorical distributions, which is why we suspect that a symmetric Dirichlet distribution with sufficiently small concentration parameter is a good proxy. Moreover, the empirical prior can also be used to address this limitation, without incurring large overhead.

We assume the same prior for all sentences in a dataset. It could be, though, that some input prompts are easier to complete than others, and the corresponding LLM's outputs are therefore generally have less entropy than on other those of other prompts. This could be counteracted by choosing a personalized hyper-parameter α. This would require deriving the prior for multiple possible values of α, which scales linearly with the number of possible values for α. However, this can be precomputed (as in Alg. 6) such that it would not affect the costs during inference.

D.2. Additional Experimental Details

URLs to the models and datasets used are provided below:

- Models:
  - https://huggingface.co/meta-llama/Llama-2-7b-hf
  - https://huggingface.co/openai-community/gpt2
  - https://huggingface.co/google-t5/t5-large
  - https://huggingface.co/deepseek-ai/deepseek-coder-1.3b-base
- Datasets:
  - https://huggingface.co/datasets/wikipedia
  - https://huggingface.co/datasets/cnn_dailymail

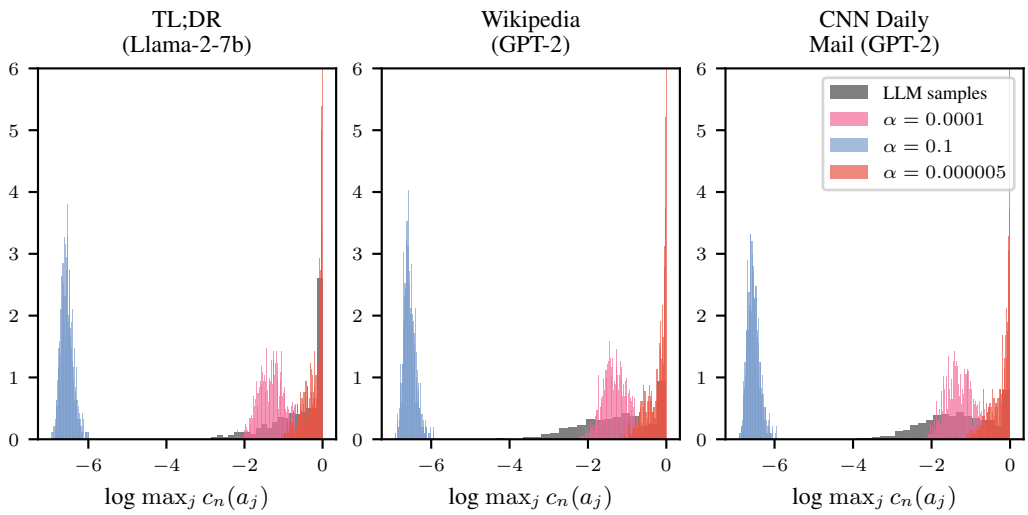


Figure D.1.: Distribution of the maximum of categorical distributions sampled from an LLM, as well as from Dirichlet priors with different concentration parameters.

- https://huggingface.co/datasets/CarperAI/openai_summarize_tldr
- https://huggingface.co/facebook/wmt19-de-en
- https://huggingface.co/datasets/openai/openai_humaneval

D.2.1. Hyperparameters

For beam search, multinomial beam search there are no hyperparameters beyond the beam size k. Speculative Search uses greedy search with n-gram based assisted decoding with `prompt_lookup_num_tokens = 10`. Best-of-k Sampling, we use top-p=0.95. For Nucleus Sampling, we use a temperature parameter of 0.2 and top-p=0.95. For contrastive search, we use penalty parameter $\alpha = 0.6$ and top-k=50. For DoLA, we set the number of DoLA layers to "high". We use version 4.38.2. of the `transformers` package.

D.2.2. Prior Choice for ULTS

We sampled a set $\mathcal{C}$ of Categorical distributions from the LLM on training sequences for each of the datasets and both of the LLMs. We used 1000 samples from the training set for each of the datasets and LLMs we considered. They are used for the empirical prior, as well as training data for fitting the concentration parameter α of the Dirichlet prior. Figure D.1 shows samples for the maximum of Categorical distributions returned by the LLM, as well as the distribution of the maximum of the Categorical distribution from a Dirichlet prior for $\alpha = 10^{-1}, 10^{-4}, 5 \times 10^{-6}$. For the experiments below, we picked a value for α based on visual overlap of those histogramms. For a comparison between different values of the concentration parameter, please refer to fig. D.3 (appendix D.3.3). Note that this way of choosing the hyperparameter α is very convenient compared to performing costly cross-validation.

Table D.1.: Results for the code completion task on HumanEval. ULTS can achieve high performance with minimal latency.

Method	Pass@1 (%) ↑	Log-likelihood ↑	Node expansions ↓	Runtime (s) ↓
Greedy	14.02	-28.000	399.369	13.557 ± 0.493
Nucleus sampling	14.63	-28.500	377.671	10.568 ± 0.429
Beamsearch-Mult (k = 2)	27.44	-24.375	496.963	7.235 ± 0.420
Beamsearch (k = 2)	29.27	-22.092	497.390	8.507 ± 0.520
A-STAR (k_{max} = 2)	34.75	-16.968	267.841	7.184 ± 0.665
ULTS (k_{max} = 2)	25.00	-16.625	134.835	4.447 ± 0.421
ULTS (k_{max} = 3)	34.76	-15.500	146.866	4.936 ± 0.454

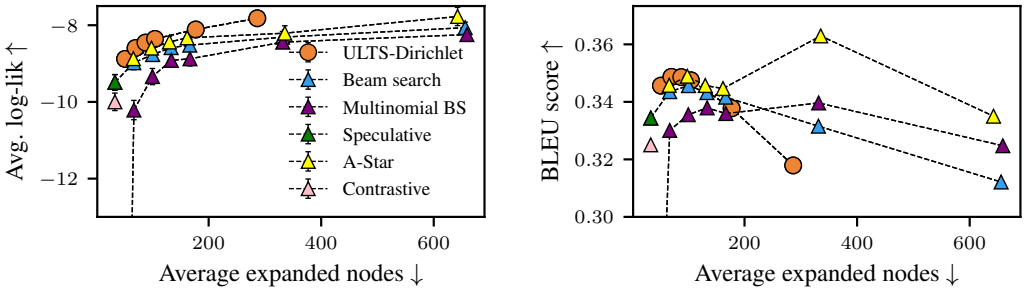


Figure D.2.: Left: Machine translation results with the WMT-19 English-to-German dataset in term of log-likelihood. **Right:** The corresponding BLEU scores.

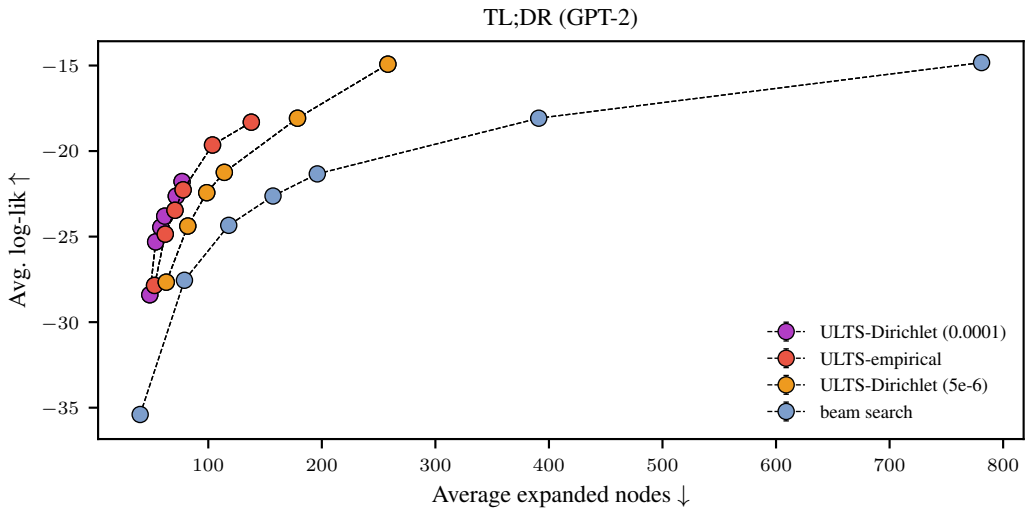
D.3. Additional Experimental Results

D.3.1. ULTS for code completion task

In addition, we test ULTS for code completion using the Deepseek Coder 1.3B LLM (Guo et al., 2024) and all 164 sequences from the OpenAI HumanEval dataset (Chen et al., 2021). The maximum tree depth is set to 500 tokens and we stop at the ⟨EOS⟩ token. We use a Dirichlet prior with $\alpha = 5 \times 10^{-6}$. To allow for a task-specific qualitative comparison, we evaluated the generated token sequences w.r.t. the Pass@1 metric (i.e. percentage of test cases passed), see table D.1. ULTS achieves favorable performance in terms of Pass@1 and log-likelihood while expanding fewer nodes and incurring lower latency.

D.3.2. Machine translation

We compare ULTS with the baselines in a machine translation task with 1000 randomly sampled sequences from the WMT-19 English to German dataset in Figure D.7 in terms of likelihood and BLEU score. For the LLM, we use T5-large (Raffel et al., 2020); for ULTS, we use a Dirichlet prior with $\alpha = 5 \times 10^{6}$. We stop the exploration of a path in the tree if the <EOS> token is found or when the maximum depth of 60 is exceeded. The results are in Figure 3. ULTS is both more efficient (fewer node expansion) and more performant (higher average log-likelihood) at all values of k/k_{max}. For all methods, small to medium beam sizes of $k = 3$ seem to work best in terms of BLEU scores—these scores decrease for a larger (maximum) beam with, in contrast to the log-likelihood in fig. D.7
.



Below, we show some example sequences found with ULTS and beam search. For these examples, we picked sentences where ULTS achieves better BLEU scores. Other examples have mainly the same outputs as beam search while ULTS is more efficient.

D.3.3. Different choice of concentration parameter for the Dirichlet prior

For the Dirichlet prior, indeed α has a meaningful impact on the performance of ULTS: It can be seen as a hyperparameter that trades exploitation for exploitation (smaller value means more exploration). So, the choice of α should be based on how much budget one has since more exploration means more node expansions. Figure D.3 shows a comparison of two different values $\alpha = 0.0001$ and $\alpha = 5e - 6$.

D.3.4. Number of samples in ULTS

Table 2 shows results for an ablation for hyperparameter N on 100 sentences from the machine translation task with maximum beam size $k_{max} = 4$. Only for $N = 1$ the performance is slightly worse, indicating that a small number like $N = 10$ might already be sufficient.

Table D.2.: Results for different choice of number of samples N.

N	Avg. log-lik	node expansions	Time in s
1	-9.454	86.64	3.488 ± 0.187
10	-9.066	90.75	3.631 ± 0.208
100	-9.097	92.28	3.690 ± 0.211
1000	-9.074	92.41	3.718 ± 0.213

Method	k or k_{max}	Time in s
Beam search	5	2.158 ± 0.029
ULTS Dirichlet	3	4.622 ± 0.114
ULTS Dirichlet	5	5.177 ± 0.164

Table D.3.: Total runtime for decoding one of the TL;DR input prompts with Llama-2-7b.

D.3.5. Runtime

We analyze the runtime overhead on top of the LLM's forward pass due to ULTS. Figure D.4 shows the wall-clock time per iteration, averaged over all sentences and all iterations, broken down into the time spent on the LLM's forward pass and the time spent on sampling the optimal values and optimizing the acquisition function. The error bars indicate the 95%- confidence intervals. Results are shown for the experiments with $k_{\text{max}} = 20$ and the Dirichlet prior—the empirical prior performs similarly since ULTS does not differentiate between them in Alg. 7. We note that the runtime overhead of ULTS is small compared to the time spent to do an LLM forward pass. However, note that our current implementation only expands one node of the search tree in each iteration and thereby only evaluate one token sequence per forward-pass. ULTS can be extended similar like Bayesian Optimization can be extended into batch Bayesian Optimization. This is outside of the present work's scope but is a promising direction for future work.

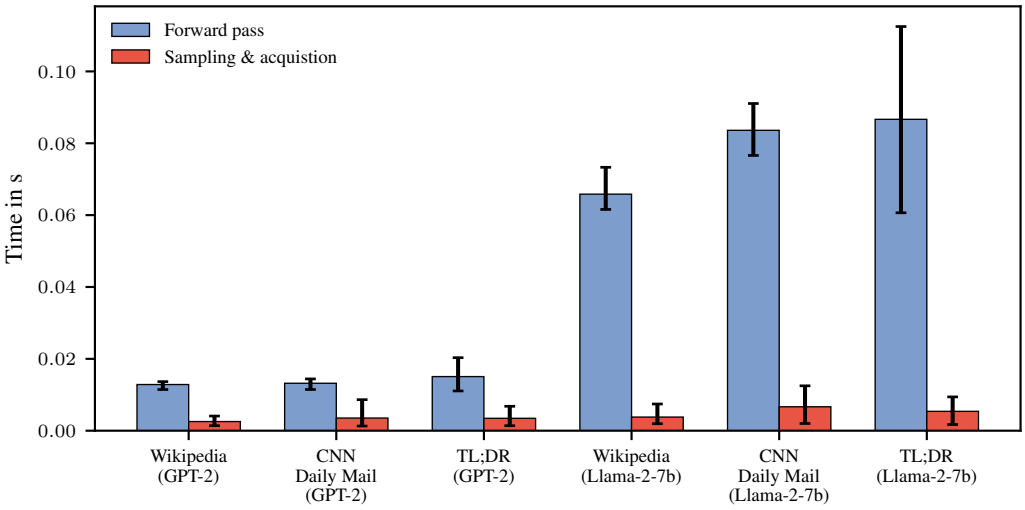


Figure D.4.: Average wall-clock time per iteration. The overhead induced by ULTS is negligible.

In Table 3 we show total runtime results for beam search with beam size $k = 5$ and ULTS with maximal beam size $k_{max} = 3$ and $k_{max} = 5$. Despite expanding fewer nodes than beam search, ULTS is currently slower in settings where different nodes expansion in beam search can be batched. However, note that batching is not always possible, e.g. in memory-constrained settings (the memory resources depend on the model size, sequence length, as well as batch size).

As a reference regarding the computation of the prior: Building the Dirichlet Prior for a tree of depth 250 an branching size 32256 with 1000 samples on a desktop machine (MacBook M1) with CPU only takes 10:50 min (2.6 secs per level of the tree).

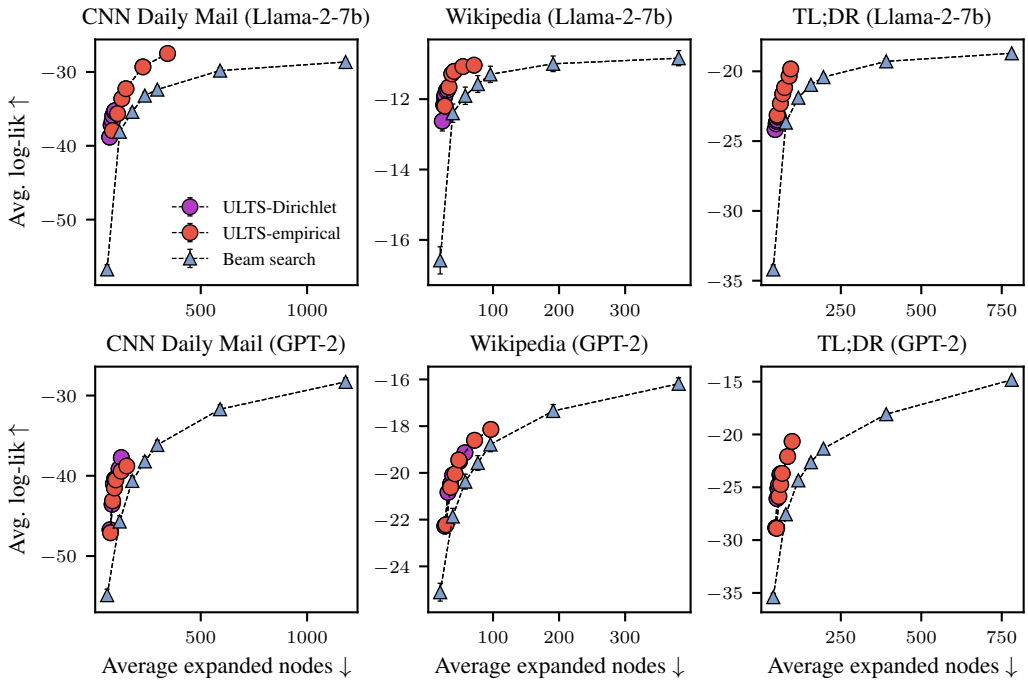


Figure D.5.: Decoding experiments with Llama-2-7b and GPT-2 with the "posterior" strategy.

D.3.6. Alternative acquisition function

As discussed in Section 6.3.2, different selection (and hence backup) strategies can be utilized. Recall that all of our results so far are obtained using the "posterior descendant" strategy defined in Equation 6.3. Here, we show the corresponding results where the other strategy ("posterior") is used instead.

First, Figure D.5 shows results under the same setting as in the main text, but both ULTS-Dirichlet and ULTS-Emp use the "posterior" strategy instead. We noticed that this strategy also performs well—it is more efficient than beam search. Moreover, it also achieves better or similar likelihood than beam search in Llama-2-7b.

We further compare the "posterior" strategy compared to the "posterior descendant" strategy in Figure D.8. We notice that the "posterior" strategy tends to under-explore compared to the "posterior descendant" strategy. Hence, we use and recommend the "posterior descendant" strategy by default. In general, the optimal value estimates from the "posterior descendant" strategy are less optimistic due to Jensen inequality. This effect is larger if the optimal values of children lie closer together. We illustrate this with the following two examples: In the first example, one of the children is much better than the other children. In such scenarios there is no big difference between the back-up-ed values. In the second example, where some children are close together in terms over their optimal value estimates, the "posterior descendant" estimate is lower making it more likely for the other siblings to get explored, too. We therefore hypothesize that the overall effect of the "posterior descendant" strategy is that exploration is a bit more encouraged as also observed empirically in Figure D.8.

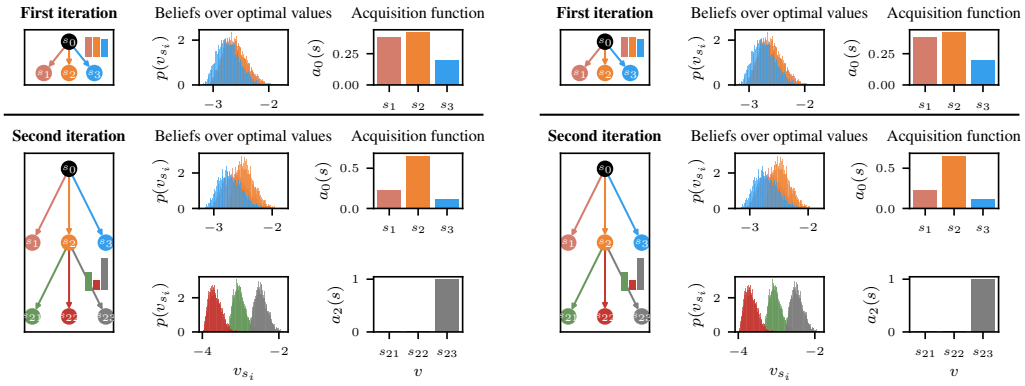


Figure D.6.: Example 1: Left: "posterior descendant" acquisition function.**Right:** "posterior" acquisition function.

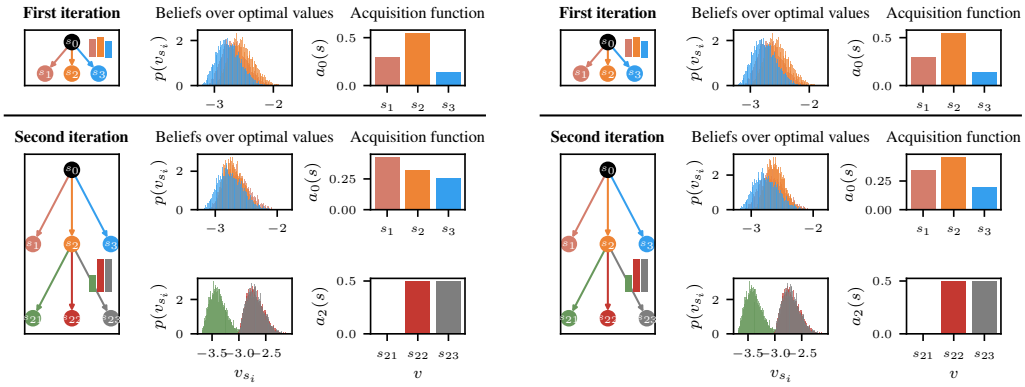


Figure D.7.: Example 2: Left: "posterior descendant" acquisition function **Right:** "posterior" acquisition function.

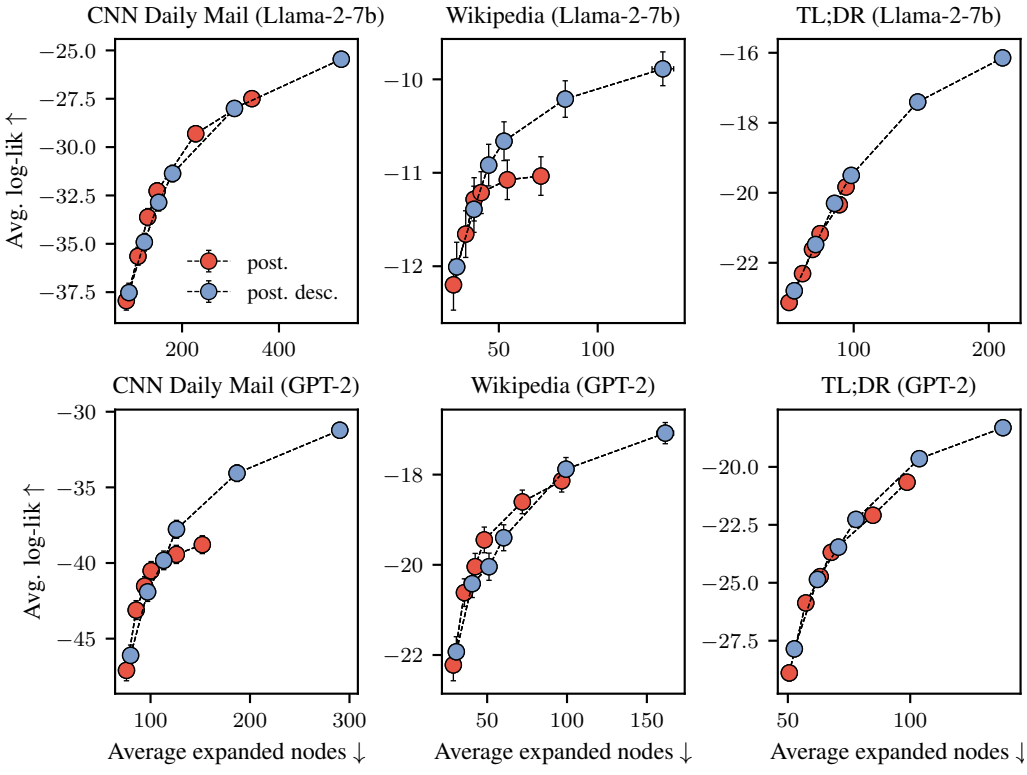


Figure D.8.: "Posterior" vs. "posterior descendant" acquisition function.

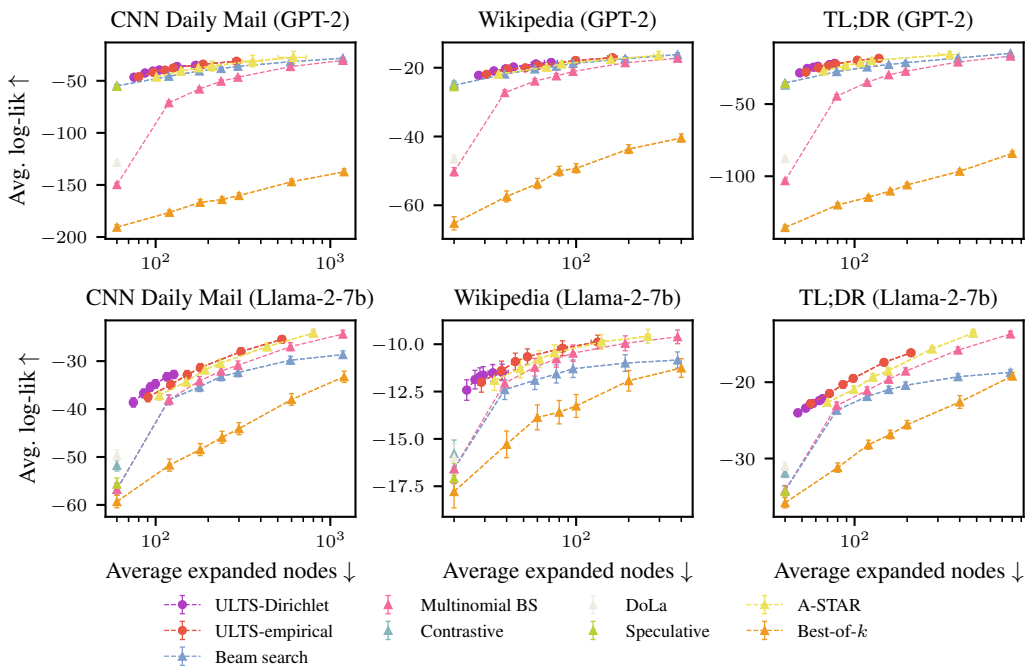


Figure D.9.: Full results for decoding experiments with LLama-2-7b and GPT-2

D.3.7. Full results for decoding experiments with LLama-2-7b and GPT-2

Figure D.9 shows the full results from the decoding experiments with LLama-2-7b and GPT-2 from section 6.5 in the main text.

D.3.8. Experiment with alternative utility function

In the following experiment, we evaluate ULTS with a different utility function – one that additionally contains a repetition-penalty term:

$$\tilde{u}(x_c, \boldsymbol{v}_i) = \mathbb{I}[(\log v_{\hat{x}_c} + \lambda\, b(\hat{x}_c)) = \max_{x_j \in \text{children}(x_i)} (\log v_{\hat{x}_j} + \lambda\, b(\hat{x}_j))], \tag{D.1}$$

where $b(\hat{x}_c)$ is a log-diversity term (Su et al., 2022). In this task, we complete 100 sentences from the Wikipedia dataset with GPT-2. We predict 100 tokens. We define the diversity term $b(x)$ of a node x based on the proportion of duplicated n-grams in the token sequence $(a_1, ..., a_i)$ corresponding to the node. For a token sequence $(a_1, ..., a_i)$, let $\textbf{rep-n}((a_1, ..., a_i)) = \left(1 - \frac{|\text{unique } n\text{-grams}((a_1,...,a_i))|}{|\text{total } n\text{-grams}((a_1,...,a_i))|}\right)$. Following Su et al. (2022), we measure diversity by taking repetitions at different n-gram levels into account:

$$\textbf{diversity}((a_1, ..., a_i)) = \prod_{n=2}^{4} (1 - \textbf{rep-n}((a_1, ..., a_i))).$$

Since we ULTS optimizes the total likelihood of a sequence and not the average likelihood of a sequence, we additionally scale the term with the tree depth, leading to $b(x) = d \cdot \log \textbf{diversity}((a_1, ..., a_i))$ in log-space. For ULTS we use a Dirichlet prior with concentration parameter $\alpha = 5 \times 10^{-6}$, $\epsilon = 0.1$, $k_{max} = 5$. and penalty parameter

Method	Perplexity ↓	Diversity ↑
Beam search	1.16±0.01	0.33±0.01
Nucleus sampling	3.69±0.07	0.82±0.01
Contrastive search	1.27±0.01	0.4±0.01
ULTS	1.16±0.01	0.33±0.01
ULTS with $\lambda = 0.2$	1.3±0.01	0.55±0.01
ULTS with $\lambda = 0.4$	1.45±0.01	0.74±0.01
ULTS with $\lambda = 0.6$	1.5±0.01	0.79±0.01
ULTS with $\lambda = 0.8$	1.53±0.01	0.82±0.01
ULTS with $\lambda = 1.0$	1.55±0.01	0.83±0.01

Table D.4.: Effects of a penalty term in ULTS' acquisition function.

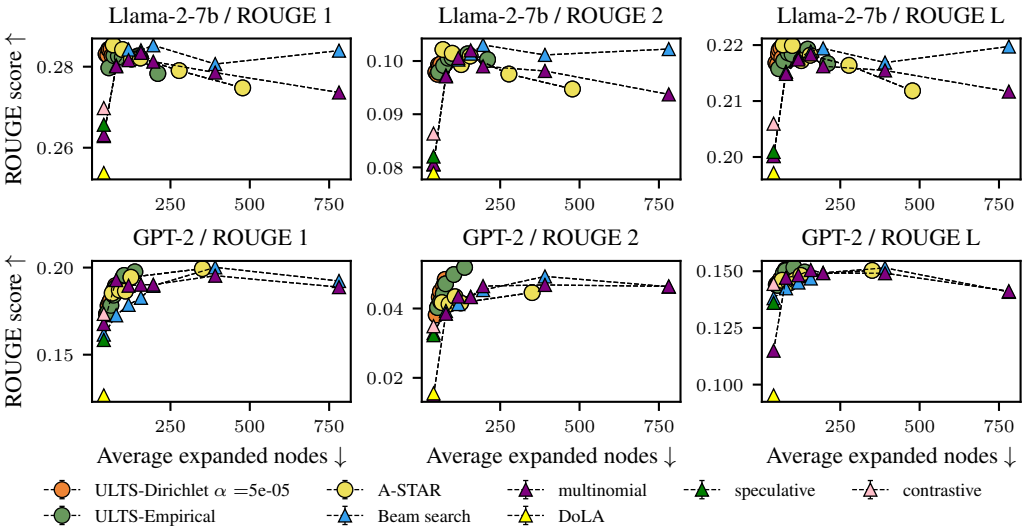


Figure D.10.: ROUGE scores for the summarization task.

$\lambda \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. The parameters for contrastive search are top-$k = 5$ and penalty parameter $\alpha = 0.6$. For beam search, we use $k = 5$. The main paper reports results for ULTS with $\lambda = 0.0$, i.w. without penalty, and $\lambda = 0.2$. The full results are given in Table 3. One can see that with increasing penalty the diversity of the decoded sequences increases, i.e. extending the acquisition function with a repetition term is effective.

D.3.9. Additional metrics

Figure D.10 shows an evaluation in terms of ROUGE metrics for the summarization task on the TL;DR dataset described in Section 6.5. ROUGE 1 quantifies the overlap of unigrams between the produced sentence and ground truth reference, ROUGE 2 refers to bigrams, and ROUGE L refers to the longest commen subsequence. Though there is no perfect correlation between the ROUGE metric and the log-likelihood, ULTS still tends to achieve a good trade-off between performance and node expansions overall. Figure D.11 shows additional results in terms of the BLEURT metric. Here, ULTS performs well for small k_{max} but shows a decrease in performance for larger k_{max}.

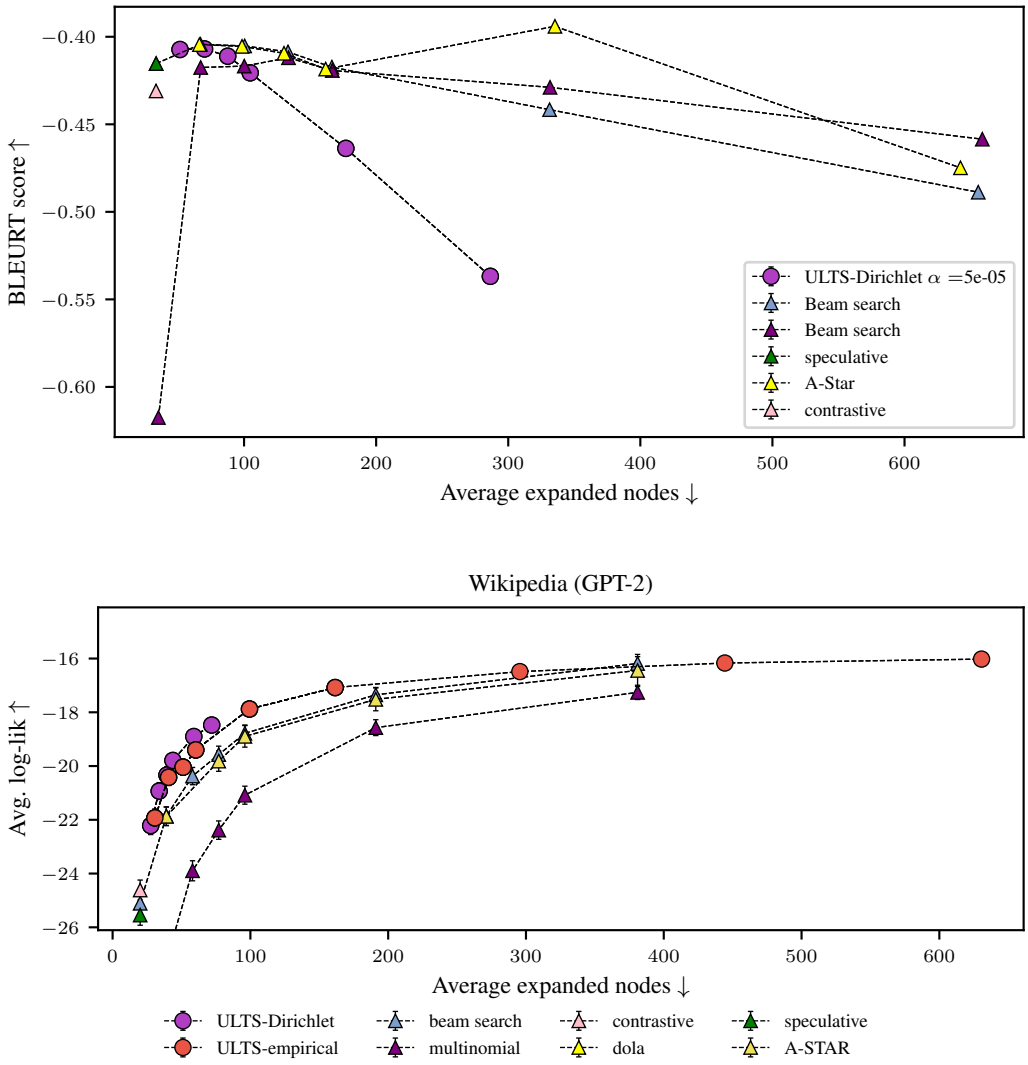


Figure D.12.: Comparison of ULTS with the baselines on the Wikipedia dataset as described in Section 6.5 with additional data points for $k_{\max} \in \{50, 100, 200\}$.

D.3.10. ULTS with larger budget

Figure D.12 shows additional results for ULTS with the empirical prior and GPT-2 on the Wikipedia dataset, as well as ULTS with the Dirichlet prior and t5 on the machine translation task (Figure D.13).

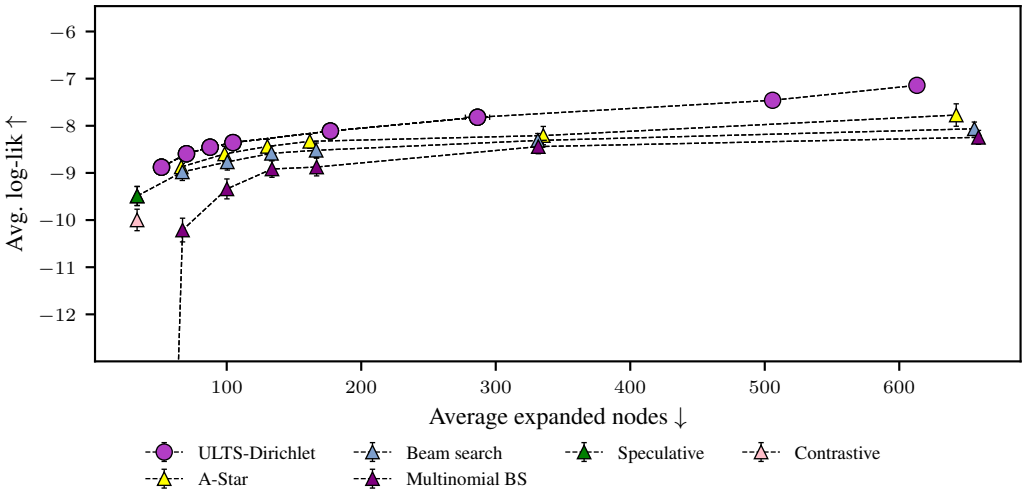


Figure D.13.: Comparison of ULTS in the machine translation task as described in Section 6.5 with additional data points for $k_{\max} \in \{50, 100\}$.

D.4. Pseudocode

Algorithm 7: ULTS

ULTS$(D, N, \{\mathcal{B}_l\}_{l=1}^{D}, \varepsilon, k_{max})$**:**
  $\mathcal{L} \leftarrow \{s_0\}$ // Initialize with root
  for $n \leftarrow 1$ **to** N **do**
    Sample $\Delta_{s_0,n} \sim \mathcal{B}_1(\Delta)$
    $v_{s_0,n} \leftarrow \log \Delta_{s_0,n}$
  $c^* \leftarrow -\infty$, $s^* \leftarrow$ None
  while $\hat{\mathbb{P}}(c^* < v_{s_0}) > \varepsilon$ **do**
    // Selection always starts from root
    $s_i \leftarrow \texttt{select}(s_0, k_{\max})$
    // Expand
    $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{s_i\}) \cup \texttt{children}(s_i)$
    for $s_c \in children(s_i)$ **do**
      // Generate posterior samples
      for $n \leftarrow 1$ **to** N **do**
        Sample $\Delta_{s_c,n} \sim \mathcal{B}_{\texttt{level}(s_c)}(\Delta)$
        $v_{s_c,n} \leftarrow \log c_{s_0 \to s_c} + \log \Delta_{s_c,n}$
      // Update best complete path so far
      if $level(s_c) = D$ **and** $\log c_{s_0 \to s_c} > c^*$ **then**
        $c^* \leftarrow \log c_{s_0 \to s_c}$
        $s^* \leftarrow s_c$
    $\texttt{backup}(\{v_{s_c,n}\}_{n=1}^{N}, s_0 \to s_c)$
    // Termination probability
    $\hat{\mathbb{P}}(c^* < v_{s_0}) \leftarrow \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[c^* \leq v_{s_0,n}]$

Appendix E

Additional Material for Chapter 7

The appendix contains a description of how the algorithm introduced in the main text for the square exponential kernel can be extended to other analytically integrable kernels in E.1, a description of the greedy sampling algorithm in pseudocode in E.2, the proofs of the theoretical results for Section 7.4 of the main paper in E.4, and the additional statements regarding the information gain in E.4.2.

E.1. Other Analytical Kernels

While the Gaussian kernel is the most widely used kernel in machine learning, it has some shortcomings, primarily that it makes very strong smoothness assumptions that can lead to instability in interpolation models. But with some algebraic elbow grease, the scheme of Section 7.5 can be extended to many other popular kernels, assuming they factorize,

$$l(a,b) = \prod_d^D l(a_d, b_d), \tag{E.1}$$

and the indefinite integrals

$$\int l(a,a)\,\mathrm{d}a \qquad \text{and} \qquad \int l(a,b)l(a,c)\,\mathrm{d}a \tag{E.2}$$

are analytically solvable. For example, the above results are applicable to the Matérn class of kernels (Stein, 1999) (including the exponential kernel, which induces the Ornstein–Uhlenbeck process), noting that, assuming w.l.o.g. $x_0 < a < b < x_1$,

$$\int_{x_0}^{x_1} \exp\left(-|x-a|\right)\exp\left(-|x-b|\right)\,\mathrm{d}x = \frac{e^{-a-b}}{2}\left(e^{2a} - e^{2x_0}\right) + (b-a)e^{a-b} + \frac{e^{a+b}}{2}\left(e^{2x_1} - e^{2b}\right), \tag{E.3}$$

and using results such as (see e.g., Gradshteyn and Ryzhik, 2007, Section 2.322)

$$\int xe^{ax}\,\mathrm{d}x = e^{ax}\left(\frac{x}{a} - \frac{1}{a^2}\right), \tag{E.4}$$

$$\int x^2e^{ax}\,\mathrm{d}x = e^{ax}\left(\frac{x^2}{a} - \frac{2x}{a^2} + \frac{2}{a^3}\right), \ldots \tag{E.5}$$

E.2. Pseudocode

Algorithm 8: Greedy sampling from approximate DPPs with analytic kernels on $[0,1]^D$

DrawFromDPP(l, D, k)**:**
  // Initialize statistics of sample as empty
  $X \leftarrow \emptyset$, $m \leftarrow \emptyset$, $\mathbf{M} \leftarrow \emptyset$, $\mathbf{L}^{-1} \leftarrow \emptyset$
  for $i = 1$ **to** k **do**
    $x_i \leftarrow \emptyset$ // Initialize current sample point
    // Draw dimensions iteratively
    for $d = 1$ **to** D **do**
      // Construct function for Equation (20)
      $\mathcal{V} \leftarrow \texttt{construct}(m, \mathbf{M}, \mathbf{L}^{-1})$
      // Draw scaled unit random number
      $u \leftarrow \mathcal{V}(1) \cdot \texttt{rand}(\cdot)$
      $I \leftarrow [0,1]$ // Initialize search interval
      while $|I| > \varepsilon$ **do**
        $\mu \leftarrow \frac{1}{2}(I_0 + I_1)$ // Interval midpoint
        if $\mathcal{V}(\mu) < u$ **then**
          $I \leftarrow [\mu, I_1]$
        else
          $I \leftarrow [I_0, \mu]$
      $x_i \leftarrow [x_i, \mu]$ // Store sampled element
    // Update sample statistics (rank-1)
    $(X, m, \mathbf{M}, \mathbf{L}^{-1}) \leftarrow \texttt{Add}(x_i \,|\, X, m, \mathbf{M}, \mathbf{L}^{-1})$

E.3. Runtime

Let k be the number of points to select, D the dimensionality, N the size of the discretization (regular grid). Then the runtime of the greedy sampling algorithm is $\mathcal{O}(k^3 \log N)$:

- The Gaussian posterior is updated once in each of the k iterations in $\mathcal{O}(k^2)$ by using the matrix-inversion-lemma (see eg. the textbook on Gaussian processes by Rasmussen and Williams (2006), A.3).

- In one iteration, along each of the D dimensions, a bisection search on a grid with discretization size $N^{(1/D)}$ is performed. This gives $\mathcal{O}(D \cdot log(N^{(1/D)})) = \mathcal{O}(log(N))$ steps in total for the bisection search. For k iterations these are $\mathcal{O}(k \cdot log(N))$ steps.

- Each step of the bisection search requires the calculation of Equation 7.29. This expression contains a double sum over the previously observed points. There are at most k of them, so there are at most $\mathcal{O}(k^2)$ summands. Each summand consists of D factors and thereby costs $\mathcal{O}(D)$. This results in $\mathcal{O}(Dk^2)$ per step of the bisection search, $\mathcal{O}(Dk^2 \log(N))$ cost per iteration and $\mathcal{O}(Dk^3 \log(N))$ in total.

E.4. Proofs

For (conditional) entropies and the expected values, we introduce the following notation:

For $\mathbf{S}_{1:i} \sim \pi$,

$$\begin{aligned}
\mathcal{H}(\mathbf{S}_{1:i}) &= -\sum_{S_{1:i}\in E^k} \pi(S_{1:i}) \log \pi(S_{1:i}) \\
\mathcal{H}(\mathbf{S}_i|\mathbf{S}_{1:i-1} = S_{1:i-1}) &= -\sum_{S_i\in E} \pi(S_i|S_{1:i-1}) \log \pi(S_i|S_{1:i-1}) \\
\mathcal{H}(\mathbf{S}_i|\mathbf{S}_{1:i-1}) &= -\sum_{S_{1:i}\in E^i} \pi(S_{1:i}) \log \pi(S_i|S_{1:i-1}) \\
\mathbb{E}_{\mathbf{S}_{1:i}}[h(S_{1:i})] &= \sum_{S_{1:i}\in E^k} \pi(S_{1:i}) h(S_{1:i}) \\
\mathbb{E}_{\mathbf{S}_i|\mathbf{S}_{1:i-1}=S_{1:i-1}}[h(S_{1:i})] &= \sum_{S_i\in E} \pi(S_i|S_{1:i-1}) h(S_{1:i}) \\
H(\mathbf{S}_{1:i}) &= \mathbb{E}_{\mathbf{S}_{1:i}}[h(S_{1:i})] + \mathcal{H}(\mathbf{S}_{1:i})
\end{aligned}$$

In general, $\mathcal{H}(\mathbf{S}_{1:i}) \neq \mathcal{H}(p_\pi)$ and thereby $H(\mathbf{S}_{1:i}) \neq H(p_\pi)$. The chain rule for the entropy is

$$\mathcal{H}(\mathbf{S}_{1:i}) = \sum_{i=1}^{k} \mathcal{H}(\mathbf{S}_i|\mathbf{S}_{1:i-1})$$

For reference, the two statements regarding submodularity and monotonicity:

Proposition E.4.1 Krause et al. (2008) Given any symmetric, positive semi-definite matrix $\mathbf{L} \in \mathbb{R}^{N\times N}$, the function $h(X) = \log\det \mathbf{L}_{XX}$ is submodular, i.e.

$$\forall X_1 \subseteq X_2 \text{ and } i \notin X_2, h(X_1 \cup \{i\}) - h(X_1) \geq h(X_2 \cup \{i\}) - h(X_2)$$

Proposition E.4.2 Sharma et al. (2015) Given any symmetric $\mathbf{L} \in \mathbb{R}^{N\times N}$ with the smallest eigenvalue $\lambda_{min}(\mathbf{L}) \geq 1$, the function $h(X) = \log\det \mathbf{L}_{XX}$ is monotone, i.e.

$$\forall X_1, X_2 \text{ with } X_1 \subseteq X_2 \subseteq X, h(X_1) \leq h(X_2)$$

Lemma E.4.3 Consider two independent random variables $\mathbf{G}_{1:k} \sim \pi_{greedy}$ and $\mathbf{O}_{1:k} \sim \pi$ for an **arbitrary** policy π. In each iteration $i = 0, ..., k-1$, we have

$$H(\mathbf{O}_{1:k}) \leq \mathbb{E}_{\mathbf{G}_{1:i}}\Big[h(G_{1:i})\Big] + k\Big[\mathbb{E}_{\mathbf{G}_{1:i+1}}[h(G_{1:i+1})] - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})] + \mathcal{H}(\mathbf{G}_{i+1}|\mathbf{G}_{1:i})\Big]$$

Proof.

$$
\begin{aligned}
& & H(\mathbf{O}_{1:k}) \\
&\overset{\text{definition of } H}{=} & \mathbb{E}_{\mathbf{O}_{1:k}}[h(O_{1:k})] + \mathcal{H}(\mathbf{O}_{1:k}) \\
&\overset{h \text{ is monotone}}{\leq} & \mathbb{E}_{\mathbf{O}_{1:k}\mathbf{G}_{1:i}}\left[h(O_{1:k} \cup G_{1:i})\right] + \mathcal{H}(\mathbf{O}_{1:k}) \\
&\overset{\text{telescoping sum}}{=} & \mathbb{E}_{\mathbf{O}_{1:k}\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} h(G_{1:i} \cup O_{1:j}) - h(G_{1:i} \cup O_{1:j-1})\right] + \mathcal{H}(\mathbf{O}_{1:k}) \\
&\overset{h \text{ is submodular}}{\leq} & \mathbb{E}_{\mathbf{O}_{1:k}\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} h(G_{1:i} \cup O_j) - h(G_{1:i})\right] + \mathcal{H}(\mathbf{O}_{1:k}) \\
&\overset{\text{entropy chain rule}}{\leq} & \mathbb{E}_{\mathbf{O}_{1:k}\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} h(G_{1:i} \cup O_j) - h(G_{1:i})\right] + \sum_{j=1}^{k} \mathcal{H}(\mathbf{O}_j|\mathbf{O}_{1:j-1}) \\
&\overset{\text{cond. can only decrease entropy}}{\leq} & \mathbb{E}_{\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} \mathbb{E}_{\mathbf{O}_j}[h(G_{1:i} \cup O_j) - h(G_{1:i})] + \mathcal{H}(\mathbf{O}_j)\right] \\
&\overset{\text{summarize}}{=} & \mathbb{E}_{\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} \mathbb{E}_{\mathbf{O}_j}[h(G_{1:i} \cup O_j) - h(G_{1:i})] + \mathcal{H}(\mathbf{O}_j)\right] \\
&\overset{\mathbf{G},\mathbf{O} \text{ indep.}}{=} & \mathbb{E}_{\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} \mathbb{E}_{\mathbf{O}_j}[h(G_{1:i} \cup O_j) - h(G_{1:i})] + \mathcal{H}(\mathbf{O}_j|\mathbf{G}_{1:i} = G_{1:i})\right] \\
&\overset{\text{greedyness}}{\leq} & \mathbb{E}_{\mathbf{G}_{1:i}}\left[h(G_{1:i}) + \sum_{j=1}^{k} \mathbb{E}_{\mathbf{G}_{i+1}}[h(G_{1:i} \cup G_{i+1}) - h(G_{1:i})] + \mathcal{H}(\mathbf{G}_{i+1}|\mathbf{G}_{1:i} = G_{1:i})\right] \\
&\overset{\text{summarize}}{=} & \mathbb{E}_{\mathbf{G}_{1:i}}\left[h(G_{1:i})\right] + k\left[\mathbb{E}_{\mathbf{G}_{1:i+1}}[h(G_{1:i+1})] - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})] + \mathcal{H}(\mathbf{G}_{i+1}|\mathbf{G}_{1:i})\right]
\end{aligned}
$$

Lemma E.4.4 Consider two independent random variables $\mathbf{G}_{1:k} \sim \pi_{greedy}$ and $\mathbf{O}_{1:k} \sim \pi$ for an **arbitrary** policy π. We have

$$(1 - 1/e)H(\mathbf{O}_{1:k}) \leq H(\mathbf{G}_{1:k})$$

Proof. By rearranging the terms from Lemma E.4.3:

$$H(\mathbf{O}_{1:k}) \leq \mathbb{E}_{\mathbf{G}_{1:i}}\left[h(G_{1:i})\right] + k\left[\mathbb{E}_{\mathbf{G}_{1:i+1}}[h(G_{1:i+1})] - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})] + \mathcal{H}(\mathbf{G}_{i+1}|\mathbf{G}_{1:i})\right],$$

we get

$$H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:i+1}}[h(G_{1:i+1})] \leq \left(1 - \frac{1}{k}\right)\left[H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})]\right] + \mathcal{H}(\mathbf{G}_{i+1}|\mathbf{G}_{1:i})$$

By induction over i, we have

$$H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})] \leq \left(1 - \frac{1}{k}\right)^i \left[H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:0}}[h(G_{1:0})]\right] + \sum_{i=1}^{k} \left(1 - \frac{1}{k}\right)^{i-1} \mathcal{H}(\mathbf{G}_i|\mathbf{G}_{1:i-1})$$

Because $\mathbb{E}_{\mathbf{G}_{1:0}}[h(G_{1:0})] = 0$:

$$H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})] \leq \left(1 - \frac{1}{k}\right)^i \left[H(\mathbf{O}_{1:k})\right] + \sum_{i=1}^{k} \left(1 - \frac{1}{k}\right)^{i-1} \mathcal{H}(\mathbf{G}_i|\mathbf{G}_{1:i-1})$$

Because $(1 - 1/k) < 1$ and the entropy chain rule:

$$H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:i}}[h(G_{1:i})] \leq \left(1 - \frac{1}{k}\right)^i \left[H(\mathbf{O}_{1:k})\right] + \mathcal{H}(\mathbf{G}_{1:k})$$

Setting $i = k$ and using the known inequality $1 - x \leq e^{-x}$:

$$H(\mathbf{O}_{1:k}) - \mathbb{E}_{\mathbf{G}_{1:k}}[h(G_{1:k})] \leq (1/e)\left[H(\mathbf{O}_{1:k})\right] + \mathcal{H}(\mathbf{G}_{1:k})$$

Rearranging terms and using the definition of H, we get the desired result:

$$(1 - 1/e)H(\mathbf{O}_{1:k}) \leq H(\mathbf{G}_{1:k})$$

Lemma E.4.5 The optimal policy is not uniquely determined due to sampling ordered sequences instead of unordered sets. Let π_{opt} be the optimal policy, that samples all sequences corresponding to the same set equally often:

$$\forall S \in \mathcal{X}\, \forall S_{1:k} \in \text{perm}(S) : \pi_{opt}(S_{1:k}) = \pi_{opt}(S_{1:k}) = \frac{1}{k!} p_{\pi_{opt}}(S)$$

For $\mathbf{O}_{1:k} \sim \pi_{opt}$, we have:

(1) $\mathbb{E}_{\mathbf{O}_{1:k}}[h(O_{1:k})] = \mathbb{E}_{O \sim p_{\pi_{opt}}}[h(O)]$

(2) $\mathcal{H}(\mathbf{O}_{1:k}) = \mathcal{H}(p_{\pi_{opt}}) + \log k!$

For the greedy policy $\mathbf{G}_{1:k} \sim \pi_{greedy}$, there is:

(3) $\mathbb{E}_{\mathbf{G}_{1:k}}[h(G_{1:k})] = \mathbb{E}_{G \sim p_{\pi_{greedy}}}[h(G)]$

(4) $\mathcal{H}(\mathbf{G}_{1:k}) \leq \mathcal{H}(p_{\pi_{greedy}}) + \log k!$

Proof. (1)+(3) h is a set function, i.e. the order does not matter for h and thereby also does not matter for expectations of h

(2)

$$
\begin{aligned}
& \mathcal{H}(O_{1:k}) \\
\overset{\text{definition of } \mathcal{H}}{=} & -\sum_{S_{1:k} \in E^k} \pi_{opt}(S_{1:k}) \log \pi_{opt}(S_{1:k}) \\
\overset{\text{definition of } \pi_{opt}}{=} & -\sum_{S_{1:k} \in E^k} \frac{1}{k!} p_{\pi_{opt}}(S) \log \frac{1}{k!} p_{\pi_{opt}}(S) \\
\overset{|\text{perm}(S)|=k!}{=} & -\sum_{S \in \mathcal{E}} p_{\pi_{opt}}(S) \log \frac{1}{k!} p_{\pi_{opt}}(S) \\
\overset{\text{summarize}}{=} & \mathcal{H}(p_{\pi_{opt}}) + \log k!
\end{aligned}
$$

(4) Consider a joint sample $\mathbf{S}_{1:k} \sim \pi_{greedy}$ and $S \sim p_{\pi_{greedy}}$, since S is fully determined by $S_{1:k}$:

$$\mathcal{H}(\mathbf{S}_{1:k}) = \mathcal{H}(\mathbf{S}_{1:k}, \mathbf{S}) = \mathcal{H}(\mathbf{S}_{1:k}|\mathbf{S}) + \mathcal{H}(\mathbf{S})$$

$\mathcal{H}(\mathbf{S}_{1:k}|\mathbf{S})$ is maximized by a uniform order over all permutations, i.e. $\mathcal{H}(\mathbf{S}_{1:k}|\mathbf{S}) \leq \log k!$

Theorem E.4.6 Let h be a submodular set function with $h(\varnothing) = 0$ and $\Delta_h(x|X) > (1/k) \log k!$ for all $X \subset \mathbb{X}$, $x \in \mathbb{X} \setminus X$. Assume $\mathbb{X}$ is finite. It holds

$$(1 - 1/e) H(p_{\pi_{opt}}) \leq H(p_{\pi_{greedy}}),$$

with $H(p_\pi) = \mathbb{E}_{X \sim p_\pi}[h(X)] + \mathcal{H}(p_\pi)$ and π_{opt} being the optimal policy.

Proof. Define a new set function $m(S) := h(S) + l(S)$ with $l(S) = -\frac{|S|}{k} \log k!$. Due to the properties of h and l being a modular function, we still have monotony and submodularity for m as well as $m(\varnothing) = 0$ such that Lemma E.4.5 applies to m, too. The greedy policy π_{greedy} and the optimal sampling policy with uniform order π_{opt} as defined in Lemma

E.4.5 the same for m and h. Using Lemma E.4.4 and Lemma E.4.5, we obtain the result:

$$
\begin{aligned}
& H(p_{\pi_{greedy}}) \\
\text{definition of } H = \quad & \mathbb{E}_{G\sim p_{\pi_{greedy}}}[h(G)] + \mathcal{H}(p_{\pi_{greedy}}) \\
\text{definition of } m = \quad & \mathbb{E}_{G\sim p_{\pi_{greedy}}}[m(G)] + \mathcal{H}(p_{\pi_{greedy}}) + \log k! \\
\text{Lemma E.4.5} \geq \quad & \mathbb{E}_{\mathbf{G}_{1:k}}[m(G_{1:k})] + \mathcal{H}(\mathbf{G}_{1:k}) \\
\text{definition of } \mathbf{M} = \quad & \mathbf{M}(\mathbf{G}_{1:k}) \\
\text{Lemma E.4.4} \geq \quad & (1-1/e)\mathbf{M}(\mathbf{O}_{1:k}) \\
\text{definition of } \mathbf{M} = \quad & (1-1/e)\left[\mathbb{E}_{\mathbf{O}_{1:k}}[m(O_{1:k})] + \mathcal{H}(\mathbf{O}_{1:k})\right] \\
\text{Lemma E.4.5} = \quad & (1-1/e)\left[\mathbb{E}_{O\sim p_{\pi_{opt}}}[m(O)] + \mathcal{H}(p_{\pi_{opt}}) + \log k!\right] \\
\text{definition of } m = \quad & (1-1/e)\left[\mathbb{E}_{O\sim p_{\pi_{opt}}}[h(O)] + \mathcal{H}(p_{\pi_{opt}})\right] \\
\text{definition of } H = \quad & (1-1/e)H(p_{\pi_{opt}})
\end{aligned}
$$

Corollary E.4.7 Running the algorithm $\pi_{greedy}(x_i \mid X_{1:i-1}) \propto \mathrm{Var}_i(x)$ as introduced in Section 7.3 for k iterations on a finite grid is a $(1-1/e)$ approximation to the exact distribution $p_{k\text{-DPP}}$ of the corresponding fixed size DPP, in the sense of

$$(1-1/e)H(p_{k\text{-DPP}}) \leq H(p_{\pi_{greedy}}),$$

with $H(p) = \mathbb{E}_{X\sim p}[h_\alpha(X)] + \mathcal{H}(P)$ and $h_\alpha(X) = \log\det\left(\alpha \mathbf{L}_{XX}\right)$ for $\alpha = \frac{k!^{1/k}}{\lambda_{min}}$. Where $\lambda_{min} > 0$ is the smallest eigenvalue of the Kernel Gram matrix over the grid.

Proof. By Proposition E.4.1, the function h_α is monotone. By Proposition E.4.2, the function h is also monotone since scaling with the reciprocal of the smallest eigenvalue ensures that all eigenvalues are larger than 1. Additionally scaling with $(k!)^{1/k}$ ensures that the marginal gains are larger than $\log k!^{1/k} = \frac{1}{k}\log k!$. Thus the assumptions in Theorem E.4.6 are met and the result follows.

E.4.1. Analysis in terms of the differential entropy with a random scaling

Corollary E.4.8 Running the algorithm $\pi_{greedy}(x_i \mid X_{1:i-1}) \propto \mathrm{Var}_i(x)$ as introduced in Section 7.3 for k iterations on a finite grid is a $(1-1/e)$ approximation to the exact

distribution $p_{ktext-DPP}$ of the corresponding fixed size DPP, in the sense of

$$(1 - 1/e)\mathbb{E}_\alpha[H_\alpha(p_{k\text{-DPP}})] \leq \mathbb{E}_\alpha[H_\alpha(p_{\pi_{greedy}})],$$

with $H(p) = \mathbb{E}_{X\sim p}[h_\alpha(X)] + \mathcal{H}(P)$ and $h_\alpha(X) = \log\det\left(\alpha \mathbf{L}_{XX}\right)$. Here, α is a random variable defined as $\alpha := (k!)^{1/k}/\lambda_{min}(\mathbf{L}_{OG})$ with $O \sim p_{k\text{-DPP}}$ and $G \sim \pi_{greedy}$.

Proof. Due to Proposition E.4.2, the function $h_\alpha(X) = \log\det(\alpha \mathbf{L}_{XX})$ with $\alpha := 1/\lambda_{min}(\mathbf{L}_{OG})$ is monotone in O and G (not necessarily in all of X). By additionally multiplying α with $(k!)^{1/k}$, all marginal gains satisfy $\Delta_\alpha(s|S) > (1/k)\log k!$ for $S \subseteq O \cup G, s \in (O \cup G) \setminus S$ under the expectation of α. Follow the proofs for Lemma E.4.3, Lemma E.4.4, Lemma E.4.5 and Theorem E.4.6 with additionally taking the expectation over α for each term and note that the condition on the slope still holds were needed. The result then follows from Theorem E.4.6.

E.4.2. Analysis in terms of the information gain

The information gain from revealing points $X_{1:k}$ in a Gaussian process regression setting is defined as $g_{IG}(X_{1:k}) = \frac{1}{2}\log\det\left(I + \sigma^{-2}\mathbf{L}_{X_{1:k}}\right)$. We'll again drop the constant factor of $1/2$ and instead analyse $g(X_{1:k}) = \log\det\left(I + \sigma^{-2}\mathbf{L}_{X_{1:k}}\right)$.

It has been pointed out before by e.g. Srinivas et al. (2009) that g is monotone and submodular and can be decomposed in terms of the posterior variances

$$g_{IG}(X_{1:k}) = \sum_{i=1}^{k} \log(1 + \sigma^2 \mathrm{Var}_{\sigma_i}(x_i)), \tag{E.6}$$

where the $\mathrm{Var}_{\sigma_i}(x) = l(x,x) - l(x, X_{1:i})(\mathbf{L}_{X_{1:i},X_{1:i}} + \sigma^2 I) l(X_{1:i}, x)$.

In the optimization setting, the greedy choice is given by

$$x_i = \arg\max_x \log(1 + \sigma^2 \mathrm{Var}_{\sigma_i}(x)), \tag{E.7}$$

so we again take the softargmax instead and sample greedily from

$$\pi_{greedy_g}(x|X_{1:i}) \propto \exp(\log(1 + \sigma^2 \mathrm{Var}_{\sigma_i}(x))) \propto 1 + \sigma^2 \mathrm{Var}_{\sigma_i}(x) \tag{E.8}$$

By direct application of Lemma E.4.4, we get:

Corollary E.4.9 Consider π_{greedy_g} as defined in Equation (E.8) and $\pi_{k\text{-DPP}}$ for an optimal policy for sampling from a k-DPP. It holds

$$(1 - 1/e) H_g\left(\pi_{k\text{-DPP}}\right) \leq H_g\left(\pi_{greedy_g}\right),$$

where $H_g(\pi) = \mathbb{E}_{X\sim\pi}(g(X)) + \mathcal{H}(\pi)$, where $g(X) = \log\det\left(I + \sigma^{-2}\mathbf{L}_X\right)$ for $\sigma^2 > 0$.

The policy to sample greedily from information gain can be implemented as efficiently as the original policy for the differential entropy:

The posterior Var_{σ_i} retains much of the structure of Var_i, so in the terms for the posterior (Equation 7.25), as well as the unnormalized cumulative density in (Equation 7.26) the matrix $\mathbf{L}$ merely has to be replaced with $\mathbf{L}+\sigma^2 I$ to obtain the corresponding terms for Var_{σ_i} and $\mathbb{V}_{\sigma_i}$. Note that we do not sample proportionally to $\mathrm{Var}_{\sigma_i}(x)$, but proportionally to $1+\sigma^2\mathrm{Var}_{\sigma_i}$. However, given the efficient computation of $\mathbb{V}_{\sigma_i}$ the unnormalized cumulative density for $1 + \sigma^2\mathrm{Var}_{\sigma_i}$ can easily be obtained due to linearity of integration.

E.5. Relation to Dirichlet Log-Likelihood

We start with the log-likelihood of the Dirichlet distribution with parameters $\tau\pi = (\tau\pi_1, \dots, \tau\pi_K)$:

$$\log \mathrm{Dir}(\mathbf{x} \mid \tau\pi) = \log\Gamma(\tau) - \sum_{i=1}^{K}\log\Gamma(\tau\pi_i) + \sum_{i=1}^{K}(\tau\pi_i - 1)\log x_i \tag{E.9}$$

Applying the logarithmic-order Stirling approximation to each gamma term

$$\log\Gamma(z) \approx z\log z - z + \frac{1}{2}\log z + \frac{1}{2}\log(2\pi),$$

we get:

$$\log\Gamma(\tau) \approx \tau\log\tau - \tau + \frac{1}{2}\log\tau + \frac{1}{2}\log(2\pi) \tag{E.10}$$

$$\sum_{i=1}^{K}\log\Gamma(\tau\pi_i) \approx \sum_{i=1}^{K}\left[\tau\pi_i\log(\tau\pi_i) - \tau\pi_i + \frac{1}{2}\log(\tau\pi_i) + \frac{1}{2}\log(2\pi)\right] \tag{E.11}$$

Using $\sum_i \pi_i = 1$, this becomes:

$$\sum_{i=1}^{K}\log\Gamma(\tau\pi_i) \approx \tau\sum_{i=1}^{K}\pi_i\log(\tau\pi_i) - \tau + \frac{1}{2}\sum_{i=1}^{K}\log(\tau\pi_i) + \frac{K}{2}\log(2\pi) \tag{E.12}$$

Insert into Equation (E.9):

$$\begin{aligned}\log \mathrm{Dir}(\mathbf{x} \mid \tau\pi) \approx\ & \tau\log\tau - \tau + \frac{1}{2}\log\tau + \frac{1}{2}\log(2\pi) \\ & - \left[\tau\sum_{i=1}^{K}\pi_i\log(\tau\pi_i) - \tau + \frac{1}{2}\sum_{i=1}^{K}\log(\tau\pi_i) + \frac{K}{2}\log(2\pi)\right] \\ & + \sum_{i=1}^{K}(\tau\pi_i - 1)\log x_i\end{aligned} \tag{E.13}$$

Now simplify and collect terms:

$$\log \mathrm{Dir}(\mathbf{x} \mid \tau\pi) \approx \tau \log \tau - \tau \sum_{i=1}^{K} \pi_i \log(\tau\pi_i) + \sum_{i=1}^{K} \tau\pi_i \log x_i + \frac{1}{2} \log \tau - \frac{1}{2} \sum_{i=1}^{K} \log(\tau\pi_i) - \frac{K-1}{2} \log(2\pi) - \sum_{i=1}^{K} \log x_i \tag{E.14}$$

Group terms:

$$\log \mathrm{Dir}(\mathbf{x} \mid \tau\pi) \approx \tau \sum_{i=1}^{K} \pi_i \log x_i - \tau \sum_{i=1}^{K} \pi_i \log \pi_i + \tau \log \tau - \tau \log \tau - \frac{1}{2} \sum_{i=1}^{K} \log \pi_i - \frac{K-1}{2} \log(2\pi\tau) - \sum_{i=1}^{K} \log x_i \tag{E.15}$$

Now divide the whole expression by τ:

$$\frac{1}{\tau} \log \mathrm{Dir}(\mathbf{x} \mid \tau\pi) \approx \sum_{i=1}^{K} \pi_i \log x_i - \sum_{i=1}^{K} \pi_i \log \pi_i - \frac{1}{\tau} \left(\frac{1}{2} \sum_{i=1}^{K} \log \pi_i + \frac{K-1}{2} \log(2\pi\tau) + \sum_{i=1}^{K} \log x_i \right) \tag{E.16}$$

Define:

$$\mathbb{E}_\pi[\log \mathbf{x}] := \sum_{i=1}^{K} \pi_i \log x_i, \quad \mathcal{H}(\pi) := -\sum_{i=1}^{K} \pi_i \log \pi_i, \quad C(\pi, \tau) := \frac{1}{2} \sum_{i=1}^{K} \log \pi_i + \frac{K-1}{2} \log(2\pi\tau) + \sum_{i=1}^{K} \log x_i$$

Thus, we obtain:

$$\frac{1}{\tau} \log \mathrm{Dir}(\mathbf{x} \mid \tau\pi) \approx \mathbb{E}_\pi[\log \mathbf{x}] + \mathcal{H}(\pi) - \frac{1}{\tau} C(\pi, \tau) \tag{E.17}$$

E.6. Additional Experimental Details

The experiments were performed on a desktop machine with a 4 GHz Quad-Core Intel Core i7 processor and 32 GB memory. Table **??** lists the scales of the squared-exponential kernels used for BQ. Since all methods share the same kernel hyperparameters, we assume that their choice only plays a subordinate role and obtained them by eyeballing the ground-truth scale from 2D plots of the integrands.

Figure E.1 shows additional runtime results in three dimensions. We draw 100 samples with $k = 10$ and $k = 100$ points from a k-DPP with square-exponential kernel with lengthscale 0.01, respectively 0.001, on the interval $[0, 1]^3$. Runs taking longer than 100 seconds for $k = 10$ or 500 seconds for $k = 1000$ were stopped, i.e. they do not appear in the figure. For $k = 10$, the experiment was repeated 5 times and for $k = 100$, 3 times.

Benchmark	Scale
Brately function	1
Continuous integrand family	1
Corner peak integrand family	1
Discontinuous integrand family	140
G function	3.5
Gaussian Peak integrand family	1
Morokoff Caflisch function 1	2.5
Morokoff Caflisch function 2	1
Oscillatroy integrand family	2
Product peak integrand family	700
Roos Arnold function	4
Zhou function	8

Table E.1.: Hyperparameters for the Bayesian Quadrature

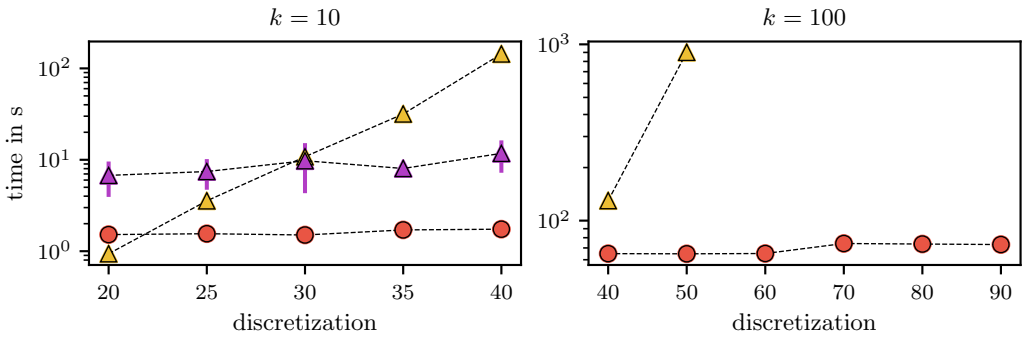


Figure E.1.: Runtime comparison of the greedy (●)algorithm with state-of-the-art baselines MCMC (▲), DPP-VFX (▲) and α-DPP (▲) in three dimensions.

Bibliography

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

M. Adachi, S. Hayakawa, M. Jørgensen, H. Oberhauser, and M. A. Osborne. Fast Bayesian inference with batch Bayesian quadrature via kernel recombination. *Advances in Neural Information Processing Systems*, 35:16533–16547, 2022.

N. Anari, S. O. Gharan, and A. Rezaei. Monte Carlo Markov chain algorithms for sampling strongly Rayleigh distributions and determinantal point processes. In *Conference on Learning Theory*, pages 103–115. PMLR, 2016.

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

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 322–331. IEEE, 1995.

M. Balog, B. Lakshminarayanan, Z. Ghahramani, D. M. Roy, and Y. W. Teh. The Mondrian kernel. *arXiv preprint arXiv:1606.05241*, 2016.

R. Bardenet and A. Hardy. Monte Carlo with determinantal point processes. *ArXiv e-print*, 1605.00361, 5 2016.

O. E. Barndorff-Nielsen and N. Shephard. Modelling by lévy processess for financial econometrics. *Lévy processes: theory and applications*, pages 283–318, 2001.

E. B. Baum and W. D. Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1-2):195–242, 1997.

R. F. Baumeister, K. D. Vohs, and G. Oettingen. Pragmatic prospection: How and why people think about the future. *Review of General Psychology*, 20(1), 2016.

A. Belhadji, R. Bardenet, and P. Chainais. Kernel quadrature with DPPs. *Advances in Neural Information Processing Systems*, 32, 2019.

J. Bertoin. *Lévy processes*, volume 121. Cambridge university press Cambridge, 1996.

A. Bertsch, A. Xie, G. Neubig, and M. R. Gormley. It's mbr all the way down: Modern generation techniques through the lens of minimum bayes risk, 2023. URL https://arxiv.org/abs/2310.01387.

C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

I. Bogunovic, J. Scarlett, S. Jegelka, and V. Cevher. Adversarially robust optimization with Gaussian processes. *Advances in neural information processing systems*, 31, 2018.

S. Borst, D. Dadush, N. Olver, and M. Sinha. Majorizing measures for the optimizer. *arXiv preprint arXiv:2012.13306*, 2020.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed Bandits. *Journal of Machine Learning Research*, 12(5), 2011.

D. Calandriello, M. Derezinski, and M. Valko. Sampling from a k-DPP without looking at all items. *Advances in Neural Information Processing Systems*, 33:6889–6899, 2020.

G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

L. Chen, G. Zhang, and E. Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity. *Advances in Neural Information Processing Systems*, 31, 2018.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code, 2021.

S. Chen, O. Hagrass, and J. M. Klusowski. Decoding game: On minimax optimality of heuristic text generation strategies. *arXiv preprint arXiv:2410.03968*, 2024a.

S. Chen, H. Yu, J. Yagoobi, and C. Shao. Reinforcement learning constrained beam search for parameter optimization of paper drying under flexible constraints. *arXiv preprint arXiv:2501.12542*, 2025.

Y. Chen, E. N. Epperly, J. A. Tropp, and R. J. Webber. Randomly pivoted cholesky: Practical approximation of a kernel matrix with few entry evaluations. *arXiv preprint arXiv:2207.06503*, 2022.

Z. Chen, M. White, R. Mooney, A. Payani, Y. Su, and H. Sun. When is tree search useful for llm planning? it depends on the discriminator. *arXiv preprint arXiv:2402.10890*, 2024b.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. PaLM: Scaling language modeling with pathways. *JMLR*, 24(240), 2023.

D. Christofidellis, G. Giannone, J. Born, O. Winther, T. Laino, and M. Manica. Unifying molecular and textual representations via multi-task language modelling. In *International Conference on Machine Learning*, pages 6140–6157. PMLR, 2023.

Y.-S. Chuang, Y. Xie, H. Luo, Y. Kim, J. Glass, and P. He. DoLa: Decoding by contrasting layers improves factuality in large language models. In *ICLR*, 2024.

M. Conforti and G. Cornuéjols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem. *Discrete applied mathematics*, 7 (3):251–274, 1984.

E. Contal, C. Malherbe, and N. Vayatis. Optimization for Gaussian processes via chaining. *arXiv preprint arXiv:1510.05576*, 2015.

R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

T. Q. Dam, C. D'Eramo, J. Peters, and J. Pajarinen. Convex regularization in Monte-Carlo tree search. In *International Conference on Machine Learning*, pages 2365–2375. PMLR, 2021.

M. DeLorenzo, A. B. Chowdhury, V. Gohil, S. Thakur, R. Karri, S. Garg, and J. Rajendran. Make every move count: Llm-based high-quality rtl code generation using mcts. *arXiv preprint arXiv:2402.03289*, 2024.

L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

M. Derezinski, D. Calandriello, and M. Valko. Exact sampling of determinantal point processes with sublinear time preprocessing. *Advances in neural information processing systems*, 32, 2019.

E. W. Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: his life, work, and legacy*, pages 287–290. 2022.

J. Djolonga, S. Jegelka, and A. Krause. Provable variational inference for constrained log-submodular models. *Advances in Neural Information Processing Systems*, 31, 2018.

J. Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1:127–136, 1971.

B. Eikema and W. Aziz. Is map decoding all you need? the inadequacy of the mode in neural machine translation. *arXiv preprint arXiv:2005.10283*, 2020.

B. Eikema and W. Aziz. Sampling-based approximations to minimum bayes risk decoding for neural machine translation. *arXiv preprint arXiv:2108.04718*, 2021.

M. Elfeki, C. Couprie, M. Riviere, and M. Elhoseiny. Gdpp: Learning diverse generations using determinantal point processes. In *International conference on machine learning*, pages 1774–1783. PMLR, 2019.

E. N. Epperly and E. Moreno. Kernel quadrature with randomly pivoted cholesky. *arXiv preprint arXiv:2306.03955*, 2023.

D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

B. Eysenbach and S. Levine. If maxent rl is the answer, what is the question? *arXiv preprint arXiv:1910.01913*, 2019.

X. Feng, Z. Wan, M. Wen, Y. Wen, W. Zhang, and J. Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

M. Freitag and Y. Al-Onaizan. Beam search strategies for neural machine translation. In *ACL Workshop on Neural Machine Translation*, 2017.

W. Gao, S. Luo, and C. W. Coley. Generative artificial intelligence for navigating synthesizable chemical space. *arXiv preprint arXiv:2410.03494*, 2024.

J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018:7576–7586, 2018.

R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

R. Gaudel and M. Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning (ICML)*, 2010.

G. Gautier, R. Bardenet, and M. Valko. On two ways to use determinantal point processes for Monte Carlo integration. *Advances in Neural Information Processing Systems*, 32, 2019a.

G. Gautier, G. Polito, R. Bardenet, and M. Valko. DPPy: DPP Sampling with Python. *Journal of Machine Learning Research - Machine Learning Open Source Software (JMLR-MLOSS)*, 2019b. URL http://jmlr.org/papers/v20/19-179.html. Code at http://github.com/guilgautier/DPPy/ Documentation at http://dppy.readthedocs.io/.

S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 273–280, 2007.

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics 195. Springer, 2009.

J. González, Z. Dai, P. Hennig, and N. Lawrence. Batch Bayesian optimization via local penalization. In *AISTATS*, 2016.

T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

M. González-Duque, R. Michael, S. Bartels, Y. Zainchkovskyy, S. Hauberg, and W. Boomsma. A survey and benchmark of high-dimensional Bayesian optimization of discrete sequences, 2024. URL https://arxiv.org/abs/2406.04739.

I. Gradshteyn and I. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, 7th edition, 2007.

J.-B. Grill, M. Valko, and R. Munos. Optimistic optimization of a Brownian. *Advances in Neural Information Processing Systems*, 31, 2018.

J.-B. Grill, F. Altché, Y. Tang, T. Hubert, M. Valko, I. Antonoglou, and R. Munos. Monte-Carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pages 3769–3778. PMLR, 2020.

J. Grosse, C. Zhang, and P. Hennig. Probabilistic DAG search. In *Uncertainty in Artificial Intelligence*, pages 1424–1433. PMLR, 2021.

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

S. Gupta, S. Rana, S. Venkatesh, et al. Bayesian Optimistic Optimisation with Exponentially Decaying Regret. In *International Conference on Machine Learning*, pages 10390–10400. PMLR, 2021.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

E. Han, I. Arora, and J. Scarlett. High-dimensional Bayesian optimization via tree-structured additive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7630–7638, 2021.

S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2), 1968a.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968b.

Y. He and F. S. Bao. Circuit routing using Monte Carlo tree search and deep neural networks. *arXiv preprint arXiv:2006.13607*, 2020.

P. Hennig. Expectation propagation on the maximum of correlated normal variables. Technical report, Cavendish Laboratory: University of Cambridge, July 2009.

P. Hennig and C. J. Schuler. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research*, 13(6), 2012.

P. Hennig, D. Stern, and T. Graepel. Coherent inference on optimal play in game trees. In *AISTATS*, 2010.

P. Hennig, M. A. Osborne, and H. P. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, page 282. Citeseer, 2013.

K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *NIPS*, 2015.

J. M. Hernández-Lobato, J. Requeima, E. O. Pyzer-Knapp, and A. Aspuru-Guzik. Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In *ICML*, 2017.

A. P. Herrmann and H. Schaub. Monte Carlo tree search methods for the earth-observing satellite scheduling problem. *Journal of Aerospace Information Systems*, 19(1):70–82, 2022.

H. Hino and K. Yano. Duality induced by an embedding structure of determinantal point process. *arXiv preprint arXiv:2404.11024*, 2024.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *ICLR*, 2019.

J. B. Hough, M. Krishnapur, Y. Peres, and B. Virág. Determinantal processes and independence. *Probability Surveys*, 3:206–229, 2006.

J. B. Hough, M. Krishnapur, Y. Peres, and B. Virág. *Zeros of Gaussian analytic functions and determinantal point processes*, volume 51, University Lecture Series. American Mathematical Society Providence, RI, 2009.

M. Hu, Y. Mu, X. Yu, M. Ding, S. Wu, W. Shao, Q. Chen, B. Wang, Y. Qiao, and P. Luo. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582*, 2023.

F. Huszár and D. Duvenaud. Optimally-weighted herding is Bayesian quadrature. *arXiv preprint arXiv:1204.1664*, 2012.

T. F. Jaeger. Redundancy and reduction: Speakers manage syntactic information density. *Cognitive psychology*, 61(1):23–62, 2010.

F. Jimenez and M. Katzfuss. Scalable Bayesian optimization using Vecchia approximations of Gaussian processes. *arXiv preprint arXiv:2203.01459*, 2022.

D. R. Jones and J. R. Martins. The direct algorithm: 25 years later. *Journal of global optimization*, 79(3): 521–566, 2021.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492, 1998.

R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv preprint arXiv:1807.02582*, 2018.

B. Kang. Fast determinantal point process sampling with application to clustering. *Advances in Neural Information Processing Systems*, 26, 2013.

S. Kapoor, N. Gruver, M. Roberts, K. Collins, A. Pal, U. Bhatt, A. Weller, S. Dooley, M. Goldblum, and A. G. Wilson. Large language models must be taught to know what they don't know. *arXiv preprint arXiv:2406.08391*, 2024.

T. Kathuria, A. Deshpande, and P. Kohli. Batched Gaussian process bandit optimization via determinantal point processes. *Advances in neural information processing systems*, 29, 2016.

M. Kemmerling, D. Lütticke, and R. H. Schmitt. Beyond games: a systematic review of neural Monte Carlo tree search applications. *Applied Intelligence*, 54(1):1020–1046, 2024.

G. Kingsley Zipf. *Selected Studies of the Principle of Relative Frequency in Language*. Harvard University Press, 1932.

E. M. Kiral, T. Möllenhoff, and M. E. Khan. The Lie-group Bayesian learning rule. In *International conference on Artificial Intelligence and Statistics*, pages 3331–3352. PMLR, 2023.

R. Kleinberg, A. Slivkins, and E. Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 681–690, 2008.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006a.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, 2006b.

P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *HLT-NAACL*, 2003.

S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448, 2005.

W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In *ICML*, 2019.

B. Korte and L. Lovász. Greedoids-a structural framework for the greedy algorithm. In *Progress in combinatorial optimization*, pages 221–243. Elsevier, 1984.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(2), 2008.

D. Kreutter and J.-L. Reymond. Multistep retrosynthesis combining a disconnection aware triple transformer loop with a route penalty score guided tree search. *Chemical Science*, 14(36):9959–9969, 2023.

A. Kristiadi, A. Immer, R. Eschenhagen, and V. Fortuin. Promises and pitfalls of the linearized Laplace in Bayesian optimization. In *AABI*, 2023.

A. Kulesza and B. Taskar. k-DPPs: Fixed-size determinantal point processes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1193–1200, 2011.

A. Kulesza and B. Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5:123–286, 2012.

S. Kumar and B. Byrne. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176, 2004.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *J. Basic Eng*, 1964.

T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

R. Leblond, J.-B. Alayrac, L. Sifre, M. Pislar, J.-B. Lespiau, I. Antonoglou, K. Simonyan, and O. Vinyals. Machine translation decoding beyond beam search. *arXiv preprint arXiv:2104.05336*, 2021.

K. Lee, J. Woo Kim, and W. Youn Kim. Efficient construction of a chemical reaction network guided by a Monte Carlo tree search. *ChemSystemsChem*, 2(5):e1900057, 2020.

Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

J. Lim and S. Yoo. Field report: Applying Monte Carlo tree search for program synthesis. In *Search Based Software Engineering: 8th International Symposium, SSBSE 2016, Raleigh, NC, USA, October 8-10, 2016, Proceedings 8*, pages 304–310. Springer, 2016.

W. Lin, M. E. Khan, and M. Schmidt. Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. In *International Conference on Machine Learning*, pages 3992–4002. PMLR, 2019.

W. Lin, F. Nielsen, K. M. Emtiyaz, and M. Schmidt. Tractable structured natural-gradient descent using local parameterizations. In *International Conference on Machine Learning*, pages 6680–6691. PMLR, 2021.

J. Liu, A. Cohen, R. Pasunuru, Y. Choi, H. Hajishirzi, and A. Celikyilmaz. Don't throw away your value model! making PPO even better via value-guided Monte-Carlo Tree Search decoding. *arXiv e-prints*, 2023.

D. J. Lizotte. Practical Bayesian optimization. 2008.

Y. Luo, M. Wang, H. Zhou, Q. Yao, W.-W. Tu, Y. Chen, W. Dai, and Q. Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1936–1945, 2019.

R. Lyons. Determinantal probability measures. *Publications Mathématiques de l'IHÉS*, 98:167–212, 2003.

C. Ma, S. Tschiatschek, K. Palla, J. M. Hernández-Lobato, S. Nowozin, and C. Zhang. Eddi: Efficient dynamic discovery of high-value information with partial vae. *arXiv preprint arXiv:1809.11142*, 2018.

O. Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, pages 83–122, 1975.

J. Mańdziuk. Mcts/uct in solving real-life problems. *Advances in Data Analysis with Computational Intelligence Methods: Dedicated to Professor Jacek Żurada*, pages 277–292, 2018.

C. Mansley, A. Weinstein, and M. Littman. Sample-based planning for continuous action Markov decision processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 21, pages 335–338, 2011.

W. Mao, K. Zhang, Q. Xie, and T. Basar. Poly-hoot: Monte-Carlo planning in continuous space mdps with non-asymptotic analysis. *Advances in Neural Information Processing Systems*, 33:4549–4559, 2020.

Z. Mariet and S. Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

G. Marsaglia and W. W. Tsang. The Ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000.

M. L. Mehta. *Random Matrices*. Academic Press, 1991.

C. Meister, T. Vieira, and R. Cotterell. Best-first beam search. *TACL*, 8, 2020a.

C. Meister, T. Vieira, and R. Cotterell. If beam search is the answer, what was the question? *arXiv preprint arXiv:2010.02650*, 2020b.

C. Meister, M. Forster, and R. Cotterell. Determinantal beam search. *arXiv preprint arXiv:2106.07400*, 2021.

J. Mern, A. Yildiz, Z. Sunberg, T. Mukerji, and M. J. Kochenderfer. Bayesian optimized Monte Carlo planning. In *AAAI*, 2021.

T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence (UAI)*, volume 17, pages 362–369. Morgan Kaufmann Publishers Inc., 2001a.

T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001b.

J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, 1975.

P. Mondrian, H. Holtzman, and M. S. James. The new art–the new life: the collected writings of piet Mondrian. *unknown*, 1993.

R. Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. *Advances in neural information processing systems*, 24:783–791, 2011.

M. Mutny and A. Krause. Efficient high dimensional Bayesian optimization with additivity and quadrature fourier features. *Advances in Neural Information Processing Systems*, 31, 2018.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

E. Nava, M. Mutny, and A. Krause. Diversified sampling for batched Bayesian optimization with determinantal point processes. In *International Conference on Artificial Intelligence and Statistics*, pages 7031–7054. PMLR, 2022.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsâ i. *Mathematical programming*, 14(1):265–294, 1978.

B. O'Donoghue, I. Osband, and C. Ionescu. Making sense of reinforcement learning and probabilistic inference. *arXiv preprint arXiv:2001.00805*, 2020.

E. Onal, K. Flöge, E. Caldwell, A. Sheverdin, and V. Fortuin. Gaussian stochastic weight averaging for Bayesian low-rank adaptation of large language models. *arXiv preprint arXiv:2405.03425*, 2024.

E. O'Reilly and N. Tran. Stochastic geometry to generalize the Mondrian process. *arXiv preprint arXiv:2002.00797*, 2020.

J. G. Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

M. Painter, M. Baioumy, N. Hawes, and B. Lacerda. Monte Carlo tree search with boltzmann exploration. *Advances in Neural Information Processing Systems*, 36, 2024.

A. Paleyes, M. Pullin, M. Mahsereci, N. Lawrence, and J. GonzÃ¡lez. Emulation of physical processes with emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*, 2019.

A. Paleyes, M. Mahsereci, and N. D. Lawrence. Emukit: A Python toolkit for decision making under uncertainty. *Proceedings of the Python in Science Conference*, 2023.

J. Pearl. Intelligent search strategies for computer problem solving. *Addision Wesley*, 1984.

A. Pélissier, A. Nakamura, and K. Tabata. Feature selection as Monte-Carlo search in growing single rooted directed acyclic graph by best leaf identification. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 450–458. SIAM, 2019.

B. L. Pellom and J. H. Hansen. An efficient scoring algorithm for Gaussian mixture model based speaker identification. *IEEE signal processing letters*, 5(11):281–284, 2002.

G. Pisier. *The volume of convex bodies and Banach space geometry*, volume 94. Cambridge University Press, 1999.

A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin. Exploiting graph properties of game trees. In *AAAI/IAAI, Vol. 1*, pages 234–239, 1996.

S. T. Rachev, Y. S. Kim, M. L. Bianchi, and F. J. Fabozzi. *Financial models with Lévy processes and volatility clustering*. John Wiley & Sons, 2011.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140), 2020.

J. Rahme and R. P. Adams. A theoretical connection between statistical physics and reinforcement learning. *arXiv preprint arXiv:1906.10228*, 2019.

S. Raible. Lévy processes in finance: Theory, numerics, and empirical facts. 2000.

M. Rando, L. Carratino, S. Villa, and L. Rosasco. Ada-BKB: Scalable Gaussian Process Optimization on Continuous Domains by Adaptive Discretization. In *International Conference on Artificial Intelligence and Statistics*, pages 7320–7348. PMLR, 2022.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT, 2006.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes in Machine Learning*. The MIT Press, 2005.

R. Rasmussen, F. Maire, and R. Hayward. A template matching table for speeding-up game-tree searches for hex. In *Australasian Joint Conference on Artificial Intelligence*, pages 283–292. Springer, 2007.

S. Rathnam, S. Parbhoo, S. Swaroop, W. Pan, S. A. Murphy, and F. Doshi-Velez. Rethinking discount regularization: New interpretations, unintended consequences, and solutions for regularization in reinforcement learning. *Journal of Machine Learning Research*, 25(255):1–48, 2024.

R. Reddy. Speech understanding systems: Summary of results of the five-year research effort at Carnegie Mellon University, 1977.

A. Rezaei and S. O. Gharan. A polynomial time MCMC method for sampling from continuous determinantal point processes. In *International Conference on Machine Learning*, pages 5438–5447. PMLR, 2019.

D. M. Roy and Y. W. Teh. The Mondrian process. In *Advances in neural information processing systems*, pages 1377–1384, 2008.

A. Saffidine, T. Cazenave, and J. Méhat. Ucd: Upper confidence bound for rooted directed acyclic graphs. *Knowledge-Based Systems*, 34:26–33, 2012.

S. Salgia, S. Vakili, and Q. Zhao. A Domain-Shrinking based Bayesian Optimization Algorithm with Order-Optimal Regret Performance. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

W. Schoutens. *Lévy processes in finance: pricing financial derivatives*. Wiley Online Library, 2003.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *ACL*, 2017.

M. H. Segler, M. Preuss, and M. P. Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, 2018.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

D. Sharma, A. Kapoor, and A. Deshpande. On greedy maximization of entropy. In *International Conference on Machine Learning*, pages 1330–1338. PMLR, 2015.

S. Shekhar and T. Javidi. Gaussian process bandits with adaptive discretization. *Electronic Journal of Statistics*, 12(2):3829–3874, 2018.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1257–1264, 2006.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

A. Soshnikov. Determinantal random point fields. *Russian Mathematical Surveys*, 55(5):923–975, 2000.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

F. Stahlberg and B. Byrne. On nmt search errors and model errors: Cat got your tongue? *arXiv preprint arXiv:1908.10090*, 2019.

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer S&B M, 1999.

D. Stern, R. Herbrich, and T. Graepel. Learning to solve game trees. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 839–846, 2007.

N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano. Learning to summarize with human feedback. In *NeurIPS*, 2020.

Y. Su, T. Lan, Y. Wang, D. Yogatama, L. Kong, and N. Collier. A contrastive framework for neural text generation. In *NeurIPS*, 2022.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved October 10, 2023, from http://www.sfu.ca/~ssurjano.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved May 16, 2022, from http://www.sfu.ca/~ssurjano, 2022.

M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

M. Talagrand. Majorizing measures: the generic chaining. *The Annals of Probability*, 24(3):1049–1103, 1996.

M. Talagrand. Empirical processes, ii. In *Upper and Lower Bounds for Stochastic Processes*, pages 433–456. Springer, 2021.

J. Tarbouriech, T. Lattimore, and B. O'Donoghue. Probabilistic inference in reinforcement learning done right. *Advances in Neural Information Processing Systems*, 36, 2024.

G. Tesauro, V. Rajan, and R. Segal. Bayesian inference in Monte-Carlo tree search. *arXiv preprint arXiv:1203.3519*, 2012.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933.

M. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

E. Todorov. General duality between optimal control and estimation. In *2008 47th IEEE conference on decision and control*, pages 4286–4292. IEEE, 2008.

F. Tonolini, N. Aletras, J. Massiah, and G. Kazai. Bayesian prompt ensembles: Model uncertainty estimation for black-box large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 12229–12272, 2024.

M. Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

M. Valko, A. Carpentier, and R. Munos. Stochastic simultaneous optimistic optimization. In *International Conference on Machine Learning*, pages 19–27. PMLR, 2013.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

J. Veness and A. Blair. Effective use of transposition tables in stochastic game tree search. In *2007 IEEE Symposium on Computational Intelligence and Games*, pages 112–116. IEEE, 2007.

A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. In *AAAI*, 2018.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

M. Völske, M. Potthast, S. Syed, and B. Stein. TL;DR: Mining Reddit to learn automatic summarization. In *Workshop on New Frontiers in Summarization*, 2017.

N. Vulkan. An economistâ s perspective on probability matching. *Journal of economic surveys*, 14(1): 101–118, 2000.

G. Wang, H. Pu, T. Song, P. Schonfeld, W. Li, H. Zhang, L. Peng, J. Hu, and J. Qiao. A 3d Monte Carlo tree search method for railway alignment optimization. *Applied Soft Computing*, 151:111158, 2024.

T. Wang, W. Ye, D. Geng, and C. Rudin. Towards practical Lipschitz bandits. In *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*, pages 129–138, 2020.

Z. Wang and S. Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

Z. Wang, B. Shakibi, L. Jin, and N. Freitas. Bayesian multi-scale optimistic optimization. In *Artificial Intelligence and Statistics*, pages 1005–1014. PMLR, 2014.

Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

Z. Wang, C. Li, S. Jegelka, and P. Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. In *International Conference on Machine Learning*, pages 3656–3664. PMLR, 2017.

T. Warnock and B. Wendroff. Search tables in computer chess. *ICGA Journal*, 11(1):10–13, 1988.

D. Weichert and A. Kister. Bayesian optimization for min max optimization. *arXiv preprint arXiv:2107.13772*, 2021.

G. Wiher, C. Meister, and R. Cotterell. On decoding strategies for neural text generators. *TACL*, 10, 2022.

M. Wilhelm, A. Ramanathan, A. Bonomo, S. Jain, E. H. Chi, and J. Gillenwater. Practical diversified recommendations on youtube with determinantal point processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2165–2173, 2018.

D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *NeurIPS*, 2020.

J. Wilson, F. Hutter, and M. Deisenroth. Maximizing acquisition functions for Bayesian optimization. In *NIPS*, 2018.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *EMNLP*, 2020.

J. Wu and P. Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *NIPS*, 2016.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

C. Xiao, R. Huang, J. Mei, D. Schuurmans, and M. Müller. Maximum entropy Monte-Carlo planning. *Advances in Neural Information Processing Systems*, 32, 2019.

A. X. Yang, M. Robeyns, X. Wang, and L. Aitchison. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*, 2023.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

C. Zhang, H. Kjellstrom, and S. Mandt. Determinantal point processes for mini-batch diversification. *arXiv preprint arXiv:1705.00607*, 2017.

D. Zhang, S. Zhoubian, Y. Yue, Y. Dong, and J. Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*, 2024.

H. Zhang, D. Duckworth, D. Ippolito, and A. Neelakantan. Trading off diversity and quality in natural language generation. *arXiv preprint arXiv:2004.10450*, 2020.

S. Zhang, Z. Chen, Y. Shen, M. Ding, J. B. Tenenbaum, and C. Gan. Planning with large language models for code generation. In *ICLR*, 2023.

A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.