Robot Learning with Strong Priors

by

Zi Wang

B.Eng., Tsinghua University (2014)

S.M., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2020

Author . . . . . .
Department of Electrical Engineering and Computer Science
January 30, 2020

Certified by . . . . . .
Leslie Pack Kaelbling
Panasonic Professor of Computer Science and Engineering
Thesis Supervisor

Certified by . . . . . .
Tomás Lozano-Pérez
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Robot Learning with Strong Priors

by

Zi Wang

Submitted to the Department of Electrical Engineering and Computer Science
on January 30, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Embedding learning ability in robotic systems is one of the long sought-after objectives of artificial intelligence research. Despite the recent advancements in hardware, large-scale machine learning algorithms and theoretical understanding of deep learning, it is still quite unrealistic to deploy an end-to-end learning agent in the wild, attempting to learn everything from scratch. Instead, we identify the importance of imposing strong prior knowledge on capable robotic systems and perform robot learning with strong priors.

In this thesis, we exemplify the value of imposing strong priors in robot learning (or machine learning in general) via both practical experiments and theories with mild assumptions. Empirically, by proposing new algorithms and systems, we show that (active) model learning with strong priors on model structures makes it feasible to adopt advanced planners to solve complicated long-horizon robotic manipulation problems that were not possible before. On the other hand, we verify our theories through mathematical analyses of data efficiency for our active data acquisition strategies based on Bayesian optimization and systems combining learning and planning. The new approaches integrate structural prior knowledge with statistical machine learning methods to achieve state-of-the-art performance on complex long-horizon robot manipulation tasks.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Panasonic Professor of Computer Science and Engineering

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Computer Science and Engineering

Acknowledgments

Throughout my PhD training, I was extremely fortunate to have the support of family, mentors, and peers, enabling me to achieve milestones and overcome hardships. This section is dedicated to every single one of them.

First, I would like to thank Leslie Kaelbling and Tomas Lozano-Perez, my ardent supporters and mentors throughout my PhD. It is through their careful guidance and constant encouragement that I was able to navigate through the maze of research at the intersection of machine learning and robotics. Despite nearing the completion of my PhD training, I still feel like I have so much to learn from them. In the whirlwind of academia where publications may often be misconstrued as the only measure of productivity and success, I have always admired my mentors' pure pursuit of solving big research problems without putting publishing pressure on their students. In an age of "publish or perish" with an explosion of publications on arXiv, it calmed me to be at this group where I had the support of my mentors and the luxury of time to explore real solutions to big problems and not to publish incremental improvements prematurely.

I am also extremely grateful for the mentorship and support I received from Stefanie Jegelka. The reading groups she organized have enriched my knowledge beyond machine learning and helped so much to broaden my horizons. The collaborations with her pushed my limits and let me realize my potential. Although there were times when I felt like I could not carry on to complete my PhD, it was often her belief in me that ultimately resulted in my growth and overcoming some of the toughest hurdles of my PhD training.

Another important figure throughout my research career is Fei Sha, my research advisor during my undergraduate internship at USC, whom I never felt hesitant to ask for advice on research and career decisions even after I entered graduate school. Fei made me realize that a PhD training followed by research as a career could be a fulfilling career. As a mentor and friend, he never failed to provide support and helpful advice, for which I am very grateful.

I would also like to thank Marc Toussaint for his valuable inputs on my research together with reading and giving feedback on the drafts of this thesis. Some of the first papers I read in the beginning of my PhD training were in fact authored by Marc. Perhaps unsurprisingly, my research has stepped towards a similar direction to Marc's. I also feel very fortunate to be able to visit his lab and learn about the latest research from his group in Stuttgart, Germany.

Besides the wisdom and vision of my mentors, this thesis has also been deeply influenced by var-

ious friends and collaborators with their expert knowledge who taught me not only new techniques, but also research philosophies. Over the years, I have greatly benefited from the discussions and collaborations with everyone I interacted with, including Jun Zhu, Ning Chen, Jianfei Chen, Xun Zheng, Zhiyun Lu, Dingchao Lu, Bolei Zhou, Pushmeet Kohli, Matt Staib, Chengtao Li, Keyulu Xu, Josh Robinson, Ilija Bogunovic, Ariel Anders, Lawson Wong, Patrick Barragan, Gustavo Goretkin, Kenji Kawaguchi, Clement Gehring, Zelda Mariet, Caelan Garrett, Beomjoon Kim, Anurag Ajay, Ferran Alet, Rohan Chitnis, Rachel Holladay, Caris Moses, Kelsey Allen, Tom Silver, Peter Karkus, Evan Pu, Brian Axelrod and Xinkun Nie.

I was also fortunate to have collaborated with and/or mentored a few brilliant undergrads, MEng students and visiting students at MIT: Victoria Xia, Alex LaGrassa, Michael Amoako, Jiayuan Mao, Kevin Chen, Skye Thompson, Nishad Gothoskar, Jingxi Xu and Ivan Jutamulia. I would like to thank all of them for making my PhD experience so special.

Outside of lab, Irene Greif has been my mentor at GWAMIT's mentorship program since the second year of my PhD. Her stories of completing a PhD in the early days of computer science were so inspiring and constantly reminded me of the big picture. The dinner meetups with her and other mentees including Vibhaa Sivaraman, Manasi Vartak and Xijia Zheng were always pleasant experiences of showing support for each other. I would like to thank the mentoring group for the company and Irene for always being available and her kind help in various issues including looking for internships, attending graduate women events among others.

During my PhD, I also had the special pleasure of being the co-president of Graduate Women of Course 6 (GW6) together with Mandy Korpusik and Xijia Zheng in 2016. Organizing fun social events with Mandy and Xijia was perhaps one of the most fulfilling moments in these past few years as I interacted with other bright colleagues and built a sense of community with them. I am deeply thankful for being a part of GW6 and the unforgettable memories we shared.

My PhD training wouldn't be as smooth without the help and support from Leslie Kolodziejski, Janet Fischer, Teresa Cataldo and Marcia Davidson. The graduate women workshop organized by Leslie and Janet during my first year was unbelievably helpful. It made me aware of all the key milestones within the seemingly chaotic PhD life and allowed me to appreciate the joy of being a graduate student at MIT. I want to specifically thank Leslie, for her understanding and unwavering confidence in me.

I would also like to thank my good friends for their friendship and company over the years, including Yurong Lu, Yilian Zhu, Qianru Zhu, Xueying Zhao, Jing Zhang, Sizi Chen, Wei Wei,

Jing Huang, Alan Yeo, Xiaoxiao Meng, Biye Jiang, Jiali Mei. If everyone's life is a straight line in high-dimensional space, I am so glad that ours crossed.

Lastly, I would like to thank my family members in Shanghai, Xi'an, Beijing, Xiaogan and Toronto. None of them really expected me to pursue a PhD, not to mention at MIT. Whenever we have family gathering, they only try to convince me to take better care of my health and enjoy life more. Special thanks to my mom, Huiyan Wang, for her unconditional love. I would also like to thank my in-law family in Wisconsin, Xiamen and Jiangxi. Finally and most importantly, I cannot appreciate my husband Yuliang Leon Sun enough. Thank you for unreservedly sharing this journey with me and always being my co-pilot no matter what hurdles we met.

Biography

Zi Wang is a researcher at the Learning and Intelligent Systems Group at MIT Computer Science and Artificial Intelligence Laboratory. Her PhD research focuses on tackling problems related to robot learning, active learning for planning and Bayesian optimization. She holds the view that robot learning, machine learning, and artificial intelligence in general need to be assisted by strong human intelligence in order to be feasible.

Zi received her S.M. in Electrical Engineering and Computer Science from MIT in February 2016 and B.Eng. in Computer Science and Technology from Tsinghua University in July 2014. Zi is a recipient of the MIT Graduate Women of Excellence Award, Rising Star in EECS, and Google Anita Borg Scholarship. While at MIT, she served as co-president of Graduate Women in Course 6 (EECS), co-organizer of the first Machine Learning Across MIT Retreat and research mentor for undergraduate and master students.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Your life has a limit, but knowledge has none. If you use what is limited to pursue what has no limit, you will be distressed, and it is going to be dangerous if you continue doing so.

Chuang Tzu

Robot learning is an emerging field in which we use existing or develop new machine-learning techniques for robotics problems in the hope that, one day, robots can be deployed in a variety of environments, capable of completing a range of complex tasks such as doing laundry, cooking meals or cleaning up rooms.

Under the umbrella of robot learning, our wildest dream is that we humans no longer need to do the hard work of hand coding behaviors for the robot and the robot can learn everything by itself through trial and error. However, it has gradually become clear that simply applying tabula rasa approaches using, for example, end-to-end deep reinforcement learning [120] based approaches, to complex long-horizon problems with sparse rewards is unfortunately very data inefficient and impractical in reality, and structures must be imposed on the learning problem to reduce the sample complexity. Behind the scenes, though not emphasized, human knowledge is still playing an indispensable role to define some structures or behaviors of the robot.

Instead of "hiding" the usage of human knowledge in learning algorithms, can we make use of it more explicitly to assist machine learning and artificial intelligence (AI)? Throughout this thesis, you will find a common theme that tries to answer this question by finding a "sweet spot" balancing between hard-coded expert knowledge and data-based statistical machine learning approaches. This thesis embodies a practical mindset of how to enable robot learning: instead of collecting large-scale

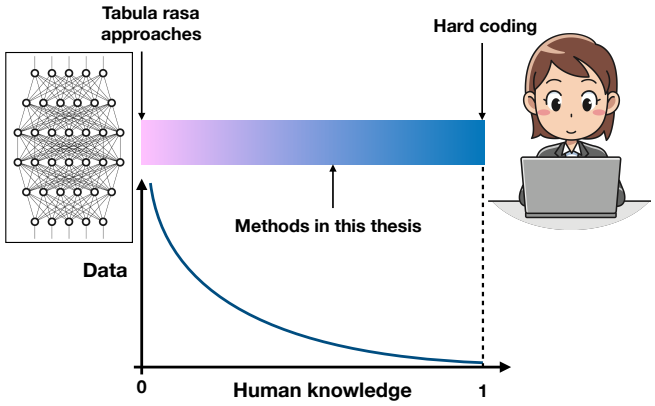


Figure 1-1: We find the balance between expert knowledge encoding and statistical machine learning. The approaches in this thesis can be viewed as a combination of data for learning and hard-coded strong priors derived from expert human intelligence.

data with an army of robots operating in parallel, we use expert domain knowledge to define strong priors to make up for the weakness of our learning robots.

Figure 1-1 illustrates roughly where this dissertation research lies in the spectrum of the mixture of human knowledge and data. In the leftmost, we do not impose much domain knowledge and tabula rasa approaches need mega-scale data to achieve good performance. In the rightmost, little data is required to be collected but an expert human engineer need to *hard code* the entire framework. My work in this thesis, in contrast to both extremes, lies in the middle.

Before we proceed to the main contents of this thesis, I would like to point out important distinctions and possible connections between this thesis and the field of Human Robot Interaction (HRI) [69], as illustrated by Figure 1-2.

The focus of HRI is to study how robots can interact with human users and vice versa. For example, the robot can be working on some tasks side by side with human workers, and the goal is to enable the robots to operate in a way that increases the performance of doing the tasks and/or potentially give the human workers a natural experience of collaborating with the robots. In addition to traditional engineering domains, HRI research can touch on many other fields due to studying both human and robots, e.g. Cognitive Science, Psychology and Social Science. Notably though, the HRI researchers themselves are usually not the humans whose interactions with robots are studied.

On the contrary, this thesis on robot learning studies how researchers can provide strong priors through hard-coding, and how those strong priors can make the robots learn and perform well. We do not study any of the cognitive parts of the human experts (a.k.a. researchers ourselves who understand relatively well how learning systems work). Our main goal is to study how to make our

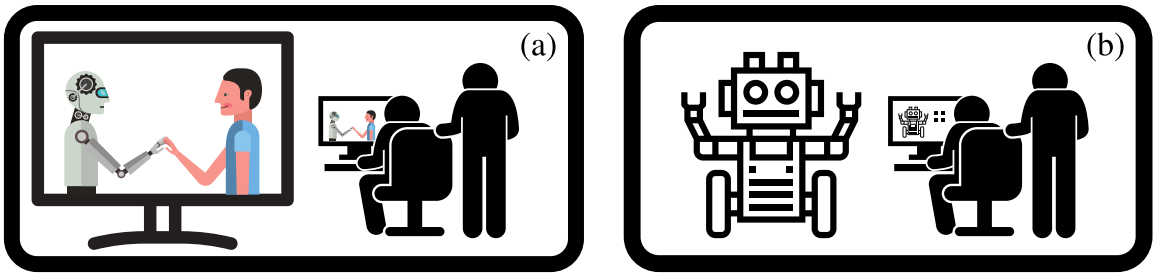


Figure 1-2: (a) An illustration of the study of Human Robot Interaction, which focuses on the interaction between human users and robots. The researchers typically do not consider themselves in the study. (b) An illustration of the focus of this thesis. We study how to enable robots to do difficult manipulation tasks with strong priors given by us the researchers.

robot learn with a realistic amount of data and expand its capabilities through integrated learning and planning. One extension of our work could be to study robot learning through interactions with human users, in which case we will be studying HRI as well as machine learning. Alternatively, if we are more lenient with definitions, our work can perhaps also be viewed as human robot interaction on the algorithmic or coding level. However, again, this thesis does not study the human side of the story.

Stemming from these viewpoints, I will discuss the challenges we are facing and my contributions towards solving them from both applied and theoretical perspectives. On the practical side, towards building a capable robotic system, I will present integrated learning and planning algorithms that has both inputs from experts and actively collected data; on the theoretical side, I will show how to better understand the sample complexity of some active data acquisition algorithms that assume different forms of priors from human, which in turn guide the development of our robot learning system.

1.1 Overview of the problems

We view a robot as an autonomous agent that actively engages in an environment. We use *states* to describe conditions of the robot and the environment. The way our robot engages with the environment is formally captured by a potentially infinite set of *actions*. For example, in Figure 1-3, a robot called PR2 is operating in an environment that has a table and several objects on that table. The robot can move its base, move its arms around, lift its torso, and open and close its grippers attached to the arms; these are typically considered its actions though there can also be higher-level actions such as push, pick, place. Notice that such actions can be continuously parameterized, e.g.

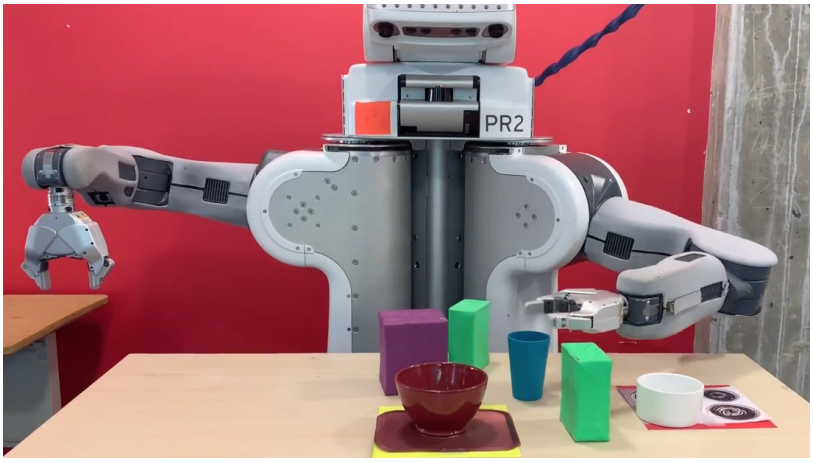


Figure 1-3: A PR2 robot is manipulating some objects on a table.

move forward 0.1 meter; and hence the set of all possible actions is infinite.

Our ultimate goal is to enable robots to solve high-level long-horizon problems as they arise in the real world. To be capable of accomplishing a range of tasks, our robot needs to have the ability of solving an important type of problems: given the current state and an objective, find a sequence of actions that achieve the objective. We as researchers and/or engineers in the lab, on the other hand, need to figure out how to design such a capable robot before it is delivered out of the lab to solve those hard problems. In particular, we address one special period of the design process for the software that is going to be the "brain" of a robot.

While the robot is in the lab, we have many options of what to do with it to build its "brain" and they mainly fall into two categories: encoding our prior knowledge to the robot or letting the robot collect its own first-hand experience. We may also anticipate that the robot will keep collecting new experience once it is delivered out of the lab. Machine learning, as a toolbox of statistical methods, now comes in handy to capture the algorithmic and theoretical foundations rooted in the software we design for the robot. This thesis mainly addresses the following robot learning challenges.

- **Model learning for planning**. Planning is necessary for long-horizon problems and it requires models of actions to be planned with. There are several key questions: (1) how to define the model? (2) how to learn the model? (3) how to use the model in the planner? The model typically describes the relations among the current state, the action taken by the robot and the resulting state after the action is completed; however, the descriptions of models could take different forms depending on the characteristics of the planner. For example, in some cases, the planner requires generative models that suggest good parameters for the planner to try out. Hence, depending on how the model is going to be used, we may set our learning

objective differently.

- **Active data acquisition**. Learning in general requires data, but data collection for robotics tasks can be very expensive both in the real-world or in high-fidelity simulations. It is important to carefully decide what data to collect to reduce the sample complexity. In particular, data collection in our robotics tasks typically reduces to the action selection problem: which action should be chosen to let the robot experiment with its domain. Note that each "experimentation" could take a long time to complete. We formulate this problem as a black-box function optimization problem where the black-box function is expensive to evaluate, noisy and no gradient information is available. The goal is to design an active data acquisition strategy so that not many evaluations on the function are necessary to find the global optimum of the function. This problem is often addressed in the framework of Bayesian optimization.

These two learning challenges are not always exclusive of each other. When we try to learn a model or use a model for planning, often it is desirable to collect data in an active way so that we can learn and plan more efficiently. When developing active data acquisition methods, it is typically very useful to know the objective we are concerned with; for example, the intuitive understanding of whether a transition model tends to be discontinuous or not can potentially give us lots of information.

It is also possible to characterize a learning problem as how to use existing data generated within a planner to solve future planning problems more efficiently. This is quite different to learning transition models, but it is indeed a type of model that models some characteristics of certain statistics in the planner. Hence one can view this problem as a model learning problem. However, one can also view this as an active data acquisition problem since this problem can be reframed as how to actively query the planner in order to get better planning performance.

Next, we will overview the main contributions of this thesis. It is perhaps good to bear in mind that though we separate the contributions into these two sets of problems corresponding to model learning and active data acquisition, the detailed solutions we provide in later chapters may be addressing these two problems jointly.

1.2 Main contributions

There are roughly two categories of key contributions in this thesis. First, we propose new problem formulations to identify the important learning problems and their possible solutions with the goal

of enabling robots to solve complex manipulation tasks that arise in the real world. Second, we abstract out a mathematical optimization problem that lies in the center of the learning problems we have identified in robotic manipulation tasks; and we offer theoretical and algorithmic insights into better solutions for optimizing black-box functions.

1.2.1 Model learning for planning

As opposed to tabular rasa approaches, our focus on robot learning is to build and use structured models of actions in a planner. We have developed data-efficient learning methods that make use of strong priors to solve questions in model learning.

Learning what conditions should hold before using an action

Solving long-horizon problems in complex domains requires flexible generative planning that can combine primitive abilities in novel combinations to solve problems as they arise in the world. In order to plan to combine primitive actions, we must have models of the preconditions and effects of those actions: under what conditions will executing this primitive achieve some particular effect in the world? We present a framework that demonstrates how active learning and hand-designed abstract models of skills can be integrated to learn abstract pre-condition models of an action. We use expert domain knowledge to define the model as a constraint on relevant parameters of the action. The constraint describes the condition of the parameters under which the action can achieve the desired effect. Then, the robot learns the constraint by actively gathering data through experimentation on using the action with different parameter settings. We present this work in Chapter 3 which previously appeared as part of [185].

Learning how to sample the skill parameters

Once we obtain models of the actions, we integrate them into an existing domain description for the other robot skills that is suitable for use with a task and motion planner. We use diversity-aware samplers (in Chapter 3) and Bayesian optimization with meta-learned priors (in Chapter 7) to solve the question of how to generate a "good" sequence of action parameters where the goodness depends on whether the planning problem can be solved efficiently. This work is part of [100, 185, 189].

Learning what parameters in the environment are relevant

We introduce SPArse RElational transition models (SPARE), which learn the combinations of relevant parameters for a model of an action. As part of the prior for learning, we assume the availability of deictic references such as *above* or *close to* that defines a sparse set of relations among objects in the scene. The learning algorithm decides which relations to use in order to find the set of appropriate parameters. This work detailed in Chapter 4 has appeared as part of [199].

Learning non-Gaussian transition models in continuous state-action spaces

We introduce a framework for model learning and planning in stochastic domains with continuous state and action spaces and non-Gaussian transition models. It is efficient because (1) local models are estimated only when the planner requires them; (2) the planner focuses on states that are most relevant to the current planning problem; and (3) the planner focuses on the most informative and/or high-value actions. Our theoretical analysis shows the validity and asymptotic optimality of the proposed approach under some prior assumptions. Empirically, we demonstrate the effectiveness of our algorithm on a simulated multi-modal pushing problem. This work shown in Chapter 5 previously appeared as [188].

1.2.2 Active data acquisition via Bayesian optimization

A key question in robot learning is how to actively acquire data for learning and decision making. This question appears in many places, e.g. when deciding which experimentation to run to gather information [185], which action to try in the planner [100, 185, 189], and what actions to focus on when learning their values [188]. We formulate this problem as a gradient-free optimization problem for non-convex, multi-peak functions where sample-complexity matters because function evaluations can be expensive. We study the underlying principles of this problem and use the intuitions gained from theoretical studies to help to design algorithms for robot learning.

Bayesian optimization is one way to formulate the gradient-free global optimization problem in both discrete and continuous input spaces, building on the ideas from Bayesian modeling. We assume there is a scoring function that measures how good each input is. The goal is to optimize the scoring function by actively querying the scores of a sequence of inputs. We show solutions to the critical computational issues of Bayesian optimization in both decision making [187, 191] and model updating [189] and give insights into the regret bound and connections among a num-

ber of existing query selection criteria. We also present new approaches that have made Bayesian optimization possible for large-scale high-dimensional problems.

Strategies for query selection

We explore the usage of *maximum values* of target functions in the design of novel Bayesian optimization approaches, and make two main contributions: First, our findings bridge missing pieces that reveal connections between entropy search methods, upper confidence bounds and probability of improvement. By exploiting these connections, we establish theoretical regret bounds for variants of entropy search and probability of improvement guided by max-values. Second, we obtain a much more efficient version of popular entropy search methods that maintain the good empirical performance. We demonstrate computational efficiency and effectiveness of Bayesian optimization approaches guided by max-values on a variety of low and high-dimensional tasks both in machine learning and simulated robot control. This work is presented in Chapter 6 and it previously appeared as [187, 191].

Empirical Bayes for model selection

Bayesian optimization usually assumes that a Bayesian prior is given. However, the strong theoretical guarantees in Bayesian optimization are often regrettably compromised in practice because of unknown parameters in the prior. We adopt a variant of empirical Bayes and show that, by estimating the Gaussian process prior from offline data sampled from the same prior and constructing unbiased estimators of the posterior, variants of both GP-UCB and *probability of improvement* achieve a near-zero regret bound, which decreases to a constant proportional to the observational noise as the number of offline data and the number of online evaluations increase. Empirically, we have verified our approach on challenging simulated robotic problems featuring task and motion planning. This work is presented in Chapter 7 and it previously appeared as [189].

Scaling up to higher dimensions and larger data sets

Many cases, such as the ones with high-dimensional inputs, may require a much larger number of observations for Bayesian optimization. Despite an abundance of observations thanks to parallel experiments, current Bayesian optimization techniques have been limited to a few thousand observations. We propose novel approaches to address three current challenges in Bayesian optimization *simultaneously*: (1) large-scale observations; (2) high dimensional input spaces; and (3) selections

of batch queries that balance quality and diversity. The key idea is to operate on an ensemble of additive Gaussian process models, each of which possesses a randomized strategy to divide and conquer. We show unprecedented, previously impossible results of scaling up Bayesian optimization to tens of thousands of observations within minutes of computation. This work is detailed in Chapter 8 and previously appeared as [190, 186].

Chapter 2

Background and related work

In this chapter, we review the technical background that lays the foundation for this thesis. In Section 2.1, we introduce Gaussian processes, a probabilistic model commonly used in machine learning. In Section 2.2, we describe how Bayesian optimization methods make use of Gaussian processes to actively acquire data, which is a critical ability for learning robots. In Section 2.3, we draw connections between our robot learning problem formulation and reinforcement learning. Lastly in Section 2.4, we explain different types of priors that play important roles in learning.

2.1 Gaussian processes

In many of our robot learning problems, it is crucial to make predictions about unseen data and model the uncertainty of our predictions. For example, one of our tasks is to model how good a pouring action is. The action is parameterized by some control parameters of the robot and some environment parameters that describe the objects (e.g. cups and liquid in the cup) the robot is interaction with. We denote these parameters as input x and we use a function f to evaluate how well the pouring action performs, measured by how much liquid is poured into the target cup. One nice tool to make predictions and model our uncertainty about the predictions is Gaussian processes.

A Gaussian process (GP) [151] is a distribution over functions and we use it to describe our prior on our function of interest. We use $f \sim GP(\mu, k)$ to denote that function f is distributed according to the GP parameterized by mean function μ and kernel k. By definition, any finite set of function values has a multivariate Gaussian distribution, namely, for function inputs $x_1, x_2, \cdots, x_n$,

the function evaluations $f(x_1), f(x_2), \cdots, f(x_n)$ have the following distribution,

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \cdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \cdots \\ \mu(x_n) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \cdots & \cdots & \cdots & \cdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix} \right)$$

where $\mathcal{N}(\mu, K)$ denotes a multivariate distribution parameterized by mean vector μ and covariance matrix K.

There are two types of uncertainty involved in modeling our function: epistemic uncertainty and aleatoric uncertainty. Continuing our example on the pouring action, notice that we are not modeling some aspects of the physical process of pouring, e.g. the air resistance or the liquid viscosity, and the execution of actions with the same parameters may not result in exactly the same movement of the robot due to noise in the control process. As a result, our observations of function f will also be noisy, and this is known as the aleatoric uncertainty. For simplicity, we are going to assume the observations y has a normal distribution centered at $f(x)$ with variance σ^2; that is, $y \sim \mathcal{N}(f(x), \sigma^2)$.

Aleatoric uncertainty models the intrinsic noise in the measurement and cannot be reduced by collecting more data. While epistemic uncertainty, on the other hand, is reflected on our belief of what function f is. As we collect more observations on function f, we can reduce the epistemic uncertainty as described by the posterior. Given a set of observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, the posterior on f is still a Gaussian process; that is, $f \sim GP(\mu_t, k_t)$. We can obtain closed-form posterior mean

$$\mu_t(x) = \mu(x) + k(x, \boldsymbol{x}_t)(K_t + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \boldsymbol{\mu}_t), \tag{2.1}$$

and posterior covariance

$$k_t(x, x') = k(x, x') - k(x, \boldsymbol{x}_t)(K_t + \sigma^2 \boldsymbol{I})^{-1} k(x', \boldsymbol{x}_t)^{\mathrm{T}},$$

where $\boldsymbol{y}_t = [y_\tau]_{\tau=1}^T$, $\boldsymbol{x}_t = [x_\tau]_{\tau=1}^T$, $\mu(x_t) = [\mu(x_\tau)]_{\tau=1}^t$, $k(x, \boldsymbol{x}_t) = [k(x, x_\tau)]_{\tau=1}^T$ and $K_t =$

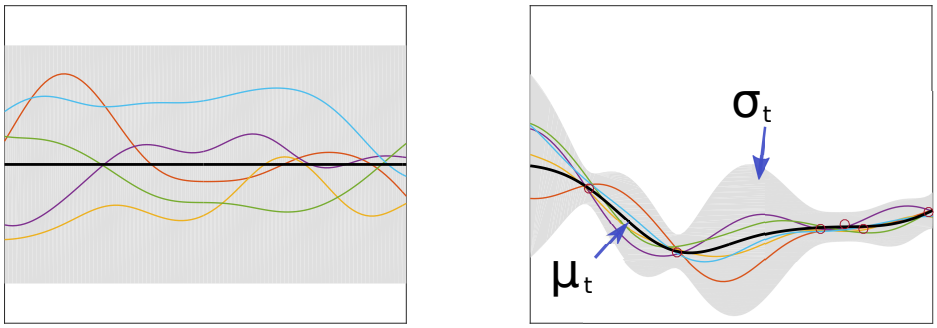


Figure 2-1: Visualization of a Gaussian process prior for a 1D function f (left) and the posterior for f (right) given some observations illustrated by the red circles. The colored lines are samples from the prior and the posterior.

$[k(x_i, x_j)]_{i\in[T],j\in[T]}$. Notice that the posterior variance is given by

$$\sigma_t^2(x) = k_t(x, x). \tag{2.2}$$

We visualize a Gaussian process modeling a 1D function f in Figure 2-1. The prior mean μ is a constant and the kernel is a squared exponential kernel $k(x, x') = a \exp b(x - x')^2$ for some constant values a, b. As more function values are observed, our uncertainty about what the function looks like also decreases, as reflected by the more constrained samples from the posterior.

2.2 Bayesian active learning

In situations where evaluating functions requires a lengthy experiment on the robot or high-fidelity simulation, we need to carefully consider what data is worth gathering. The value of the data depends on our goal of what to do with them. We here introduce two types of closely related goals, optimization and level set estimation, and show the role of active learning in these two tasks.

2.2.1 Bayesian optimization

Bayesian optimization [163, 169] is a popular framework for sequential decisions making to optimize black-box functions that are expensive to evaluate. This optimization problem is closely related to experimental design, bandit problems, global optimization and derivative-free optimization.

$$\textbf{Problem: } \underset{x\in\mathfrak{X}}{\text{maximize}} \quad f(x)$$

We sequentially select inputs to evaluate f and our goal here is to reach the global optimum with as few evaluations as possible. We assume the search space $\mathfrak{X}$ is a compact set in $\mathbb{R}^d$. In Bayesian optimization, a common strategy is to model function f with a Gaussian process $GP(\mu, k)$. In each iteration t, we use *acquisition functions* $\alpha_t(x)$ as the cheap surrogate to evaluate how beneficial it is to evaluate different inputs, and choose to evaluate the input that optimizes acquisition function $\alpha_t(x)$. Once we have the full dataset D_T, one option is to recommend the input that achieves the highest observed function value as the $\arg\max$. We show the pseudo code in Algorithm 1.

Algorithm 1 A typical paradigm of Bayesian optimization

function BAYESIANOPTIMIZATION (f, D_0)
  for $t = 1, \cdots, T$ **do**
    $\mu_{t-1}(\cdot), \sigma_{t-1}(\cdot) \leftarrow$ MODEL(D_{t-1})
    $\alpha_{t-1}(\cdot) \leftarrow$ACQUISITION ($\mu_{t-1}, \sigma_{t-1}$)
    $x_t \leftarrow \arg\max_{x \in \mathfrak{X}} \alpha_{t-1}(x)$
    Observe $y_t \sim \mathcal{N}(f(x_t), \sigma^2)$
    $D_t \leftarrow D_{t-1} \cup \{(x_t, y_t)\}$
  end for
end function

There are many options to define the acquisition function. One idea is to use the upper confidence bound (UCB) [8, 171] derived from the Gaussian process posterior model. That is, we use

$$\alpha_t(x) = \mu_t(x) + \beta\sigma_t(x)$$

where the posterior mean μ_t is defined in Equation (2.1) and the posterior standard deviation in Equation (2.2). The coefficient β is a parameter balancing between exploration and exploitation. If β is high, the standard deviation plays a more important role in the UCB acquisition function. The standard deviation is high in the regions where we have not obtained many observations yet, and thus we focus more on exploring the uncharted search space. If β is low, we focus more on exploiting what we have discovered so far, represented by the posterior mean; this typically results in choosing to evaluate inputs that are close to the best input(s) we have observed so far.

2.2.2 Bayesian level set estimation

Level set estimation [28, 70] is about identifying the boundaries that separate the search space into two parts: those who have function values above a threshold (super-level set) and those below

(sub-level set). For example, when we are scoring the pour action, it is satisfactory as long as the action parameters result in having at least 95% liquid in the target cup. We would like to identify the region containing these good kinds of action parameters, which is a super-level set estimation problem. Note that we can always subtract the threshold from the function.

This problem is slightly different to active learning problems for Gaussian process classification, in which case a binary or multi-class label is used to describe the function output for each data point. For level set estimation, the function output can be continuous.

Without loss of generality, we use a threshold of 0 and state the problem below.

$$\textbf{Problem:} \text{ identify } \{x \in \mathfrak{X} \mid f(x) > 0\}$$

Here we only included identifying the super-level set because the sub-level set is simply the complement of the super-level set.

Similar to Bayesian optimization, we want to evaluate the function on few inputs to recover the level set. Again, we use a Gaussian process to model function f and we use acquisition functions α_t to guide the selection of inputs in each iteration t. Once we finish collecting the dataset D_T, we can predict the probability of $f(x) > 0$ for each $x \in \mathfrak{X}$.

Typical algorithms for Bayesian level set estimation follows the same paradigm as Algorithm 1, except defining acquisition functions differently. For example, [28] uses

$$\alpha_t(x) = |\mu_t(x)| + \beta\sigma_t(x),$$

which gains high value when the posterior mean $\mu_t(x)$ is close to the threshold 0 or has high uncertainty reflected by the posterior standard deviation $\sigma_t(x)$.

2.3 Reinforcement learning

Reinforcement learning [173, 175, 123, 86] is a general framework that studies how an autonomous agent can take actions in an environment to receive as much reward as possible. It is closely related to optimal control [19, 21, 18]. In the most basic form, we can describe a reinforcement learning problem with a finite Markov decision process (MDP).

An MDP is specified by a tuple (S, A, P, R, γ) consisting of a finite set of states S, a finite set of actions A, transition model P, reward model R and a discount factor γ. The transition model

$P(s' \mid s, a)$ describes the probability of transitioning to state s' after taking action a at state s. The reward model $R(s, a, s')$ uses a real value to describe the reward being received at state s' after taking action a at state s. The discount factor γ satisfies $1 > \gamma > 0$. Beginning from the start state s_0, the agent follows a policy $\pi : S \mapsto A$ that defines what action should be taken at any possible state. The goal here is to find the policy π that maximizes the cumulative rewards.

$$\begin{aligned} \textbf{Problem: } & \underset{\pi}{\text{maximize}} \;\; \mathbb{E}\left[\sum_{t=0}^{\infty} R(s_t, a_t, s_{t+1})\right] \\ & \text{subject to} \;\; \pi(s_t) = a_t, \;\; s_{t+1} \sim P(s_{t+1} \mid s_t, a_t) \end{aligned}$$

Our robot learning problems can be viewed as a variant of the reinforcement learning problem: we are also interested in making a sequence of decisions on what actions to take in order to achieve an objective. It is indeed possible to characterize our objective as a reward function, e.g. setting a very high reward at a goal state in our objective. However, unlike typical reinforcement learning paradigms, our states and actions may contain not only discrete variables (e.g. which object to operate with) but also continuous parameters (e.g. the control for grasping an object); our problems are much more complex (e.g. cook a meal) comparing to typical control problems solved by reinforcement learning methods (e.g. low-level control tasks); instead of finding a policy that maps from every state to an action, it is enough for us to have a "path" that leads the robot from its current state to a goal state, with the option to replan when necessary; These differences in assumptions and objectives result in learning strategies different to classic reinforcement learning methods as exemplified by this thesis.

2.4 Types of priors in learning

Priors are the built-in knowledge that exists in a learning agent before it starts learning by either acquiring data in the world or training on a given dataset. Typically, priors are hand-designed by experts, but they can also be learned via meta-learning. Priors can take many forms and we summarize below.

Model structures

Human experts can also assist machines to learn more efficiently by specifying model structures a priori while potentially leaving the rest of the model to be learned from data. For example,

convolutional structures in neural networks [119, 106] build in translation invariance to the model and they are especially successful for image recognition tasks. Graphical model approaches such as topic modeling [22] use expert-provided hypotheses on how data is generated to enable efficient Bayesian inference. One more example is value iteration network [177] and QMDP-Net [94] type methods, which construct fully differentiable neural networks with built-in planning capabilities as part of the model. Humans specified model structures can be viewed as constraints on the search space of machine learning models, and they typically enable more data-efficient learning. In this thesis, for example, Chapter 3 describes a method that learns constraints in the descriptions of models of actions partly specified by human and Chapter 8 uses a special graphical model for additive Gaussian processes to scale up Bayesian optimization to higher dimensions.

Assumptions on what is given

Sometimes machine learning can be made easier to achieve by utilizing unique assumptions specified by human experts. These assumptions are given a priori to the learning approaches as guidance to the model. For example, Andreas et. al. [5] used the assumption that symbolic labels of components of a task are available to the learner to ease the learning process of reusable components in the policies for different tasks. In Chapter 4, we assume the availability of symbolic relations among objects, which can then be used to build abstract models of actions. These assumptions can be viewed as a type of prior since they exist before the learning process begins.

Priors in Bayesian modeling

In Bayesian modeling, priors are explicitly specified as initial probabilistic beliefs on certain variables. For example, in topic modeling [22, 34], probabilistic priors are specified to describe the distribution of words in each topic and the distribution of topics in each document; in Bayesian optimization explained in Section 2.2.1, a Gaussian process prior is used to describe the initial belief on what the function looks like.

In this thesis, we explore different usages of priors in robot learning to significantly reduce sample complexity and make it feasible for robots to learn with small data.

Part I

Learning models for planning

Chapter 3

Active model learning and diverse action sampling for task and motion planning

For a robot to be effective in a domain that combines novel sensorimotor primitives, such as pouring or stirring, with long-horizon, high-level task objectives, such as cooking a meal or making a cup of coffee, it is necessary to acquire models of these primitives to use in planning robot motions and manipulations. These models characterize (a) conditions under which the primitive is likely to succeed and (b) the effects of the primitive on the state of the world.

Figure 3-1 illustrates several instances of a parameterized motor primitive for pouring in a simple two-dimensional domain. The primitive action has control parameters θ that govern the rate at which the cup is tipped and target velocity of the poured material. In addition, several properties of the situation in which the pouring occurs are very relevant for its success: robot configuration c_R, pouring cup pose and size p_A, s_A, and target cup pose and size p_B, s_B. To model the effects of the action we need to specify c'_R and p'_A, the resulting robot configuration and pose of the pouring cup A. Only for some settings of the parameters $(c_R, p_A, s_A, p_B, s_B, \theta, c'_R, p'_A) \in \chi$ is the action feasible: one key objective of our work is to efficiently learn a representation of the feasible region χ.

For learning this model, each training example requires running the primitive, which is expensive on real robot hardware and even in high-fidelity simulation. To minimize the amount of training data required, we actively select each setting in which the primitive is executed, with the goal of

Part of this chapter was previously published as
Zi Wang and Caelan Reed Garrett and Leslie Pack Kaelbling and Tomás Lozano-Pérez. Active model learning and diverse action sampling for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

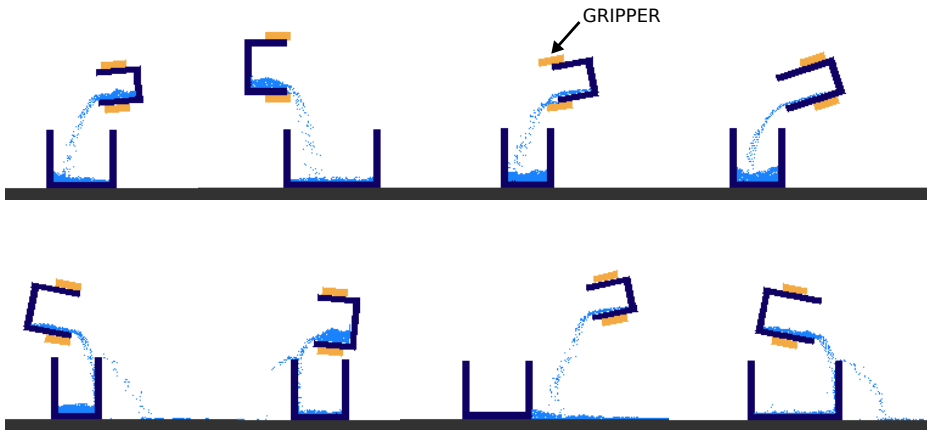


Figure 3-1: Several examples of executing a pouring primitive with different settings, including control parameters, cup sizes, and relative placements.

obtaining as much information as possible about how to use the primitive. This results in a dramatic reduction in required examples over our preliminary work [88] on this problem.

Given a model of a primitive, embodied in χ, we utilize existing sample-based algorithms for *task and motion planning* (TAMP) to find plans. To use the model within the planner, it is necessary to select candidate instances of the primitives for expansion during the search for a plan. The objective here is not to gain information, but to select primitive instances that are likely to be successful. It is not enough to select a single instance, however, because there may be other circumstances that make a particular instance infeasible within a planning context: for example, although the most reliable way to grasp a particular object might be from the top, the robot might encounter the object situated in a cupboard in a way that makes the top grasp infeasible. Thus, our action-sampling mechanism must select instances that are both likely to succeed and are *diverse* from one another, so that the planner has coverage of the space of possible actions.

One difficulty in sampling χ is that it inhabits a lower-dimensional submanifold of the space it is defined in, because some relations among robot configurations and object poses, for example, are functional. The STRIPStream planner [64, 65] introduced a strategy for sampling from such *dimensionality-reducing* constraints by constructing *conditional samplers* that, given values of some variables, generate values of the other variables that satisfy the constraint. Our goal in this chapter is to learn and use conditional samplers within the STRIPStream planner.

Our technical strategy for addressing the problems of (a) learning success constraints and (b) generating diverse samples is based on an explicit representation of uncertainty about an underlying *scoring* function that measures the quality or likelihood of success of a parameter vector, and

uses Gaussian process (GP) techniques to sample for information-gathering during learning and for success probability and diversity during planning. We begin by describing some basic background, discuss related work, describe our methods in technical detail, and then present experimental results of learning and planning with several motor primitives in a two-dimensional dynamic simulator.

3.1 Problem formulation and background

We will focus on the formal problem of learning and using a conditional sampler of the form $\phi(\theta \mid \alpha)$, where $\alpha \in R^{d_\alpha}$ is a vector of contextual parameters and $\theta \in B$ is a vector of parameters that are to be generated, conditioned on α. We assume in the following that the domain of θ is a hyper-rectangular space $B = [0,1]^{d_\theta} \subset R^{d_\theta}$, but generalization to other topologies is possible. The conditional sampler generates samples of θ such that $(\theta, \alpha) \in \chi$ where $\chi \subset R^{d_\alpha + d_\theta}$ characterizes the set of world states and parameters for which the skill is feasible. We assume that χ can be expressed in the form of an inequality constraint $\chi(\theta, \alpha) = (g(\theta, \alpha) > 0)$, where $g(\cdot)$ is a scoring function with arguments θ and α and χ can be viewed as a classifier based on the scoring function g.

We denote the *super level-set* of the scoring function given α by $A_\alpha = \{\theta \in B \mid g(\theta, \alpha) > 0\}$. For example, the scoring function $g(\cdot)$ for pouring might be the proportion of poured liquid that actually ends up in the target cup, minus some target proportion. We assume the availability of values of such a score function during training rather than just binary labels of success or failure. In the following, we give basic background on two important components of our method: Gaussian processes and STRIPStream.

As introduced in Section 2.1, Gaussian processes (GPs) are distributions over functions, and popular priors for Bayesian non-parametric regression. For compactness, we use slightly different notations. In this chapter, we use the Gaussian process $\mathrm{GP}(0, k)$ which has mean zero and covariance (kernel) function $k(x, x')$. Let f be a true underlying function sampled from $\mathrm{GP}(0, k)$. Given a set of observations $\mathcal{D} = \{(x_t, y_t)\}_{t=1}^{|\mathcal{D}|}$, where y_t is an evaluation of f at x_t corrupted by i.i.d additive Gaussian noise $\mathcal{N}(0, \zeta^2)$, we obtain a posterior GP, with mean $\mu(x) = \boldsymbol{k}^{\mathcal{D}}(x)^{\mathrm{T}}(\boldsymbol{K}^{\mathcal{D}} + \zeta^2 \boldsymbol{I})^{-1}\boldsymbol{y}^{\mathcal{D}}$ and covariance $\sigma^2(x, x') = k(x, x') - \boldsymbol{k}^{\mathcal{D}}(x)^{\mathrm{T}}(\boldsymbol{K}^{\mathcal{D}} + \zeta^2 \boldsymbol{I})^{-1}\boldsymbol{k}_t(x')$ where the kernel matrix $\boldsymbol{K}^{\mathcal{D}} = [k(x_i, x_j)]_{x_i, x_j \in \mathcal{D}}$ and $\boldsymbol{k}^{\mathcal{D}}(x) = [k(x_i, x)]_{x_i \in \mathcal{D}}$ [151]. With slight abuse of notation, we denote the posterior variance by $\sigma^2(x) = \sigma^2(x, x)$, and the posterior GP by $\mathrm{GP}(\mu, \sigma)$.

STRIPStream [65] is a framework for incorporating blackbox sampling procedures in a planning

language. It extends the STRIPS planning language [58] by adding *streams*, declarative specifications of conditional generators. Streams have previously been used to model off-the-shelf motion planners, collision checkers, inverse kinematic solvers. In this work, we learn new conditional generators, such as samplers for pouring, and incorporate them using streams.

3.2 Related Work

Our work draws ideas from model learning, probabilistic modeling of functions, and task and motion planning (TAMP).

There is a large amount of work on learning individual motor primitives such pushing [107, 77], scooping [159], and pouring [143, 178, 25, 200, 158]. We focus on the task of learning models of these primitives suitable for multi-step planning. We extend a particular formulation of planning model learning [88], where constraint-based pre-image models are learned for parameterized action primitives, by giving a probabilistic characterization of the pre-image and using these models during planning.

Other approaches exist to learning models of the preconditions and effects of sensorimotor skills suitable for planning. One [101] constructs a completely symbolic model of skills that enable purely symbolic task planning. Our method, on the other hand, learns hybrid models, involving continuous parameters. Another [108] learns image classifiers for pre-conditions but does not support general-purpose planning.

We use GP-based level set estimation [28, 70, 151, 23] to model the feasible regions (super level set of the scoring function) of action parameters. We use the *straddle* algorithm [28] to actively sample from the function threshold, in order to estimate the super level set that satisfy the constraint with high probability. Our methods can be extended to other function approximators that gives uncertainty estimates, such as Bayesian neural networks and their variants [63, 112].

Alternatively, one can use GP classification methods with active learning [93] to model our constraints. Active learning of GP classifiers was often used for modeling safety constraints to help perform safe exploration [161, 54]. The focus of this work, however, is to present a suite of approaches to address not only how to actively learn a model but also how to use learned models and solve complex long-horizon manipulation tasks. In this work, we only focus on one setting of the active model learning problem (level set estimation with GP regression) but other active learning approaches can certainly be used.

Determinantal point processes (DPPs) [110] are typically used for diversity-aware sampling. However, both sampling from a continuous DPP [74] and learning the kernel of a DPP [1] are challenging.

Several approaches to TAMP utilize generators to enumerate infinite sequences of values [87, 172, 64]. Our learned samplers can be incorporated in any of these approaches. Additionally, some recent papers have investigated learning effective samplers within the context of TAMP. Chitnis et al. [36] frame learning plan parameters as a reinforcement learning problem and learn a randomized policy that samples from a discrete set of robot base and object poses. Kim et al. [98] proposed a method for selecting from a discrete set of samples by scoring new samples based on their correlation with previously attempted samples. In subsequent work, they instead train a Generative Adversarial Network to directly produce a distribution of satisfactory samples [99].

3.3 Active sampling for learning and planning

Our objective in the learning phase is to efficiently gather data to characterize the conditional super-level-sets A_α with high confidence. We use a GP on the score function g to select informative queries using a level-set estimation approach. Our objective in the planning phase is to select a diverse set of samples $\{\theta_i\}$ for which it is likely that $\theta \in A_\alpha$. We do this in two steps: first, we use a novel risk-aware sampler to generate θ values that satisfy the constraint with high probability; second, we integrate this sampler with STRIPStream, where we generate samples from this set that represent its diversity, in order to expose the full variety of choices to the planner.

3.3.1 Actively learning the constraint with a GP

Our goal is to be able to sample from the super level set $A_\alpha = \{\theta \in B \mid g(\theta, \alpha) > 0\}$ for any given context α, which requires learning the decision boundary $g(\theta, \alpha) = 0$. During training, we select α values from a distribution reflecting naturally occurring contexts in the underlying domain. Note that learning the level-set is a different objective from learning all of the function values well, and so it must be handled differently from typical GP-based active learning.

For each α value in the training set, we apply the *straddle* algorithm [28] to actively select samples of θ for evaluation by running the motor primitive. After we obtain each new evaluation of $g(\theta, \alpha)$, the data-set $\mathcal{D}$ is augmented with pair $\langle(\theta, \alpha), g(\theta, \alpha)\rangle$, and used to update the GP. The straddle algorithm selects θ that maximizes the acquisition function $\psi(\theta; \alpha, \mu, \sigma) = -|\mu(\theta, \alpha)| +$

$1.96\sigma(\theta, \alpha)$. It has a high value for values of θ that are near the boundary for the given α or for which the score function is highly uncertain. The parameter 1.96 is selected such that if $\psi(\theta)$ is negative, θ has less than 5 percent chance of being in the level set. In practice, this heuristic has been observed to deliver state-of-the-art learning performance for level set estimation [23, 70]. After each new evaluation, we retrain the Gaussian process by maximizing its marginal data-likelihood with respect to its hyper-parameters. Alg. 2 specifies the algorithm; GP-predict$(\cdot)$ computes the posterior mean and variance as explained in Sec. 3.1.

Algorithm 2 Active Bayesian Level Set Estimation

1: Given initial data set $\mathcal{D}$, context α, desired number of samples T
2: **for** $t = 1 \rightarrow T$ **do**
3: $\quad \mu, \sigma \leftarrow$ GP-predict($\mathcal{D}$)
4: $\quad \theta \leftarrow \arg\max_\theta \psi(\theta; \alpha, \mu, \sigma)$
5: $\quad y \leftarrow g(\theta, \alpha)$
6: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{((\theta, \alpha), y)\}$
7: **end for**
8: **return** $\mathcal{D}$

3.3.2 Risk-aware adaptive sampling for constraint satisfaction

Now we can use this Bayesian estimate of the scoring function g to select action instances for planning. Given a new context α, which need not have occured in the training set—the GP will provide generalization over contexts—we would like to sample a sequence of $\theta \in B$ such that with high probability, $g(\theta, \alpha) \geq 0$. In order to guarantee this, we adopt a concentration bound and a union bound on the predictive scores of the samples. Notice that by construction of the GP, the predictive scores are Gaussian random variables. The following is a direct corollary of Lemma 3.2 of [191].

Corollary 3.3.1. *Let* $g(\theta) \sim \mathrm{GP}(\mu, \sigma)$, $\delta \in (0, 1)$ *and set* $\beta_i^* = (2\log(\pi_i/2\delta))^{\frac{1}{2}}$, *where* $\sum_{i=1}^{T} \pi_i^{-1} \leq 1$, $\pi_i > 0$. *If* $\mu(\theta_i) > \beta_i^* \, \sigma(\theta_i), \forall i = 1, \cdots, T$, *then* $\Pr[g(\theta_i) > 0, \forall i] \geq 1 - \delta$.

Define the high-probability super-level-set of θ given context α as $\hat{A}_\alpha = \{\theta \mid \mu(\theta, \alpha) > \beta^* \, \sigma(\theta, \alpha)\}$ where β^* is picked according to Corollary 3.3.1. If we draw T samples from $\hat{A}_\alpha$, then with probability at least $1 - \delta$, all of the samples will satisfy the constraint $g(\theta, \alpha) > 0$.

In practice, however, for any given α, using the definition of β^* from Corollary 3.3.1, the set $\hat{A}_\alpha$ may be empty. In that case, we can relax our criterion to include the set of θ values whose score is

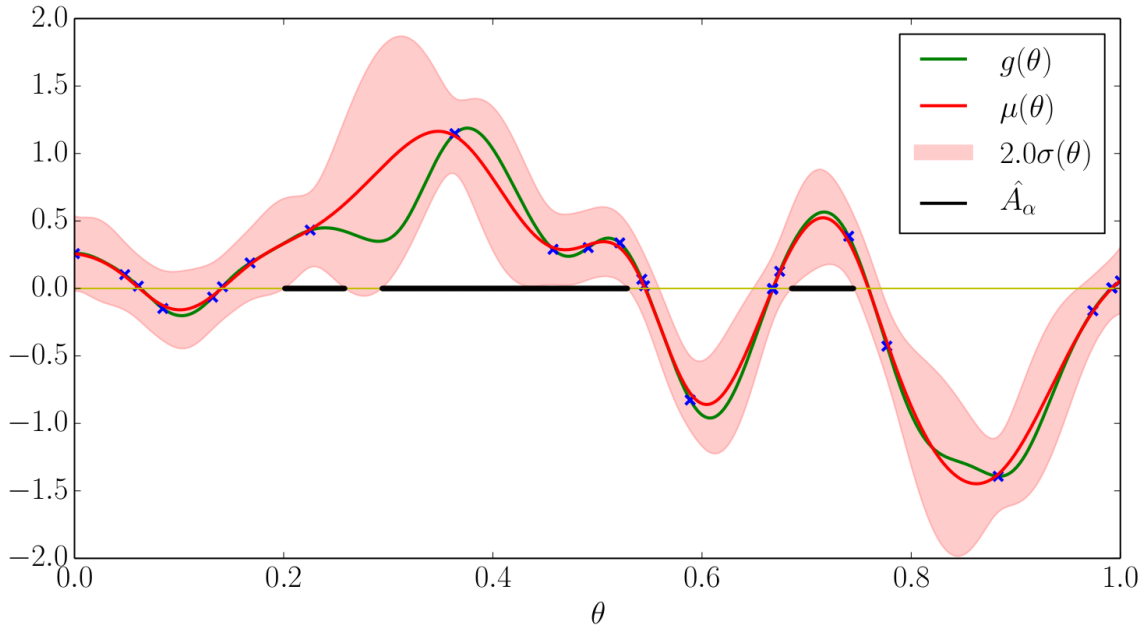


Figure 3-2: High-probability super-level-set in black.

within 5% of the θ value that is currently estimated to have the highest likelihood of satisfying the constraint: $\beta = \Phi^{-1}(0.95\Phi(\max_{\theta \in B} \mu(\theta, \alpha)/\sigma(\theta, \alpha)))$ where Φ is the cumulative density function of a normal distribution.

Figure 3-2 illustrates the computation of $\hat{A}_\alpha$. The green line is the true hidden $g(\theta)$; the blue $\times$ symbols are the training data, gathered using the straddle algorithm in $[0, 1]$; the red line is the posterior mean function $\mu(\theta)$; the pink regions show the two-standard-deviation bounds on $g(\theta)$ based on $\sigma(\theta)$; and the black line segments are the high-probability super-level-set $\hat{A}_\alpha$ for $\beta = 2.0$. We can see that sampling has concentrated near the boundary, that $\hat{A}_\alpha$ is a subset of the true super-level-set, and that as σ decreases through experience, $\hat{A}_\alpha$ will approach the true super-level set.

To sample from $\hat{A}_\alpha$, one simple strategy is to do rejection sampling with a proposal distribution that is uniform on the search bounding-box B. However, in many cases, the feasible region of a constraint is typically much smaller than B, which means that uniform sampling will have a very low chance of drawing samples within $\hat{A}_\alpha$, and so rejection sampling will be very inefficient. We address this problem using a novel adaptive sampler, which draws new samples from the neighborhood of the samples that are already known to be feasible with high probability and then re-weights these new samples using importance weights.

The algorithm ADAPTIVESAMPLER takes as input the posterior GP parameters μ and σ and context vector α, and yields a stream of samples. It begins by computing β and sets Θ_{init} to contain

the θ that is most likely to satisfy the constraint. It then maintains a buffer Θ of at least $m/2$ samples, and yields the first one each time it is required to do so; it technically never actually returns, but generates a sample each time it is called. The main work is done by SAMPLEBUFFER, which constructs a mixture of truncated Gaussian distributions (TGMM), specified by mixture weights p, means Θ, circular variance with parameter v, and bounds B. Parameter v indicates how far from known good θ values it is reasonable to search; it is increased if a large portion of the samples from the TGMM are accepted and decreased otherwise. The algorithm iterates until it has constructed a set of at least m samples from $\hat{A}_\alpha$. It samples n elements from the TGMM and retains those that are in $\hat{A}_\alpha$ as Θ_a. Then, it computes "importance weights" p_a that are inversely related to the probability of drawing each $\theta_a \in \Theta_a$ from the current TGMM. This will tend to spread the mass of the sampling distribution away from the current samples, but still concentrated in the target region. A set of n uniform samples is drawn and filtered, again to maintain the chance of dispersing to good regions that are far from the initialization. The p values associated with the old Θ as well as the newly sampled ones are concatenated and then normalized into a distribution, the new samples added to Θ, and the loop continues. When at least m samples have been obtained, m elements are sampled from Θ according to distribution p, without replacement.

It is easy to see that as n goes to infinity, by sampling from the discrete set according to the re-weighted probability, we are essentially sampling uniformly at random from $\hat{A}_\alpha$. This is because $\forall \theta \in \Theta, p(\theta) \propto \frac{1}{p_{sample}(\theta)} p_{sample}(\theta) = 1$. For uniform sampling, $p_{sample}(\theta) = \frac{1}{Vol(B)}$, where $Vol(B)$ is the volume of B; and for sampling from the truncated mixture of Gaussians, $p_{sample}(\theta)$ is the probability density of θ. In practice, n is finite, but this method is much more efficient than rejection sampling.

3.3.3 Diversity-aware sampling for planning

Now that we have a sampler that can generate approximately uniformly random samples within the region of values that satisfy the constraints with high probability, we can use it inside a planning algorithm for continuous action spaces. Such planners perform backtracking search, potentially needing to consider multiple different parameterized instances of a particular action before finding one that will work well in the overall context of the planning problem. The efficiency of this process depends on the order in which samples of action instances are generated. Intuitively, when previous samples of this action for this context have failed to contribute to a successful plan, it would be wise to try new samples that, while still having high probability of satisfying the constraint, are as

Algorithm 3 Super Level Set Adaptive Sampling

function ADAPTIVESAMPLER($\mu, \sigma, \alpha, n, m, B$)
  $\beta \leftarrow \Phi^{-1}(0.95\Phi(\max_{\theta \in B} \mu(\theta, \alpha)/\sigma(\theta, \alpha)))$
  $\Theta_{init} \leftarrow \{\arg\max_{\theta \in B} \frac{\mu(\theta)}{\sigma(\theta)}\}$; $\Theta \leftarrow \emptyset$
  while True do
    if $|\Theta| < m/2$ **then**
      $\Theta \leftarrow$ SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}, n, m, B$)
    end if
    $\theta \leftarrow \Theta[0]$
    yield θ
    $\Theta \leftarrow \Theta \setminus \{\theta\}$
  end while
end function
function SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}, n, m, B$)
  $v \leftarrow [1]_{d=1}^{d_\theta}$; $\Theta \leftarrow \Theta_{init}$; $p \leftarrow [1]_{i=1}^{|\Theta|}$
  while True **do**
    $\Theta' \leftarrow$ SampleTGMM($n; p, \Theta, v, B$)
    $\Theta_a \leftarrow \{\theta \in \Theta' \mid \mu(\theta) > \beta\sigma(\theta)\}$
    $p_a \leftarrow 1/p_{\text{TGMM}}(\Theta_a; p, \Theta, v, B)$
    $v \leftarrow v/2$ **if** $|\Theta_a| < |\Theta'|/2$ **else** $v \times 2$
    $\Theta'' \leftarrow$ SampleUniform($n; B$)
    $\Theta_r \leftarrow \{\theta \in \Theta'' \mid \mu(\theta) > \beta\sigma(\theta)\}$
    $p_r \leftarrow [Vol(B)]_{i=1}^{|\Theta_r|}$
    $p \leftarrow$ Normalize($[p, p_r, p_a]$)
    $\Theta \leftarrow [\Theta, \Theta_r, \Theta_a]$
    if $|\Theta| > m$ **then**
      return Sample($m; \Theta, p$)
    end if
  end while
end function

different from those that were previously tried as possible. We need, therefore, to consider diversity when generating samples; but the precise characterization of useful diversity depends on the domain in which the method is operating. We address this problem by adapting a kernel that is used in the sampling process, based on experience in previous planning problems.

Diversity-aware sampling has been studied extensively with determinantal point processes (DPPs) [110]. We begin with similar ideas and adapt them to the planning domain, quantifying diversity of a set of samples S using the determinant of a Gram matrix: $D(S) = \log\det(\Xi^S\zeta^{-2} + \boldsymbol{I})$, where $\Xi^S_{ij} = \xi(\theta_i, \theta_j), \forall\theta_i, \theta_j \in S$, ξ is a covariance function, and ζ is a free parameter (we use $\zeta = 0.1$). In DPPs, the quantity $D(S)$ can be interpreted as the volume spanned by the feature space of the kernel $\xi(\theta_i, \theta_j)\zeta^{-2} + \mathbf{1}_{\theta_i \equiv \theta_j}$ assuming that $\theta_i = \theta_j \iff i = j$. Alternatively, one can interpret the quantity $D(S)$ as the information gain of a GP when the function values on S are observed [171]. This GP has kernel ξ and observation noise $\mathcal{N}(0, \zeta^2)$. Because of the submodularity and monotonicity of $D(\cdot)$, we can maximize $D(S)$ greedily with the promise that $D([\theta_i]_{i=1}^N) \geq (1 - \frac{1}{e}) \max_{|S| \leq N} D(S)$ $\forall N = 1, 2, \cdots$, where $\theta_i = \arg\max_\theta D(\theta \cup \{\theta_j\}_{j=1}^{i-1})$. In fact, maximizing $D(\theta \cup S)$ is equivalent to maximizing

$$\eta_S(\theta) = \xi(\theta, \theta) - \boldsymbol{\xi}^S(\theta)^\mathsf{T}(\Xi^S + \zeta^2\boldsymbol{I})^{-1}\boldsymbol{\xi}^S(\theta)$$

which is exactly the same as the posterior variance for a GP.

The DIVERSESAMPLER procedure is very similar in structure to the ADAPTIVESAMPLER procedure, but rather than selecting an arbitrary element of Θ, the buffer of good samples, to return, we track the set S of samples that have already been returned and select the element of Θ that is most diverse from S as the sample to yield on each iteration. In addition, we yield S to enable kernel learning as described in Alg 5, to yield a kernel η.

Algorithm 4 Super Level Set Diverse Sampling

function DIVERSESAMPLER($\mu, \sigma, \alpha, \eta, n, m, B$)
  $\beta \leftarrow \lambda(\max_{\theta \in B} \frac{\mu(\theta)}{\sigma(\theta)})$; $\Theta \leftarrow \emptyset$
  $\theta \leftarrow \arg\max_{\theta \in B} \frac{\mu(\theta)}{\sigma(\theta)}$; $S \leftarrow \emptyset$
  while planner requires samples **do**
    yield θ, S
    if $|\Theta| < m/2$ **then**
      $\Theta \leftarrow$ SAMPLEBUFFER($\mu, \sigma, \alpha, \beta, \Theta_{init}, n, m, B$)
    end if
    $S \leftarrow S \cup \{\theta\}$ ▷ S contains samples before θ
    $\theta \leftarrow \arg\max_{\theta \in \Theta} \eta_S(\theta)$
    $\Theta \leftarrow \Theta \setminus \{\theta\}$
  end while
end function

It is typical to learn the kernel parameters of a GP or DPP given supervised training examples of function values or diverse sets, but those are not available in our setting; we can only observe which samples are accepted by the planner and which are not. We derive our notion of similarity by assuming that all samples that are rejected by the planner are similar. Under this assumption, we develop an online learning approach that adapts the kernel parameters to learn a good diversity metric for a sequence of planning tasks.

We use the squared exponential kernel of the form $\xi(\theta, \gamma; l) = \exp(-\sum_d r_d^2)$, where $r_d = |l_d(\theta_d - \gamma_d)|$ is the rescaled "distance" between θ and γ on the d-th feature and l is the inverse lengthscale. Let θ be the sample that failed and the set of samples sampled before θ be S. We define the importance of the d-th feature as

$$\tau_S^\theta(d) = \xi(\theta_d, \theta_d; l_d) - \boldsymbol{\xi}^S(\theta_d; l_d)^{\mathrm{T}}(\Xi^S + \zeta^2 \boldsymbol{I})^{-1}\boldsymbol{\xi}^S(\theta_d; l_d) \ ,$$

which is the conditional variance if we ignore the distance contribution of all other features except the d-th; that is, $\forall k \neq d, l_k = 0$. Note that we keep $\Xi_i + \zeta^2 \boldsymbol{I}$ the same for all the features so that the inverse only needs to be computed once.

The diverse sampling procedure is analogous to the weighted majority algorithm [61] in that each feature d is seen as an expert that contributes to the conditional variance term, which measures how diverse θ is with respect to S. The contribution of feature d is measured by $\tau_S^\theta(d)$. If θ was rejected by the planner, we decrease the inverse lengthscale l_d of feature $d = \arg\max_{d \in [d_\theta]} \tau_S^\theta(d)$

to be $(1-\epsilon)l_d$, because feature d contributed the most to the decision that θ was most different from S.

Algorithm 5 Task-level Kernel Learning

for task in T **do**
  $\alpha \leftarrow$ current context
  $\mu, \sigma \leftarrow$ GP-predict(α); $S \leftarrow \emptyset$
  while plan not found **do**
    if $|S| > 0$ **then**
      $d \leftarrow \arg\max_{d \in [d_\theta]} \tau_S^\theta(d)$
      $l_d \leftarrow (1-\epsilon)l_d$
    end if
    $\theta, S \leftarrow$ DIVERSESAMPLER$(\mu, \sigma, \alpha, \xi(\cdot,\cdot;l), n, m, B)$
    Check if a plan exist using θ
  end while
end for

Alg. 5 depicts a scenario in which the kernel is updated during interactions with a planner; it is simplified in that it uses a single sampler, but in our experimental applications there are many instances of action samplers in play during a single execution of the planner. Given a sequence of tasks presented to the planner, we can continue to apply this kernel update, molding our diversity measure to the demands of the distribution of tasks in the domain. This simple strategy for kernel learning may lead to a significant reduction in planning time, as we demonstrate in the next section.

3.4 Experiments

We show the effectiveness and efficiency of each component of our method independently, and then demonstrate their collective performance in the context of planning for long-horizon tasks in a high-dimensional continuous domain. We first construct the experiments in a simple simulated 2D setting, then extend it to a more complex simulated 3D domain, and finally show promising results on a real-world physical robot.

3.4.1 Implementation of Kitchen2D

To test our algorithms, we implemented a simulated 2D kitchen based on the physics engine Box2D [32]. Fig. 3-3 shows several scenes indicating the variability of arrangements of objects in the domain. We use bi-directional RRT [109] to implement motion planning. The parameterized primitive motor

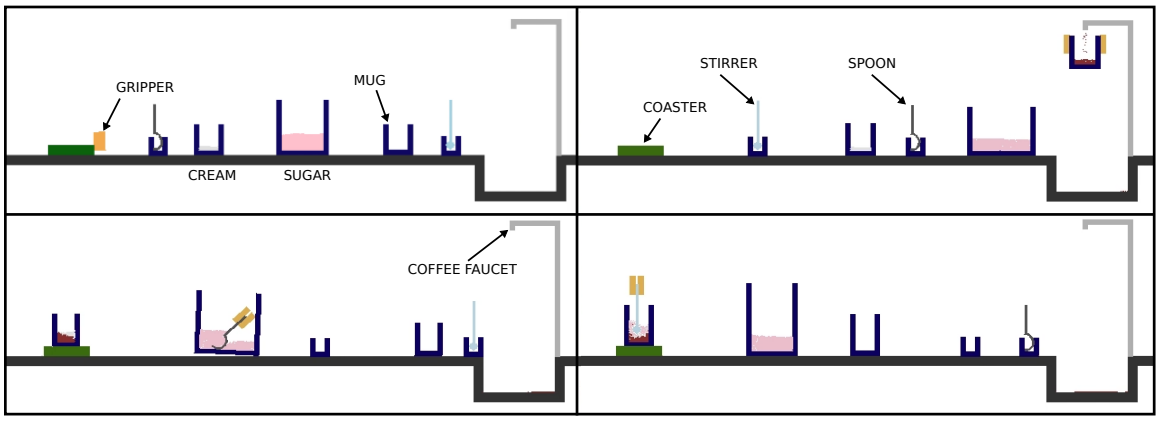


Figure 3-3: Four arrangements of objects in 2D kitchen, including: green coaster, coffee faucet, yellow robot grippers, sugar scoop, stirrer, coffee mug, small cup with cream, larger container with pink sugar.

actions are: moving the robot (a simple "free-flying" hand), picking up an object, placing an object, pushing an object, filling a cup from a faucet, pouring a material out of a cup, scooping material into a spoon, and dumping material from a spoon. The gripper has 3 degrees of freedom (2D position and rotation). The material to be poured or scooped is simulated as small circular particles.

We learn models and samplers for three of these action primitives: pouring (4 context parameters, 4 predicted parameters, scooping (2 context parameters, 7 predicted parameters), and pushing (2 context parameters, 6 predicted parameters). The actions are represented by a trajectory of way points for the gripper, relative to the object it is interacting with. For pouring, we use the scoring function $g_{pour}(x) = \exp(2*(x*10-9.5))-1$, where x is the proportion of the liquid particles that are poured into the target cup. The constraint $g_{pour}(x) > 0$ means at least 95% of the particles are poured correctly to the target cup. The context of pouring includes the sizes of the cups, with widths ranging from 3 to 8 (units in Box2D), and heights ranging from 3 to 5. For scooping, we use the proportion of the capacity of the scoop that is filled with liquid particles, and the scoring function is $g_{scoop}(x) = x - 0.5$, where x is the proportion of the spoon filled with particles. We fix the size of the spoon and learn the action parameters for different cup sizes, with width ranging from 5 to 10 and height ranging from 4 to 8. For pushing, the scoring function is $g_{push}(x) = 2 - \|x - x_{goal}\|$ where x is the position of the pushed object after the pushing action and x_{goal} is the goal position; here the goal position is the context. The pushing action learned in Sec. 3.4.3 has the same setting as [88], viewing the gripper/object with a bird-eye view. The code for the simulation and learning methods is public at `https://ziw.mit.edu/projects/kitchen2d/`.

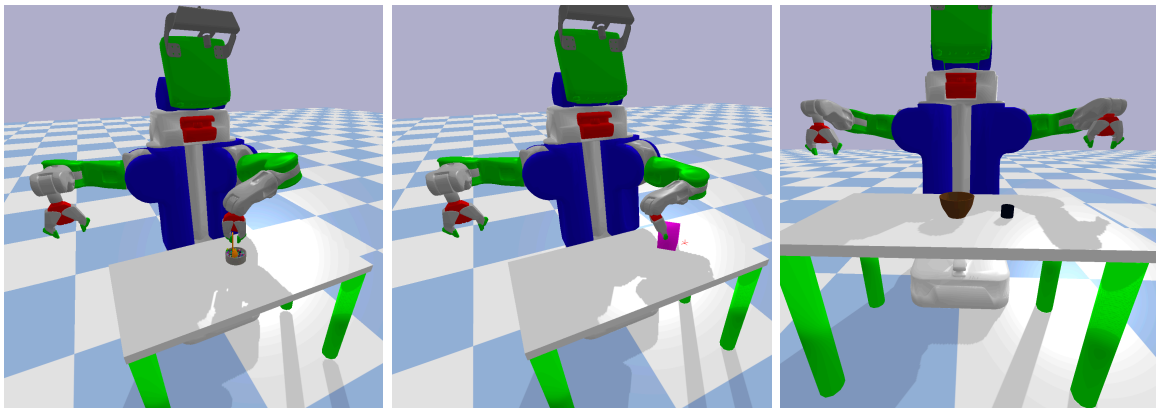


Figure 3-4: Scenes of a simulated PR2 robot trying to scoop, push and pour.

3.4.2 Implementation of Kitchen3D

Simulation To make the experiments more complete, we also implemented a simulated 3D environment with a PR2 robot operating by a table with objects on it. This 3D environment aims to provide a simulated version of our real-world robot operating scenario. Our 3D simulation is based on the physics engine PyBullet [39]. An illustration of our implementation is shown in Figure 3-4.

We define the following parameterized motion primitives: moving an arm from one pose to another, grasping an object, picking up an object, placing an object, pouring from a cup, scooping from a bowl, pushing an object and stirring in a cup with a spoon. All the primitives have their parameterized policies hand-specified.

Among the pre-specified primitives, pouring and scooping are the ones that we use machine learning methods to learn the conditional sampler for. Their scoring function is defined in a similar way to the ones in Kitchen2D in the section above. We define the generators for the parameters of other primitives by hand, e.g. using uniform sampling in a predefined region. For simplicity, we parameterize primitives by object poses or robot gripper location relative to an object instead of robot poses.

For inverse kinematics, we use IKFast [48, 47]. We use Moveit! [37, 142] for pruning points that collide with the robot. Similar to Kitchen2D, we also use bi-directional RRT [109] to implement motion planning.

Real-world We use exactly the same set of primitives and their implementations as in the 3D simulation. We present a demonstration of the PR2 robot operating in the real world in Figure 3-5.

In this chapter we are assuming that states are fully observable. For the real-world setting, we use the open-sourced Tensorflow Object Detection API [81] and train it with hand-labeled data to

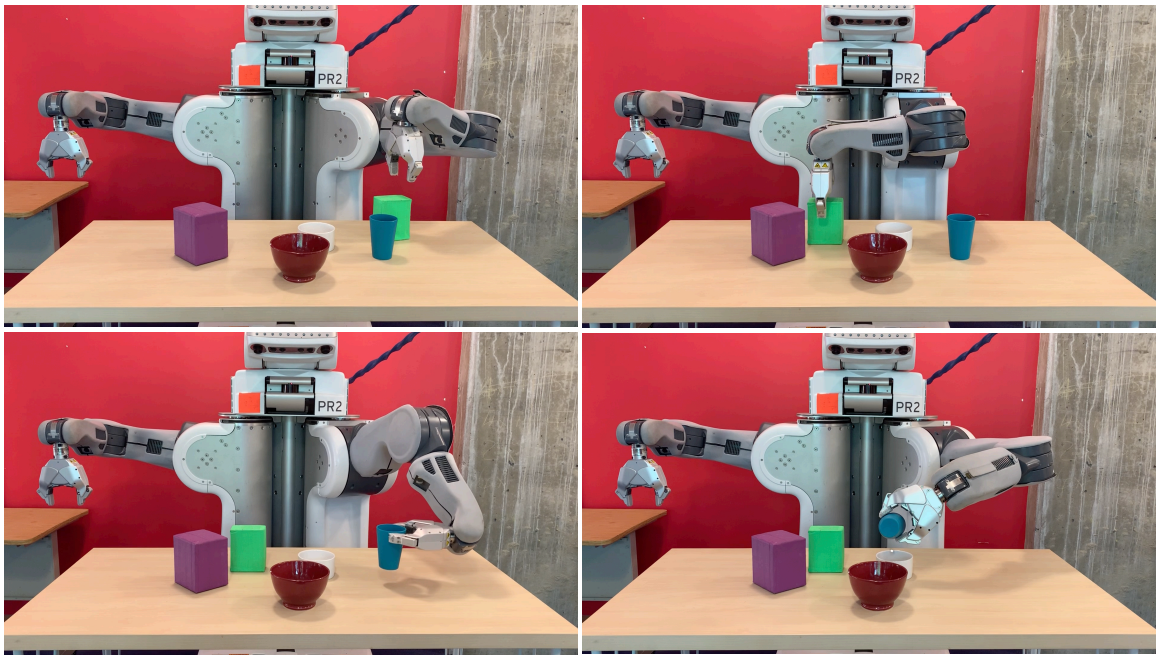


Figure 3-5: Our PR2 robot is operating by a table on top of which some blocks, cups and bowls lie. The goal is to pour from the blue cup to the white bowl. The planner decides to move away the green block obstacle first and then approach the blue cup.

obtain the detection information of objects on the table. We then use the detection information (including bounding boxes and labels of object types) to crop out the point cloud of the detected objects. The poses of the objects are estimated using an optimization based pose estimator built by Dr. Jared Glover. The pose of the table plane is estimated using the Point Cloud Library (PCL) [156]. We provide the pose estimation with 3D models of all of the objects. Once we obtain the estimated poses of all the objects, we assume there is no uncertainty in the estimated poses.

We use beads and bead-like small objects as the material to be poured and scooped. In order to obtain scores on pouring or scooping, we use USB scales underneath the cups and bowls to estimate the number of beads inside. The USB scales are directly connected to our computer to provide accurate real-time readings. Once we have the estimated number of beads, the same scoring metric is used in real world as in the 3D simulator.

3.4.3 Active learning for conditional samplers

We demonstrate the performance of using a GP with the straddle algorithm (GP-LSE) to estimate the level set of the constraints on parameters for pushing, pouring and scooping. For comparison, we also implemented a simple method [88], which uses a neural network to map (θ, α) pairs to predict

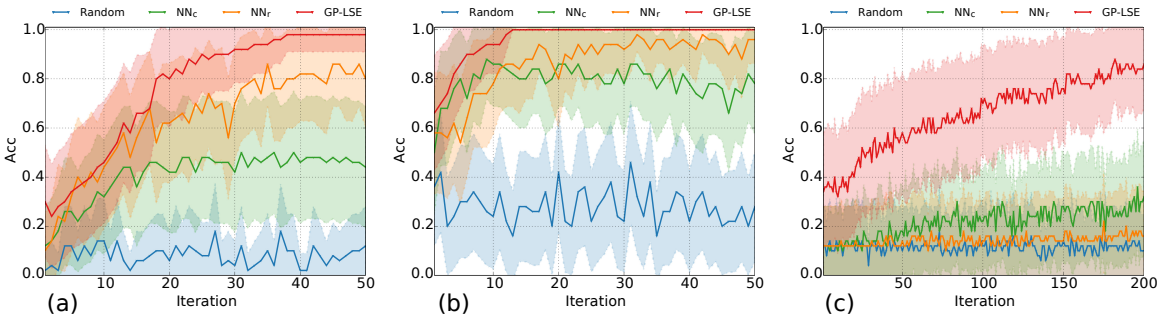


Figure 3-6: Mean accuracy (with 1/2 stdev on mean shaded) of the first action recommended by random selection (Random), regression-based neural network (NN_r), classification-based neural network (NN_c) and Gaussian process using level-set estimation (GP-LSE) on (a) a pouring task with 8 parameters (4 are context parameters); (b) a scooping task with 9 parameters (2 are context parameters) , and (c) a pushing task with 6 parameters (2 are context parameters).

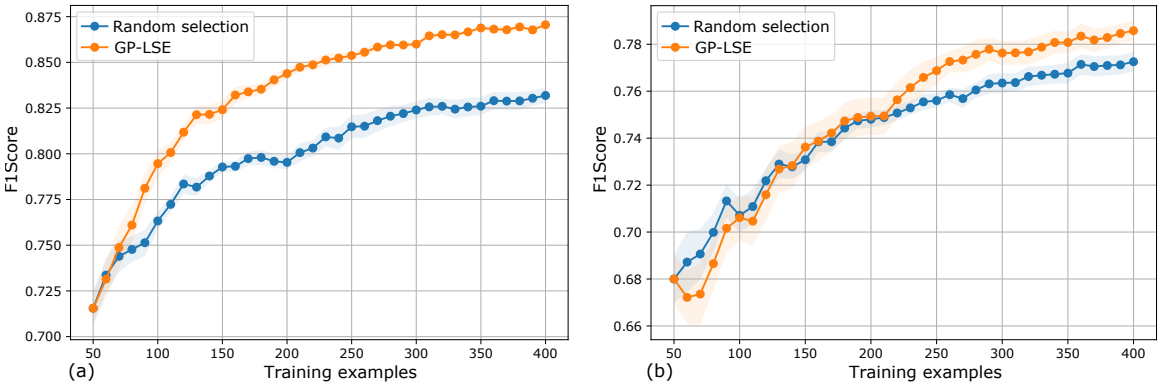


Figure 3-7: Mean accuracy (with standard error on mean shaded) of the F1 score evaluated on a set of randomly generated test examples. We compare GP-LSE with *Random selection*, which uses random training examples to train a GP instead of using active learning. (a) A pouring task with 11 parameters (6 are context parameters). (b) A scooping task with 8 parameters (3 are context parameters).

the probability of success using a logistic output. Given a partially trained network and a context α, the $\theta^* = \arg\max_\theta \mathrm{NN}(\alpha, \theta)$ which has the highest probability of success with α is chosen for execution. Its success or failure is observed, and then the network is retrained with this added data point. This method is called NN_c in the results. In addition, we implemented a regression-based variation that predicts $g(\theta, \alpha)$ with a linear output layer, but given an α value still chooses the maximizing θ. This method is called NN_r. We also compare to random sampling of θ values, without any training.

GP-LSE is able to learn much more efficiently than the other methods. Figure 3-6 shows the accuracy of the first action parameter vector θ (value 1 if the action with parameters θ is actually successful and 0 otherwise) recommended by each of these methods as a function of the number

Figure 3-8: Comparing the first 5 samples generated by DIVERSE (left) and ADAPTIVE (right) on one of the experiments for pouring. The more transparent the pose, the later it gets sampled.

of actively gathered training examples. The results are evaluated using Kitchen2D. GP-LSE recommends its first θ by maximizing the probability that $g(\theta, \alpha) > 0$. The neural-network methods recommend their first θ by maximizing the output value, while RANDOM always selects uniformly randomly from the domain of θ.

Figure 3-7 shows the F1 scores evaluated on a set of 1000 randomly selected test examples, and we compare GP-LSE with the GP model using random selection of training examples. This experiment is based on the Kitchen3D simulation. The results shows the effectiveness of active learing with GP.

In every case, the GP-based method achieves perfect or high accuracy well before the others, demonstrating the effectiveness of uncertainty-driven active sampling methods.

3.4.4 Adaptive sampling and diverse sampling

Given a probabilistic estimate of a desirable set of θ values, obtained by a method such as GP-LSE, the next step is to sample values from that set to use in planning. We compare simple rejection sampling using a uniform proposal distribution (REJECTION), the basic adaptive sampler from Sec. ssec:adaptive, and the diversity-aware sampler from Sec. 3.3.3 with a fixed kernel: the results are shown in Table. 3.1. For all the results, we use $\Phi^{-1}(0.99\Phi(\beta_*))$ to construct the high probability super level set.

We report the false positive rate (proportion of samples that do not satisfy the true constraint) on 50 samples (FP), the time to sample these 50 samples (T_{50}), the total number of samples required to find 5 positive samples (N_5), and the diversity of those 5 samples. The experiments are repeated

Table 3.1: Effectiveness of adaptive and diverse sampling. FP: the false positive rate of 50 samples. T_{50}: the total sampling time of the 50 samples. N_5: number of samples required to achieve 5 positive ones. Diversity: the diversity rate of the 5 positive samples.

		REJECTION	ADAPTIVE	DIVERSE
Pour (2D)	FP (%)	6.45 ± 8.06*	4.04 ± 6.57	5.12 ± 6.94
	T_{50} (s)	3.10 ± 1.70*	0.49 ± 0.10	0.53 ± 0.09
	N_5	5.51 ± 1.18*	5.30 ± 0.92	5.44 ± 0.67
	Diversity	17.01 ± 2.90*	16.24 ± 3.49	18.80 ± 3.38
Scoop (2D)	FP (%)	$0.00^{\dagger}$	2.64 ± 6.24	3.52 ± 6.53
	T_{50} (s)	$9.89 \pm 0.88^{\dagger}$	0.74 ± 0.10	0.81 ± 0.11
	N_5	$5.00^{\dagger}$	5.00 ± 0.00	5.10 ± 0.41
	Diversity	$21.1^{\dagger}$	20.89 ± 1.19	21.90 ± 1.04
Push (2D)	FP (%)	$68.63 \pm 46.27^{\ddagger}$	21.36 ± 34.18	38.56 ± 37.60
	T_{50} (s)	$7.50 \pm 3.98^{\ddagger}$	3.58 ± 0.99	3.49 ± 0.81
	N_5	$5.00 \pm 0.00^{\ddagger}$	$5.56 \pm 1.51^{\triangle}$	$6.44 \pm 2.11^{\clubsuit}$
	Diversity	$23.06 \pm 0.02^{\ddagger}$	$10.74 \pm 4.92^{\triangle}$	$13.89 \pm 5.39^{\clubsuit}$
Pour (3D)	FP (%)	0.03 ± 0.10	0.02 ± 0.07	0.02 ± 0.08
	T_{50} (s)	143.56 ± 176.05	72.84 ± 71.26	65.93 ± 72.93
	N_5	5.14 ± 0.45	5.10 ± 0.58	5.15 ± 0.71
	Diversity	15.29 ± 3.44	15.40 ± 2.94	18.78 ± 3.07
Scoop (3D)	FP (%)	0.13 ± 0.17	0.16 ± 0.16	0.12 ± 0.10
	T_{50} (s)	265.57 ± 118.24	72.84 ± 71.26	35.11 ± 18.73
	N_5	5.77 ± 1.82	6.11 ± 1.77	5.66 ± 1.09
	Diversity	10.93 ± 2.50	11.82 ± 1.63	14.57 ± 2.13

*1 out of 50 experiments failed (to generate 50 samples within 10 seconds); †49 out of 50 failed; ‡34 out of 50 failed; 5 out of 16 experiments failed (to generate 5 positive samples within 100 samples); $^{\triangle}$7 out of 50 failed; $^{\clubsuit}$11 out of 50 failed.

over 50 such samplers for each method. We limit cpu time for gathering 50 samples to 10 seconds for 2D experiments and unlimited for 3D simulated experiments (running with Python 2.7.13 and Ubuntu 14.04 on Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 64GB memory). If no sample is returned within 10 seconds for 2D experiments, we do not include that experiment in the reported results except the sampling time. Hence the reported sampling time may be a lower bound on the actual sampling time. The diversity term is measured by $D(S) = \log\det(\Xi^S\zeta^{-2} + \boldsymbol{I})$ using a squared exponential kernel with inverse lengthscale $l = [1, 1, \cdots, 1]$ and $\zeta = 0.1$. We run the sampling algorithm for an additional 50 iterations (a maximum of 100 samples in total) until we have 5 positive examples and use these samples to report $D(S)$. We also report the total number of samples needed to achieve 5 positive ones (N_p). If the method is not able to get 5 positive samples within 100 samples, we report failure and do not include them in the diversity metric or the N_p metric.

DIVERSE uses slightly more samples than ADAPTIVE to achieve 5 positive ones, and its false positive rate is slightly higher than ADAPTIVE, but the diversity of the samples is notably higher. The FP rate of diverse can be decreased by increasing the confidence bound on the level set.

As described by Alg. 4, on each iteration DIVERSE tries to sample something different from the previous samples. This also means that Alg. 4 may sample closer to the boundary of the high-probability super-level-set $\hat{A}_\alpha = \{\theta \mid \mu(\theta, \alpha) > \beta\sigma(\theta, \alpha)\}$, which may lead to a slightly lower probability of satisfying the constraint. One may make up for this false positive rate by increasing β. For example, for the 2D pouring action, we also run DIVERSE with $\beta = \Phi^{-1}(0.998\Phi(\beta_*))$. The false positive rate of DIVERSE is 3.52 ± 4.74, N_p is 5.38 ± 0.56 while the diversity is 18.38 ± 3.66. The false positive rate of DIVERSE decreases by almost 2%, but the diversity does not degrade very much. Similarly for 2D scooping, if we use $\beta = \Phi^{-1}(0.998\Phi(\beta_*))$ in DIVERSE, the false positive rate of DIVERSE is 3.24 ± 6.76, N_p is 5.04 ± 0.20 while the diversity is 21.91 ± 0.63.

We illustrate the ending poses of the 5 2D pouring actions generated by adaptive sampling with DIVERSE and ADAPTIVE in Fig. 3-8, illustrating that DIVERSE is able to generate more diverse action parameters, which may facilitate planning.

3.4.5 Learning kernels for diverse sampling in planning

In the final set of experiments, we explore the effectiveness of the diverse sampling algorithm with task-level kernel learning We compare ADAPTIVE, DIVERSE-GK with a fixed kernel, and diverse sampling with learned kernel (DIVERSE-LK), in every case using a high-probability super-level-

Table 3.2: Effect of distance metric learning on sampling. We compare the performance of ADAPTIVE, DIVERSE-GK and DIVERSE-LK in terms of their average runtime and success rate (SR) within a certain time for solving five different tasks.

Task I	Runtime (ms)	0.2s SR (%)	0.02s SR (%)
ADAPTIVE	8.16 ± 12.16	100.0 ± 0.0	87.1 ± 0.8
DIVERSE-GK	9.63 ± 9.69	100.0 ± 0.0	82.2 ± 1.2
DIVERSE-LK	5.87 ± 4.63	100.0 ± 0.0	99.9 ± 0.1
Task II	**Runtime (s)**	**60s SR (%)**	**6s SR (%)**
ADAPTIVE	3.22 ± 6.51	91.0 ± 2.7	82.4 ± 5.6
DIVERSE-GK	2.06 ± 1.76	95.0 ± 1.8	93.6 ± 2.2
DIVERSE-LK	1.71 ± 1.23	95.0 ± 1.8	94.0 ± 1.5
Task III	**Runtime (s)**	**60s SR (%)**	**6s SR (%)**
ADAPTIVE	5.79 ± 11.04	51.4 ± 3.3	40.9 ± 4.1
DIVERSE-GK	3.90 ± 5.02	56.3 ± 2.0	46.3 ± 2.0
DIVERSE-LK	4.30 ± 6.89	59.1 ± 2.6	49.1 ± 2.6
Task IV	**Runtime (s)**	**60s SR (%)**	**6s SR (%)**
ADAPTIVE	18.41 ± 8.87	42.0 ± 10.3	28.0 ± 15.4
DIVERSE-GK	18.22 ± 9.70	48.0 ± 7.5	26.0 ± 16.6
DIVERSE-LK	17.07 ± 9.72	53.0 ± 6.0	40.0 ± 11.8
Task V	**Runtime (s)**	**60s SR (%)**	**6s SR (%)**
ADAPTIVE	44.20 ± 22.05	23.0 ± 12.5	5.0 ± 3.2
DIVERSE-GK	44.85 ± 23.47	21.0 ± 9.2	5.0 ± 3.2
DIVERSE-LK	42.86 ± 23.34	23.0 ± 12.1	6.0 ± 5.8

set estimated by a GP. All the experiments are repeated 5 times with random scene settings. In DIVERSE-LK, we use $\epsilon = 0.3$.

The first set of tasks (Task I) we consider is a simple controlled example where the goal is to push an object off a 2D table with the presence of an obstacle on either one side of the table or the other (both possible situations are equally likely). The presence of these obstacles is not represented in the context of the sampler, but the planner will reject sample action instances that generate a collision with an object in the world and request a new sample. We use a fixed range of feasible actions sampled from two rectangles in 2D of unequal sizes. The optimal strategy is to first randomly sample from one side of the table and if no plan is found, sample from the other side. For DIVERSE-GK, the kernel inverse is initialized as $[1, 1]$ and if, for example, it sampled on the left side of the object (pushing to the right) and the obstacle is on the right, it may not choose to sample on the right side because the kernel indicates that the other feature is has more diversity. However, after a few planning instances, DIVERSE-LK is able to figure out the right configuration of the kernel and its sampling strategy becomes the optimal one.

We also tested these three sampling algorithms on several more complicated tasks. For Task

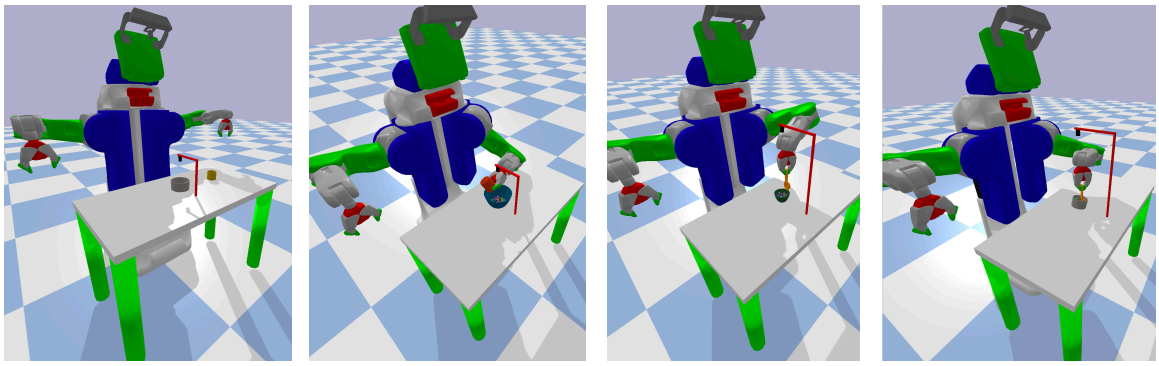


Figure 3-9: Illustrations of Task IV and V. In Task IV, the robot is tasked to pour from a cup to a bowl while avoiding the faucet (red structure next to the bowl). In Task V, the goal is to scoop from a bowl while avoiding collisions with the faucet.

II and III, we select a fixed test set with 50 task specifications and repeat the evaluation 5 times. The first one (Task II) involves picking up cup A, getting water from a faucet, move to a pouring position, pour water into cup B, and finally placing cup A back in its initial position. Cup B is placed randomly either next to the wall on the left or right. The second task is a harder version of Task II, with the additional constraint that cup A has a holder and the sampler also has to figure out that the grasp location must be close to the top of the cup (Task III). Both Task II and III are in Kitchen2D.

For Task IV and V, illustrated in Figure 3-9, we test our methods in the simulated Kitchen3D environment. Task IV's goal is to pour from a cup to a bowl while avoiding the faucet next to the bowl, and Task V is to scoop from a bowl while avoiding the faucet next to the bowl.

We show the timing and success rate results in Tab. 3.2 (after training). Our empirical results shows that, in general, DIVERSE-LK is able to find a better solution than the alternatives in all the tasks. Moreover, the two diverse sampling methods achieve lower variance on the success rate and perform more stably after training.

3.4.6 Integrated system

Finally, we integrate the learned action sampling models for pour and scoop with 7 pre-existing robot operations (move, push, pick, place, fill, dump, stir) in a domain specification for STRIPStream. The robot's goal is to "serve" a cup of coffee with cream and sugar by placing it on the green coaster near the edge of the table. Accomplishing this requires general-purpose planning, including picking where to grasp the objects, where to place them back down on the table, and what the pre-operation poses of the cups and spoon should be before initiating the sensorimotor primitives for pouring and scooping should be. Significant perturbations of the object arrangements are handled without

difficulty[1]. Some example resulting plans and execution sequences are shown in https://www.youtube.com/playlist?list=PLoWhBFPMfSzDbc8CYelsbHZa1d3uz-W_c.

This work illustrates a critical ability: to augment the existing competences of a robotic system (such as picking and placing objects) with new sensorimotor primitives by learning probabilistic models of their preconditions and effects and using a state-of-the-art domain-independent continuous-space planning algorithm to combine them fluidly and effectively to achieve complex goals.

3.5 Conclusion

In this chapter, we augment the basic abilities of a robot by learning to use new sensorimotor primitives to enable the solution of complex long-horizon problems. Our approach combines active learning and diverse sampling methods with expert-specified structures of action models and efficiently learns a probabilistic description of the models that can be used for task and motion planning. We demonstrated state-of-the-art performance of our approach on a set of challenging task and motion planning problems.

[1]We use the focused algorithm within STRIPStream, and it solves the task in 20-40 seconds for a range of different arrangements of objects.

Chapter 4

Learning sparse relational transition models

Many complex domains are appropriately described in terms of sets of objects, properties of those objects, and relations among them. We are interested in the problem of taking actions to change the state of such complex systems, in order to achieve some objective. To do this, we require a *transition model*, which describes the system state that results from taking a particular action, given the previous system state. In many important domains, ranging from interacting with physical objects to managing the operations of an airline, actions have *localized* effects: they may change the state of the object(s) being directly operated on, as well as some objects that are *related* to those objects in important ways, but will generally not affect the vast majority of other objects.

In this chapter, we present a strategy for learning state-transition models that embodies these assumptions. We structure our model in terms of *rules*, each of which only depends on and affects the properties and relations among a small number of objects in the domain, and only very few of which may apply for characterizing the effects of any given action. Our primary focus is on learning the *kernel* of a rule: that is, the set of objects that it depends on and affects. At a moderate level of abstraction, most actions taken by an intentional system are inherently directly parametrized by at least one object that is being operated on: a robot pushes a block, an airport management system reschedules a flight, an automated assistant commits to a venue for a meeting. It is clear that properties of these "direct" objects are likely to be relevant to predicting the action's effects and that some properties of these objects will be changed. But how can we characterize which other objects,

Victoria Xia*, Zi Wang*, Kelsey Allen, Tom Silver, and Leslie Pack Kaelbling. Learning sparse relational transition models. In *International Conference on Learning Representations (ICLR)*, 2019. (* indicates equal contribution.)

out of all the objects in a household or airline network, are relevant for prediction or likely to be affected?

To do so, we make use of the notion of a *deictic reference*. In linguistics, a deictic (literally meaning "pointing") reference, is a way of naming an object in terms of its relationship to the current situation rather than in global terms. So, "the object I am pushing," "all the objects on the table nearest me," and "the object on top of the object I am pushing" are all deictic references. This style of reference was introduced as a representation strategy for AI systems by [2], under the name *indexical-functional* representations, for the purpose of compactly describing policies for a video-game agent, and has been in occasional use since then.

We will learn a set of deictic references, for each rule, that characterize, relative to the object(s) being operated on, which other objects are relevant. Given this set of relevant objects, the problem of describing the transition model on a large, variable-size domain, reduces to describing a transition model on fixed-length vectors characterizing the relevant objects and their properties and relations, which we represent and learn using standard feed-forward neural networks.

Next, we briefly survey related work, describe the problem more formally, and then provide an algorithm for learning both the structure, in terms of deictic references, and parameters, in terms of neural networks, of a sparse relational transition model. We go on to demonstrate this algorithm in a simulated robot-manipulation domain in which the robot pushes objects on a cluttered table.

4.1 Problem formulation

We assume we are working on a class of problems in which the domain is appropriately described in terms of objects. This method might not be appropriate for a single high-dimensional system in which the transition model is not sparse or factorable, or can be factored along different lines (such as a spatial decomposition) rather than along the lines of objects and properties. We also assume a set of primitive *actions* defined in terms of control programs that can be executed to make actual changes in the world state and then return. These might be robot motor skills (grasping or pushing an object) or virtual actions (placing an order or announcing a meeting). In this section, we formalize this class of problems, define a new rule structure for specifying probabilistic transition models for these problems, and articulate an objective function for estimating these models from data.

4.1.1 Relational domain

A problem *domain* is given by tuple $\mathcal{D} = (\Upsilon, \mathcal{P}, \mathcal{F}, \mathcal{A})$ where Υ is a countably infinite universe of possible objects, $\mathcal{P}$ is a finite set of properties $P_i : \Upsilon \mapsto \mathbb{R}, i \in [N_{\mathcal{P}}] = \{1, \cdots, N_{\mathcal{P}}\}$, and $\mathcal{F}$ is a finite set of deictic reference functions $F_i : \Upsilon^{m_i} \mapsto \wp(\Upsilon), i \in [N_{\mathcal{F}}]$ where $\wp(\Upsilon)$ denotes the powerset of Υ. Each function $F_i \in \Upsilon$ maps from an ordered list of objects to a set of objects, and we define it as

$$F_i(o_1, \ldots, o_{m_i}) = \{o \mid f_i(o, o_1, \ldots, o_{m_i}) = \text{True},\ o, o_j \in \Upsilon, \forall j \in [m_i]\}\ ,$$

where the relation $f_i : \Upsilon^{m_i+1} \mapsto \{\text{True}, \text{False}\}$ is defined in terms of the object properties in $\mathcal{P}$. For example, if we have a location property P_{loc} and $m_i = 1$, we can define $f_i(o, o_1) = \mathbb{1}_{\|P_{\text{loc}}(o) - P_{\text{loc}}(o_1)\| < 0.5}$ so that the function F_i associated with f_i maps from one object to the set of objects that are within 0.5 distance of its center; here $\mathbb{1}$ is an indicator function. Finally, $\mathcal{A}$ is a set of *action templates* $A_i : \mathbb{R}^{d_i} \times \Upsilon^{n_i} \mapsto \Psi, i \in [N_{\mathcal{A}}]$, where Ψ is the space of executable control programs. Each action template is a function parameterized by continuous parameters $\alpha_i \in \mathbb{R}^{d_i}$ and a tuple of n_i objects that the action operates on. In this work, we assume that $\mathcal{P}, \mathcal{F}$ and $\mathcal{A}$ are given.[1]

A problem *instance* is characterized by $\mathcal{I} = (\mathcal{D}, \mathcal{U})$, where $\mathcal{D}$ is a domain defined above and $\mathcal{U} \subset \Upsilon$ is a finite universe of objects with $|\mathcal{U}| = N_{\mathcal{U}}$. For simplicity, we assume that, for a particular instance, the universe of objects remains constant over time. In the problem instance $\mathcal{I}$, we characterize a *state* s in terms of the concrete values of all properties in $\mathcal{P}$ on all objects in $\mathcal{U}$; that is, $s = [P_i(o_j)]_{i=1,j=1}^{N_{\mathcal{P}}, N_{\mathcal{U}}} \in \mathbb{R}^{N_{\mathcal{P}} \times N_{\mathcal{U}}} = \mathfrak{S}$. A problem instance induces the definition of its *action* space $\mathfrak{A}$, constructed by applying every action template $A_i \in \mathcal{A}$ to all tuples of n_i elements in $\mathcal{U}$ and all assignments α_i to the continuous parameters; namely, $\mathfrak{A} = \{A_i(\alpha_i, [o_{ij}]_{j=1}^{n_i}) \mid o_{ij} \in \mathcal{U}, \alpha_i \in \mathbb{R}^{d_i}\}$.

4.1.2 Sparse relational transition models

In many domains, there is substantial uncertainty, and the key to robust behavior is the ability to model this uncertainty and make plans that respect it. A *sparse relational transition model* (SPARE) for a domain $\mathcal{D}$, when applied to a problem instance $\mathcal{I}$ for that domain, defines a probability density function on the resulting state s' resulting from taking action a in state s. Our objective is to specify

[1] There is a direct extension of this formulation in which we encode relations among the objects as well. Doing so complicates notation and adds no new conceptual ideas, and in our example domain it suffices to compute spatial relations from object properties so there is no need to store relational information explicitly, so we omit it from our treatment.

this function in terms of domain elements $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{F}$ in such a way that it will apply to any problem instance, independent of the number and properties of the objects in its universe. We achieve this by defining the transition model in terms of a set of *transition rules*, $\mathcal{T} = \{T_k\}_{k=1}^{K}$ and a score function $C : \mathcal{T} \times \mathfrak{S} \mapsto \mathbb{N}$. The score function takes in as input a state s and a rule $T \in \mathcal{T}$, and outputs a non-negative integer. If the output is 0, the rule does not apply; otherwise, the rule can predict the distribution of the next state to be $p(s' \mid s, a; T)$. The final prediction of SPARE is

$$p(s' \mid s, a; \mathcal{T}) = \begin{cases} \frac{1}{|\hat{\mathcal{T}}|} \sum_{T \in \hat{\mathcal{T}}} p\left(s' \mid s, a; T\right) & \text{if } |\hat{\mathcal{T}}| > 0 \\ \mathcal{N}(s, \Sigma_{\text{default}}) & \text{otherwise} \end{cases}, \tag{4.1}$$

where $\hat{\mathcal{T}} = \arg\max_{T \in \mathcal{T}} C(T, s)$ and the matrix $\Sigma_{\text{default}} = \boldsymbol{I}_{N_{\mathcal{U}}} \otimes \text{diag}([\sigma_i]_{i=1}^{N_{\mathcal{P}}})$ is the default predicted covariance for any state that is not predicted to change, so that our problem is well-formed in the presence of noise in the input. Here $\boldsymbol{I}_{N_{\mathcal{U}}}$ is an identity matrix of size $N_{\mathcal{U}}$, and $\text{diag}([\sigma_i]_{i=1}^{N_{\mathcal{P}}})$ represents a square diagonal matrix with σ_i on the main diagonal, denoting the default variance for property P_i if no rule applies. Note that the transition rules will be learned from past experience with a loss function specified in Section 4.1.3. In the rest of this section, we formalize the definition of transition rules and the score function.

Transition rule $T = (A, \Gamma, \Delta, \phi_\theta, \boldsymbol{v}_{\text{default}})$ is characterized by an action template A, two ordered lists of *deictic references* Γ and Δ of size N_Γ and N_Δ, a predictor ϕ_θ and the default variances $\boldsymbol{v}_{\text{default}} = [v_i]_{i=1}^{N_{\mathcal{P}}}$ for each property P_i under this rule. The action template is defined as operating on a tuple of n object variables, which we will refer to as $O^{(0)} = (O_i)_{i=1}^{n}, O_i \in \mathcal{U}, \forall i$. A reference list uses functions to designate a list of additional objects or sets of objects, by making *deictic* references based on previously designated objects. In particular, Γ generates a list of objects whose properties affect the prediction made by the transition rule, while Δ generates a list of objects whose properties are affected after taking an action specified by the action template A.

We begin with the simple case in which every function returns a single object, then extend our definition to the case of sets. Concretely, for the t-th element γ_t in Γ ($t \in [N_\Gamma]$), $\gamma_t = (F, (O_{k_j})_{j=1}^{m})$ where $F \in \mathcal{F}$ is a deictic reference function in the domain, m is the arity of that function, and integer $k_j \in [n + t - 1]$ specifies that object O_{n+t} in the object list can be determined by applying function F to objects $(O_{k_j})_{j=1}^{m}$. Thus, we get a new list of objects, $O^{(t)} = (O_i)_{i=1}^{n+t}$. So, reference γ_1 can only refer to the objects $(O_i)_{i=1}^{n}$ that are named in the action, and determines an object O_{n+1}. Then, reference γ_2 can refer to objects named in the action or those that were determined by reference γ_1, and so

on.

When the function F in $\gamma_t = (F, (O_{k_j})_{j=1}^m) \in \Gamma$ returns a set of objects rather than a single object, this process of adding more objects remains almost the same, except that the O_t may denote sets of objects, and the functions that are applied to them must be able to operate on sets. In the case that a function F returns a set, it must also specify an *aggregator*, g, that can return a single value for each property $P_i \in \mathcal{P}$, aggregated over the set. Examples of aggregators include the mean or maximum values or possibly simply the cardinality of the set.

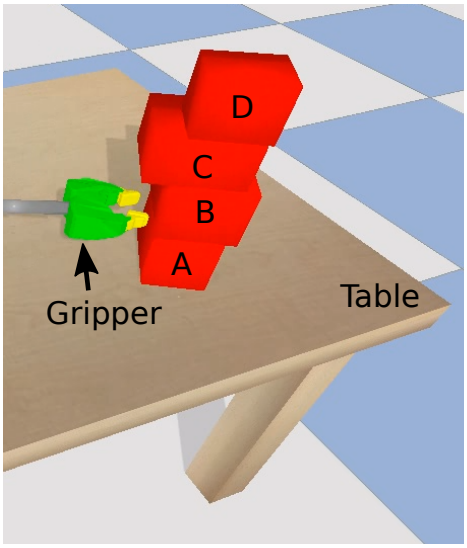


Figure 4-1: A robot gripper is pushing a stack of 4 blocks on a table.

For example, consider the case of pushing the bottom (block A) of a stack of 4 blocks, depicted in Figure 4-1. Suppose the deictic reference is F =`above`, which takes one object and returns a set of objects immediately on top of the input object. Then, by applying F =`above` starting from the initial set $O_0 = \{A\}$, we get an ordered list of sets of objects (O_0, O_1, O_2) where $O_1 = F(O_0) = \{B\}, O_2 = F(O_1) = \{C\}$.

Returning to the definition of a transition rule, we now can see informally that if the parameters of action template A are instantiated to actual objects in a problem instance, then Γ and Δ can be used to determine lists of *input* and *output* objects (or sets of objects). We can use these lists, finally, to construct input and output vectors. The input vector $\boldsymbol{x}$ consists of the continuous action parameters α of action A and property $P_i(O_t)$ for all properties $P_i \in \mathcal{P}$ and objects $O_t \in O^{N_\Gamma}$ that are selected by Γ in arbitrary but fixed order. In the case that O_t is a set of size greater than one, the aggregator associated with the function F that computed the reference is used to compute $P_i(O_t)$. Similar for the desired output construction, we use the references in the list Δ, initialize $\hat{O}^{(0)} = O^{(0)}$, and gradually add more objects to construct the output set of objects $\hat{O} = \hat{O}^{(N_\Delta)}$. The output vector is $\boldsymbol{y} = [P(\hat{o})]_{\hat{o} \in \hat{O}, P \in \mathcal{P}}$ where if $\hat{o}$ is a set of objects, we apply a mean aggregator on the properties of all the objects in $\hat{o}$.

The *predictor* ϕ_θ is some functional form ϕ (such as a feed-forward neural network) with parameters (weights) θ that will take values $\boldsymbol{x}$ as input and predict a distribution for the output vector $\boldsymbol{y}$. It is difficult to represent arbitrarily complex distributions over output values. In this work, we

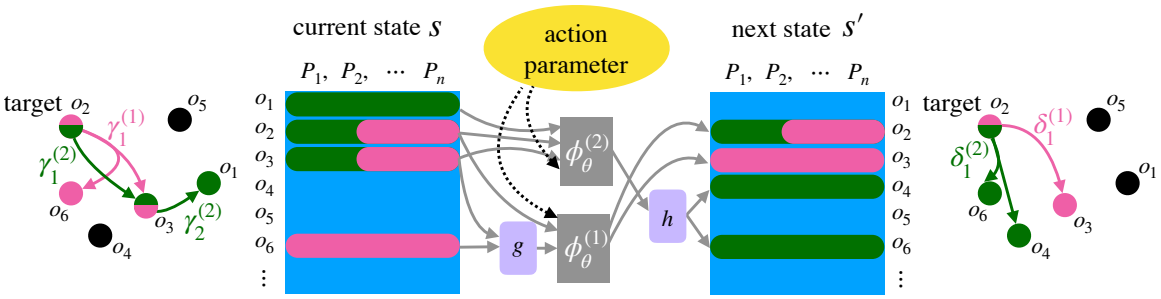


Figure 4-2: Instead of directly mapping from current state s to next state s', our prediction model uses deictic references to find subsets of objects for prediction. In the left most graph, we illustrate what relations are used to construct the input objects with two rules for the same action template, $T_1 = (A, \Gamma^{(1)}, \Delta^{(1)}, \phi_\theta^{(1)}, \boldsymbol{v}_{\text{default}}^{(1)})$ and $T_2 = (A, \Gamma^{(2)}, \Delta^{(2)}, \phi_\theta^{(2)}, \boldsymbol{v}_{\text{default}}^{(2)})$, where the reference list $\Gamma^{(1)} = [(\gamma_1^{(1)}, o_2)]$ applied a deictic reference $\gamma_1^{(1)}$ to the target object o_2 and added input features computed by an aggregator g on o_3, o_6 to the inputs of the predictor of rule T_1. Similarly for $\Gamma^{(2)} = [(\gamma_1^{(2)}, o_2), (\gamma_2^{(2)}, o_3)]$, the first deictic reference selected o_3 and then $\gamma_2^{(2)}$ is applied on o_3 to get o_1. The predictors $\phi_\theta^{(1)}$ and $\phi_\theta^{(2)}$ are neural networks that map the fixed-length input to a fixed-length output, which is applied to a set of objects computed from a relational graph on all the objects, derived from the reference list $\Delta^{(1)} = [(\delta_1^{(1)}, o_2)]$ and $\Delta^{(2)} = [(\delta_1^{(2)}, o_2)]$, to compute the whole next state s'. Because $\delta_1^{(2)}(o_2) = (o_4, o_6)$ and the $\phi_\theta^{(2)}$ is only predicting a single property, we use a "de-aggregator" function h to assign its prediction to both objects o_4, o_6.

restrict ourselves to representing a Gaussian distributions on all property values in $\boldsymbol{y}$, encoded with a mean and independent variance for each dimension.

Now, we describe how a transition rule can be used to map a state and action into a distribution over the new state. A transition rule $T = (A, \Gamma, \Delta, \phi_\theta, \boldsymbol{v}_{\text{default}})$ *applies* to a particular state-action (s, a) pair if a is an instance of A and if none of the elements of the input or output object lists is empty. To construct the input (and output) list, we begin by assigning the actual objects $o_1, \ldots, o_n$ to the object variables $O_1, \ldots, O_n$ in action instance a, and then successively computing references $\gamma_i \in \Gamma$ based on the previously selected objects, applying the definition of the deictic reference F in each γ_i to the actual values of the properties as specified in the state s. If, at any point, a $\gamma_i \in \Gamma$ or $\delta_i \in \Delta$ returns an empty set, then the transition rule does not apply. If the rule does apply, and successfully selects input and output object lists, then the values of the input vector $\boldsymbol{x}$ can be extracted from s, and predictions are made on the mean and variance values $\Pr(\boldsymbol{y} \mid \boldsymbol{x}) = \phi_\theta(\boldsymbol{x}) = \mathcal{N}(\mu_{\theta_1}(\boldsymbol{x}), \Sigma_{\theta_2}(\boldsymbol{x}))$.

Let $\left(\mu_{\theta_1}^{(ij)}(\boldsymbol{x}), \Sigma_{\theta 2}^{(ij)}(\boldsymbol{x})\right)$ be the vector entry corresponding to the predicted Gaussian parameters of property P_i of j-th output object set $\hat{o}_j$ and denote $s[o, P_i]$ as the property P_i of object o in state s,

for all $o \in \mathcal{U}$. The predicted distribution of the resulting state $p(s' \mid s, a; T)$ is computed as follows:

$$p(s'[o, P_i] \mid s, a; T) = \begin{cases} \frac{1}{|\{j:o \in \hat{o}_j\}|} \sum_{\{j:o \in \hat{o}_j\}} \mathcal{N}(\mu_{\theta_1}^{(ij)}(\boldsymbol{x}), \Sigma_{\theta 2}^{(ij)}(\boldsymbol{x})) & \text{if } |\{j : o \in \hat{o}_j\}| > 0 \\ \mathcal{N}(s[o, P_i], v_i) & \text{otherwise} \end{cases}$$

where $v_i \in \boldsymbol{v}_{\text{default}}$ is the default variance of property P_i in rule T. There are two important points to note. First, it is possible for the same object to appear in the object-list more than once, and therefore for more than one predicted distribution to appear for its properties in the output vector. In this case, we use the mixture of all the predicted distributions with uniform weights. Second, when an element of the output object list is a set, then we treat this as predicting the same single property distribution for all elements of that set. This strategy has sufficed for our current set of examples, but an alternative choice would be to make the predicted values be *changes* to the current property value, rather than new absolute values. Then, for example, moving all of the objects on top of a tray could easily specify a change to each of their poses. We illustrate how we can use transition rules to build a SPARE in Fig. 4-2.

For each transition rule $T_k \in \mathcal{T}$ and state $s \in \mathfrak{S}$, we assign the ***score function*** value to be 0 if T_k does not apply to state s. Otherwise, we assign the total number of deictic references plus one, $N_\Gamma + N_\Delta + 1$, as the score. The more references there are in a rule that is applicable to the state, the more detailed the match is between the rules conditions and the state, and the more specific the predictions we expect it to be able to make.

4.1.3 Learning SPAREs from data

We frame the problem of learning a transition model from data in terms of conditional likelihood. The learning problem is, given a *problem domain description* $\mathcal{D}$ and a set of experience $\mathcal{E}$ tuples, $\mathcal{E} = \{(s^{(i)}, a^{(i)}, s'^{(i)})\}_{i=1}^n$, find a SPARE $\mathcal{T}$ that minimizes the loss function:

$$\mathcal{L}(\mathcal{T}; \mathcal{D}, \mathcal{E}) = -\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s'^{(i)} \mid s^{(i)}, a^{(i)}; \mathcal{T}) \ . \tag{4.2}$$

Note that we require all of the tuples in $\mathcal{E}$ to belong to the same *domain* $\mathcal{D}$, and require for any $(s^{(i)}, a^{(i)}, s'^{(i)}) \in \mathcal{E}$ that $s^{(i)}$ and $s'^{(i)}$ belong to the same problem *instance*, but individual tuples may be drawn from different problem instances (with, for example, different numbers and types of objects). In fact, to get good generalization performance, it will be important to vary these aspects

across training instances.

4.2 Related work

Rule learning has a long history in artificial intelligence. The novelty in our approach is the combination of learning discrete structures with flexible parametrized models in the form of neural networks.

Rule learning We are inspired by very early work on rule learning by [50], which sought to find predictive rules in simple noisy domains, using Boolean combinations of binary input features to predict the effects of actions. This approach has a modern re-interpretation in the form of schema networks [92]. The rules we learn are *lifted*, in the sense that they can be applied to objects, generally, and are not tied to specific bits or objects in the input representation and *probabilistic*, in the sense that they make a distributional prediction about the outcome. In these senses, this work is similar to that of [144] and methods that build on it ([138], [137], [115].) In addition, the approach of *learning* to use deictic expressions was inspired by Pasula et al. and used also by [130] in the form of object-oriented reinforcement learning and by [20]. [20], however, relies on a full description of the states in ground first-order logic and does not have a mechanism to introduce new deictic references to the action model. Our representation and learning algorithm improves on the Pasula et al. strategy by using the power of feed-forward neural networks as a local transition model, which allows us to address domains with real-valued properties and much more complex dependencies. In addition, our EM-based learning algorithm presents a much smoother space in which to optimize, making the overall learning faster and more robust. We do not, however, construct new functional terms during learning; that would be an avenue for future work for us.

Graph network models There has recently been a great deal of work on learning graph-structured (neural) network models [16]. There is a way in which our rule-based structure could be interpreted as a kind of graph network, although it is fairly non-standard. We can understand each object as being a node in the network, and the deictic functions as being labeled directed hyper-edges (between sets of objects). Unlike the typical graph network models, we do not condition on a fixed set of neighboring nodes and edges to compute the next value of a node; in fact, a focus of our learning method is to determine which neighbors (and neighbors of neighbors, etc.) to condition on, depending on the current state of the edge labels. This means that the relevant neighborhood structure of any node changes dynamically over time, as the state of the system changes.

This style of graph network is not inherently better or worse than others: it makes a different set of assumptions (including a strong default that most objects do not change state on any given step and the dynamic nature of the neighborhoods) which are particularly appropriate for modeling an agent's interactions with a complex environment using actions that have relatively local effects.

4.3 Our approach

We describe our learning algorithm in three parts. First, we introduce our strategy for learning ϕ_θ, which predicts a Gaussian distribution on $\boldsymbol{y}$, given $\boldsymbol{x}$. Then, we describe our algorithm for learning reference lists Γ and Δ for a single transition rule, which enable the extraction of $\boldsymbol{x}$ and $\boldsymbol{y}$ from $\mathcal{E}$.

4.3.1 Distributional prediction

For a particular transition rule T with associated action template A, once Γ and Δ have been specified, we can extract input and output features $\boldsymbol{x}$ and $\boldsymbol{y}$ from a given set of experience samples $\mathcal{E}$. We would like to learn the transition rule's predictor ϕ_θ to minimize Eq. (4.2). Our predictor takes the form $\phi_\theta(\boldsymbol{x}) = \mathcal{N}(\mu_\theta(\boldsymbol{x}), \Sigma_\theta(\boldsymbol{x}))$ and a neural network is used to predict both the mean $\mu_\theta(\boldsymbol{x})$ and the diagonal variance $\Sigma_\theta(\boldsymbol{x})$. We directly optimize the negative data-likelihood loss function

$$\mathcal{L}(\theta, \Gamma, \Delta; \mathcal{D}, \mathcal{E}) = \frac{1}{n} \sum_{i=1}^{n} \left((\boldsymbol{y}^{(i)} - \mu_\theta(\boldsymbol{x}^{(i)}))^{\mathrm{T}} \Sigma_\theta(\boldsymbol{x})^{-1} (\boldsymbol{y}^{(i)} - \mu_\theta(\boldsymbol{x}^{(i)})) + \log\det \Sigma_\theta(\boldsymbol{x}^{(i)}) \right).$$

Let $\mathcal{E}_T \in \mathcal{E}$ be the set of experience tuples to which rule T applies. Then once we have θ, we can optimize the default variance of the rule $T = (A, \Gamma, \Delta, \phi_\theta, \boldsymbol{v}_{\text{default}})$ by optimizing $\mathcal{L}(\{T\}; \mathcal{D}, \mathcal{E}_T)$. It can be shown that these loss-minimizing values for the default predicted variances $\boldsymbol{v}_{\text{default}}$ are the empirical averages of the squared deviations for all unpredicted objects (i.e., those for which ϕ_θ does not explicitly make predictions), where averages are computed separately for each object property.

We use $\theta, \boldsymbol{v}_{\text{default}} \leftarrow \textsc{LearnDist}(\mathcal{D}, \mathcal{E}, \Gamma, \Delta)$ to refer to this learning and optimization procedure for the predictor parameters and default variance.

4.3.2 Rule learning

We consider the setting where only one transition rule T exists in our domain $\mathcal{D}$, we show how to construct the input and output reference lists Γ and Δ that will determine the vectors $\boldsymbol{x}$ and

Algorithm 6 Greedy procedure for constructing Γ.

procedure GREEDYSELECT($\mathcal{D}, \mathcal{E}, A, \Delta, N_\Gamma$)
  train model using $\Gamma_0 = \emptyset$, save loss L_0
  $i \leftarrow 1$
  while $i \leq N_\Gamma$ **do**
    $\gamma_i \leftarrow$ None ; $L_i \leftarrow \infty$
    for all $\gamma \in R_i$ **do**
      $\Gamma_i \leftarrow \Gamma_{i-1} \cup \{\gamma\}$
      $\theta, \boldsymbol{v}_{\text{default}} \leftarrow$ LEARNDIST($\mathcal{D}, \mathcal{E}_{train}, \Gamma_i, \Delta$)
      $l \leftarrow \mathcal{L}(\mathcal{T}_\gamma; \mathcal{D}, \mathcal{E}_{val})$
      if $l < L_i$ **then** $L_i \leftarrow l$; $\gamma_i \leftarrow \gamma$
      end if
    end for
    if $L_i < L_{i-1}$ **then** $\Gamma_i \leftarrow \Gamma_{i-1} \cup \{\gamma_i\}$; $i \leftarrow i + 1$
    else break
    end if
  end while
end procedure

$\boldsymbol{y}$. Suppose for now that Δ and $\boldsymbol{v}_{\text{default}}$ are fixed, and we wish to learn Γ. Our approach is to incrementally build up Γ by adding $\gamma_i = (F, (O_{k_j})_{j=1}^m)$ tuples one at a time via a greedy selection procedure. Specifically, let R_i be the universe of possible γ_i, split the experience samples $\mathcal{E}$ into a training set $\mathcal{E}_{train}$ and a validation set $\mathcal{E}_{val}$, and initialize the list Γ to be $\Gamma_0 = \emptyset$. For each i, compute $\gamma_i = \arg\min_{\gamma \in R_i} \mathcal{L}(\mathcal{T}_\gamma; \mathcal{D}, \mathcal{E}_{val})$, where $\mathcal{L}$ in Eq. (4.2) evaluates a SPARE $\mathcal{T}_\gamma$ with a single transition rule $T = (A, \Gamma_{i-1} \cup \{\gamma\}, \Delta, \phi_\theta, \boldsymbol{v}_{\text{default}})$, where θ and $\boldsymbol{v}_{\text{default}}$ are computed using the LEARNDIST described in Section 4.3.1[2]. If the value of the loss function $\mathcal{L}(\mathcal{T}_{\gamma_i}; \mathcal{D}, \mathcal{E}_{val})$ is less than the value of $\mathcal{L}(\mathcal{T}_{\gamma_{i-1}}; \mathcal{D}, \mathcal{E}_{val})$, then we let $\Gamma_i = \Gamma_{i-1} \cup \{\gamma_i\}$ and continue. Else, we terminate the greedy selection process with $\Gamma = \Gamma_{i-1}$, since further growing the list of deictic references hurts the loss. We also terminate the process when i exceeds some predetermined maximum allowed number of input deictic references, N_Γ. Pseudocode for this algorithm is provided in Algorithm 6.

In our experiments we set $\Delta = \Gamma$ and construct the lists of deictic references using a single pass of the greedy algorithm described above. This simplification is reasonable, as the set of objects that are relevant to predicting the transition outcome often overlap substantially with the objects that are affected by the action. Alternatively, we could learn Δ via an analogous greedy procedure nested around or, as a more efficient approach, interleaved with, the one for learning Γ.

[2]When the rule T does not apply to a training sample, we use for its loss the loss that results from having empty reference lists in the rule. Alternatively, we can compute the default variance Σ_{default} to be the empirical variances on all training samples that cannot use rule T.

4.3.3 Multiple rules

Our training data in robotic manipulation tasks are likely to be best described by many rules instead of a single one, since different combinations of relations among objects could be present in different states. For example, we may have one rule for pushing a single object and another rule for pushing a stack of objects. We now address the case where we wish to learn K rules from a single experience set $\mathcal{E}$, for $K > 1$. We do so via initial clustering to separate experience samples into K clusters, one for each rule to be learned, followed by an EM-like approach to further separate samples and simultaneously learn rule parameters.

To facilitate the learning of our model, we will additionally learn *membership probabilities* $Z = ((z_{i,j})_{i=1}^{|\mathcal{E}|})_{j=1}^{K}$, where $z_{i,j}$ represents the probability that the i-th experience sample is assigned to transition rule T_j, and $\sum_{j=1}^{K} z_{i,j} = 1$ for all $i \in [|\mathcal{E}|]$. We initialize membership probabilities via clustering, then refine them through EM.

Because the experience samples $\mathcal{E}$ may come from different problem instances and involve different numbers of objects, we cannot directly run a clustering algorithm such as k-means on the (s, a, s') samples themselves. Instead we first learn a single transition rule $T = (A, \Gamma, \Delta, \phi_\theta, \boldsymbol{v}_{\text{default}})$ from $\mathcal{E}$ using the algorithm in Section 4.3.2, use the resulting Γ and Δ to transform $\mathcal{E}$ into $\boldsymbol{x}$ and $\boldsymbol{y}$, and then run k-means clustering on the concatenation of $\boldsymbol{x}$, $\boldsymbol{y}$, and values of the loss function when T is used to predict each of the samples. For each experience sample, the squared distance from the sample to each of the K cluster centers is computed, and membership probabilities for the sample to each of the K transition rules to be learned are initialized to be proportional to the (multiplicative) inverses of these squared distances.

Before introducing the EM-like algorithm that simultaneously improves the assignment of experience samples to transition rules and learns details of the rules themselves, we make a minor modification to transition rules to obtain *mixture rules*. Whereas a probabilistic transition rule has been defined as $T = (A, \Gamma, \Delta, \phi_\theta, \boldsymbol{v}_{\text{default}})$, a mixture rule is $T = (A, \pi_\Gamma, \pi_\Delta, \Phi)$, where π_Γ represents a *distribution* over all possible lists of input references Γ (and similarly for π_Δ and Δ), of which there are a finite number, since the set of available reference functions $\mathcal{F}$ is finite, and there is an upper bound N_Γ on the maximum number of references Γ may contain. For simplicity of terminology, we refer to each possible list of references Γ as a *shell*, so π_Γ is a distribution over possible shells. Finally, $\Phi = (\Gamma^{(k)}, \Delta^{(k)}, \phi_{\theta^{(k)}}, \boldsymbol{v}_{\text{default}}^{(k)})_{k=1}^{\kappa}$ is a collection of κ transition rules (i.e., predictors $\phi_{\theta^{(k)}}$, each with an associated $\Gamma^{(k)}$, $\Delta^{(k)}$, and $\boldsymbol{v}_{\text{default}}^{(k)}$). To make predictions for a sam-

ple (s, a) using a mixture rule, predictions from each of the mixture rule's κ transition rules are combined according to the probabilities that π_Γ and π_Δ assign to each transition rule's $\Gamma^{(k)}$ and $\Delta^{(k)}$. Rather than having our EM approach learn K transition rules, we instead learn K mixture rules, as the distributions π_Γ and π_Δ allow for smoother sorting of experience samples into clusters corresponding to the different rules, in contrast to the discrete Γ and Δ of regular transition rules.

As before, we focus on the case where for each mixture rule, $\Gamma^{(k)} = \Delta^{(k)}$, $k \in [\kappa]$, and $\pi_\Gamma = \pi_\Delta$ as well. Our EM-like algorithm is then as follows:

1. For each $j \in [K]$, initialize distributions $\pi_\Gamma = \pi_\Delta$ for mixture rule T_j as follows. First, use the algorithm in Section 4.3.2 to learn a transition rule on the *weighted* experience samples $\mathcal{E}_{Z_j}$ with weights equal to the membership probabilities $Z_j = (z_{i,j})_{i=1}^{|\mathcal{E}|}$. In the process of greedily assembling reference lists $\Gamma = \Delta$, data likelihood loss function values are computed for multiple *explored* shells, in addition to the shell $\Gamma = \Delta$ that was ultimately selected. Initialize $\pi_\Gamma = \pi_\Delta$ to distribute weight proportionally, according to data likelihood, for these explored shells: $\pi_\Gamma(\Gamma) = \exp(-\mathcal{L}(\mathcal{T}_\Gamma; \mathcal{D}, \mathcal{E}_{Z_j}))/\chi$, where $\mathcal{T}_\Gamma$ is the SPARE model with a single transition rule $T = (A, \Gamma, \Delta = \Gamma, \phi_\theta)$, and $\chi = (1 - \epsilon) \sum_\Gamma \exp(-\mathcal{L}(\mathcal{T}_\Gamma; \mathcal{D}, \mathcal{E}_{Z_j}))$, with the summation taken over all explored shells Γ, is a normalization factor so that the total weight assigned by π_Γ to explored shells is $1 - \epsilon$. The remaining ϵ probability weight is distributed uniformly across unexplored shells.

2. For each $j \in [K]$, let $T_j = (A, \pi_\Gamma, \pi_\Delta, \Phi)$, where we have dropped subscripting according to j for notational simplicity:

   (a) For $k \in [\kappa]$, train predictor $\Phi_k = (\Gamma^{(k)}, \Delta^{(k)}, \phi_{\theta^{(k)}}, \boldsymbol{v}^{(k)}_{\text{default}})$ using the procedure in Section 4.3.2 on the weighted experience samples $\mathcal{E}_{Z_j}$, where we choose $\Gamma^{(k)} = \Delta^{(k)}$ to be the list of references with k-th highest weight according to $\pi_\Gamma = \pi_\Delta$.

   (b) Update $\pi_\Gamma = \pi_\Delta$ by redistributing weight among the top κ shells according to a voting procedure where each training sample "votes" for the shell whose predictor minimizes the validation loss for that sample. In other words, the i-th experience sample $\mathcal{E}^{(i)}$ votes for mixture rule $v(i) = k$ for $k = \arg\min_{k \in [\kappa]} \mathcal{L}(\Phi_k; \mathcal{D}, \mathcal{E}^{(i)})$. Then, shell weights are assigned to be proportional to the sum of the sample weights (i.e., membership probability of belonging to this rule) of samples that voted for each particular shell: the number of votes received by the k-th shell is $V(k) = \sum_{i=1}^{|\mathcal{E}|} \mathbb{1}_{v(i)=k} \cdot z_{i,j}$, for indicator function $\mathbb{1}$ and $k \in [\kappa]$. Then, $\pi_\Gamma(k)$, the current k-th highest value of π_Γ, is updated

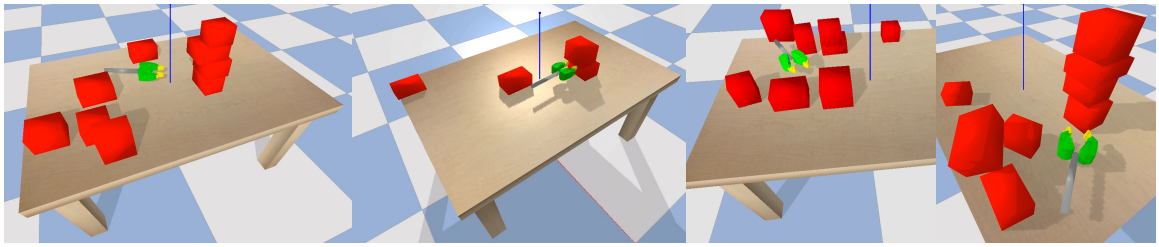


Figure 4-3: Representative problem instances sampled from the domain.

to become $V(k)/\xi$, where ξ is a normalization factor to ensure that π_Γ remains a valid probability distribution. (Specifically, $\xi = (\sum_{k=1}^{\kappa} \pi_\Gamma(k))/(\sum_{k=1}^{\kappa} V(k))$.)

(c) Repeat Step 2a, in case the κ shells with highest π_Γ values have changed, in preparation for using the mixture rule to make predictions in the next step.

3. Update membership probabilities by scaling by data likelihoods from using each of the K rules to make predictions: $z_{i,j} = z_{i,j} \cdot \exp(-\mathcal{L}(T_j; \mathcal{D}, \mathcal{E}^{(i)}))/\zeta$, where $\exp(-\mathcal{L}(T_j; \mathcal{D}, \mathcal{E}^{(i)}))$ is the data likelihood from using mixture rule T_j to make predictions for the i-th experience sample $\mathcal{E}^{(i)}$, and $\zeta = \sum_{j=1}^{K} z_{i,j} \cdot \exp(-\mathcal{L}(T_j; \mathcal{D}, \mathcal{E}^{(i)}))$ is a normalization factor to maintain $\sum_{j=1}^{K} z_{i,j} = 1$.

4. Repeat Steps 2 and 3 some fixed number of times, or until convergence.

4.4 Experiments

We apply our approach, SPARE, to a challenging problem of predicting pushing stacks of blocks on a cluttered table top. We describe our domain, the baseline that we compare to and report our results.

4.4.1 Object manipulation domain

In our domain $\mathcal{D} = (\Upsilon, \mathcal{P}, \mathcal{F}, \mathcal{A})$, the object universe Υ is composed of blocks of different sizes and weight, the property set $\mathcal{P}$ includes shapes of the blocks (width, length, height) and the position of the block ((x, y, z) location relative to the table). We have one action template, *push*(α, o), which pushes toward a *target object* o with parameters $\alpha = (x_g, y_g, z_g, d)$, where (x_g, y_g, z_g) is the 3D position of the gripper before the push starts and d is the distance of the push. The orientation of the gripper and the direction of the push are computed from the gripper location and the target object location.

We simulate this 3D domain using the physically realistic PyBullet [39] simulator. In real-world scenarios, an action cannot be executed with the exact action parameters due to the inaccuracy in the motor and hence in our simulation, we add Gaussian noise on the action parameters during execution to imitate this effect.

We consider the following deictic references in the reference collection $\mathcal{F}$: (1) *identity* (O), which takes in an object O and returns O; (2) *above* (O), which takes in an object O and returns the object immediately above O; (3) *below* (O), which takes in an object O and returns the object immediately below O; (4) *nearest* (O), which takes in an object O and returns the object that is closest to O.

4.4.2 Baseline methods

Neural network (NN)

We compare to a neural network function approximator that takes in as input the current state $s \in \mathbb{R}^{N_{\mathcal{P}} \times N_{\mathcal{U}}}$ and action parameter $\alpha \in \mathbb{R}^{N_{\mathcal{A}}}$, and outputs the next state $s' \in \mathbb{R}^{N_{\mathcal{P}} \times N_{\mathcal{U}}}$. The list of objects that appear in each state is ordered: the target objects appear first and the remaining objects are sorted by their poses (first sort by x coordinate, then y, then z).

Each network was implemented as a fully-connected network with two hidden layers of 64 nodes each in Keras, used ReLU activations between layers, and the Adam optimizer with default parameters. Predictors for the templates approach were trained for 1000 epochs each with a decaying learning rate starting at 1e-2 and decreasing by a factor of 0.6 every 100 epochs. The baseline NN predictor was implemented in exactly the same way.

Graph NN

We compare to a fully connected graph NN. Each node of the graph corresponds to an object in the scene, and the action α is concatenated to the state of each object. Bidirectional edges connect every node in the graph. The graph NN consists of encoders for the nodes and edges, propagation networks for message passing, and a node decoder to convert back to predict the mean and variance of the next state of each object.

We used a node encoder and edge encoder to map to latent spaces of 16 dimensions. The propagation networks consisted of 2 fully connected layers of 16 units each, and the decoder mapped back to 6 dimensions: 3 for the mean, and 3 for the variance. The GNN was trained using a decaying

learning rate starting at 1e-2, and decreasing by a factor of 0.5 every 100 epochs. A total of 900 epochs were used.

4.4.3 Results

Effects of deictic rules

As a sanity check, we start from a simple problem where a gripper is pushing a stack of three blocks with two extra blocks on the table. We randomly sampled 1250 problem instances by drawing random block shapes and locations from a uniform distribution within a range while satisfying the condition that the stack of blocks is stable and the extra blocks do not affect the push. In each problem instance, we uniformly randomly sample the action parameters and obtain the training data, a collection of tuples of state, action and next state, where the target object of the push action is always the one at the bottom of the stack. We held out 20% of the training data as the validation set. We found that our approach is able to reliably select the correct combinations of the references that select all the blocks in the problem instance to construct inputs and outputs. In Fig. 4-4(a), we show how the performance varies as deictic references are added during a typical run of this experiment. The solid purple curves show training performance, as measured by data likelihood on the validation set, while the dashed purple curve shows performance on a held-out test set with 250 unseen problem instances. As expected, performance improves noticeably from the addition of the first two deictic references selected by the greedy selection procedure, but not from the 4th. The brown curve shows the learned default standard deviations, used to compute data likelihoods for features of objects not explicitly predicted by the rule. As expected, the learned default standard deviation drops as deictic references are added, until it levels off after the third reference is added since at that point the set of references captures all moving objects in the scene.

Sensitivity analysis on the number of objects

We compare our approach to the baselines in terms of how sensitive the performance is to the number of objects that exist in the problem instance. We continue the setting where a stack of three blocks lie on a table, with extra blocks that may affect the prediction of the next state. Figure 4-4(b) shows the performance, as measured by the log data likelihood, as a function of the number of extra blocks. For each number of extra blocks, we used 1250 training problem instances with 20% as the validation set and 250 testing problem instances. When there are no extra blocks, SPARE learns a

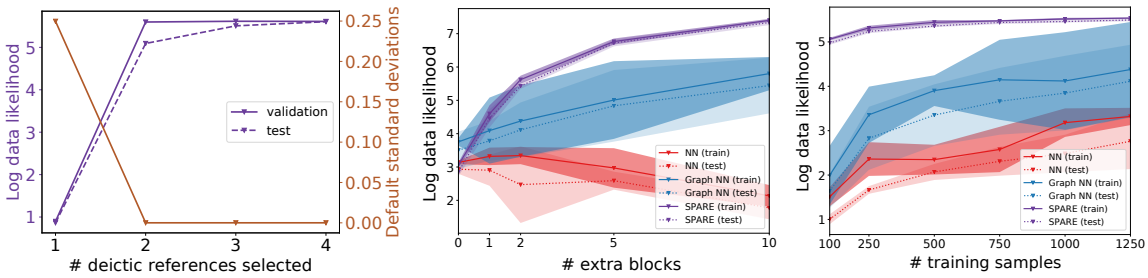


Figure 4-4: (a) In a simple 3-block pushing problem instance, data likelihood and learned default standard deviation both improve as more deictic references are added. (b) Comparing performance as a function of number of distractors with a fixed amount of training data. (c) Comparing sample efficiency of SPARE to the baselines. Shaded regions represent 95% confidence interval.

single rule whose $\boldsymbol{x}$ and $\boldsymbol{y}$ contain the same information as the inputs and outputs for the baselines. As more objects are added to the table, NN's performance drops as the presence of these additional objects appear to complicate the scene and NN is forced to consider more objects when making its predictions. SPARE outperforms graph NN, as the good predictions for the extra blocks contribute to the log data likelihood.

Note that, performance aside, NN is limited to problems for which the number of objects in the scenes is fixed, as the it requires a fixed-size input vector containing information about all objects. Our SPARE approach does not have this limitation, and could have been trained on a single, large dataset that is the combination of the datasets with varying numbers of extra objects. However, we did not do this in our experiments for the sake of providing a more fair comparison against NN.

Sample efficiency

We evaluate our approach on more challenging problem instances where the robot gripper is pushing blocks on a cluttered table top and there are two additional blocks on the table that do not interfere or get affected by the pushing action. Fig. 4-4(c) plots the data likelihood as a function of the number of training samples. We evaluate with training samples varying from 100 to 1250 and in each setting, the test dataset has 250 samples. Both our approach and the baselines benefit from having more training samples, but our approach is much more sample efficient and achieves good performance within only 500 training samples.

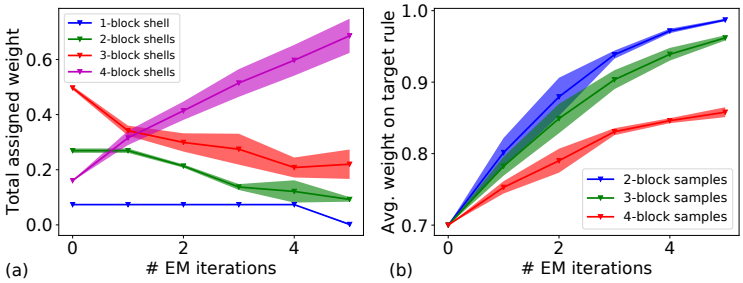


Figure 4-5: (a) Shell weights per iteration of our EM-like algorithm. (b) Membership probabilities of training samples per iteration.

Learning multiple transition rules

Now we put our approach in a more general setting where multiple transition rules need to be learned for prediction of the next state. Our approach adopts an EM-like procedure to assign each training sample its distribution on the transition rules and learn each transition rule with re-weighted training samples. First, we construct a training dataset and 70% of it is on pushing 4-block stack. Our EM approach is able to concentrate to the 4-block case as shown in Fig. 4-5(a).

Fig. 4-5(b) tracks the assignment of samples to rules over the same five runs of our EM procedure. The three curves correspond to the three stack heights in the original dataset, and each shows the average weight assigned to the "target" rule among samples of that stack height, where the target rule is the one that starts with a high concentration of samples of that particular height. At iteration 0, we see that the rules were initialized such that samples were assigned 70% probability of belonging to specific rules, based on stack height. As the algorithm progresses, the samples separate further, suggesting that the algorithm is able to separate samples into the correct groups.

4.5 Conclusion

These results demonstrate the power of combining relational abstraction with neural networks, to learn probabilistic state transition models for an important class of domains from very little training data. In addition, the structural nature of the learned models will allow them to be used in factored search-based planning methods that can take advantage of sparsity of effects to plan efficiently.

Chapter 5

Focused Model-Learning and Planning for Non-Gaussian Continuous State-Action Systems

Most real-world domains are sufficiently complex that it is difficult to build an accurate deterministic model of the effects of actions. Even with highly accurate actuators and sensors, stochasticity still widely appears in basic manipulations, especially non-prehensile ones [203]. The stochasticity may come from inaccurate execution of actions as well as from lack of detailed information about the underlying world state. For example, rolling a die is a deterministic process that depends on the forces applied, air resistance, etc.; however, we are not able to model the situation sufficiently accurately to plan reliable actions, nor to execute them repeatably if we could plan them. We can plan using a stochastic model of the system, but in many situations, such as rolling dice or pushing a can shown in Fig. 5-1, the stochasticity is not modeled well by additive single-mode Gaussian noise, and a more sophisticated model class is necessary.

In this chapter, we address the problem of learning and planning for non-Gaussian stochastic systems in the practical setting of continuous state and action spaces. Our framework learns transition models that can be used for planning to achieve different objectives in the same domain, as well as to be potentially transferred to related domains or even different types of robots. This strategy is in contrast to most reinforcement-learning approaches, which build the objective into

Wang, Zi and Jegelka, Stefanie and Kaelbling, Leslie Pack and Lozano-Pérez, Tomás. Focused model learning and planning for non-Gaussian continuous state-action systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

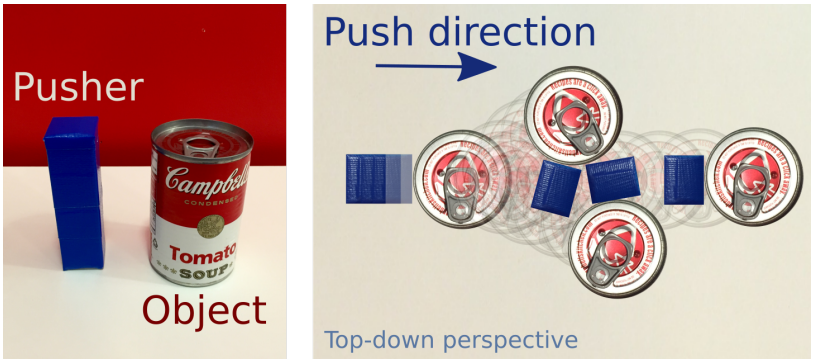


Figure 5-1: A quasi-static pushing problem: the pusher has a velocity controller with low gain, resulting in non-Gaussian transitions. We show trajectories for object and pusher resulting from the same push velocity.

the structure being learned. In addition, rather than constructing a single monolithic model of the entire domain which could be difficult to represent, our method uses a memory-based lazy learning scheme [157, 6]: it computes localized models on the fly, only when the planner requires them. To avoid constructing models that do not contribute to improving the policy, the planner should focus only on states relevant to the current planning problem, and actions that can lead to high reward.

We propose a closed-loop planning algorithm that applies to stochastic continuous state-action systems with arbitrary transition models. It is assumed that the transition models are represented by a function that may be expensive to evaluate. Via two important steps, we focus the computation on the current problem instance, defined by the starting state and goal region. To focus on relevant states, we use real time dynamic programming (RTDP) [15] on a set of states strategically sampled by a rapidly-exploring random tree (RRT) [117, 82]. To focus selection of actions from a continuous space, we develop a new batch Bayesian optimization (BO) technique that selects and tests, in parallel, action candidates that will lead most quickly to a near-optimal answer.

We show theoretically that the expected accumulated difference between the optimal value function of the original problem and the value of the policy we compute vanishes to 0 as the number of actions we test increases, under mild assumptions. Finally we evaluate our approach empirically on a simulated multi-modal pushing problem, and demonstrate the effectiveness and efficiency of the proposed algorithm.

5.1 Problem formulation

Let the state space $S \subset \mathbb{R}^{d_s}$ with metric d and the control space $U \subset \mathbb{R}^{d_u}$ both be compact and measurable sets. The interior of the state space S is S^o and the boundary is ∂S. For the control space U, there exists an open set U^o in R^{d_u} such that U is the closure of U^o. We assume the state is fully observed (any remaining latent state will manifest as stochasticity in the transition models). Actions $a = (u, \Delta t)$ are composed of both a control on the robot and the duration for which it will be exerted, so the action space is $A = U \times [T_{min}, T_{max}]$, where $T_{min}, T_{max} \in \mathbb{R}_+ \setminus \{\infty\}$ are the minimum and the maximum amount of duration allowed. The action space A is also a compact set. The starting state is s_0, and the goal region is denoted as $\mathcal{G} \subset S$, in which all states are terminal states. We assume $\mathcal{G}$ has non-zero measure, and S has finite measure. The *transition model* has the form of a continuous probability density function $p_{s'|s,a}$ on the resulting state s', given previous state s and action a, such that $\forall s' \in S, p_{s'|s,a}(s' \mid s, a) \geq 0, \int_S p(s' \mid s, a)\, \mathrm{d}s' = 1$.

Given a transition model and a cost function $C : S \times S \times A \to \mathbb{R}$ associated with a goal region, we can formulate the problem as a continuous state-action MDP $(S, A, p_{s'|s,a}, R, \gamma)$, where $R(s' \mid s, a) = -C(s' \mid s, a)$ is the immediate reward function and γ is the discount factor. A high reward is assigned to the states in the goal region $\mathcal{G}$, and a cost is assigned to colliding with obstacles or taking any action. We would like to solve for the optimal policy $\pi : S \to A$, for which the value of each state s is

$$V^\pi(s) = \max_{a \in A} \int_{s' \in S} p_{s'|s,a}(s' \mid s, a) \left(R(s' \mid s, a) + \gamma^{\Delta t} V^\pi(s') \right) \mathrm{d}s'.$$

5.2 Related Work

Learning The class of problems that we address may be viewed as reinforcement-learning (RL) problems in observable continuous state-action spaces. It is possible to address the problem through model-free RL, which estimates a value function or policy for a specific goal directly through experience. Though the majority of work in RL addresses domains with discrete action spaces, there has been a thread of relevant work on value-function-based RL in continuous action spaces [73, 11, 149, 183, 141]. An alternative approach is to do direct search in the space of policies [42, 84].

In continuous state-action spaces, model-based RL, where a model is estimated to optimize a policy, can often be more effective. Gaussian processes (GP) can help to learn the dynamics [44, 153, 140], which can then be used by GP-based dynamic programming [45, 153] to determine a

continuous-valued closed-loop policy for the whole state space. More details can be found in the excellent survey [66].

Unfortunately, the common assumption of i.i.d Gaussian noise on the dynamics is restrictive and may not hold in practice [203], and the transition model can be multi-modal. It may additionally be difficult to obtain a good GP prior. The basic GP model is can capture neither the multi-modality nor the heteroscedasticity of the noise. While more advanced GP algorithms may address these problems, they often suffer from high computational cost [181, 204].

Moldovan et al. [136] addressed the problem of multi-modality by using Dirichlet process mixture models (DPMMs) to learn the density of the transition models. Their strategies for planning were limited by deterministic assumptions, appropriate for their domains of application, but potentially resulting in collisions in ours. Kopicki et al. [103, 102, 104] addressed the problem of learning to predict the behavior of rigid objects under manipulations such as pushing, using kernel density estimation. In this chapter, we propose an efficient planner that can work with arbitrary, especially multi-modal stochastic models in continuous state-action spaces. Our learning method in the experiment resembles DPMMs but we estimate the density on the fly when the planner queries a state-action pair. We were not able to compare our approach with DPMMs because we found DPMMs not computationally feasible for large datasets.

Planning We are interested in domains for which queries are made by specifying a starting state and a goal set, and in which the solution to the given query can be described by a policy that covers only a small fraction of the state space that the robot is likely to encounter.

Planning only in the fraction of the state-action space that the robot is likely to encounter is, in general, very challenging. Other related work uses tree-based search methods [194, 129, 201], where the actions are selected by optimizing an optimistic heuristic. These algorithms are impractical for our problem because of the exponential growth of the tree and the lack of immediate rewards that can guide the pruning of the tree.

In contrast to the tree-search algorithms, iMDP [82], which is most related to our work, uses sampling techniques from RRTs to create successively more accurate discrete MDP approximations of the original continuous MDP, ultimately converging to the optimal solution to the original problem. Their method assumes the ability to solve the Bellman equation optimally (e.g. for a simple stochastic LQR problem), the availability of the backward transition models, and that the dynamics is modeled by a Wiener process, in which the transition noise is Gaussian with execution-time-dependent variance. However, the assumptions are too restrictive to model our domains of interest

where the dynamics is non-closed-form, costly to evaluate, non-reversible, and non-Gaussian. Furthermore, iMDP is designed for stochastic control problems with multiple starting states and a single goal, while we are interested in multiple start-goal pairs.

Our work builds on the idea of constructing a sequence of MDPs from iMDP [82], and aims at practically resolving the challenges of state/action selection faced by both iMDP and tree-search-based planners [194].

Bayesian optimization There have been a number of applications of BO in optimal control, although to our knowledge, it has not been previously applied to action-selection in continuous-action MDPs. BO has been used to find weights in a neural network controller [62], to solve for the parameters of a hierarchical MDP [27], and to address safe exploration in finite MDPs [182]. To our knowledge, BO has not been previously applied to action-selection in continuous-action MDPs.

5.3 Our method: BOIDP

We describe our algorithm Bayesian Optimization Incremental-realtime Dynamic Programming (BOIDP) in this section. At the highest level, BOIDP in Alg. 7 operates in a loop, in which it samples a discrete set of states $\tilde{S} \subset S$ and attempts to solve the discrete-state, continuous-action MDP $\tilde{\mathcal{M}} = (\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$. Here $\hat{P}_{s'|s,a}(s' \mid s, a)$ is the probability mass function for the transition from state $s \in \tilde{S}$ and action $a \in A$ to a new state $s' \in \tilde{S}$. The value function for the optimal policy of the approximated MDP $\tilde{\mathcal{M}}$ is $V(s) = \max_{a \in A} Q_s(a)$, where

$$Q_s(a) = \sum\nolimits_{s' \in \tilde{S}} \hat{P}_{s'|s,a}(s' \mid s, a) \left(R(s' \mid s, a) + \gamma^{\Delta t} V(s') \right). \tag{5.1}$$

If the value of the resulting policy is satisfactory according to the task-related stopping criterion[1], we can proceed; otherwise, additional state samples are added and the process is repeated. Once we have a policy π on $\tilde{S}$ from RTDP, the robot can iteratively obtain and execute the policy for the nearest state to the current state in the sampled set $\tilde{S}$ by the metric d.

There are a number of challenges underlying each step of BOIDP. First, we need to find a way of accessing the transition probability density function $p_{s'|s,a}$, which is critical for the approximation of $\hat{P}_{s'|s,a}(s' \mid s, a)$ and the value function. We describe our "lazy access" strategy in Sec. 5.3.1. Second, we must find a way to compute the values of as few states as possible to fully exploit the

[1]For example, one stopping criterion could be the convergence of the starting state's value $V(s_0)$.

Algorithm 7 BOIDP

function BOIDP($s_0, \mathcal{G}, S, p_{s'|s,a}, N_{\min}$)
  $\tilde{S} \leftarrow \{s_0\}$
  loop
    $\tilde{S} \leftarrow$ SAMPLESTATES($N_{\min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
    $\pi, V =$ RTDP($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  end loop
  until stopping criteria reached
  EXECUTEPOLICY($\pi, \tilde{S}, \mathcal{G}$)
end function

function EXECUTEPOLICY($\pi, \tilde{S}, \mathcal{G}$)
  loop
    $s_c \leftarrow$ current state
    $\tilde{s} \leftarrow \arg\min_{s \in \tilde{S}} d(s, s_c)$
    Execute $\pi(\tilde{s})$
  end loop
  until current state is in $\mathcal{G}$
end function

"lazy access" to the transition model. Our solution is to first use an RRT-like process [117, 82] to generate the set of states that asymptotically cover the state space with low dispersion (Sec. 5.3.2), and then "prune" the irrelevant states via RTDP [15] (Sec. 5.3.3). Last, each dynamic-programming update in RTDP requires a maximization over the action space; we cannot achieve this analytically and so must sample a finite set of possible actions. We develop a new batch BO algorithm to focus action sampling on regions of the action space that are informative and/or likely to be high-value, as described in Sec. 5.3.4.

Both the state sampling and transition estimation processes assume that there exists a collision checker EXISTSCOLLISION(s, a, s') that checks the path from s to s' induced by action a for collisions with permanent objects in the map.

5.3.1 Estimating transition models in BOIDP

In a typical model-based learning approach, first a monolithic model is estimated from the data and then that model is used to construct a policy. Here, however, we aim to scale to large spaces with non-Gaussian dynamics, a setting where it is very difficult to represent and estimate a single monolithic model. Hence, we take a different approach via "lazy access" to the model: we estimate local models on demand, as the planning process requires information about relevant states and actions.

We assume a dataset $D = \{s_i, a_i, s'_i\}_{i=0}^{N}$ for the system dynamics and the dataset is large enough to provide a good approximation to the probability density of the next state given any state-action pair. If a stochastic simulator exists for the transition model, one may collect the dataset dynamically in response to queries from BOIDP. The "lazy access" provides a flexible interface, which can accommodate a variety of different density-estimation algorithms with asymptotic theoretical guarantees, such as kernel density estimators [197] and Gaussian mixture models [135]. In our experiments, we focus on learning Gaussian mixture models with the assumption that $p_{s'|s,a}(s' \mid s, a)$ is distributed according to a mixture of Gaussians $\forall (s, a) \in S \times A$. Note that here although the action space A has infinite elements, our method does not require computing the transition model for any (s, a) pairs, as detailed in Section 5.3.4.

Given a discrete set of states $\tilde{S}$, starting state s and action a, we compute the approximate discrete transition model $\hat{P}_{s'|s,a}$ as shown in Algorithm 8. We use the function HIGHPROBNEXTSTATES to select the largest set of next states $\hat{S} \subseteq \tilde{S}$ such that $\forall s' \in \hat{S}, p_{s'|s,a}(s' \mid s, a) > \epsilon$. ϵ is a small threshold parameter, e.g. we can set $\epsilon = 10^{-5}$. If $p_{s'|s,a}$ does not take obstacles into account, we have to check the path from state s to next state $s' \in \tilde{S}$ induced by action a for collisions, and model their effect in the approximate discrete transition model $\hat{P}_{s'|s,a}$. To achieve this, we add a dummy terminal state s_{obs}, which represents a collision, to the selected next-state set $\hat{S}$. Then, for any s, a, s' transition that generates a collision, we move the probability mass $\hat{P}_{s'|s,a}(s' \mid s, a)$ to the transition to the collision state $\hat{P}_{s'|s,a}(s_{obs} \mid s, a)$. Finally, $\hat{P}_{s'|s,a}(\hat{S} \mid s, a)$ is normalized and returned together with the selected set $\hat{S}$.

These approximated discrete transition models can be indexed by state s and action a and cached for future use in tasks that use the same set of states $\tilde{S}$ and the same obstacle map. The memory-based essence of our modeling strategy is similar to the strategy of non-parametric models such as Gaussian processes, which make predictions for new inputs via smoothness assumptions and similarity between the query point and training points in the data set.

For the case where the dynamics model $p_{s'|s,a}$ is given, computing the approximated transition $\hat{P}_{s'|s,a}$ could still be computationally expensive because of the collision checking. Our planner is designed to alleviate the high computation in $\hat{P}_{s'|s,a}$ by focusing on the relevant states and actions, as detailed in the next sections.

Algorithm 8 Transition model for discrete states

function TRANSITIONMODEL($s, a, \tilde{S}, p_{s'|s,a}$)
  $\hat{S} \leftarrow$ HIGHPROBNEXTSTATES($p_{s'|s,a}(\tilde{S} \mid s, a)) \cup \{s_{obs}\}$ ▷ s_{obs} is a terminal state
  $\hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow p_{s'|s,a}(\hat{S} \mid s, a)$
  for s' **in** $\hat{S}$ **do**
    if $s' \in S^o$ **and** EXISTSCOLLISION(s, a, s') **then**
      $\hat{P}_{s'|s,a}(s_{obs} \mid s, a) \leftarrow \hat{P}_{s'|s,a}(s_{obs} \mid s, a) + \hat{P}_{s'|s,a}(s' \mid s, a)$
      $\hat{P}_{s'|s,a}(s' \mid s, a) \leftarrow 0$
    end if
  end for
  $\hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow$ NORMALIZE($\hat{P}_{s'|s,a}(\hat{S} \mid s, a)$)
  return $\hat{S}, \hat{P}_{s'|s,a}(\hat{S} \mid s, a)$
end function

Algorithm 9 RRT states sampling for BOIDP

function SAMPLESTATES($N_{min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  $\tilde{S}^o \leftarrow$ SAMPLEINTERIORSTATES($\lceil N_{min}/2 \rceil, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  $\partial\tilde{S} \leftarrow$ SAMPLEBOUNDARYSTATES($\lceil N_{min}/2 \rceil, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  return $\tilde{S}^o \cup \partial\tilde{S}$
end function

function SAMPLEINTERIORSTATES($N_{min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  while $| \tilde{S} | < N_{min}$ or $\mathcal{G} \cap \tilde{S} = \emptyset$ **do**
    $s_{rand} \leftarrow$ UNIFORMSAMPLE(S)
    $s_{nearest} \leftarrow$ NEAREST($s_{rand}, \tilde{S}$)
    $s_n, a_n \leftarrow$ RRTEXTEND($s_{nearest}, s_{rand}, p_{s'|s,a}$)
    if found s_n, a_n **then**
      $\tilde{S} \leftarrow \tilde{S} \cup \{s_n\}$
    end if
  end while
  return $\tilde{S}$
end function

function RRTEXTEND($s_{nearest}, s_{rand}, p_{s'|s,a}$)
  $d_n = \infty$
  while stopping criterion not reached **do**
    $a \leftarrow$ UNIFORMSAMPLE(A)
    $s' \leftarrow$ SAMPLE $\left(p_{s'|s,a}(\cdot \mid s_{nearest}, a)\right)$
    if (**not** EXISTSCOLLISION(s, s', a)) **and** $d_n > d(s_{rand}, s')$ **then**
      $d_n \leftarrow d(s_{rand}, s')$
      $s_n, a_n \leftarrow s', a$
    end if
  end while
  return s_n, a_n
end function

5.3.2 Sampling states

Algorithm 9 describes the state sampling procedures. The input to SAMPLESTATES in Alg. 9 includes the minimum number of states, $N_{\min}$, to sample at each iteration of BOIDP. It may be that more than $N_{\min}$ states are sampled, because sampling must continue until at least one terminal goal state is included in the resulting set $\tilde{S}$. To generate a discrete state set, we sample states both in the interior of S^o and on its boundary ∂S. Notice that we can always add more states by calling SAMPLESTATES.

To generate one interior state sample, we randomly generate a state s_{rand}, and find $s_{nearest}$ that is the nearest state to s_{rand} in the current sampled state set $\tilde{S}$. Then we sample a set of actions from A, for each of which we sample the next state s_n from the dataset D given the state-action pair $s_{neareast}, a$ (or from $p_{s'|s,a}$ if given). We choose the action a that gives us the s_n that is the closest to s_{rand}. To sample states on the boundary ∂S, we assume a uniform random generator for states on ∂S is available. If not, we can use something similar to SAMPLEINTERIORSTATES but only sample inside the obstacles uniformly in line 8 of Algorithm 9. Once we have a sample s_{rand} in the obstacle, we try to reach s_{rand} by moving along the path $s_{rand} \to s_n$ incrementally until a collision is reached.

5.3.3 Focusing on the relevant states via RTDP

We apply our algorithm with a known starting state s_0 and goal region $\mathcal{G}$. Hence, it is not necessary to compute a complete policy, and so we can use RTDP [15] to compute a value function focusing on the relevant state space and a policy that, with high probability, will reach the goal before it reaches a state for which an action has not been determined. We assume an upper bound of the values for each state s to be $h_u(s)$. One can approximate $h_u(s)$ via the shortest distance from each state to the goal region on the fully connected graph with vertices $\tilde{S}$. We show the pseudocode in Algorithm 10. When doing the recursion (TRIALRECURSE), we can save additional computation when maximizing $Q_s(a)$. Assume that the last time $\arg\max_a Q_s(a)$ was called, the result was a^* and the transition model tells us that $\bar{S}$ is the set of possible next states. The next time we call $\arg\max_a Q_s(a)$, if the values for $\bar{S}$ have not changed, we can just return a^* as the result of the optimization. This can be done easily by caching the current (optimistic) policy and transition model for each state.

Algorithm 10 RTDP for BOIDP

function RTDP($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  for s **in** $\tilde{S}$ **do**
    $V(s) = h_u(s)$ ▷ Compute the value upper bound
  end for
  while $V(\cdot)$ not converged **do**
    $\pi, V \leftarrow$ TRIALRECURSE$(s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a})$
  end while
  return π, V
end function

function TRIALRECURSE($s, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  if reached cycle **or** $s \in \mathcal{G}$ **then**
    return
  end if
  $\pi(s) \leftarrow \arg\max_a Q(s, a, \tilde{S}, p_{s'|s,a})$ ▷ Max via BO
  $s' \leftarrow$ SAMPLE$(\hat{P}_{s'|s,a}(\tilde{S}|s, \pi(s)))$
  TRIALRECURSE$(s', \mathcal{G}, \tilde{S}, p_{s'|s,a})$
  $\pi(s) \leftarrow \arg\max_a Q(s, a, \tilde{S}, p_{s'|s,a})$ ▷ Max via BO
  $V(s) \leftarrow Q(s, \pi(s), \tilde{S}, p_{s'|s,a})$
  return π, V
end function

function Q($s, a, \tilde{S}, p_{s'|s,a}$)
  if $\hat{P}_{s'|s,a}(\tilde{S}|s, a)$ has not been computed **then**
    $\hat{P}_{s'|s,a}(\tilde{S}|s, a) = \mathbf{0}$ ▷ $\hat{P}_{s'|s,a}$ is a shared matrix
    $\hat{S}, \hat{P}_{s'|s,a}(\hat{S}|s, a) \leftarrow$ TRANSITIONMODEL$(s, a, \tilde{S}, p_{s'|s,a})$
  end if
  return $R(s, a) + \gamma^{\Delta t} \sum_{s' \in \hat{S}} \hat{P}_{s'|s,a}(s'|s, a) V(s')$
end function

5.3.4 Focusing on good actions via BO

RTDP in Algorithm 10 relies on a challenging optimization over a continuous and possibly high-dimensional action space. Queries to $Q_s(a)$ in Eq. (5.1) can be very expensive because in many cases a new model must be estimated. Hence, we need to limit the number of points queried during the optimization. There is no clear strategy for computing the gradient of $Q_s(a)$, and random sampling is very sample-inefficient especially as the dimensionality of the space grows. We will view the optimization of $Q_s(a)$ as a black-box function optimization problem, and use batch BO to efficiently approximate the solution and make full use of the parallel computing resources.

Algorithm 11 Optimization of $Q_s(a)$ via sequential GP optimization

$\mathfrak{D}_0 \leftarrow \emptyset$
for $t = 1 \rightarrow T$ **do**
  $\mu_{t-1}, \sigma_{t-1} \leftarrow$ GP-predict($\mathfrak{D}_{t-1}$)
  $a_t \leftarrow \arg\min_{a \in A} \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)}$
  $y_t \leftarrow Q_s(a_t)$
  $\mathfrak{D}_t \leftarrow \mathfrak{D}_{t-1} \cup \{a_t, y_t\}$
end for

Algorithm 12 Optimization of $Q_s(a)$ via batch GP optimization

$\mathfrak{D}_0 \leftarrow \emptyset$
for $t = 1 \rightarrow T$ **do**
  $\mu_{t-1}, \sigma_{t-1} \leftarrow$ GP-predict($\mathfrak{D}_{t-1}$)
  $B \leftarrow \emptyset$
  for $i = 1 \rightarrow M$ **do**
    $B \leftarrow B \cup \{\arg\max_{a \in A} F_s(B \cup \{a\}) - F_s(B)\}$
  end for
  $\boldsymbol{y}_B \leftarrow Q_s(B)$ ▷ Test Q_s in parallel
  $\mathfrak{D}_t \leftarrow \mathfrak{D}_{t-1} \cup \{B, \boldsymbol{y}_B\}$
end for

We first briefly review a sequential Gaussian-process optimization method, GP-EST [191], shown in Algorithm 11. For a fixed state s, we assume $Q_s(a)$ is a sample from a Gaussian process with zero mean and kernel κ. At iteration t, we select action a_t and observe the function

value $y_t = Q_s(a_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. Given the observations $\mathfrak{D}_t = \{(a_\tau, y_\tau)\}_{\tau=1}^t$ up to time t, we obtain the posterior mean and covariance of the $Q_s(a)$ function via the kernel matrix $\boldsymbol{K}_t = [\kappa(a_i, a_j)]_{a_i, a_j \in \mathfrak{D}_t}$ and $\boldsymbol{\kappa}_t(a) = [\kappa(a_i, a)]_{a_i \in \mathfrak{D}_t}$ [151]:

$$\begin{aligned} \mu_t(a) &= \boldsymbol{\kappa}_t(a)^{\mathrm{T}} (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}_t \\ \kappa_t(a, a') &= \kappa(a, a') - \boldsymbol{\kappa}_t(a)^{\mathrm{T}} (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{\kappa}_t(a') \ . \end{aligned}$$

The posterior variance is given by $\sigma_t^2(a) = \kappa_t(a, a)$. We can then use the posterior mean function $\mu_t(\cdot)$ and the posterior variance function $\sigma_t^2(\cdot)$ to select which action to test in the next iteration. We here make use of the assumption that we have an upper bound $h_u(s)$ on the value $V(s)$. We select the action that is most likely to have a value greater than or equal to $h_u(s)$ to be the next one to evaluate. Algorithm 11 relies on sequential tests of $Q_s(a)$, but it may be much more effective to test $Q_s(a)$ for multiple values of a in parallel. This requires us to choose a diverse subset of actions that are expected to be informative and/or have good values.

We propose a new batch Bayesian optimization method that selects a query set that has large diversity and low values of the acquisition function $G_{s,t}(a) = \left(\frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)}\right)$. The key idea is to maximize a submodular objective function with a cardinality constraint on $B \subset A, |B| = M$ that characterize both diversity and quality:

$$F_s(B) = \log\det \boldsymbol{K}_B - \lambda \sum\nolimits_{a \in B} \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{5.2}$$

where $\boldsymbol{K}_B = [\kappa(a_i, a_j)]_{a_i, a_j \in B}$ and λ is a trade-off parameter for diversity and quality. If λ is large, F_s will prefer actions with lower $G_{s,t(a)}$, which means a better chance of having high values. If λ is low, $\log\det \boldsymbol{K}_B$ will dominate F_s and a more diverse subset B is preferred. λ can be chosen by cross-validation. We optimize the heuristic function F_s via greedy optimization which yield a $1 - \frac{1}{e}$ approximation to the optimal solution. We describe the batch GP optimization in Algorithm 12.

The greedy optimization can be efficiently implemented using the following property of the

determinant:

$$F_s(B \cup \{a\}) - F_s(B) \tag{5.3}$$

$$= \log\det \boldsymbol{K}_{B\cup\{a\}} - \log\det \boldsymbol{K}_B - \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{5.4}$$

$$= \log(\kappa_a - \boldsymbol{\kappa}_{Ba}^{\mathrm{T}} \boldsymbol{K}_B^{-1} \boldsymbol{\kappa}_{Ba}) - \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{5.5}$$

where $\kappa_a = \kappa(a, a), \boldsymbol{\kappa}_{Ba} = [\kappa(a_i, a)]_{a_i \in B}$.

5.4 Theoretical analysis

In this section, we characterize the theoretical behavior of BOIDP. Thm. 5.4.1 establishes the error bound for the value function on the $\hat{\pi}^*$-*relevant* set of states [15], where $\hat{\pi}^*$ is the optimal policy computed by BOIDP. A set $B \subseteq S$ is called π-*relevant* if all the states in B is reachable via finite actions from the starting state s_0 under the policy π. We denote $|\cdot|_B$ as the L_∞ norm of a function $\cdot$ over the set B.

We assume the existence of policies whose relevant sets intersect with $\mathcal{G}$. If there exists no solution to the continuous state-action MDP $\mathcal{M}$, our algorithm will not be able to generate an RRT whose vertices contain a state in the goal region $\mathcal{G}$, and hence no policy will be generated. We use the reward setup described in Sec. 5.1. For the simplicity of the analysis, we set the reward for getting to the goal large enough such that the optimal value function $V^*(s) = \max_{a\in A} Q_s(a)$ is positive for any state s on the path to the goal region under the optimal policy π^*.

We denote the measure for the state space S to be ρ and the measure for the action space A to be ψ. Both ρ and ψ are absolutely continuous with respect to Lebesgue measure. The metric for A is g, and for S is d. We also assume the transition density function $p_{s'|s,a}$ is not a generalized function and satisfies the property that if $\int_{\mathcal{F}} p_{s'|s,a}(s'|s,a)\,\mathrm{d}s' > 0$, then $\rho(\mathcal{F}) > 0$. Without loss of generality, we assume $\min \Delta t = 1$ and $\max \Delta t = \mathcal{T}$.

Under mild conditions on $Q_s(a)$ specified in Thm. 5.4.1, we show that with finitely many actions selected by BO, the expected accumulated error expressed by the difference between the optimal value function V^* and the value function $\hat{V}$ of the policy computed by BOIDP in Alg. 7 on the $\hat{\pi}^*$-*relevant* set decreases to 0 as the number of actions selected for optimizing $Q_s(\cdot)$ in Eq. (5.1) increases.

Theorem 5.4.1 (Error bound for BOIDP)**.** *Let $D = \{s_i, a_i, s_i'\}_{i=0}^N$ be the dataset that is collected*

from the true transition probability $p_{s'|s,a}$, $\forall(s,a) \in S \times A$. We assume that the transition model $p_{s'|s,a}$ estimated by the density estimator asymptoticly converges to the true model. $\forall s \in S$, we assume $Q_s(a) = \int_{s' \in S} p_{s'|s,a}(s' \mid s,a) \left(R(s' \mid s,a) + \gamma^{\Delta t} V^*(s')\right) \mathrm{d}s'$ is a function locally continuous at $\arg\max_{a \in A} Q_s(a)$, where $V^*(\cdot) = \max_{a \in A} Q_\cdot(a)$ is the optimal value function for the continuous state-action MDP $\mathcal{M} = (S, A, p_{s'|s,a}, R, \gamma)$. $V^*(\cdot)$ is associated with an optimal policy whose relevant set contains at least one state in the goal region $\mathcal{G}$. At iteration k of RTDP in Alg. 10, we define $\hat{V}_k$ to be the value function for $\tilde{\mathcal{M}} = (\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$ approximated by BOIDP*, $\hat{\pi}_k$ to be the policy corresponding to $\hat{V}_k$, and B_k to be the $\hat{\pi}_k$-*relevant set*. We assume that*

$$Q_{s,k}(a) = \sum\nolimits_{s' \in \tilde{S}} \hat{P}_{s'|s,a}(s' \mid s,a) \left(R(s' \mid s,a) + \gamma^{\Delta t} \hat{V}_{k-1}(s')\right)$$

is a function sampled from a Gaussian process with known priors and i.i.d Gaussian noise $\mathcal{N}(0, \sigma)$. If we allow Bayesian optimization for $Q_{s,k}(a)$ to sample T actions for each state and run RTDP in Alg. 10 until it converges with respect to the Cauchy's convergence criterion [33] with $\mathcal{K} < \infty$ iterations, in expectation,

$$\lim\nolimits_{|\tilde{S}|,|D| \to \infty} |\hat{V}_{\mathcal{K}}(\cdot) - V^*(\cdot)|_{B_{\mathcal{K}}} \leq \frac{\nu}{1-\gamma} \sqrt{\frac{2\eta_T}{T \log(1+\sigma^2)}},$$

where η_T is the maximum information gain of the selected actions [171, Theorem 5], $\nu = \max\limits_{s,t,k} \min\limits_{a \in A} G_{s,t,k}(a)$, and $G_{s,t,k}(\cdot)$ is the acquisition function in [191, Theorem 3.1] for state $s \in \tilde{S}$, iteration $t = 1, 2, \cdots, T$ in Alg. 1 or 12, and iteration $k = 1, 2, \cdots, \mathcal{K}$ of the loop in Alg. 10.

Proof. To prove Thm. 5.4.1, we first show the following facts: (1) The state sampling procedure in Alg. 9 stops in finite steps; (2) the difference between the value function computed by BOIDP and the optimal value function computed via asynchronous dynamic programming with an exact optimizer is bounded in expectation; (3) the optimal value function of the approximated MDP $\tilde{\mathcal{M}}$ asymptotically converges to that of the original problem defined by the MDP $\mathcal{M}$.

Claim 5.4.1.1: The expected number of iterations for the set of sampled states $\tilde{S}$ computed by Alg. 9 to contain one state in $\mathcal{G}$ is finite.

Proof of Claim 5.4.1.1: Let S_{good} be the set of states with non-zero probability to reach the goal region via finite actions. Clearly, $\mathcal{G} \subset S_{good}$. Because there exists a state in the goal region that is reachable with finite steps following the policy π^* starting from s_0, we have $s_0 \in S_{good}$. Hence $\tilde{S} \cap S_{good}$ is non-empty in any iteration of SAMPLEINTERIORSTATES of Alg. 9. We can show

that if the nearest state selected in Line 8 of Alg. 9 is in S_{good}, there is non-zero probability to extend another state in S_{good} with the RRT procedure. To prove this, we first show for every state $s \in S_{good} \cap \tilde{S}$ there exists a set of actions with non-zero measure, in which each action a satisfies $\int_{S_{good}} p_{s'|s,a}(s' \mid s, a)\, \mathrm{d}s' > 0$.

For every state $s \in S_{good} \cap \tilde{S}$, because $Q_s(a)$ is locally continuous at $a = \pi^*(s)$, for any positive real number ζ, there exists a positive real number δ such that $\forall a \in \{a : g(a, \pi^*(s)) < \delta, a \in A\}$, we have

$$|Q_s(a) - Q_s(\pi^*(s))| < \zeta. \tag{5.6}$$

Notice that by the design of the reward function R, we have

$$\begin{aligned} Q_s(a) &= \int_{S_{good} \cup (S \setminus S_{good})} p_{s'|s,a}(s' \mid s, a) \left(R(s' \mid s, a) + \gamma^{\Delta t} V^*(s')\right) \mathrm{d}s' && (5.7) \\ &\leq \int_{S_{good}} p_{s'|s,a}(s' \mid s, a) \left(R(s' \mid s, a) + \gamma^{\Delta t} V^*(s')\right) \mathrm{d}s' \\ &\quad + \frac{C_a}{1 - \gamma^{\mathcal{T}}} \int_{S \setminus S_{good}} p_{s'|s,a}(s' \mid s, a)\, \mathrm{d}s' && (5.8) \end{aligned}$$

where $-C_a > 0$ is the smallest cost for either executing one action or colliding with obstacles. The inequality is because $\forall s' \in S \setminus S_{good}$,

$$R(s' \mid s, a) + \gamma^{\Delta t} V^*(s') \leq \frac{C_a}{1 - \gamma^{\mathcal{T}}} < 0. \tag{5.9}$$

Because $\pi^*(s) = \arg\max_{a \in A} Q_s(a)$ and the reward for the goal region is set large enough so that $Q_s(\pi^*(s)) > 0$, there exists $r > 0$ such that $\forall q \in \mathbb{R}$ satisfying $q > Q_s(\pi^*(s)) - r$, we have $q > 0$. Let the arbitrary choice of ζ in Eq. (5.6) be $\zeta = r$. Because $Q_s(\pi^*(s)) - \zeta = Q_s(\pi^*(s)) - r < Q_s(a)$, we have

$$Q_s(a) > 0, \forall a \in \{a : g(a, \pi^*(s)) < \delta, a \in A\},$$

and

$$\int_{S_{good}} p_{s'|s,a}(s' \mid s, a) \left(R(s' \mid s, a) + \gamma^{\Delta t} V^*(s')\right) \mathrm{d}s' > -\frac{C_a}{1 - \gamma^{\mathcal{T}}} \int_{S \setminus S_{good}} p_{s'|s,a}(s' \mid s, a)\, \mathrm{d}s' > 0$$

Hence $\int_{s' \in S_{good}} p_{s'|s,a}(s' \mid s, a)\, \mathrm{d}s' > 0$ must hold for any action $a \in \{a : g(a, \pi^*(s)) < \delta, a \in A\}$.

Recall that one dimension of $a = (u, \Delta t)$ is the duration Δt of the control u. Because the dynamics of the physics world is continuous, for any $a = (u, \Delta t') \in A$ such that $g((u, \Delta t), \pi^*(s)) < \delta$ and $1 \leq \Delta t' \leq \Delta t$, we have $\int_{S_{good}} p_{s'|s,a}(s' \mid s, a)\, \mathrm{d}s' > 0$. Let $A_s = \{(u, \Delta t') : g((u, \Delta t), \pi^*(s)) < \delta, 1 \leq \Delta t' \leq \Delta t\} \cap A$.

What remains to be shown is $\psi(A_s) > 0$. Because there exists an open set A^o such that A is the closure of A^o, $\pi^*(s)$ is either in A^o or a limit point of A^o. If $\pi^*(s)$ is in the open set A^o, there exist $0 < \delta' \leq \delta$ such that $\{a : g(a, \pi^*(s)) < \delta'\} \subset A$, and so we also have $\psi(A_s) > 0$. If $\pi^*(s)$ is a limit point of the open set A^o, there exist $a' \in A^o$ such that $g(a', \pi^*(s)) < \delta/2$ and $a' \neq \pi^*(s)$. Because $a' \in A^o$, there exist $0 < \delta' \leq \delta/2$ such that $\{a : g(a', a) < \delta'\} \subset A$. For any $a \in \{a : g(a', a) < \delta'\}$, we have $g(a, \pi^*(s)) \leq g(a, a') + g(a', \pi^*(s)) < \delta' + \delta/2 \leq \delta$. Hence $\{a : g(a', a) < \delta'\} \subset A_s$, and so $\psi(A_s) > 0$. Thus for any $\pi^*(s) \in A$, we have $\psi(A_s) > 0$.

So, for every state $s \in S_{good} \cap \tilde{S}$ and action $a \in A_s$ with $\psi(A_s) > 0$, $\int_{S_{good}} p_{s'|s,a}(s' \mid s, a)\, \mathrm{d}s' > 0$. As a corollary, $\rho(\{s' : p(s' \mid s, a) > 0\} \cap S_{good}) > 0$ holds $\forall s \in S_{good} \cap \tilde{S}, a \in A_s$.

Now we can show that there is non-zero probability to extend a state on the near-optimal path in S_{good} with the RRT procedure in one iteration of SAMPLEINTERIORSTATES in Alg. 9. Let $\theta = \min_{s \in \tilde{S} \cap S_{good}, a \in A_s} \rho(\{s' : p(s' \mid s, a) > 0\} \cap S_{good}) > 0$. With the finite $\tilde{S}$ for some iteration, we can construct a Voronoi diagram based on the vertices from the current set of sampled states $\tilde{S}$. $\forall s \in S_{good} \cap \tilde{S}$, there exists a Voronoi region $\mathrm{Vor}(s)$ associated with state s. We can partition this Voronoi region $\mathrm{Vor}(s)$ to one part, $\mathrm{Vor}(A_s) \subset \mathrm{Vor}(s)$, containing states in S_{good} generated by actions in A_s and its complement, $\mathrm{Vor}(A \setminus A_s) = \mathrm{Vor}(s) \setminus \mathrm{Vor}(A_s)$. Notice that A_s includes actions with the minimum duration, and the unit for the minimum duration can be set small enough so that $\rho(\{s' : p(s' \mid s, a) > 0, a \in A_s, s' \in S_{good}\} \cap \mathrm{Vor}(s)) > 0$[2]. Since $\{s' : p(s' \mid s, a) > 0, a \in A_s\} \cap S_{good} \cap \mathrm{Vor}(s) \subset \mathrm{Vor}(A_s)$, we have $\rho(\mathrm{Vor}(A_s)) > 0, \forall s \in \tilde{S}$. We denote $p_s = \min_s \frac{\rho(\mathrm{Vor}(A_s))}{\rho(S)} > 0$ and $p_a = \min_s \frac{\psi(A_s)}{\psi(A)} > 0$. With probability at least p_s, there is a random state sampled in $\mathrm{Vor}(A_s)$ in this iteration. With probability at least p_a, at least an action in A_s is selected to test distance, and with probability at least θ, a state in S_{good} can be sampled from the transition model conditioned on the state s and the selected action in A_s.

Next we show that SAMPLEINTERIORSTATES in Alg. 9 constructs an RRT whose finite set of sampled states $\tilde{S}$ contains at least one goal state in expectation.

By assumption, the goal state is reachable with finite actions. For any $s \in S_{good} \cap \tilde{S}$, the goal

[2]This is because $\mathrm{Vor}(s)$ is a neighborhood of s, and there exists an action $a \in A_s$ such that a next state $s' \in S_{good}$ is in the interior of $\mathrm{Vor}(s)$ given the current state s. So there exists a small ball in S with s' as the center such that this ball is a subset of both $\mathrm{Vor}(s)$ and $\{s' : p(s' \mid s, a) > 0, a \in A_s, s' \in S_{good}\}$ (by the continuity of $p_{s'|s,a}$).

region is reachable from s in finite steps. Notice that once a new state in $\{s' : p(s' \mid s, a) > 0, a \in A_s\} \cap S_{good}$ is sampled, s' uses one less step than s to reach the goal region. Let K be the largest finite number of actions necessary to reach the goal region $\mathcal{G}$ from the initial state s_0. Hence, with at most a finite number of $\frac{K}{\theta p_s p_a}$ iterations in expectation (including both loops for sampling actions and loops for sampling states), at least a goal state will be added to $\tilde{S}$.

Q.E.D.

Claim 5.4.1.2: Let $\tilde{V}^*$ be the optimal value function computed via asynchronous dynamic programming with an exact optimizer. If Alg. 10 converges with $\mathcal{K} < \infty$ iterations,

$$|\hat{V}_{\mathcal{K}}(\cdot) - \tilde{V}^*(\cdot)|_{B_{\mathcal{K}}} \leq \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}.$$

Proof of Claim 5.4.1.2: The RTDP process of BOIDP in Alg. 10 searches for the relevant set $B_{\mathcal{K}}$ of BOIDP's policy $\hat{\pi}$ via recursion on stochastic paths (trials). If BOIDP converges, all states in $B_{\hat{\pi}}$ should have been visited and their values $\hat{V}(s), \forall s \in B_{\mathcal{K}}$ have converged. Compared to asynchronous dynamic programming (ADP) with an exact optimizer, our RTDP process introduces small errors at each trial, but eventually the difference between the optimal value function computed by ADP and the value function computed by BOIDP is bounded.

In the following, the order of states to be updated in ADP is set to follow RTDP. This order does not matter for the convergence of ADP as any state in $B_{\hat{\pi}}$ will eventually be visited infinitely often if $\mathcal{K} \to \infty$ [173]. We denote the value for the i-th state updated at iteration k of RTDP to be $\hat{V}_{ki}$, the corresponding value function updated by RTDP with an exact optimizer only for this update to be $\hat{V}^*_{ki}$, and the difference between them to be $\epsilon_{ki} = |\hat{V}^*_{ki} - \hat{V}_{ki}|$.

According to [191, Theorem 3.1], in expectation, for any i-th state s_{ki} to be updated at iteration k, the following inequality holds:

$$\epsilon_{ki} \leq \nu_{ki}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}},$$

where

$$\nu_{ki} = \max_{t\in[1,T]} \min_{a\in A} G_{s_i,t,k}(a),$$

and

$$G_{s_i,t,\cdot}(a) = \frac{h_u(s_i) - \mu_{t-1}(a)}{\sigma_{t-1}(a)}$$

is the acquisition function in [191, Theorem 3.1], which makes use of the assumed upper bound $h_u(\cdot)$ on the value function. Let the sequence of states to be updated at iteration k be $s_{k1}, s_{k2}, \cdots, s_{kn_k}$ and $\nu = \max_{k\in[1,\mathcal{K}],i\in[1,n_k]} \nu_{ki}$. For any iteration k and state s_{ki}, we have

$$\epsilon_{ki} \leq \nu\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}} = \epsilon.$$

So our optimization introduces error of at most ϵ to the optimization of the Bellman equation at any iteration. Furthermore, we can bound the difference between the value for the i-th state updated at iteration k of RTDP ($\hat{V}_{ki}$) and the corresponding value function updated by ADP ($\tilde{V}^*_{ki}$). More specifically, the following inequalities hold for any $\mathcal{K} = 1, 2, \cdots, \infty$:

$$\begin{aligned}
|\hat{V}_{11} - \tilde{V}^*_{11}| &\leq \epsilon,\\
|\hat{V}_{12} - \tilde{V}^*_{12}| &\leq \epsilon + \gamma\epsilon,\\
&\cdots,\\
|\hat{V}_{1n_1} - \tilde{V}^*_{1n_1}| &\leq \sum\nolimits_{i=1}^{n_1} \gamma^{i-1}\epsilon,\\
&\cdots,\\
&\cdots,\\
|\hat{V}_{\mathcal{K}1} - \tilde{V}^*_{\mathcal{K}1}| &\leq \epsilon + \gamma\sum\nolimits_{i=1}^{n_1+\cdots+n_{\mathcal{K}-1}} \gamma^{i-1}\epsilon,\\
|\hat{V}_{\mathcal{K}2} - \tilde{V}^*_{\mathcal{K}2}| &\leq \epsilon + \gamma\epsilon + \gamma^2\sum\nolimits_{i=1}^{n_1+\cdots+n_{\mathcal{K}-1}} \gamma^{i-1}\epsilon,\\
&\cdots,\\
|\hat{V}_{\mathcal{K}n_{\mathcal{K}}} - \tilde{V}^*_{\mathcal{K}n_{\mathcal{K}}}| &\leq \sum\nolimits_{i=1}^{n_1+\cdots+n_{\mathcal{K}}} \gamma^{i-1}\epsilon < \frac{\epsilon}{1-\gamma} = \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}.
\end{aligned}$$

Notice that $\hat{V}_{ki}$ converges because it monotonically decreases for any state index i and it is lower bounded by $\left(\frac{C}{1-\gamma^{\mathcal{T}}}\right)$ where $C < 0$ is the highest cost, e.g. colliding with obstacles. Since we use Cauchy's convergence test [33], we can set the threshold for the convergence test to be negligible. Hence for some $\mathcal{K} < \infty$, both $\hat{V}_k$ and $\tilde{V}_k$ converge according to Cauchy's convergence test, and we

have

$$|\hat{V}_{\mathcal{K}}(\cdot) - \tilde{V}^*(\cdot)|_{B_{\mathcal{K}}} \leq \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}.$$

Q.E.D.

Claim 5.4.1.3: $\tilde{V}^*$, the optimal value function of the approximated MDP $\tilde{\mathcal{M}}$, asymptotically converges to V^*, the optimal value function of the original problem defined by the MDP $\mathcal{M}$.

Proof of Claim 5.4.1.3: We consider the asymptotic case where the size of the dataset $|D| \to \infty$ and the number of states sampled $|\tilde{S}| \to \infty$. Notice that these two limit does not contradict Claim 1.1 because BOIDP operates in a loop and we can iteratively sample more states by calling SAMPLESTATES in Alg. 7. When $|D| \to \infty$, $p_{s'|s,a}$ converges to the true transition model.

Because the states are sampled uniformly randomly from the state space S in Line 8 of Alg. 9, when $|\tilde{S}| \to \infty$, the set $\tilde{S}$ can be viewed as uniform random samples from the reachable state space[3]. So the value function for the optimal policy of $\tilde{\mathcal{M}}$ asymptotically converges to that of $\mathcal{M}$:

$$\lim_{|\tilde{S}|,|D|\to\infty} |\tilde{V}^* - V^*|_\infty = 0.$$

Q.E.D.

Thm. 5.4.1 directly follows Claim 1.1, 1.2, and 1.3. By the triangle inequality of L_∞, we have

$$\begin{aligned}\lim_{|\tilde{S}|,|D|\to\infty} |\hat{V}_{\mathcal{K}} - V^*|_{B_{\mathcal{K}}} &\leq \lim_{|\tilde{S}|,|D|\to\infty} |\hat{V}_{\mathcal{K}} - \tilde{V}^*|_{B_{\mathcal{K}}} + \lim_{|\tilde{S}|,|D|\to\infty} |\tilde{V}^* - V^*|_{B_{\mathcal{K}}} \\ &\leq \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}\end{aligned}$$

holds in expectation. □

5.5 Implementation and Experiments

We tested our approach in a quasi-static problem, in which a robot pushes a circular object through a planar workspace with obstacles in simulation[4]. We represent the action by the robot's initial

[3]Alg. 9 does not necessarily lead to uniform samples of states in the state space. However, as the number of states sampled approaches infinity, $|\tilde{S}| \to \infty$, we can construct a set of finite and arbitrarily small open balls that cover the (reachable) state space such that there exists at least one sampled state in any of those balls. Such a cover exists because the state space is compact. If the samples are not uniform, we can simply adopt a uniform sampler on top of Alg. 9, and throw away a fixed proportion of states so that the remaining set of states are uniform samples from the state space S.

[4]All experiments were run with Python 2.7.6 on Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 64GB memory.

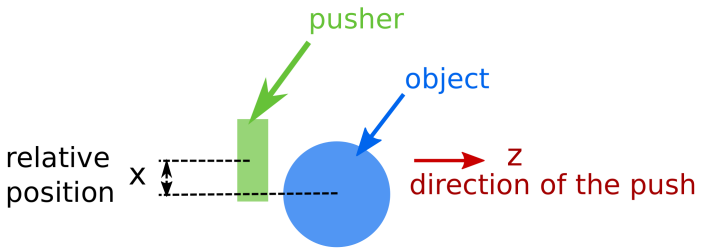


Figure 5-2: Pushing a circular object with a rectangle pusher.

relative position x to the object (its distance to the object center is fixed), the direction of the push z, and the duration of the push Δt, which are illustrated in Fig. 5-2. The companion video shows the behavior of this robot, controlled by a policy derived by BOIDP from a set of training examples.

In this problem, the basic underlying dynamics in free space with no obstacles are location invariant; that is, that the change in state Δs resulting from taking action $a = (u, \Delta t)$ is independent of the state s in which a was executed. We are given a training dataset $D = \{\Delta s_i, a_i\}_{i=0}^{N}$, where a_i is an action and Δs_i is the resulting state change, collected in the free space in a simulator. Given a new query for action a, we predict the distribution of Δs by looking at the subset $D' = \{\Delta s_j, a_j\}_{j=0}^{M} \subseteq D$ whose actions a_j are the most similar to a (in our experiments we use 1-norm distance to measure similarity), and fit a Gaussian mixture model on Δs_j using the EM algorithm, yielding an estimated continuous state-action transition model $p_{s'|s,a}(s+\Delta_s \mid s, a) = p_{\Delta s|a}(\Delta s \mid a)$. We use the Bayesian information criterion (BIC) to determine the number of mixture components.

5.5.1 Importance of learning accurate models

Our method was designed to be appropriate for use in systems whose dynamics are not well modeled with uni-modal Gaussian noise. The experiments in this section explore the question of whether a uni-modal model could work just as well, using a simple domain with known dynamics $s' = s + T(a)\rho$, where the relative position $x = 0$ and duration $\Delta t = 1$ are fixed, the action is the direction of motion, $a = z \in [0, 2\pi)$, $T(a)$ is the rotation matrix for angle, and the noise is

$$\rho \sim 0.6\mathcal{N}(\begin{bmatrix}5.0\\5.0\end{bmatrix}, \begin{bmatrix}2.0 & 0.0\\0.0 & 2.0\end{bmatrix}) + 0.4\mathcal{N}(\begin{bmatrix}5.0\\-5.0\end{bmatrix}, \begin{bmatrix}2.0 & 0.0\\0.0 & 2.0\end{bmatrix}).$$

We sample ρ from its true distribution and fit a Gaussian ($K = 1$) and a mixture of Gaussians ($K = 2$). The samples from $K = 1$ and $K = 2$ are shown in Fig. 5-3 (a). We plan with both models

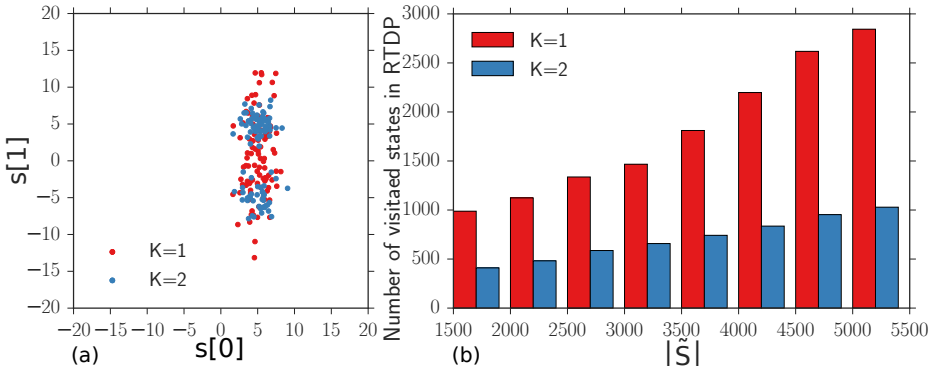


Figure 5-3: (a) Samples from the single-mode Gaussian transition model ($K = 1$) and the two-component Gaussian mixture transition model ($K = 2$) in the free space when $a = 0$. (b) The number of visited states (y-axis) increases with the number of sampled states $|\tilde{S}|$ (x-axis). Planning with $K = 2$ visits fewer states in RTDP than with $K = 1$.

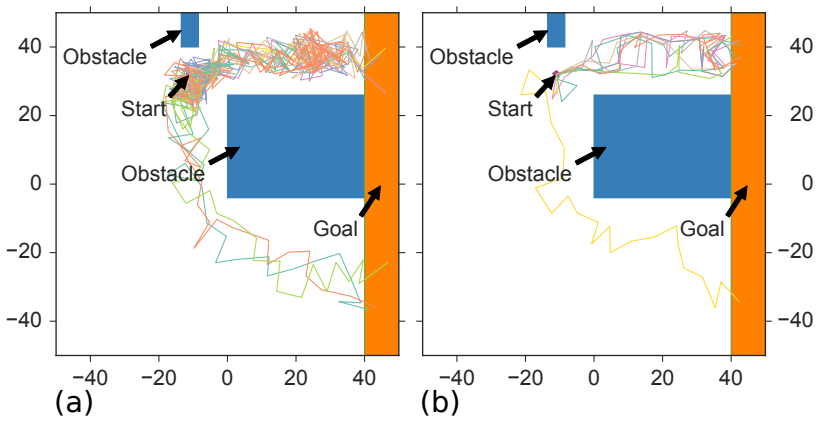


Figure 5-4: (a) Samples of 10 trajectories with $K = 1$. (b) Samples of 10 trajectories with $K = 2$. Using the correct number of components for the transition model improves the quality of the trajectories.

where each action has an instantaneous reward of -1, hitting an obstacle has a reward of -10, and the goal region has a reward of 100. The discount factor $\gamma = 0.99$. To show that the results are consistent, we use Algorithm 9 to sample 1500 to 5000 states to construct $\tilde{S}$, and plan with each of them using 100 uniformly discretized actions within 1000 iterations of RTDP.

To compute the Monte Carlo reward, we simulated 500 trajectories for each computed policy with the true model dynamics, and for each simulation, at most 500 steps are allowed. We show 10 samples of trajectories for both $K = 1$ and $K = 2$ with $|\tilde{S}| = 5000$, in Fig 5-4. Planning with the right model $K = 2$ tends to find better trajectories, while because $K = 1$ puts density on many states that the true model does not reach, the policy of $K = 1$ in Fig 5-4 (a) causes the robot to do extra maneuvers or even choose a longer trajectory to avoid obstacles that it actually has

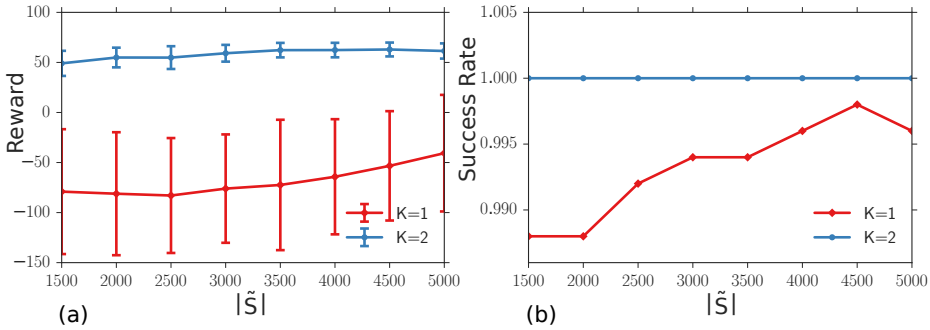


Figure 5-5: (a): Reward. (b): Success rate. Using two components ($K = 2$) performs much better than using one component ($K = 1$) in terms of reward and success rate.

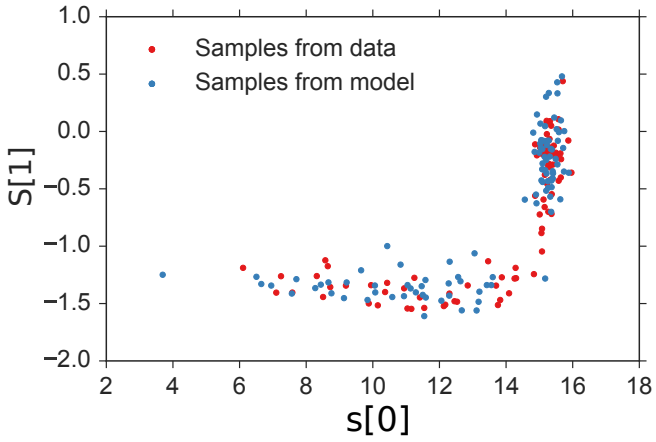


Figure 5-6: The conditional distribution of Δs given $a = (z, x, \Delta t) = (0.0, 0.3, 2.0)$ is a multi-modal Gaussian.

very low probability of hitting. As a result, the reward and success rate for $K = 2$ are both higher than $K = 1$, as shown in Fig. 5-5. Furthermore, because the single-mode Gaussian estimates the noise to have a large variance, it causes RTDP to visit many more states than necessary, as shown in Fig. 5-3 (b).

5.5.2 Focusing on the good actions and states

In this section we demonstrate the effectiveness of our strategies for limiting the number of states visited and actions modeled. We denote using Bayesian optimization in Lines 14 and 17 in Algorithm 10 as BO and using random selections as Rand.

We first demonstrate why BO is better than random for optimizing $Q_s(a)$ with the simple example from Sec. 5.5.1. We plot the $Q_s(a)$ in the first iteration of RTDP where $s = [-4.3, 33.8]$, and let random and BO in Algorithm 1 each pick 10 actions to evaluate sequentially as shown in

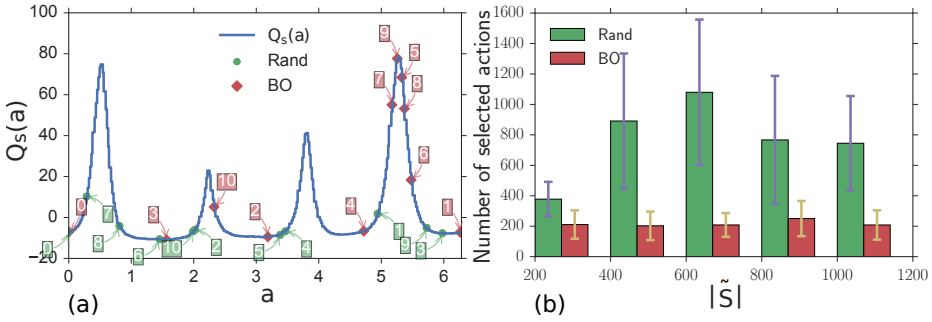


Figure 5-7: (a) We optimize $Q_s(a)$ with BO and Rand by sequentially sampling 10 actions. BO selects actions more strategically than Rand. (b) BO samples fewer actions than Rand in the pushing problem for all settings of $|\tilde{S}|$.

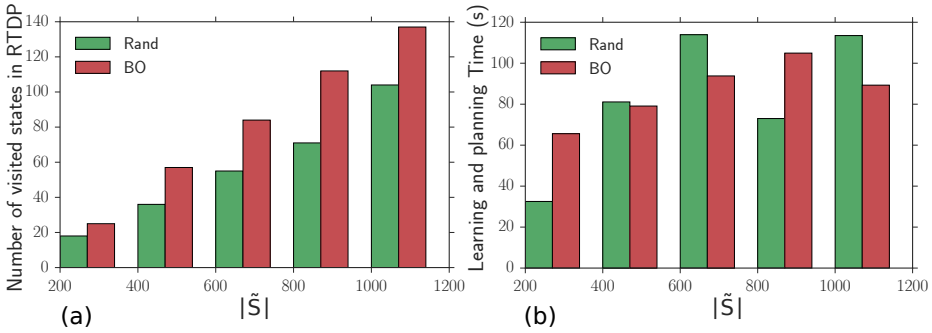


Figure 5-8: (a) Number of visited states in RTDP. Both of Rand and BO consistently focus on about 10% states for planning. (b) Learning and planning time of BO and Rand.

Fig 5-7 (a). We use the GP implementation and the default Matern52 kernel implemented in the GPy module [71] and optimize its kernel parameters every 5 selections. The first point for both BO and Rand is fixed to be $a = 0.0$. We observe that BO is able to focus its action selections in the high-value region, and BO is also able to explore informative actions if it has not found a good value or if it has finished exploiting a good region (see selection 10). Random action selection wastes choices on regions that have already been determined to be bad.

Next we consider a more complicated problem in which the action is the high level control of a pushing problem $a = (z, x, \Delta t)$, $z \in [0, 2\pi], x \in [-1.0, 1.0], \Delta t \in [0.0, 3.0]$ as illustrated in Fig. 5-2. The instantaneous reward is -1 for each free-space motion, -10 for hitting an obstacle, and 100 for reaching the goal; $\gamma = 0.99$. We collected 1.2×10^6 data points of the form $(a, \Delta s)$ with x and Δt as variables in the Box2D simulator [32] where noise comes from variability of the executed action. We make use of the fact that the object is cylindrical (with radius 1.0) to reuse data. An example of the distribution of Δs given $a = (0.0, 0.3, 2.0)$ is shown in Fig. 5-6.

We compare policies found by Rand and BO with the same set of sampled states ($|\tilde{S}| =$

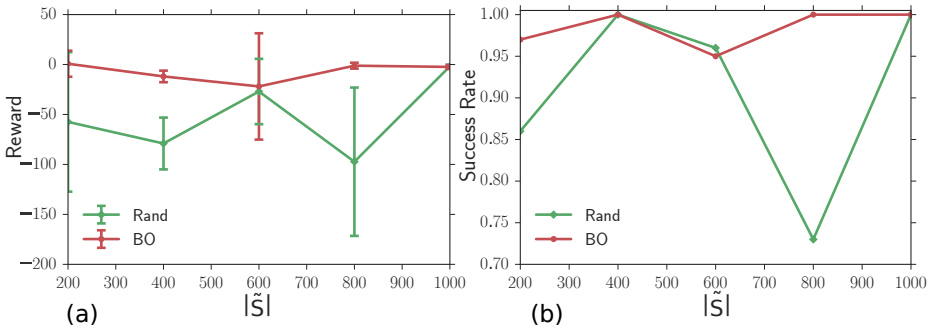


Figure 5-9: (a) Reward. (b) Success rate. BO achieves better reward and success rate, with many fewer actions and slightly more visited states.

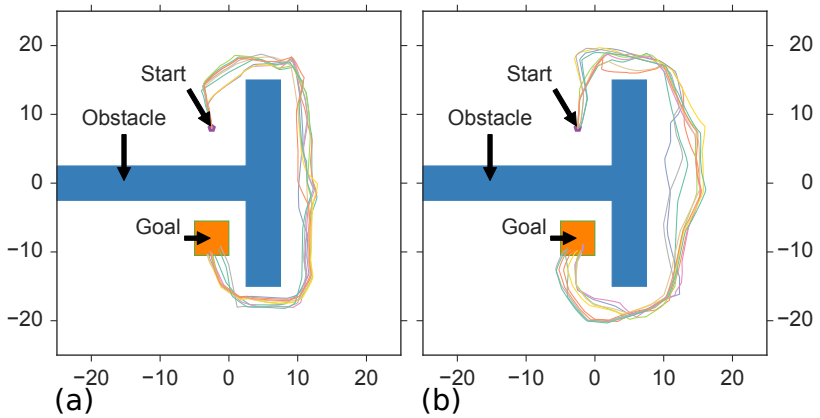


Figure 5-10: (a) 10 samples of trajectories generated via Rand with 1000 states. (b) 10 samples of trajectories generated via BO with 1000 states.

$200, 400, 600, 800, 1000$) within approximately the same amount of total computation time. They are both able to compute the policy in $30 \sim 120$ seconds, as shown in Fig. 5-8 (b). In more realistic domains, it is possible that learning the transition model will take longer and dominate the action-selection computation. We simulate 100 trajectories in the Box2D simulator for each planned policy with a maximum of 200 seconds. We show the result of the reward and success rate in Fig. 5-9, and the average number of actions selected for visited states in Fig. 5-7(b). In our simulations, BO consistently performs approximately the same or better than Rand in terms of reward and success rate while BO selects fewer actions than Rand. We show 10 simulated trajectories for Rand and BO with $|\tilde{S}| = 1000$ in Fig. 5-10.

From Fig. 5-8 (a), it is not hard to see that RTDP successfully controlled the number of visited states to be only a small fraction of the whole sampled set of states. Interestingly, BO was able to visit slightly more states with RTDP and as a result, explored more possible states that it is likely to encounter during the execution of the policy, which may be a factor that contributed to its better performance in terms of reward and success rate in Fig. 5-9. We did not compare with pure value iteration because the high computational cost of computing models for all the states made it infeasible.

BOIDP is able to compute models for only around 10% of the sampled states and about 200 actions per state. If we consider a naive grid discretization for both action (3 dimension) and state (2 dimension) with 100 cells for each dimension, the number of models we would have to compute is on the order of 10^{10}, compared to our approach, which requires only 10^4.

5.6 Conclusion

An important class of robotics problems are intrinsically continuous in both state and action space, and may demonstrate non-Gaussian stochasticity in their dynamics. We have provided a framework to plan and learn effectively for these problems. We achieve efficiency by focusing on relevant subsets of state and action spaces, while retaining guarantees of asymptotic optimality.

Part II

Active data acquisition with Bayesian optimization

Chapter 6

Bayesian Optimization Guided by Max-values

Data collection can be very expensive for robotics tasks in high-fidelity simulation or real world. In Part II of this thesis, we build the theoretical and algorithmic foundations on active data acquisition and we specifically focus on the zero-th order optimization problem using Bayesian optimization (BO). This problem is also called the blackbox function optimization problem, and the functions of interest typically have multiple peaks and are expensive to evaluate. In fact, this problem does not only appear in robot learning. The success of BO was witnessed by applications in many areas of science and engineering including robotics [30, 125, 188], chemistry [79], aerospace engineering [114] and machine learning [169, 179], among others.

BO techniques pose a prior on the unknown objective function, and the uncertainty given by the associated posterior is the basis for an acquisition function that guides the selection of the next point to query the function. The selection of query points and hence the acquisition function is critical for the success of the method. A vast amount of research has been devoted to the design of acquisition functions. The earliest work that we know of is by [111], who proposed to use the *probability of improvement* over a certain threshold as the acquisition function. However, the algorithm is very sensitive to the choice of threshold. Since then, more robust and theoretically justified acquisition functions appeared such as *expected improvement* [134] and *upper confidence bounds* [7, 171].

This chapter contains contents from the following two publications.

Zi Wang, Bolei Zhou, and Stefanie Jegelka. Optimization as estimation with Gaussian processes in bandit settings. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning (ICML)*, 2017.

Particularly successful recent additions are *entropy search* [75] and *predictive entropy search* [78] that aim to maximize the mutual information between the queried points and the location of the global optimum.

Despite the development of a variety of acquisition functions, we have very little understanding about the relations between them, and how to choose the "right one" in practice. In this chapter, we study these questions from the perspective of BO methods guided by the *optimal value* of the objective function. For consistency and without loss of generality, we assume throughout the chapter that the objective function is to be maximized. Hence we use max-value and optimal value interchangeably.

Our motivating intuition is that max-values may provide enough information about the location of the global maximizer. By analyzing acquisition functions that rely on max-values from both greedy and information-theoretic viewpoints, we reveal previously unknown connections among entropy search methods, upper confidence bounds and probability of improvement, bridged by max-value based approaches. Building upon these connections, we establish regret bounds for variants of *entropy search* and *probability of improvement* that are guided by max-values. Our empirical evaluations demonstrate that the BO methods guided by max-values identify good points as quickly or better than the state-of-the-art approaches, while retaining computational efficiency and robustness to unspecified hyper-parameters.

6.1 Background

Our goal is to maximize a black-box function $f : \mathfrak{X} \to \mathbb{R}$ where $\mathfrak{X} \subset \mathbb{R}^d$ and $\mathfrak{X}$ is compact. For each iteration t, we select an input $\boldsymbol{x}_t$ and observe a possibly noisy function evaluation $y_t = f(\boldsymbol{x}_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. Gaussian variables. The key idea of BO is to build a probabilistic model of function f using current observations $D_t = \{(\boldsymbol{x}_\tau, y_\tau)\}_{\tau=1}^t$ and define an acquisition function $\alpha_t : \mathfrak{X} \to \mathbb{R}$ based on the model. The probabilistic model is typically represented by a mean prediction $\mu_t(\cdot)$ and confidence level $\sigma_t(\cdot)$. The input $\boldsymbol{x}_t$ to evaluate is the maximizer of the acquisition function $\alpha_t(\cdot)$. A common paradigm of BO is given in Alg. 1

6.1.1 Bayesian models for functions

A Gaussian process (GP) [151] is a distribution over functions. In BO, GPs are typically used to build a probabilistic model of black-box function f if input dimension d is small (for example,

smaller than 10). For high dimensional cases, it was shown that modeling function f with a variant of additive Gaussian processes (add-GPs) [51, 91] could boost the performance of BO [190]. The methods introduced in this chapter can be extended to other function approximation approaches that predicts uncertainties; for example, Bayesian neural networks and their variants [63, 112]. In this chapter, we use GPs and add-GPs.

Gaussian processes In a GP, any finite set of function values has a multivariate Gaussian distribution. A Gaussian Process $GP(\mu, k)$ is fully specified by a mean function $\mu(\boldsymbol{x})$ and covariance (kernel) function $k(\boldsymbol{x}, \boldsymbol{x}')$.

Let f be a function sampled from a Gaussian process $GP(\mu, k)$. Given the observations $D_t = \{(\boldsymbol{x}_\tau, y_\tau)\}_{\tau=1}^t$, we obtain the posterior mean

$$\mu_t(\boldsymbol{x}) = \boldsymbol{k}_t(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{y}_t$$

and posterior covariance

$$k_t(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}_t(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{k}_t(\boldsymbol{x}')$$

of the function via the kernel matrix $\boldsymbol{K}_t = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{\boldsymbol{x}_i, \boldsymbol{x}_j \in D_t}$ and $\boldsymbol{k}_t(\boldsymbol{x}) = [k(\boldsymbol{x}_i, \boldsymbol{x})]_{\boldsymbol{x}_i \in D_t}$ [151]. The posterior variance is given by $\sigma_t^2(\boldsymbol{x}) = k_t(\boldsymbol{x}, \boldsymbol{x})$.

Additive Gaussian processes Additive Gaussian processes (add-GP) were proposed in [51], and first analyzed in the BO setting in [91]. Following the latter, we assume that the function f is a sum of independent functions sampled from Gaussian processes that are active on disjoint sets A_m of input dimensions. Precisely,

$$f(x) = \sum\nolimits_{m=1}^{M} f^{(m)}(x^{A_m})$$

, with $A_i \cap A_j = \emptyset$ for all $i \neq j$, $|\cup_{i=1}^M A_i| = d$, and $f^{(m)} \sim GP(\mu^{(m)}, k^{(m)})$, for all $m \leq M$ ($M \leq d < \infty$). As a result of this decomposition, the function f is distributed according to $GP(\sum_{m=1}^M \mu^{(m)}, \sum_{m=1}^M k^{(m)})$.

Given a set of noisy observations $D_t = \{(\boldsymbol{x}_\tau, y_\tau)\}_{\tau=1}^t$ where $y_\tau \sim \mathcal{N}(f(x_\tau), \sigma^2)$, the posterior mean and covariance of the function component $f^{(m)}$ can be inferred as

$$\mu_t^{(m)}(\boldsymbol{x}) = \boldsymbol{k}_t^{(m)}(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{y}_t$$

and

$$k_t^{(m)}(\boldsymbol{x}, \boldsymbol{x}') = k^{(m)}(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}_t^{(m)}(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{k}_t^{(m)}(\boldsymbol{x}'),$$

where $\boldsymbol{k}_t^{(m)}(\boldsymbol{x}) = [k^{(m)}(\boldsymbol{x}_i, \boldsymbol{x})]_{\boldsymbol{x}_i \in D_t}$ and $\boldsymbol{K}_t = \left[\sum_{m=1}^{M} k^{(m)}(\boldsymbol{x}_i, \boldsymbol{x}_j)\right]_{\boldsymbol{x}_i, \boldsymbol{x}_j \in D_t}$. For simplicity, we use the shorthand $k^{(m)}(\boldsymbol{x}, \boldsymbol{x}') = k^{(m)}(\boldsymbol{x}^{A_m}, \boldsymbol{x}'^{A_m})$.

Other notations Throughout the chapter, we use ψ to denote the probability density function, Ψ the cumulative density function of a normal distribution, $Q(\cdot) = 1 - \Psi(\cdot)$, and $\gamma_\theta^{(t)}(\boldsymbol{x}) = \frac{\theta - \mu_t(\boldsymbol{x})}{\sigma_t(\boldsymbol{x})}$, unless otherwise mentioned.

6.1.2 Acquisition functions

Active data acquisition in BO is guided by an acquisition function $\alpha_t : \mathfrak{X} \to \mathbb{R}$, which indicates how prominent a function input is. Usually the prominence of an input is high if it potentially has high output value or it is informative or both. The design of acquisition functions plays a key role in the performance of BO. In the following, we review historical landmarks of the development of acquisition functions.

Probability of improvement (PI) [111] introduced the first BO acquisition function, PI, that maximizes the probability of improving over a threshold θ:

$$\alpha_t^{\mathrm{PI}}(\boldsymbol{x}) = \Pr[f(\boldsymbol{x}) > \theta] = 1 - \Psi(\gamma_\theta^{(t)}(\boldsymbol{x})),$$

which is equivalent to selecting

$$\boldsymbol{x}_t = \arg\min_{\boldsymbol{x} \in \mathfrak{X}} \gamma_\theta^{(t)}(\boldsymbol{x}).$$

For each iteration, the threshold is set to be $\theta = \max_{\tau \in [1,t]} y_\tau + \eta_t$. In practice, PI is very sensitive to the choice of η_t and choosing η_t can be very difficult in practice [85].

Expected improvement (EI) [134] proposed an alternative acquisition function called EI that alleviated the problem of selecting the hyper-parameter η_t in PI. The EI criterion selects the input maximizing the expected improvement over the best observation $\theta = \max_{\tau \in [1,t]} y_\tau$; namely,

$$\alpha_t^{\mathrm{EI}}(\boldsymbol{x}) = \mathbb{E}[(f(\boldsymbol{x}) - \theta)_+].$$

For GPs, this improvement is given in closed form as

$$\alpha_t^{\mathrm{EI}}(\boldsymbol{x}) = \left[\psi(\gamma_\theta^{(t)}(\boldsymbol{x})) - \gamma_\theta^{(t)}(\boldsymbol{x})Q(\gamma_\theta^{(t)}(\boldsymbol{x}))\right]\sigma_t(\boldsymbol{x}).$$

[29] showed that EI may not converge to the global optimum, but a more sophisticated ϵ-greedy version of EI can converge to near-optimality.

Upper confidence bound (GP-UCB) The idea of using upper confidence bounds for bandit problems was introduced by [8]. The upper confidence bound on the quality of each candidate is calculated, and the candidate with the highest upper confidence bound is evaluated. [171] provided a detailed analysis for using upper confidence bounds with GP bandits. They propose the upper confidence bound for $\boldsymbol{x}$ to be

$$\alpha_t^{\mathrm{GP-UCB}}(\boldsymbol{x}) = \mu_t(\boldsymbol{x}) + \lambda_t\sigma_t(\boldsymbol{x}),$$

where $\lambda_t = \left(2\log(|\mathfrak{X}|\pi^2t^2/(6\delta))\right)^{\frac{1}{2}}$ if $\mathfrak{X}$ is discrete[1]. However, [171] also admitted that the theoretically derived value of λ_t may be suboptimal for practical performance.

Entropy search (ES) and predictive entropy search (PES) Entropy search methods use an information-theoretic perspective to select where to evaluate. They find a query point that maximizes the information about the location $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}\in\mathfrak{X}} f(x)$ whose value $y^* = f(\boldsymbol{x}^*)$ achieves the global maximum of the function f. Using the negative differential entropy of $p(\boldsymbol{x}^*|D_t)$ to characterize the uncertainty about $\boldsymbol{x}^*$, ES [75] and PES [78] use the acquisition functions

$$\begin{aligned}
\alpha_t^{\mathrm{ES/PES}}(\boldsymbol{x}) &= I(\{\boldsymbol{x}, y\}; \boldsymbol{x}^* \mid D_t) && (6.1)\\
&= H\left(p(\boldsymbol{x}^* \mid D_t)\right) - \mathbb{E}\left[H(p(\boldsymbol{x}^* \mid D_t \cup \{\boldsymbol{x}, y\}))\right] && (6.2)\\
&= H(p(y \mid D_t, \boldsymbol{x})) - \mathbb{E}\left[H(p(y \mid D_t, \boldsymbol{x}, \boldsymbol{x}^*))\right]. && (6.3)
\end{aligned}$$

ES uses formulation (6.2), in which the expectation is over $p(y|D_t, \boldsymbol{x})$, while PES uses the equivalent, symmetric formulation (6.3), where the expectation is over $p(\boldsymbol{x}^*|D_t)$. Unfortunately, both $p(\boldsymbol{x}^*|D_t)$ and its entropy is analytically intractable and have to be approximated via expensive computations. Moreover, the optimum may not be unique, adding further complexity to this distribution.

[1] Please refer to [171] for continuous $\mathfrak{X}$.

6.1.3 Evaluation Criteria

We study two types of the evaluation criteria for BO, *best-sample simple regret* and *inference regret*. In each iteration, we choose to evaluate one input $\boldsymbol{x}_t$ to "learn" where the $\arg\max$ of the function is. The best-sample simple regret $r_T = \max_{\boldsymbol{x}\in\mathfrak{X}} f(\boldsymbol{x}) - \max_{t\in[1,T]} f(\boldsymbol{x}_t)$ measures the value of the best queried point. After all queries, we may infer an $\arg\max$ of the function, which is usually chosen as $\tilde{\boldsymbol{x}}_T = \arg\max_{\boldsymbol{x}\in\mathfrak{X}} \mu_T(\boldsymbol{x})$ [75, 78]. We denote the inference regret as $R_T = \max_{\boldsymbol{x}\in\mathfrak{X}} f(\boldsymbol{x}) - f(\tilde{\boldsymbol{x}}_T)$ which characterizes how satisfying our inference of the $\arg\max$ is.

6.2 Acquisition functions based on max-values

We study a group of acquisition functions that rely on the *maximum value* $y^* = f(\boldsymbol{x}^*)$. These acquisition functions are derived from the intuition that the max-value y^* may provide enough information to guide data acquisition. One simple idea (Sec. 6.2.1) is to greedily evaluate inputs that are most likely to achieve the max-value y^*, while the other acquisition function introduced in Sec. 6.2.2 makes use of the max-value y^* by analyzing information gain. We will see in Sec. 6.3 that, despite their different viewpoints, these two approaches are closely connected.

6.2.1 Optimization as argmax estimation (EST)

Consider the probability of event $M_{\boldsymbol{x}}$: a fixed $\boldsymbol{x} \in \mathfrak{X}$ is an argmax of f. The event $M_{\boldsymbol{x}}$ is equivalent to the event that for all $\boldsymbol{x}' \in \mathfrak{X}$, we have $v_{\boldsymbol{x}}(\boldsymbol{x}') := f(\boldsymbol{x}') - f(\boldsymbol{x}) \leq 0$. Given observations $D_t = \{(\boldsymbol{x}_\tau, y_\tau)\}_{\tau=1}^t$, the difference $v_{\boldsymbol{x}}(\boldsymbol{x}')$ between two Gaussian variables is Gaussian:

$$v_{\boldsymbol{x}}(\boldsymbol{x}') \sim \mathcal{N}(\mu_t(\boldsymbol{x}') - \mu_t(\boldsymbol{x}), \sigma_t(\boldsymbol{x})^2 + \sigma_t(\boldsymbol{x}')^2 - 2k_t(\boldsymbol{x}, \boldsymbol{x}')).$$

The covariance for any $\boldsymbol{x}', \boldsymbol{x}'' \in \mathfrak{X}$ is

$$\mathrm{Cov}(v_{\boldsymbol{x}}(\boldsymbol{x}'), v_{\boldsymbol{x}}(\boldsymbol{x}'')) = \sigma_t(\boldsymbol{x})^2 + k_t(\boldsymbol{x}', \boldsymbol{x}'') - k_t(\boldsymbol{x}, \boldsymbol{x}') - k_t(\boldsymbol{x}, \boldsymbol{x}'').$$

The random variables $\{v_{\boldsymbol{x}}(\boldsymbol{x}')\}_{\boldsymbol{x}'\in\mathfrak{X}}$ determine the cumulative probability

$$\Pr[M_{\boldsymbol{x}}|D_t] = \Pr[\forall \boldsymbol{x}' \in \mathfrak{X}, v(\boldsymbol{x}') \leq 0|D_t]. \tag{6.4}$$

This probability may be specified via limits as e.g. in [75, App.A]. Moreover, due to the assumed smoothness of f, it is reasonable to work with a discrete approximation and restrict the set of candidate points to be finite for now. So the quantity in Eqn. (6.4) is well-defined. However, the computation of Eq. 6.4 requires integrating over a $|\mathfrak{X}|$-dimensional multivariate Gaussian. The complexity of a numerical integration approach using Fubini's theorem grows exponentially as $|\mathfrak{X}|$ increases.

Suppose the max-value of the function f is given as $y^* \triangleq f(\boldsymbol{x}^*) = \theta$, can we estimate the probability of $M_{\boldsymbol{x}}$ more efficiently? The answer is yes. We approximate $\{f(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathfrak{X}}$ by independent Gaussian random variables with means $\mu_t(\boldsymbol{x})$ and variances $\sigma_t(\boldsymbol{x})^2$ for all $\boldsymbol{x} \in \mathfrak{X}$. Our first acquisition function, EST, is the probability of event $M_{\boldsymbol{x}}|y^*, D_t$:

$$\alpha_t^{\mathrm{EST}}(\boldsymbol{x}) = \Pr[M_{\boldsymbol{x}}|y^* = \theta, D_t] \approx Q\Big(\gamma_\theta^{(t)}(\boldsymbol{x})\Big) \prod\nolimits_{\boldsymbol{x}'\neq\boldsymbol{x}} \Psi\Big(\gamma_\theta^{(t)}(\boldsymbol{x}')\Big),$$

which is the probability that $f(\boldsymbol{x}) \geq \theta$ and $f(\boldsymbol{x}') < \theta$. Intuitively, our estimation strategy EST chooses to evaluate the function input that is approximately most likely to achieve the highest function value $y^* = \theta$. Evaluating $\Pr[M_{\boldsymbol{x}}|y^* = \theta, D_t]$ takes only $O(|\mathfrak{X}|)$ time[2], and as we will see in Sec. 6.3, the complexity of evaluating this acquisition function can be reduced to $O(1)$ time.

It is important to point out that similar ideas on greedily evaluating the input that achieves best value have inspired other existing acquisition functions. For example, Jones [85] presented "one-stage approach for goal seeking" which assumes that the max-value of the function f is known. This approach (or "Method 6" refered in [85]) uses the likelihood of max-value y^* under the predictive distribution as the acquisition function: $p(f(\boldsymbol{x}) = y^* \mid D_t)$. In essence, Method 6 of [85] is also trying to compute the probability of event $M_{\boldsymbol{x}}|y^*, D_t$. The approach we showed above, however, uses a different approximation for the event M_x. Intuitively, our approximation can be more stable because it also considers that other inputs have values less than max-value y^*. If one uses the probability density function (PDF) $p(f(\boldsymbol{x}) = y^* \mid D_t)$ and have an input $\boldsymbol{x}$ that has been estimated to have higher value than y^*, Method 6 of [85] will unlikely choose to evaluate $\boldsymbol{x}$, because the evaluation $f(\boldsymbol{x})$ could have a very low PDF for y^*.

We will show both theoretically and empirically that this max-value parameter θ does not have to be exactly the (often unknown) max-value of the function f in order for EST to perform well. The theoretical justification in Sec. 6.4.1 allows θ to be an upper bound on the max-value and removed

[2]This is excluding the time complexity of Gaussian process posterior inference.

the assumption that $\mathfrak{X}$ is a discrete set. The tightness of the bound θ determines the magnitude of the expected regret (Sec. 6.4.3). Moreover, using an upper bound on the max-value also yields good experimental results, as shown in Sec. 6.6.4 and 6.6.5.

6.2.2 Max-value entropy search (MES)

The above observations suggest that the max-value y^* plays a critical role in deciding where the $\arg\max$ of a function could be. If the max-value y^* is unknown, we can adopt the same information-theoretic idea as entropy search to gather information about the max-value y^*. That means, instead of measuring the information about the maximizing argument $\boldsymbol{x}^*$, we compute the information about the *maximum value* $y^* = f(\boldsymbol{x}^*)$. Our second acquisition function is the gain in mutual information between the max-value y^* and the next point we query, which can be approximated analytically by evaluating the entropy of the predictive distribution:

$$\alpha_t^{\mathrm{MES}}(\boldsymbol{x}) = I(\{\boldsymbol{x}, y\}; y^* \mid D_t) \tag{6.5}$$

$$= H(p(y \mid D_t, \boldsymbol{x})) - \mathbb{E}[H(p(y \mid D_t, \boldsymbol{x}, y^*))] \tag{6.6}$$

$$\approx \frac{1}{K} \sum\nolimits_{\hat{y}^* \in Y_t^*} \left[\frac{\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}) \psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))}{2\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))} - \log(\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))) \right]. \tag{6.7}$$

Recall that $\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}) = \frac{\hat{y}^* - \mu_t(\boldsymbol{x})}{\sigma_t(\boldsymbol{x})}$ (Sec. 6.1.1). The expectation in Eq. (6.6) is over $p(y^*|D_n)$, which is approximated using Monte Carlo estimation by sampling a set of K function maxima. Notice that the probability in the first term $p(y|D_t, \boldsymbol{x})$ is a Gaussian distribution with mean $\mu_t(\boldsymbol{x})$ and variance $k_t(\boldsymbol{x}, \boldsymbol{x})$. The probability in the second term $p(y|D_n, \boldsymbol{x}, y^*)$ is a truncated Gaussian distribution: given y^*, the distribution of y needs to satisfy $y \leq y^*$. Importantly, while ES and PES rely on the expensive, d-dimensional distribution $p(\boldsymbol{x}^*|D_t)$, here, we use the one-dimensional $p(y^*|D_n)$, which is computationally much easier.

It remains to determine how to sample the max-value y^* in MES. We propose two approaches to sample y^*: (a) sampling from an approximation via a Gumbel distribution; and (b) sampling functions from the posterior Gaussian distribution and maximizing the functions to obtain samples of y^*. We present the MES algorithm in Alg. 13.

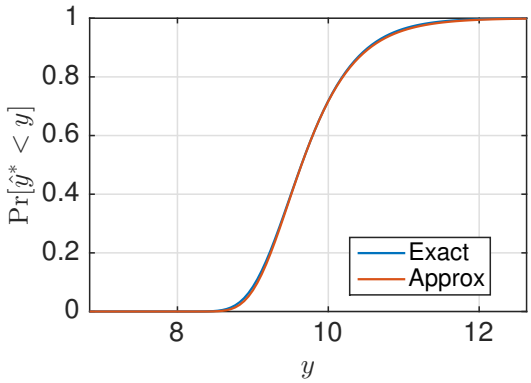


Figure 6-1: An example of approximating the cumulative probability of the maximum of independent differently distributed Gaussians $\widehat{\Pr}[y^* < y]$ (Exact) with a Gumbel distribution $\mathcal{G}(a, b)$ (Approx) via percentile matching.

Gumbel sampling

The marginal distribution of $f(\boldsymbol{x})$ for any $\boldsymbol{x}$ is a one-dimensional Gaussian, and hence the distribution of y^* may be viewed as the maximum of an infinite collection of dependent Gaussian random variables. Since this distribution is difficult to compute, we make two simplifications. First, we replace the continuous set $\mathfrak{X}$ by a discrete (finite), dense subset $\hat{\mathfrak{X}}$ of representative points. If we select $\hat{\mathfrak{X}}$ to be an ϵ-cover of $\mathfrak{X}$ and the function f is Lipschitz continuous with constant L, then we obtain a valid upper bound on $f(\mathfrak{X})$ by adding ϵL to any upper bound on $f(\hat{\mathfrak{X}})$.

Second, we use a "mean field" approximation and treat the function values at the points in $\hat{\mathfrak{X}}$ as independent. If posterior covariance $k_t(\boldsymbol{x}, \boldsymbol{x}') \geq 0$, this "mean field" approximation tends to over-estimate the maximum by Slepian's lemma [165, 131].

We sample from the approximation $\hat{p}(y^*|D_n)$ via its cumulative distribution function (CDF) $\widehat{\Pr}[y^* < z \mid D_t] = \prod_{\boldsymbol{x} \in \hat{\mathfrak{X}}} \Psi(\gamma_z^{(t)}(\boldsymbol{x}))$. That means we sample r uniformly from $[0, 1]$ and find z such that $\Pr[y^* < z \mid D_t] = r$. A binary search for z to accuracy δ requires $O(\log \frac{1}{\delta})$ queries to the CDF, and each query takes $O(|\hat{\mathfrak{X}}|) \approx O(n^d)$ time, so we obtain an overall time of $O(M|\hat{\mathfrak{X}}| \log \frac{1}{\delta})$ for drawing M samples.

To sample more efficiently, we propose a $O(M + |\hat{\mathfrak{X}}| \log \frac{1}{\delta})$-time strategy, by approximating the CDF by a Gumbel distribution: $\widehat{\Pr}[y^* < z \mid D_t] \approx \mathcal{G}(a, b) = e^{-e^{-\frac{z-a}{b}}}$. This choice is motivated by the Fisher-Tippett-Gnedenko theorem [59], which states that the maximum of a set of i.i.d. Gaussian variables is asymptotically described by a Gumbel distribution. This does not in general extend to non-i.i.d. Gaussian variables, but we nevertheless observe that in practice, this approach yields a good and fast approximation. Figure 6-1 shows an example of the approximation for the distribution

Algorithm 13 Max-value Entropy Search (MES)

function MES (f, K, D_0)
  for $t = 1, \cdots, T$ **do**
    $\mu_{t-1}(\cdot), \sigma_{t-1}(\cdot) \leftarrow$ MODEL(D_{t-1})
    $Y_t^* \leftarrow$SAMPLEMAXVALUE (K, D_{t-1})
    $\alpha_{t-1}(\cdot) \leftarrow$ Eq. (6.7)
    $\boldsymbol{x}_t \leftarrow \arg\max_{\boldsymbol{x} \in \mathfrak{X}} \alpha_{t-1}(\boldsymbol{x})$
    $y_t \leftarrow f(\boldsymbol{x}_t) + \epsilon_t, \epsilon_t \sim \mathcal{N}(0, \sigma^2)$
    $D_t \leftarrow D_{t-1} \cup \{\boldsymbol{x}_t, y_t\}$
  end for
end function

function SAMPLEMAXVALUE (K, D_t)
  if Sample with Gumbel **then**
    approximate $\Pr[y^* < z \mid D_t]$ with $\mathcal{G}(a, b)$
    sample a K-length vector $\boldsymbol{r} \sim$ Unif$([0, 1])$
    $Y_t^* \leftarrow a - b \log(-\log \boldsymbol{r})$
  else
    for $i = 1, \cdots, K$ **do**
      sample $\tilde{f} \sim GP(\mu_t, k_t \mid D_t)$
      $\hat{y}_{(i)}^* \leftarrow \max_{\boldsymbol{x} \in \mathfrak{X}} \tilde{f}(\boldsymbol{x})$
    end for
    $Y_t^* \leftarrow [\hat{y}_{(i)}^*]_{i=1}^K$
  end if
  return Y_t^*
end function

of the maximum of a 1-D function $f_t \sim GP(\mu_t, k_t)$; 50 observed data points are randomly selected from a function sampled from a GP with mean zero and Gaussian kernel.

We sample from the Gumbel distribution via the Gumbel quantile function: we sample r uniformly from $[0, 1]$, and let the sample be $y = \mathcal{G}^{-1}(a, b) = a - b\log(-\log r)$. We set the appropriate Gumbel distribution parameters a and b by solving the two-variable linear equations $a - b\log(-\log r_1) = y_1$ and $a - b\log(-\log r_2) = y_2$, where $\Pr[y^* < y_1] = r_1$ and $\Pr[y^* < y_2] = r_2$. In practice, we use $r_1 = 0.25$ and $r_2 = 0.75$ so that the scale of the approximated Gumbel distribution is proportional to the interquartile range of the CDF $\hat{\Pr}[y^* < z]$.

Optimizing posterior functions

For an alternative sampling strategy we follow [78]: we draw functions from the posterior GP and then maximize each of the sampled functions. Given the observations $D_t = \{(\boldsymbol{x}_\tau, y_\tau)_{\tau=1}^t\}$, we can approximate the posterior Gaussian process using a 1-hidden-layer neural network $\tilde{f}(\boldsymbol{x}) = \boldsymbol{a}_t^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^D$ is a vector of feature functions [139, 150] and the Gaussian weight $\boldsymbol{a}_t \in \mathbb{R}^D$ is distributed according to a multivariate Gaussian $\mathcal{N}(\boldsymbol{\nu}_t, \boldsymbol{\Sigma}_t)$.

Computing $\boldsymbol{\phi}(\boldsymbol{x})$. By Bochner's theorem [155], the Fourier transform $\hat{k}$ of a continuous and translation-invariant kernel k is guaranteed to be a probability distribution. Hence we can write the kernel of the GP to be $k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\omega\sim\hat{k}(\omega)}[e^{i\omega^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}')}] = \mathbb{E}_{c\sim U[0,2\pi]}\mathbb{E}_{\hat{k}}[2\cos(\omega^{\mathrm{T}}\boldsymbol{x} + c)\cos(\omega^{\mathrm{T}}\boldsymbol{x}' + c)]$ and approximate the expectation by $k(\boldsymbol{x}, \boldsymbol{x}') \approx \boldsymbol{\phi}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}')$ where $\phi_i(\boldsymbol{x}) = \sqrt{\frac{2}{D}}\cos(\omega_i^{\mathrm{T}}\boldsymbol{x} + c_i)$, $\omega_i \sim \hat{\kappa}(\omega)$, and $c_i \sim U[0, 2\pi]$ for $i = 1, \ldots, D$.

Computing $\boldsymbol{\nu}_t, \boldsymbol{\Sigma}_t$. By writing the GP as a random linear combination of feature functions $\boldsymbol{a}_t^T\boldsymbol{\phi}(\boldsymbol{x})$, we are defining the mean and covariance of the GP to be $\mu_t(\boldsymbol{x}) = \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ and $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Sigma}_t\boldsymbol{\phi}(\boldsymbol{x}')$. Let $Z = [z_1, \cdots, z_t] \in \mathbb{R}^{D\times t}$, where $z_\tau \coloneqq \boldsymbol{\phi}(\boldsymbol{x}_\tau) \in \mathbb{R}^D$. The GP posterior mean and covariance in Section 6.1.1 become $\mu_t(\boldsymbol{x}) = z^{\mathrm{T}}Z(Z^{\mathrm{T}}Z + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}_t$ and $k_t(\boldsymbol{x}, \boldsymbol{x}') = z^{\mathrm{T}}z' - z^{\mathrm{T}}Z(Z^{\mathrm{T}}Z + \sigma^2\boldsymbol{I})^{-1}Z^{\mathrm{T}}z'$. Because $Z(Z^{\mathrm{T}}Z + \sigma^2\boldsymbol{I})^{-1} = (ZZ^{\mathrm{T}} + \sigma^2\boldsymbol{I})^{-1}Z$, we can simplify the above equations and obtain $\boldsymbol{\nu}_t = \sigma^{-2}\boldsymbol{\Sigma}_t Z_t\boldsymbol{y}_t$ and $\boldsymbol{\Sigma}_t = (ZZ^{\mathrm{T}}\sigma^{-2} + \boldsymbol{I})^{-1}$.

To sample a function from this random 1-hidden-layer neural network, we sample $\tilde{\boldsymbol{a}}$ from $\mathcal{N}(\boldsymbol{\nu}_t, \boldsymbol{\Sigma}_t)$ and construct the sampled function $\tilde{f} = \tilde{\boldsymbol{a}}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$. Then we optimize $\tilde{f}$ with respect to its input to get a sample of the maximum of the function $\max_{\boldsymbol{x}\in\mathfrak{X}} \tilde{f}(\boldsymbol{x})$.

Optimizing by iterative sampling

Numerical gradient-based optimization methods typically all adopt an iterative strategy and require a sequence of function evaluations and corresonding gradient values of the form $[f(x_i), f'(x_i)]_{i=0}^{K}$ where K is the number of steps that the optimizer takes. Using this property, we can derive an iterative sampling approach to obtain a max-value sample of a GP.

More specifically, we start with the posterior GP $GP(\mu_t, k_t)$ conditioned on the current observations D_t at the t-th iteration of BO. At step 0 of the optimization process for the max-value, We can sample the function and gradient values $[f(\boldsymbol{x}_0), f'(\boldsymbol{x}_0)]$ from the joint distribution over function and gradient values, i.e.

$$\begin{bmatrix} f(\boldsymbol{x}_0) \\ f'(\boldsymbol{x}_0) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_0) \\ \mu'(\boldsymbol{x}_0) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_0, \boldsymbol{x}_0) & \frac{\mathrm{d}k(\boldsymbol{x}_0, \boldsymbol{x}_0)}{\mathrm{d}\boldsymbol{x}_0} \\ (\frac{\mathrm{d}k(\boldsymbol{x}_0, \boldsymbol{x}_0)}{\mathrm{d}\boldsymbol{x}_0})^T & \frac{\mathrm{d}^2 k(\boldsymbol{x}_0, \boldsymbol{x}_0)}{\mathrm{d}\boldsymbol{x}_0^2} \end{bmatrix} \right).$$

Here $\boldsymbol{x}_0$ is an initial guess and the optimization process can be restarted using different $\boldsymbol{x}_0$. The sampled $[f(\boldsymbol{x}_0), f'(\boldsymbol{x}_0)]$ becomes our pseudo-observations. At step i of the optimization, We sample from a GP conditioned on both D_t and all the previous pseudo-observations $[f(\boldsymbol{x}_j), f'(\boldsymbol{x}_j)]_{j=i-1}^{K}$ to obtain $[f(x_i), f'(x_i)]$. Each x_i is chosen by the numerical optimizer.

Once the optimizer converges, we will obtain a sample of the max-value. In the next iteration of BO, we can discard all the pseudo-observations from the last iteration.

If we use K steps in this approach, for iteration t of BO, the optimization for sampling a max-value has a time complexity of $O((t + D)^3 K^4)$. This approach is an exact sampling method to obtain a max-value if the underlying function we are optimizing is concave. However, a function sampled from a GP can often be multi-peak, which makes this approach very sensitive to the initial guess $\boldsymbol{x}_0$. We did not implement this method or compare with it in the experiment, but encourage interested readers to try.

6.3 Connections among acquisition functions

In Sec. 6.2, we introduced two acquisition functions, EST and MES, to enable max-values to guide BO. In particular, MES resembles ES and PES from the information-theoretic viewpoint, and extends the entropy search family of methods by using the entropy of the maximum value y^*. Next, we show that the max-value viewpoint also leads to new connections between known acquisition functions in the literature. In particular, EST is equivalent to a variant of MES, and they connect

entropy search approaches with GP-UCB and PI. Note that the equivalence between GP-UCB and PI was pointed out in [85], which also included part of the proof. We show a complete proof of the relations among the methods in this section.

Lemma 6.3.1. *The following methods are equivalent:*

1. *MES with $K = 1$ and a single max-value sample $\hat{y}^*$ for $\alpha_t^{\mathrm{MES}}(\boldsymbol{x})$;*
2. *EST with max-value parameter $\theta = \hat{y}^*$;*
3. *GP-UCB with parameter $\lambda_t = \min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})$, where $\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}) = \frac{\hat{y}^* - \mu_t(\boldsymbol{x})}{\sigma_t(\boldsymbol{x})}$;*
4. *PI with threshold parameter $\theta = \hat{y}^*$.*

Proof. We first prove the equivalence between 2 and 4. Let $\boldsymbol{b}$ be the input selected by EST. Without loss of generality, we assume $\boldsymbol{b}$ is unique. For all $\boldsymbol{x} \in \mathfrak{X}$ and $t = 0, \cdots, T-1$, we have

$$\frac{\Pr[M_{\boldsymbol{b}}|y^* = \hat{y}^*, D_t]}{\Pr[M_{\boldsymbol{x}}|y^* = \hat{y}^*, D_t]} \propto \frac{Q(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b})) \prod_{\boldsymbol{x}'\neq\boldsymbol{b}} \Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}'))}{Q(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})) \prod_{\boldsymbol{x}'\neq\boldsymbol{x}} \Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}'))} = \frac{Q(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b}))\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))}{Q(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b}))} \geq 1,$$

and so, $\frac{\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b}))}{1-\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b}))} \leq \frac{\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))}{1-\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))}$. This inequality holds if and only if $\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b}) \leq \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathfrak{X}$ because Ψ monotonically increases and $0 < \Psi(\gamma) < 1$ for any $-\infty < \gamma < \infty$. Hence, EST chooses $\boldsymbol{b} = \arg\min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})$. On the other hand, PI chooses $\arg\min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{\theta}^{(t)}(\boldsymbol{x})$, where $\theta = \hat{y}^*$. Hence EST and PI are equivalent under the parameterization specified in the lemma.

We next prove the equivalence between 2 and 3. Assume $\boldsymbol{a}$ is the unique input selected by GP-UCB. By the definition of λ_t, we have $\lambda_t = \min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}) = \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{b}) \leq \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{a})$. This implies that $\mu_t(\boldsymbol{b}) + \lambda_t\sigma_t(\boldsymbol{b}) = \hat{y}^* \geq \mu_t(\boldsymbol{a}) + \lambda_t\sigma_t(\boldsymbol{a}) \geq \mu_t(\boldsymbol{b}) + \lambda_t\sigma_t(\boldsymbol{b})$, because by the input selection rule of GP-UCB, we also have $\mu_t(\boldsymbol{a}) + \lambda_t\sigma_t(\boldsymbol{a}) = \max_{\boldsymbol{x}\in\mathfrak{X}} \mu_t(\boldsymbol{x}) + \lambda_t\sigma_t(\boldsymbol{x})$. Hence, by uniqueness of $\boldsymbol{a}$ and $\boldsymbol{b}$, we have $\boldsymbol{a} = \boldsymbol{b}$. So GP-UCB and EST select the same input.

What remains to be shown is the equivalence between 1 and 2. When using a single $\hat{y}^*$ in MES, the next point to evaluate is chosen by maximizing $\alpha_t(\boldsymbol{x}) = \gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})\frac{\psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))}{2\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))} - \log(\Psi(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})))$. For EST with $\theta = \hat{y}^*$, the next point to evaluate is chosen by minimizing $\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})$. Let us define a function $g(u) = u\frac{\psi(u)}{2\Psi(u)} - \log(\Psi(u)), u \in \mathbb{R}$. Clearly, $\alpha_t(\boldsymbol{x}) = g(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))$. Because $g(u)$ is a monotonically decreasing function, maximizing $g(\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x}))$ is equivalent to minimizing $\gamma_{\hat{y}^*}^{(t)}(\boldsymbol{x})$. Hence 1 and 2 are equivalent. □

Notice that, despite the fact that GP-UCB typically uses $\lambda_t > 0$, the proof of Lemma 6.3.1 does not depend on the sign of λ_t. Corollary 6.3.2 shows that, no matter what value is used for

λ_t in GP-UCB, there is a corresponding parameterization for PI and other methods so that they are equivalent to GP-UCB.

Corollary 6.3.2. *The following methods are equivalent:*

1. *MES with $K = 1$ and a single sample $\hat{y}^* = \max_{\boldsymbol{x}\in\mathfrak{X}} \mu_t(\boldsymbol{x}) + \lambda_t\sigma_t(\boldsymbol{x})$ for $\alpha_t^{\mathrm{MES}}(\boldsymbol{x})$;*
2. *EST with max-value parameter $\theta = \max_{\boldsymbol{x}\in\mathfrak{X}} \mu_t(\boldsymbol{x}) + \lambda_t\sigma_t(\boldsymbol{x})$;*
3. *GP-UCB with parameter $\lambda_t \in \mathbb{R}$;*
4. *PI with threshold parameter $\theta = \max_{\boldsymbol{x}\in\mathfrak{X}} \mu_t(\boldsymbol{x}) + \lambda_t\sigma_t(\boldsymbol{x})$.*

Proof. The equivalence of 1,2 and 4 directly follows Lemma 6.3.1. So we only need to show the equivalence between 2 and 3. Assume $\boldsymbol{a}$ is the unique input selected by GP-UCB and $\boldsymbol{b}$ is the unique input selected by EST. By definition, $\theta = \max_{\boldsymbol{x}\in\mathfrak{X}} \mu_t(\boldsymbol{x}) + \lambda_t\sigma_t(\boldsymbol{x}) = \mu_t(\boldsymbol{a}) + \lambda_t\sigma_t(\boldsymbol{a})$. By Lemma 6.3.1, we have $\frac{\theta-\mu_t(\boldsymbol{b})}{\sigma_t(\boldsymbol{b})} \leq \frac{\theta-\mu_t(\boldsymbol{a})}{\sigma_t(\boldsymbol{a})} = \lambda_t$. Combining both equations, we have

$$\mu_t(\boldsymbol{a}) + \lambda_t\sigma_t(\boldsymbol{a}) = \theta \leq \mu_t(\boldsymbol{b}) + \lambda_t\sigma_t(\boldsymbol{b}) \leq \mu_t(\boldsymbol{a}) + \lambda_t\sigma_t(\boldsymbol{a}).$$

Hence, by the uniqueness of $\boldsymbol{a}$ and $\boldsymbol{b}$, we have $\boldsymbol{a} = \boldsymbol{b}$, so 2 and 3 are equivalent. □

Alg. 14 compares the pseudocode for all the methods mentioned in Lemma 6.3.1. At the t-th iteration, we compute the target value, $\hat{y}^*$, with different strategies. Then, $\boldsymbol{x}_t = \arg\min_{\boldsymbol{x}\in\mathfrak{X}} \frac{\hat{y}^*-\mu_{t-1}(\boldsymbol{x})}{\sigma_{t-1}(\boldsymbol{x})}$ serves as the next input to evaluate.

Algorithm 14 A unified BO algorithm for MES, EST, GP-UCB and PI

function BO-UNIFIED (f, D_0)
  for $t = 1, \cdots, T$ **do**
    $\mu_{t-1}(\cdot), \sigma_{t-1}(\cdot) \leftarrow$ MODEL(D_{t-1})
    $\theta =$ { SAMPLEMAXVALUE$(1, D_{t-1})$ — MES with K=1; y^* — EST; $\max_{\boldsymbol{x}\in\mathfrak{X}} \mu_{t-1}(\boldsymbol{x}) + \lambda_{t-1}\sigma_{t-1}(\boldsymbol{x})$ — GP-UCB; $\max_{\tau\in[1,t-1]} y_\tau + \eta_{t-1}$ — PI
    $\boldsymbol{x}_t \leftarrow \arg\min_{\boldsymbol{x}\in\mathfrak{X}} \frac{\theta-\mu_{t-1}(\boldsymbol{x})}{\sigma_{t-1}(\boldsymbol{x})}$
    $y_t \leftarrow f(\boldsymbol{x}_t) + \epsilon_t, \epsilon_t \sim \mathcal{N}(0, \sigma^2)$
    $D_t \leftarrow D_{t-1} \cup \{\boldsymbol{x}_t, y_t\}$
  end for
end function

One interpretation of Lemma 6.3.1 is that all these acquisition functions (MES, EST, GP-UCB and PI) share the same idea of reaching a target value ($\hat{y}^*$ in this case), and thereby trading off exploration and exploitation. GP-UCB in [171] can be interpreted as setting the target value to be a loose upper bound $\max_{\boldsymbol{x}\in\mathfrak{X}} \mu_{t-1}(\boldsymbol{x}) + \lambda_t \sigma_{t-1}(\boldsymbol{x})$ with $\lambda_t = (2\log(|\mathfrak{X}|\pi^2 t^2/6\delta))^{\frac{1}{2}}$, as a result of applying a union bound on all the members of $\mathfrak{X}$[3]. PI applies a upwards shift of η_{t-1} to the current best observation $\max_{\tau\in[1,t-1]} y_\tau$. In both cases, the exploration-exploitation tradeoff depends on the parameter to be set.

On the other hand, MES and EST implicitly and automatically balance exploration and exploitation guided by the *maximum value* of the function. Viewed as GP-UCB or PI, MES and EST adaptively set the respective parameter in a data-dependent way. If we write EST or MES as PI, we see that target value used in PI becomes $\theta = \hat{y}^* > \max_{\tau\in[1,t-1]} y_\tau$. Having a target value larger than the current best observation is known to be advantageous in practice [124]. These analogies likewise suggest that the target value $\theta = \max_{\tau\in[1,t]} y_\tau$ in PI corresponds to a very small coefficient λ_t in GP-UCB, and results in very little exploration, offering an explanation for the known shortcomings of this target value θ.

From an information-theoretic view, we can also interpret Lemma 6.3.1 as a bridge between modern entropy search methods and classic BO acquisition functions: PI and GP-UCB are approximately maximizing the information gain about the function max-value if their parameters are set as in Lemma 6.3.1.

Note that max-value based acquisition functions are not only intuitively reasonable, but they also enable us to prove the first regret bounds for special cases of PI and entropy search methods, as detailed in Sec. 6.4.

6.4 Regret Bounds

In this section, we analyze regret bounds for EST with known max-value parameter θ and MES with a single max-value $\hat{y}_*$ sampled. By the connections analyzed in Sec. 6.3, EST with known max-value parameter θ is exactly the same method as PI with a given threshold parameter θ; MES with a single max-value sample $\hat{y}_*$ is a point estimate of the information gain in Eq. 6.5. To the best of our knowledge, the results shown in this section are the first regret bound for a variant of PI and

[3] Since $\Pr[|f(\boldsymbol{x})-\mu(\boldsymbol{x})| > \lambda_t\sigma(\boldsymbol{x})] \leq e^{\frac{-\lambda_t^2}{2}}$, applying the union bound results in $\Pr[|f(\boldsymbol{x})-\mu(\boldsymbol{x})| > \lambda_t\sigma(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathfrak{X}] \leq |\mathfrak{X}|e^{\frac{-\lambda_t^2}{2}}$. This means $f(\boldsymbol{x}) \leq \max_{\boldsymbol{x}\in\mathfrak{X}} \mu(\boldsymbol{x}) + \lambda_t\sigma(\boldsymbol{x})$ with probability at least $1 - |\mathfrak{X}|e^{\frac{-\lambda_t^2}{2}}$ [171, Lemma 5.1].

a variant of ES, both of which are guided by max-values.

6.4.1 Regret Bounds for EST and PI

We begin with a high-probability bound on the best-sample simple regret r_T. In particular, Theorem 6.4.1 establishes that EST and PI asymptotically achieves no regret if its max-value/threshold parameter θ is a strict upper bound on the maximum function value. Notice that this upper bound θ does not have to be the same across iterations. Hence in the following, we add subscript t and use θ_t for t-th iteration of EST and PI.

Theorem 6.4.1. *Assume the objective function $f \sim GP(\mu, k)$ where $k(\boldsymbol{x}, \boldsymbol{x}') \leq 1, \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathfrak{X}$ and the observations on f have iid additive Gaussian noise $\mathcal{N}(0, \sigma^2)$. If in each iteration $t \in [1, T]$, the query point is chosen as $\boldsymbol{x}_t = \arg\min_{\boldsymbol{x} \in \mathfrak{X}} \gamma_{\theta_t}^{(t-1)}(\boldsymbol{x})$, where $\gamma_{\theta_t}^{(t-1)}(\boldsymbol{x}) \triangleq \frac{\theta_t - \mu_{t-1}(\boldsymbol{x})}{\sigma_{t-1}(\boldsymbol{x})}$ and $\theta_t \geq \max_{\boldsymbol{x} \in \mathfrak{X}} f(\boldsymbol{x})$, then with probability at least $1 - \delta$, the best-sample simple regret up to T iterations is bounded as*

$$r_T \leq (\nu_{t^*} + \zeta_{t^*}) \sqrt{\frac{C \rho_T}{T}},$$

where $C = 2/\log(1 + \sigma^{-2})$ and $\zeta_t = (2 \log(\frac{\pi_t}{2\delta}))^{\frac{1}{2}}$, π_t satisfy $\sum_{t=1}^{T} \pi_t^{-1} \leq 1$, and $\pi_t > 0$; $t^* = \arg\max_t (\nu_t + \zeta_t)$ with $\nu_t \triangleq \min_{\boldsymbol{x} \in \mathfrak{X}} \gamma_{\theta_t}^{(t-1)}(\boldsymbol{x})$, and ρ_T is the maximum information gain of at most T selected points (Eq. (6.8)*).*

Let $\boldsymbol{f}_A = [f(\boldsymbol{x})]_{\boldsymbol{x} \in A}$ be the function values at an input set A and $\boldsymbol{y}_A$ be the observations at the same inputs. The information gain ρ_T after T rounds is the maximum mutual information between function values $\boldsymbol{f}_A$ and observations $\boldsymbol{y}_A$.

$$\rho_T = \max_{A \subseteq \mathfrak{X}, |A| \leq T} I(\boldsymbol{y}_A, \boldsymbol{f}_A) = \max_{A \subseteq \mathfrak{X}, |A| \leq T} \frac{1}{2} \log\det(\boldsymbol{I} + \sigma^{-2} \boldsymbol{K}_A). \tag{6.8}$$

If $k(\boldsymbol{x}, \boldsymbol{x}) \leq 1$ and $\mathfrak{X} \subset \mathbb{R}^d$, then we can bound $\rho_T = O((\log T)^{d+1})$ for the Gaussian kernel, and $\rho_T = O(T^{d(d+1)/(2\xi + d(d+1))} \log T)$ for the Matérn kernel, where ξ is the roughness parameter of the Matérn kernel [171, Theorem 5].

Before proving Theorem 6.4.1, we state a few lemmas that we will need.

Lemma 6.4.2. *Pick $\delta \in (0, 1)$ and set $\zeta_t = (2 \log(\frac{\pi_t}{2\delta}))^{\frac{1}{2}}$, where $\sum_{t=1}^{T} \pi_t^{-1} \leq 1$, $\pi_t > 0$. Then it holds that $\Pr[\mu_{t-1}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t) \leq \zeta_t \sigma_{t-1}(\boldsymbol{x}_t)] \geq 1 - \delta$, for all $t \in [1, T]$.*

Proof. Let $z_t = \frac{\mu_{t-1}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t)}{\sigma_{t-1}(\boldsymbol{x}_t)} \sim \mathcal{N}(0,1)$. It holds that

$$\begin{aligned}\Pr[z_t > \zeta_t] &= \int_{\zeta_t}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-z^2/2}\,\mathrm{d}z = \int_{\zeta_t}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-(z-\zeta_t)^2/2 - \zeta_t^2/2 - z\zeta_t}\,\mathrm{d}z \\ &\leq e^{-\zeta_t^2/2} \int_{\zeta_t}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-(z-\zeta_t)^2/2}\,\mathrm{d}z = \frac{1}{2} e^{-\zeta_t^2/2}.\end{aligned}$$

A union bound extends this bound to all rounds: $\Pr[z_t > \zeta_t \text{ for some } t \in [1,T]] \leq \sum_{t=1}^{T} \frac{1}{2} e^{-\zeta_t^2/2}$. With $\zeta_t = (2\log(\frac{\pi_t}{2\delta}))^{\frac{1}{2}}$ and $\sum_{t=1}^{T} \pi_t^{-1} = 1$, this implies that with probability at least $1-\delta$, it holds that $\mu_{t-1}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t) \leq \zeta_t \sigma_{t-1}(\boldsymbol{x}_t)$ for all $t \in [1,T]$. One may set $\pi_t = \frac{1}{6}\pi^2 t^2$, or $\pi_t = T$, in which case $\zeta_t = \zeta = (2\log(\frac{T}{2\delta}))^{\frac{1}{2}}$. □

Lemma 6.4.3 (Lemma 5.3 in [171])**.** *The information gain for the selected points can be expressed in terms of the predictive variances:*

$$I(\boldsymbol{y}_T; \boldsymbol{f}_T) = \frac{1}{2} \sum_{t=1}^{T} \log(1 + \sigma^{-2}\sigma_{t-1}^2(\boldsymbol{x}_t))).$$

The proof of Theorem 6.4.1 now follows from the above lemmas.

Proof. (Theorem 6.4.1) Let $\zeta_t = 2\log(\frac{\pi_t}{2\delta})$, $\pi_t = \frac{\pi^2 t^2}{6}$. By Lemma 6.4.2, $\mu_{t-1}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t) \leq \zeta_t \sigma_{t-1}(\boldsymbol{x}_t)\ \forall t \in [1,T]$ holds with probability at least $1-\delta$. We define immediate regret at t-th iteration as $\tilde{r}_t \triangleq \max_{\boldsymbol{x}\in\mathfrak{X}} f(\boldsymbol{x}) - f(\boldsymbol{x}_t)$. If $\mu_{t-1}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t) \leq \zeta_t \sigma_{t-1}(\boldsymbol{x}_t)$, $\tilde{r}_t$ is upper bounded as

$$\tilde{r}_t \leq \theta_t - f(\boldsymbol{x}_t) \leq \theta_t - \mu_{t-1}(\boldsymbol{x}_t) + \zeta_t \sigma_{t-1}(\boldsymbol{x}_t) = (\nu_t + \zeta_t)\sigma_{t-1}(\boldsymbol{x}_t), \leq (\nu_{t^*} + \zeta_{t^*})\sigma_{t-1}(\boldsymbol{x}_t)$$

where $\nu_t \triangleq \min_{\boldsymbol{x}\in\mathfrak{X}} \frac{\theta_t - \mu_{t-1}(\boldsymbol{x})}{\sigma_{t-1}(\boldsymbol{x})}$, $\theta_t \geq \max_{\boldsymbol{x}\in\mathfrak{X}} f(\boldsymbol{x})$ and $t^* = \arg\max_t(\nu_t + \zeta_t)$. The best-sample simple regret r_T satisfies

$$r_T \leq \frac{1}{T} \sum_{t=1}^{T} \tilde{r}_t \leq \frac{1}{T} \sum_{t=1}^{T} \sigma_{t-1}(\boldsymbol{x}_t)(\nu_{t^*} + \zeta_{t^*}).$$

To bound the sum of variances, we first use that $(1+a)^x \leq 1 + ax$ for $0 \leq x \leq 1$ and the assumption $\sigma_{t-1}(\boldsymbol{x}_t) \leq \sqrt{k(\boldsymbol{x}_t, \boldsymbol{x}_t)} \leq 1$ to obtain

$$\sigma_{t-1}^2(\boldsymbol{x}_t) \leq \frac{\log(1 + \sigma^{-2}\sigma_{t-1}^2(\boldsymbol{x}_t))}{\log(1+\sigma^{-2})}. \tag{6.9}$$

Lemma 6.4.3 now implies that $\sum_{t=1}^{T} \sigma_{t-1}^2(\boldsymbol{x}_t) \leq \frac{2}{\log(1+\sigma^{-2})} I(\boldsymbol{y}_T; \boldsymbol{f}_T)$. Using the shorthand $\rho_T =$

$I(\boldsymbol{y}_T; \boldsymbol{f}_T)$ and the Cauchy-Schwarz inequality leads to

$$\sum_{t=1}^{T} \sigma_{t-1}(\boldsymbol{x}_t) \leq \sqrt{T \sum_{t=1}^{T} \sigma_{t-1}^2(\boldsymbol{x}_t)} \leq \sqrt{\frac{2T}{\log(1+\sigma^{-2})} \rho_T}. \tag{6.10}$$

Therefore, with probability at least $1-\delta$,

$$r_T \leq (\nu_{t^*} + \zeta_{t^*}) \sqrt{\frac{2T\rho_T}{\log(1+\sigma^{-2})}}. \qquad \square$$

The derivation of the above regret bound for EST is similar to that for GP-UCB in [171], due to the connections between the two methods. Yet, the obviation of applying union bound over all of $\mathfrak{X}$, and the freedom of choosing the function maximum upper bound makes our bounds more flexible and adaptive to EST/PI with different choices of target values θ_t.

Consistency In order for this regret bound to be consistent, a.k.a. converging to 0, however, it is important to choose the target value θ_t in such a way that the regret bound's coefficient $\nu_{t^*} = \max_t \min_{\boldsymbol{x}} \frac{\theta_t - \mu_{t-1}(\boldsymbol{x})}{\sigma_{t-1}(\boldsymbol{x})}$ is $o(1)$. Admittedly, the choice of GP-UCB reflected by Corrollary 6.3.2 will ensure having a consistent bound by setting the target value θ_t to be $\max_{\boldsymbol{x}} \mu_{t-1}(\boldsymbol{x}) + \lambda_t \sigma_{t-1}(\boldsymbol{x})$ and appropriately setting λ_t, e.g. setting λ_t to be a constant. Alternatively, if we set θ_t to be the exact max-value f^*, we can also ensure that ν_t is bounded by a constant with high probability by applying Lemma 6.4.2 to the function argmax $\boldsymbol{x}^*$. It is unclear if a consistent regret bound exists for target values strictly larger than the function max-value, but empirically, as shown in Section 6.6, our method stills works well with approximate estimates of max-value f^*.

6.4.2 Regret Bounds for MES

The connection with EST and PI directly leads to a bound on the best-sample simple regret of MES, when using a single max-value sample $\hat{y}_t^*$ in the t-th iteration of the algorithm.

Theorem 6.4.4. *Assume the objective function $f \sim GP(\mu, k)$ where $k(\boldsymbol{x}, \boldsymbol{x}') \leq 1, \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathfrak{X}$ and the observations on f have iid additive Gaussian noise $\mathcal{N}(0, \sigma^2)$. Let F be a cumulative density function such that $F(y^*) \leq w \in [0, 1)$, $y^* = \max_{\boldsymbol{x} \in \mathfrak{X}} f(\boldsymbol{x})$. If in each iteration t, the query point is chosen as $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x} \in \mathfrak{X}} \gamma_{\hat{y}_t^*}^{(t-1)}(\boldsymbol{x}) \frac{\psi(\gamma_{\hat{y}_t^*}^{(t-1)}(\boldsymbol{x}))}{2\Psi(\gamma_{\hat{y}_t^*}^{(t-1)}(\boldsymbol{x}))} - \log(\Psi(\gamma_{\hat{y}_t^*}^{(t-1)}(\boldsymbol{x})))$, where $\gamma_{\hat{y}_t^*}^{(t-1)}(\boldsymbol{x}) = \frac{\hat{y}_t^* - \mu_{t-1}(\boldsymbol{x})}{\sigma_{t-1}(\boldsymbol{x})}$ and $\hat{y}_t^*$ is independently drawn from F, then with probability at least $1-\delta$,*

in $T' = \sum_{i=1}^{T} \log_w \frac{\delta}{2\pi_i}$ *number of iterations, the best-sample simple regret satisfies*

$$r_{T'} \leq \sqrt{\frac{C\rho_T}{T}}(\nu_{t^*} + \zeta_{t^*}) \tag{6.11}$$

where $C = 2/\log(1+\sigma^{-2})$ *and* $\zeta_i = (2\log(\frac{\pi_i}{\delta}))^{\frac{1}{2}}$, π_i *satisfies* $\sum_{i=1}^{T} \pi_i^{-1} \leq 1$ *and* $\pi_i > 0$, $i \in [1, T]$*;* $t^* = \arg\max_{i\in[1,T]}(\nu_i+\zeta_i)$ *with* $\nu_i \triangleq \min_{\boldsymbol{x}\in\mathfrak{X}, \hat{y}^*_{t_i} > y^*} \gamma^{(t_i-1)}_{\hat{y}^*_{t_i}}(\boldsymbol{x})$*, and* ρ_T *is the maximum information gain of at most* T *selected points.*

Proof. (Theorem 6.4.4) By Lemma 6.3.1, we know that the theoretical results from Theorem 6.4.1 can be adapted to MES if $\hat{y}^*_t \geq y^*$. The key question is when a sampled $\hat{y}^*_t$ that can satisfy this condition. Because the cumulative density $F(y^*) \leq w \in [0, 1)$ and $\hat{y}^*_t$ are independent samples from F, there exists at least one $\hat{y}^*_t$ that satisfies $\hat{y}^*_t \geq y^*$ with probability at least $1 - w^{k_i}$ in k_i iterations.

Let $T' = \sum_{i=1}^{T} k_i$ be the total number of iterations. We split these iterations to T parts where each part have k_i iterations, $i = 1, \cdots, T$. By union bound, with probability at least $1 - \sum_{i=1}^{T} w^{k_i}$, in all the T parts of iterations, we have at least one iteration t_i which samples $\hat{y}^*_{t_i}$ satisfying $\hat{y}^*_{t_i} \geq y^*, \forall i = 1, \cdots, T$.

Let $\sum_{i=1}^{T} w^{k_i} = \frac{\delta}{2}$, we can set $k_i = \log_w \frac{\delta}{2\pi_i}$ for any $\sum_{i=1}^{T}(\pi_i)^{-1} = 1$. A convenient choice for π_i is $\pi_i = \frac{\pi^2 i^2}{6}$. Hence with probability at least $1 - \frac{\delta}{2}$, there exist a sampled $\hat{y}^*_{t_i}$ satisfying $\hat{y}^*_{t_i} \geq y^*, \forall i = 1, \cdots, T$.

Now let $\zeta_i = (2\log\frac{\pi_i}{\delta})^{\frac{1}{2}}$. By Lemma 6.4.2, the immediate regret $r_{t_i} = y^* - f(\boldsymbol{x}_{t_i})$ can be bounded as $r_{t_i} \leq (\nu_i + \zeta_i)\sigma_{t_i-1}(\boldsymbol{x}_{t_i})$. Note that by assumption $0 \leq \sigma^2_{t_i-1}(\boldsymbol{x}_{t_i}) \leq 1$, so we have $\sigma^2_{t_i-1} \leq \frac{\log(1+\sigma^{-2}\sigma^2_{t_i-1}(\boldsymbol{x}_{t_i}))}{\log(1+\sigma^{-2})}$. Then by Lemma 6.4.3, we have $\sum_{i=1}^{T} \sigma^2_{t_i-1}(\boldsymbol{x}_{t_i}) \leq \frac{2}{\log(1+\sigma^{-2})} I(\boldsymbol{y}_T; \boldsymbol{f}_T)$ where $\boldsymbol{f}_T = (f(\boldsymbol{x}_{t_i}))_{i=1}^{T} \in \mathbb{R}^T, \boldsymbol{y}_T = (y_{t_i})_{i=1}^{T} \in \mathbb{R}^T$. From assumptions, we have $I(\boldsymbol{y}_T; \boldsymbol{f}_T) \leq \rho_T$. By Cauchy-Schwarz inequality, $\sum_{i=1}^{T} \sigma_{t_i-1}(\boldsymbol{x}_{t_i}) \leq \sqrt{T\sum_{i=1}^{T} \sigma^2_{t_i-1}(\boldsymbol{x}_{t_i})} \leq \sqrt{\frac{2T\rho_T}{\log(1+\sigma^{-2})}}$. It follows that with probability at least $1 - \delta$,

$$\sum_{i=1}^{T} r_{t_i} \leq (\nu_{t^*} + \zeta_{t^*})\sqrt{\frac{2T\rho_T}{\log(1+\sigma^{-2})}}.$$

As a result, the best-sample simple regret is bounded as

$$r_{T'} \leq \frac{1}{T}\sum_{i=1}^{T} r_{t_i} \leq (\nu_{t^*} + \zeta_{t^*})\sqrt{\frac{2\rho_T}{T\log(1+\sigma^{-2})}},$$

where $T' = \sum_{i=1}^{T} k_i = \sum_{i=1}^{T} \log_w \frac{\delta}{2\pi_i}$ is the total number of iterations. □

Notice that Theorem 6.4.4 removed the assumption in [187, Theorem 3.2] that F is the ground truth probability density function of the function maximum y^*. The only requirement on F is that there is non-zero probability to sample $\hat{y}^* \geq y^*$. Together with Lemma 6.4.6, we can show that the regret bound in Theorem 6.4.4 holds for the sampling approach described in Section 6.2.2 with some additional assumptions on function f. The proof of Lemma 6.4.6 uses Slepian's comparison lemma stated in Lemma 6.4.5.

Lemma 6.4.5 (Slepian's Comparison Lemma [165, 131])**.** *Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ be two multivariate Gaussian random vectors with the same mean and variance, such that*

$$\mathbb{E}[\boldsymbol{v}_i \boldsymbol{v}_j] \leq \mathbb{E}[\boldsymbol{u}_i \boldsymbol{u}_j], \forall i, j.$$

Then for every y

$$\Pr[\sup_{i\in[1,n]} \boldsymbol{v}_i \leq y] \leq \Pr[\sup_{i\in[1,n]} \boldsymbol{u}_i \leq y].$$

Lemma 6.4.6. *In addition to the assumptions made in Theorem 6.4.4, we assume f is a Lipschitz continuous function with Lipschitz constant L. Let $\hat{\mathfrak{X}}$ be an ϵ-covering of $\mathfrak{X}$. For any iteration $t \in [1, T]$, if $\hat{z}^* \sim F_t(y) = \prod_{\boldsymbol{x}\in\hat{\mathfrak{X}}} \Psi(\gamma_y^{(t-1)}(\boldsymbol{x}))$ and $k_t(\boldsymbol{x}, \boldsymbol{x}') \geq 0, \forall \boldsymbol{x}, \boldsymbol{x}' \in \hat{\mathfrak{X}}$, then there exists $w \in [0, 1)$ such that $\Pr[\hat{z}^* + \epsilon L < y^*] \leq w$.*

Proof. If $k_t(\boldsymbol{x}, \boldsymbol{x}') \geq 0, \forall \boldsymbol{x}, \boldsymbol{x}' \in \hat{\mathfrak{X}}$, we have $F_t(y) = \Pr[\sup_{\boldsymbol{x}\in\hat{\mathfrak{X}}} g(\boldsymbol{x}) \leq y]$ where $g(\boldsymbol{x}) \sim \mathcal{N}(\mu_t(\boldsymbol{x}), \sigma_t^2(\boldsymbol{x}))$ and $\mathrm{Cov}(g(\boldsymbol{x}), g(\boldsymbol{x}')) = 0 \leq k_t(\boldsymbol{x}, \boldsymbol{x}'), \forall \boldsymbol{x}, \boldsymbol{x}' \in \hat{\mathfrak{X}}$. By Lemma 6.4.5, the cumulative density function F_t satisfies $F_t(y) \leq \Pr[\sup_{\boldsymbol{x}\in\hat{\mathfrak{X}}} f(\boldsymbol{x}) < y \mid D_t]$. Let z^* be the ground truth maximizer of $f(\boldsymbol{x}), \boldsymbol{x} \in \hat{\mathfrak{X}}$. Because $z^* \sim \Pr[\sup_{\boldsymbol{x}\in\hat{\mathfrak{X}}} f(\boldsymbol{x}) < y \mid D_t]$, $\Pr[\sup_{\boldsymbol{x}\in\hat{\mathfrak{X}}} f(\boldsymbol{x}) < z^* \mid D_t] \in (0, 1)$ by the sampling procedure: uniformly randomly sample p from Unif$[0, 1]$ and find z^* such that $\Pr[\sup_{\boldsymbol{x}\in\hat{\mathfrak{X}}} f(\boldsymbol{x}) < z^* \mid D_t] = p; p \neq 1$ w.p. 1. Hence, $F_t(z^*) \leq w$ and $w \in [0, 1)$.

Because $\hat{\mathfrak{X}}$ is an ϵ-covering of $\mathfrak{X}$, for any $\boldsymbol{x} \in \mathfrak{X}$, there exists $\boldsymbol{x}' \in \hat{\mathfrak{X}}$ such that $\|\boldsymbol{x}' - \boldsymbol{x}\| \leq \epsilon$. By Lipschitz assumption on f, we have $|f(\boldsymbol{x}) - f(\boldsymbol{x}')| \leq L\epsilon, \forall \|\boldsymbol{x} - \boldsymbol{x}'\| \leq \epsilon$. So there exist $\boldsymbol{x} \in \hat{\mathfrak{X}}$ such that $|f(\boldsymbol{x}) - y^*| \leq L\epsilon$. This implies that $y^* \leq f(\boldsymbol{x}) + L\epsilon, \exists \boldsymbol{x} \in \hat{\mathfrak{X}}$ and $y^* \leq z^* + L\epsilon$.

By assumption, we have $\hat{z}^* \sim F_t(\cdot)$. Because we have $F_t(z^*) \leq w$ and $w \in [0, 1)$, $\hat{z}^*$ satisfies $\Pr[\hat{z}^* \leq z^*] \leq w$, and so $\Pr[\hat{z}^* \leq y^* - L\epsilon] \leq w$. □

Thus, a regret bound for MES with max-values sampled according to Section 6.2.2 directly follows by using w in Lemma 6.4.6 in Theorem 6.4.4.

Comparing Theorem 6.4.4 and Theorem 6.4.1, it might seem like MES with a single max-value sample does not have a converging rate as good as EST or GP-UCB. However, as we will see in Lemma 6.4.7, $\min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{y_1}^{(t-1)}(\boldsymbol{x}) < \min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{y_2}^{(t-1)}(\boldsymbol{x})$ if $y_1 < y_2$. Here $\min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{\cdot}^{(t-1)}(\boldsymbol{x})$ decides the rate of convergence in Eq. 6.11. So larger $\hat{y}_t^*$ may lead to worse regret bound. If we use $\hat{y}_t^*$ that is smaller than y^*, however, its value is not count in the total regret in our proof and its effect in terms of reducing the regret is unknown. With no easy way of setting $\hat{y}_t^*$ since y^* is unknown, our regret bound in Theorem 6.4.4 is a randomized trade-off between sampling large and small $\hat{y}_t^*$.

6.4.3 Effects of Target Values

In both Section 6.4.1 and Section 6.4.2, we have seen how the target value θ in Algorithm 14 influences the theoretical analyses of regrets. In this section, we focus on one iteration of Algorithm 14 and investigate the effects of different target values.

Lemma 6.4.7. *For any iteration t in Algorithm 14, suppose we have two target values $\theta^{(1)} < \theta^{(2)}$. Let $\boldsymbol{x}^{(i)} = \arg\min_{\boldsymbol{x}\in\mathfrak{X}} \gamma_{\theta^{(i)}}^{(t-1)}(\boldsymbol{x})$ be the selected input when using $\theta^{(i)}$ and $\tilde{r}^{(i)} = y^* - f(\boldsymbol{x}^{(i)})$ be the immediate regret, $i = 1, 2$. Then $\mathbb{E}[\tilde{r}^{(1)}] \leq \mathbb{E}[\tilde{r}^{(2)}]$, $\mu_{t-1}(\boldsymbol{x}^{(1)}) \geq \mu_{t-1}(\boldsymbol{x}^{(2)})$, $\sigma_{t-1}(\boldsymbol{x}^{(1)}) \leq \sigma_{t-1}(\boldsymbol{x}^{(2)})$ and $\gamma_{\theta^{(1)}}^{(t-1)}(\boldsymbol{x}^{(1)}) < \gamma_{\theta^{(2)}}^{(t-1)}(\boldsymbol{x}^{(2)})$.*

Proof. We use the shorthands $\mu_i = \mu_{t-1}(\boldsymbol{x}^{(i)})$, $\sigma_i = \sigma_{t-1}(\boldsymbol{x}^{(i)})$ and $\theta_i = \theta^{(i)}$, $i = 1, 2$. By definition of $\boldsymbol{x}^{(1)}$, we have $\frac{\theta_1-\mu_1}{\sigma_1} \leq \frac{\theta_1-\mu_2}{\sigma_2} < \frac{\theta_2-\mu_2}{\sigma_2} \implies \gamma_{\theta^{(1)}}^{(t-1)}(\boldsymbol{x}^{(1)}) < \gamma_{\theta^{(2)}}^{(t-1)}(\boldsymbol{x}^{(2)})$.

The expected immediate regret is $\mathbb{E}[\tilde{r}^{(i)}] = \max_{\boldsymbol{x}\in\mathfrak{X}} f(\boldsymbol{x}) - \mu_i$, so $\mu_1 \geq \mu_2$ directly implies $\mathbb{E}[\tilde{r}^{(1)}] \leq \mathbb{E}[\tilde{r}^{(2)}]$. We prove $\mu_1 \geq \mu_2$ by contradiction.

Case 1: Assume $\mu_1 < \mu_2$ and $\sigma_1 \leq \sigma_2$. This implies $\frac{\theta_1-\mu_1}{\sigma_1} > \frac{\theta_1-\mu_2}{\sigma_2}$, but it contradicts with the definition of $\boldsymbol{x}^{(1)}$. Hence the assumptions in Case 1 do not hold.

Case 2: Assume $\mu_1 < \mu_2$ and $\sigma_1 > \sigma_2$. This implies $\frac{\sigma_1}{\sigma_2} - 1 > 0$. By definition of $\boldsymbol{x}^{(1)}$,

$$\frac{\theta_1-\mu_1}{\sigma_1} \leq \frac{\theta_1-\mu_2}{\sigma_2} \implies \left(\frac{\sigma_1}{\sigma_2}-1\right)^{-1}\left(\frac{\sigma_1}{\sigma_2}\mu_2-\mu_1\right) \leq \theta_1 < \theta_2 \implies \frac{\theta_2-\mu_1}{\sigma_1} < \frac{\theta_2-\mu_2}{\sigma_2}.$$

However, by definition of $\boldsymbol{x}^{(2)}$, $\frac{\theta_2-\mu_1}{\sigma_1} \geq \frac{\theta_2-\mu_2}{\sigma_2}$. So the assumptions in Case 2 do not hold.

Since both Case 1 and Case 2 do not hold, $\mu_1 \geq \mu_2$ and $\mathbb{E}[\tilde{r}^{(1)}] \leq \mathbb{E}[\tilde{r}^{(2)}]$ must hold.

We next prove $\sigma_1 \leq \sigma_2$ by contradiction. Suppose $\sigma_1 > \sigma_2$ is true. Then we must have $\frac{\theta_2-\mu_1}{\sigma_1} < \frac{\theta_2-\mu_2}{\sigma_2}$, but this contradicts with the definition of $\boldsymbol{x}^{(2)}$. So $\sigma_1 \leq \sigma_2$. □

An implication of this lemma is that using smaller target values leads to less exploration and

smaller expected one-step regret. While a higher target value lets Algorithm 14 explore inputs with higher variance. These explanations are consistent with the connection to GP-UCB. By Lemma 6.3.1, Algorithm 14 with target value $\theta^{(i)}$ corresponds to GP-UCB with parameter $\lambda_t = \gamma^{(t)}_{\theta^{(i)}}(\boldsymbol{x}^{(i)})$. So from Lemma 6.4.7 we know that a larger target value corresponds to a larger λ_t in GP-UCB, which is known to emphasize more on exploration than exploitation.

6.5 High Dimensional MES with Add-GP

The high-dimensional input setting has been a challenge for many BO methods. We extend MES to this setting via additive Gaussian processes (Add-GP). In the past, Add-GP has been used and analyzed for GP-UCB by [91], who assumed the high dimensional black-box function is a summation of several disjoint lower dimensional functions: $f(\boldsymbol{x}) = \sum_{m=1}^{M} f^{(m)}(\boldsymbol{x}^{A_m})$, with $A_i \cap A_j = \emptyset$ for all $i \neq j$, $|\cup_{i=1}^{M} A_i| = d$. Utilizing this special additive structure, we overcome the statistical problem of having insufficient data to recover a complex function, and the difficulty of optimizing acquisition functions in high dimensions.

Since the function components $f^{(m)}$ are operating on disjoint lower dimensional space, we can maximize the mutual information between the input in the active dimensions A_m and maximum of $f^{(m)}$ for each component separately. Hence, we have a separate acquisition function for each function component, where $y^{(m)}$ is the evaluation of $f^{(m)}$:

$$\alpha_t^{(m)}(\boldsymbol{x}) = I(\{\boldsymbol{x}^{A_m}, y^{(m)}\}; y_m^* \mid D_t) \tag{6.12}$$

$$= H(p(y^{(m)} \mid D_t, \boldsymbol{x}^{A_m})) - \mathbb{E}[H(p(y^{(m)} \mid D_t, \boldsymbol{x}^{A_m}, y_m^*))] \tag{6.13}$$

$$\approx \sum\nolimits_{\hat{y}^* \in Y^*_{(m)}} \gamma^{(m)}_{\hat{y}^*}(\boldsymbol{x}) \frac{\psi(\gamma^{(m)}_{\hat{y}^*}(\boldsymbol{x}))}{2\Psi(\gamma^{(m)}_{\hat{y}^*}(\boldsymbol{x}))} - \log(\Psi(\gamma^{(m)}_{\hat{y}^*}(\boldsymbol{x}))) \tag{6.14}$$

where $\gamma^{(m)}_{\hat{y}^*}(\boldsymbol{x}) = \frac{\hat{y}^*_m - \mu_t^{(m)}(\boldsymbol{x})}{\sigma_t^{(m)}(\boldsymbol{x})}$. Analogously to the non-additive case, we sample the function max-value y_m^*, but now we do it separately for each function component. We select the final $\boldsymbol{x}_t$ by choosing a sub-vector $\boldsymbol{x}_t^{(m)} = \arg\max_{\boldsymbol{x}^{(m)} \in A_m} \alpha_t^{(m)}(\boldsymbol{x}^{(m)})$ and concatenating the components.

Sampling y_m^* with a Gumbel distribution The Gumbel sampling approach from Section 6.2.2 directly extends to sampling y_m^*. We can use the same "mean-field" independence assumption and sample from the component-wise CDF $\widehat{\Pr}[y_*^{(m)} < z] = \prod_{\boldsymbol{x} \in \hat{\mathfrak{X}}} \Psi(\gamma_y^{(m)}(\boldsymbol{x})))$. We can also use the same Gumbel approximation to further speed up the computations.

Sampling y_m^* via posterior functions The additive structure removes some connections of the input-to-hidden weight layer of our 1-hidden-layer neural network approximation $\tilde{f}(\boldsymbol{x}) = \boldsymbol{a}_t^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$. Namely, for each feature function ϕ there exists a unique group m such that ϕ is only active on $\boldsymbol{x}^{A_m}$, and $\phi(\boldsymbol{x}) = \sqrt{\frac{2}{D}}\cos(\omega^{\mathrm{T}}\boldsymbol{x}^{A_m} + c)$ where $\mathbb{R}^{|A_m|} \ni \omega \sim \hat{\kappa}^{(m)}(\omega)$ and $c \sim U[0, 2\pi]$. Similar to the non-additive case, we may draw a posterior sample $\boldsymbol{a}_t \sim \mathcal{N}(\boldsymbol{\nu}_t, \boldsymbol{\Sigma}_t)$ where $\boldsymbol{\nu}_t = \sigma^{-2}\boldsymbol{\Sigma}_t Z_t \boldsymbol{y}_t$ and $\boldsymbol{\Sigma}_t = (ZZ^{\mathrm{T}}\sigma^{-2} + \boldsymbol{I})^{-1}$. Let $B_m = \{i : \boldsymbol{\phi}_i(\boldsymbol{x}) \text{ is active on } \boldsymbol{x}^{A_m}\}$. The posterior sample for the function component $f^{(m)}$ is $\tilde{f}^{(m)}(\boldsymbol{x}) = (\boldsymbol{a}_t^{B_m})^{\mathrm{T}}\phi^{B_m}(\boldsymbol{x}^{A_m})$. Then we can maximize $\tilde{f}^{(m)}$ to obtain a sample for y_m^*.

The algorithm for the additive max-value entropy search method (add-MES) is shown in Algorithm 15. The function SAMPLEMAXVALUE samples max-values for approximating the mutual information in a similar way as in Algorithm 13, except that it only acts on the active dimensions in the m-th group.

Algorithm 15 Additive Max-value Entropy Search

function ADD-MES(f, D_0)
  for $t = 1, \cdots, T$ **do**
    for $m = 1, \cdots, M$ **do**
      $Y^*_{(m)} \leftarrow$SAMPLEMAXVALUE $^{(m)}(K, D_{t-1})$
      $\alpha^{(m)}_{t-1}(\cdot) \leftarrow$ Eq. (6.14)
      $\boldsymbol{x}_t^{A_m} \leftarrow \arg\max_{\boldsymbol{x}^{A_m} \in \mathfrak{X}^{A_m}} \alpha^{(m)}_{t-1}(\boldsymbol{x})$
    end for
    $y_t \leftarrow f(\boldsymbol{x}_t) + \epsilon_t, \epsilon_t \sim \mathcal{N}(0, \sigma^2)$
    $D_t \leftarrow D_{t-1} \cup \{\boldsymbol{x}_t, y_t\}$
  end for
end function

6.6 Experiments

In this section, we probe the empirical performance of acquisition functions guided by max-value on a variety of tasks. To avoid confusion, we assume MES, EST, GP-UCB, and PI use their default ways of setting parameters. Here, MES-G denotes MES with y^* sampled from the approximate Gumbel distribution, and MES-R denotes MES with y^* computed by maximizing a sampled function represented by random features.

6.6.1 Implementation details

We evaluate BO performance according to the best-sample simple regret and inference regret defined in Section 6.1.3. We used the open source Matlab implementation of PES, ES [75, 78]. Our Matlab code and test functions are available at

`https://github.com/zi-w/Max-value-Entropy-Search/`.

Choice of GP priors Following [78], we adopt the zero mean function and non-isotropic squared exponential kernel as the prior for the GP.

Model adaptation In practice we do not know the hyper-parameters of the GP, so we must adapt our GP model as we observe more data. A standard way to learn the GP hyper-parameters is to optimize the marginal data likelihood with respect to the hyper-parameters. As a full Bayesian treatment, we can also draw samples of the hyper-parameters using slice sampling [184], and then marginalize out the hyper-parameters in our acquisition function in Eq. (6.7). Namely, if we use E to denote the set of sampled settings for the GP hyper-parameters, our acquisition function becomes

$$\alpha_t(\boldsymbol{x}) = \sum\nolimits_{\eta\in E}\sum\nolimits_{y^*\in Y^*}\left[\frac{\gamma^\eta_{y^*}(\boldsymbol{x})\psi(\gamma^\eta_{y^*}(\boldsymbol{x}))}{2\Psi(\gamma^\eta_{y^*}(\boldsymbol{x}))} - \log(\Psi(\gamma^\eta_{y^*}(\boldsymbol{x})))\right],$$

where $\gamma^\eta_{y^*}(\boldsymbol{x}) = \frac{y^*-\mu^\eta_t(\boldsymbol{x})}{\sigma^\eta_t(\boldsymbol{x})}$ and the posterior inference on the mean function μ^η_t and σ^η_t depends on the GP hyper-parameter setting η. Similar approaches have been used by [78, 169].

Hyper-parameters of acquisition functions We compare to methods from the entropy search family, i.e., ES and PES, and to other popular Bayesian optimization methods including GP-UCB (denoted by UCB), PI, EI and EST. For EST, we use the function maximum value as the max-value parameter if it is available; otherwise we use an estimation method described by [191] to estimate the max-value. The parameter for GP-UCB was set according to Theorem 2 in [171]; the parameter η_t for PI was set to be the observation noise σ. For the functions with unknown GP hyper-parameters, every 10 iterations, we learn the GP hyper-parameters using the same approach as was used by [78] for PES. For the high dimensional tasks, we follow [91]; when the additive decomposition is unknown, we sample 10^4 additive structures/GP parameters and use a fixed one with the highest data likelihood based on 500 data points uniformly randomly sampled from the search space $\mathfrak{X}$.

Global optimization of acquisition functions Because acquisition functions are by themselves non-concave non-convex functions, optimizing them accurately can be challenging. We fol-

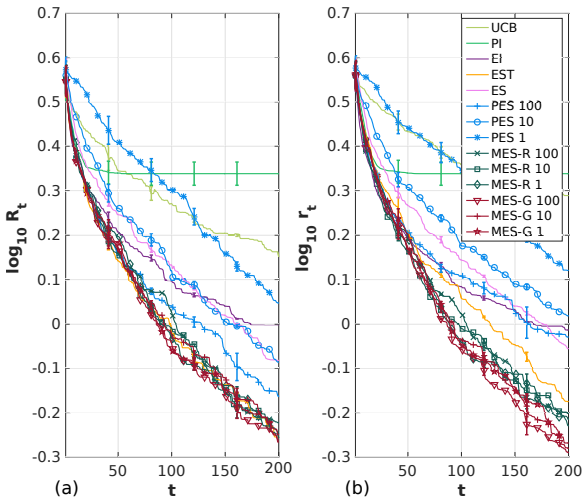


Figure 6-2: (a) Inference regret; (b) best-sample simple regret. MES methods are much less sensitive to the number of maxima y^* sampled for the acquisition function (1, 10 or 100) than PES is to the number of argmaxes $\boldsymbol{x}_*$.

METHOD	TIME (S)	METHOD	TIME (S)	METHOD	TIME (S)
UCB	0.08 ± 0.05	PES 100	15.24 ± 4.44	MES-R 1	0.13 ± 0.03
PI	0.10 ± 0.02	PES 10	1.61 ± 0.50	MES-G 100	0.12 ± 0.02
EI	0.07 ± 0.03	PES 1	0.20 ± 0.06	MES-G 10	0.09 ± 0.02
EST	0.15 ± 0.02	MES-R 100	5.85 ± 0.86	MES-G 1	0.09 ± 0.03
ES	8.07 ± 3.02	MES-R 10	0.67 ± 0.11		

Table 6.1: The runtime of selecting the next input. PES 100 is significantly slower than other methods. MES-G's runtime is comparable to the fastest method EI while it performs better in terms of simple and inference regrets.

low a simple approach PES [78] adopted: select a random set of inputs to evaluate their acquisition function values and then run an optimization algorithm, such as the interior point method [24], starting from the input with the highest acquisition function value.

6.6.2 Synthetic Functions

We begin with a comparison on synthetic functions sampled from a 3-dimensional GP, to probe our conjecture that MES is much more robust to the number of y^* sampled to estimate the acquisition function than PES is to the number of $\boldsymbol{x}_*$ samples. For PES, we sample 100 (PES 100), 10 (PES 10) and 1 (PES 1) argmaxes for the acquisition function. Similarly, we sample 100, 10, 1 y^* values for MES-R and MES-G. We average the results on 100 functions sampled from the same Gaussian kernel with scale parameter 5.0 and bandwidth parameter 0.0625, and observation noise $\mathcal{N}(0, 0.01^2)$.

Figure 6-2 shows the simple and inference regrets. For both regret measures, PES is very sensitive to the the number of $\boldsymbol{x}_*$ sampled for the acquisition function: 100 samples lead to much better results than 10 or 1. In contrast, MES-G and MES-R perform competitively even with 1 or 10 samples. Overall, MES-G is slightly better than MES-R, and both MES methods performed better than other ES methods. MES methods performed better than all other methods with respect to best-sample simple regret. For inference regret, MES methods performed similarly to EST, and much better than all other methods including PES and ES.

In Table 6.1, we show the runtime of selecting the next input per iteration[4] using GP-UCB, PI, EI, EST, ES, PES, MES-R and MES-G on the synthetic data with fixed GP hyper-parameters. EST used the max-value estimation method introduced by [191] so it was not as fast as PI. For PES and MES-R, every $\boldsymbol{x}_*$ or y^* requires running an optimization sub-procedure, so their running time grows noticeably with the number of samples. MES-G avoids this optimization, and competes with the fastest methods EI and UCB.

In the following experiments, we set the number of $\boldsymbol{x}_*$ sampled for PES to be 200, and the number of y^* sampled for MES-R and MES-G to be 100 unless otherwise mentioned.

6.6.3 Optimization Test Functions

We test on 6 challenging optimization test functions: the 2-D eggholder function, the 10-D Shekel function, the 2-D, 5-D, and 10-D Michalewicz function and the 6-D Hartmann function. All of these functions have many local optima. We randomly sample 1000 points to learn a good GP hyper-parameter setting, and then run the BO methods with the same hyper-parameters. The first observation is the same for all methods. We repeat the experiments 10 times.

The averaged best-sample simple regret is shown in Fig. 6-3, and the inference regret is shown in Table 6.2. On the 2-D eggholder function and 2-D Michalewicz function, PES was able to achieve good function values in a fast manner, which verified the good performance of PES when sufficiently many $\boldsymbol{x}_*$ are sampled. However, for higher-dimensional test functions, MES and EST methods performed much better than PES. Overall MES-G performed better than all of other methods on 3 out of 6 functions. Both EST and MES-G achieved (overall) better inference regrets than other methods.

[4] All the timing experiments were run exclusively on an Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz. The function evaluation time is excluded.

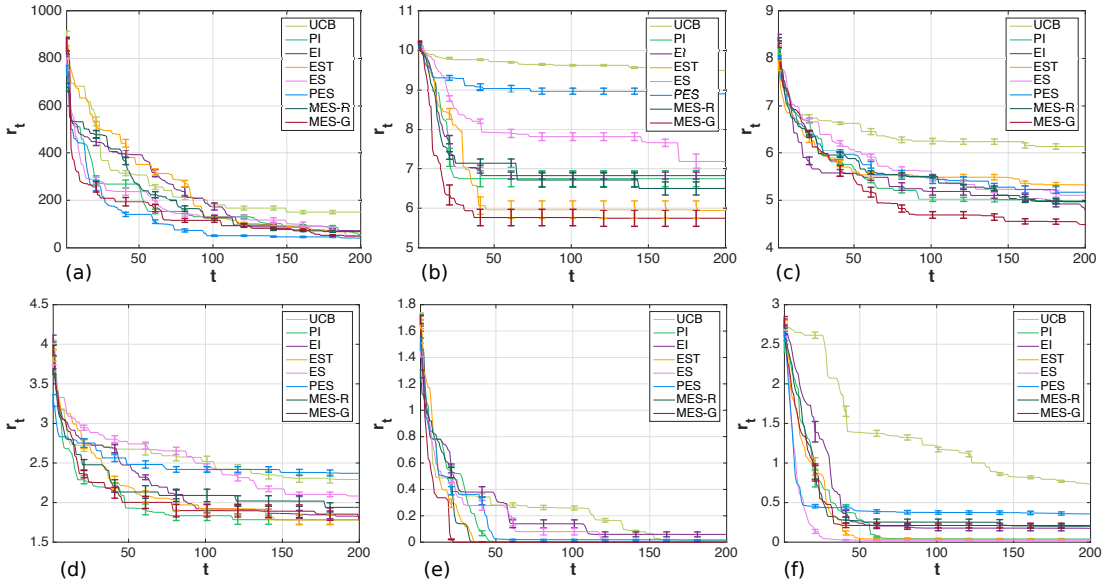


Figure 6-3: (a) 2-D eggholder function; (b) 10-D Shekel function; (c) 10-D Michalewicz function; (d) 5-D Michalewicz function; (e) 2-D Michalewicz function; (f) 6-D Hartmann function. The results are mixed, but in general, EST, MES-R and MES-G performed competitively comparing to other approaches. In particular, MES-G was able to achieve the lowest best-sample simple regret on 3 out of 6 functions.

6.6.4 Tuning Hyper-parameters for Neural Networks

Next, we experiment with Levenberg-Marquardt optimization for training a 1-hidden-layer neural network. The 4 parameters we tune with BO are the number of neurons, the damping factor μ, the μ-decrease factor, and the μ-increase factor. We test regression on the Boston housing dataset and classification on the breast cancer dataset [9]. The experiments are repeated 20 times, and the neural network's weight initialization and all other parameters are set to be the same to ensure a fair comparison. Both of the datasets were randomly split into train/validation/test sets. We initialize the observation set to have 10 random function evaluations which were set to be the same across all the methods. The averaged best-sample simple regret for the regression L2-loss on the validation set of the Boston housing dataset is shown in Fig. 6-4(a), and the classification accuracy on the validation set of the breast cancer dataset is shown in Fig. 6-4(b). For the classification problem on the breast cancer dataset, MES-G, PES and UCB achieved a similar best-sample simple regret. On the Boston housing dataset, MES methods and PES achieved a lower best-sample simple regret. We also show the inference regrets for both datasets in Table 6.3.

METHOD	EGGHOLDER	SHEKEL	10-D MICH	5-D MICH	2-D MICH	6-D HART
UCB	141.00 ± 70.96	9.40 ± 0.26	6.07 ± 0.53	2.55 ± 0.45	0.02 ± 0.01	0.76 ± 0.11
PI	52.04 ± 39.03	6.64 ± 2.00	4.97 ± 0.39	1.68 ± 0.43	0.01 ± 0.00	0.01 ± 0.04
EI	71.18 ± 59.18	6.63 ± 0.87	4.80 ± 0.60	1.74 ± 0.22	0.06 ± 0.19	0.17 ± 0.32
EST	55.84 ± 24.85	5.57 ± 2.56	5.33 ± 0.46	1.72 ± 0.39	0.00 ± 0.00	0.03 ± 0.06
ES	48.85 ± 29.11	6.43 ± 2.73	5.11 ± 0.73	1.94 ± 0.42	0.01 ± 0.00	0.02 ± 0.05
PES	37.94 ± 26.05	8.73 ± 0.67	5.17 ± 0.74	1.95 ± 0.36	0.03 ± 0.02	0.42 ± 0.21
MES-R	54.47 ± 37.71	6.17 ± 1.80	4.97 ± 0.59	1.88 ± 0.64	0.01 ± 0.00	0.21 ± 0.27
MES-G	46.56 ± 27.05	5.45 ± 2.07	4.49 ± 0.51	1.72 ± 0.66	0.00 ± 0.00	0.21 ± 0.30

Table 6.2: Inference regret R_T for optimizing the eggholder function, Shekel function, Michalewicz function and Hartmann function.

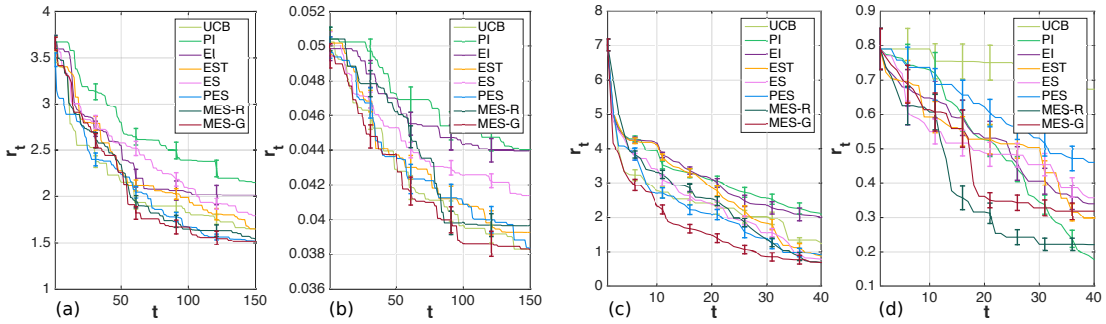


Figure 6-4: Tuning hyper-parameters for training a neural network, (a) Boston housing dataset; (b) breast cancer dataset. MES methods and PES perform better than other methods on (a), while for (b), MES-G, UCB, PES perform similarly and better than others. BO for active data selection on two robot pushing tasks for minimizing the distance to a random goal with (c) 3-D actions and (d) 4-D actions. MES methods perform better than other methods on the 3-D function. For the 4-D function, MES methods converge faster to a good regret, while PI achieves lower regret in the very end.

6.6.5 Active Learning for Robot Pushing

We use BO to do active learning for the pre-image learning problem for pushing [89]. The function we optimize takes as input the pushing action of the robot, and outputs the distance of the pushed object to the goal location. We use BO to minimize the function in order to find a good pre-image for pushing the object to the designated goal location. The first function we tested has a 3-dimensional input: robot location (r_x, r_y) and pushing duration t_r. We initialize the observation size to be one, the same across all methods. The second function has a 4-dimensional input: robot location and angle (r_x, r_y, r_θ), and pushing duration t_r. We initialize the observation to be 50 random points and set them the same for all the methods. We select 20 random goal locations for each function to test if BO can learn where to push for these locations. We show the best-sample simple regret in Fig. 6-4 (c,d) and the inference regret in Table 6.3. MES methods performed on a par with or better than their competitors.

METHOD	BOSTON	CANCER (%)	3-D ACTION	4-D ACTION
UCB	1.64 ± 0.43	3.83 ± 0.01	1.10 ± 0.66	0.56 ± 0.44
PI	2.15 ± 0.99	4.40 ± 0.01	2.03 ± 1.77	0.16 ± 0.20
EI	1.99 ± 1.03	4.40 ± 0.01	1.89 ± 1.87	0.30 ± 0.33
EST	1.65 ± 0.57	3.93 ± 0.01	0.70 ± 0.90	0.24 ± 0.17
ES	1.79 ± 0.61	4.14 ± 0.00	0.62 ± 0.59	0.25 ± 0.20
PES	1.52 ± 0.32	3.84 ± 0.01	0.81 ± 1.27	0.38 ± 0.38
MES-R	1.54 ± 0.56	3.96 ± 0.01	0.61 ± 1.23	0.16 ± 0.10
MES-G	1.51 ± 0.61	3.83 ± 0.01	0.61 ± 1.26	0.24 ± 0.25

Table 6.3: Inference regret R_T for tuning neural network hyper-parameters on the Boston housing and breast cancer datasets in Sec. 6.6.4 and for action selection in robot pushing in Sec. 6.6.5.

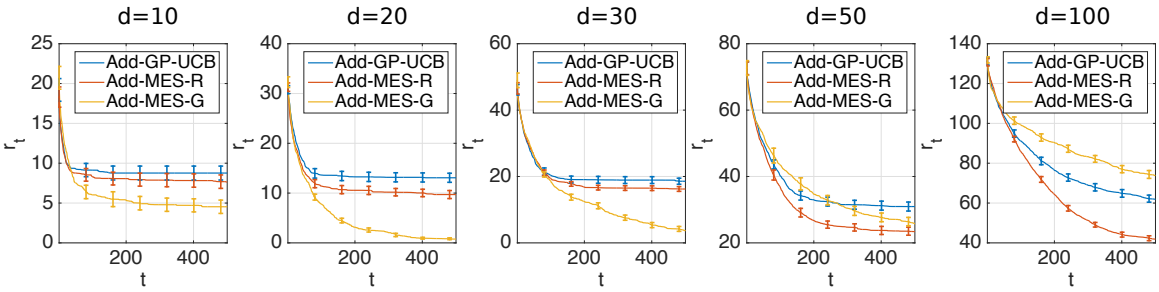


Figure 6-5: best-sample simple regrets for add-GP-UCB and add-MES methods on the synthetic add-GP functions. Both add-MES methods outperform add-GP-UCB except for add-MES-G on the input dimension $d = 100$. Add-MES-G achieves the lowest best-sample simple regret when d is relatively low, while for higher d add-MES-R becomes better than add-MES-G.

6.6.6 High Dimensional BO with Add-MES

In this section, we test our add-MES algorithm on high dimensional black-box function optimization problems. First we compare add-MES and add-GP-UCB [91] on a set of synthetic additive functions with known additive structure and GP hyper-parameters. Each function component of the synthetic additive function is active on at most three input dimensions, and is sampled from a GP with zero mean and Gaussian kernel (bandwidth = 0.1 and scale = 5). For the parameter of add-GP-UCB, we follow [91] and set $\beta_t^{(m)} = |A_m| \log 2t/5$. We set the number of $y_*^{(m)}$ sampled for each function component in add-MES-R and add-MES-G to be 1. We repeat each experiment for 50 times for each dimension setting. The results for best-sample simple regret are shown in Fig. 6-5. Add-MES methods perform much better than add-GP-UCB in terms of best-sample simple regret. Interestingly, add-MES-G works better in lower dimensional cases where $d = 10, 20, 30$, while add-MES-R outperforms both add-MES-G and add-GP-UCB for higher dimensions where $d = 50, 100$. In general, MES-G tends to overestimate the maximum of the function because of the independence assumption, and MES-R tends to underestimate the maximum of the function because of the imperfect global optimization of the posterior function samples. Hence, intuitively, MES-R

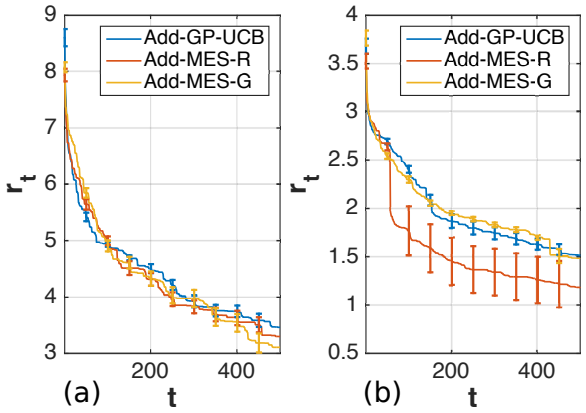


Figure 6-6: best-sample simple regrets for add-GP-UCB and add-MES methods on (a) a robot pushing task with 14 parameters and (b) a planar bipedal walker optimization task with 25 parameters. Both MES methods perform competitively comparing to add-GP-UCB.

may perform better for settings where exploitation is preferred over exploration (e.g., not too many local optima), and MES-G may work better if exploration is preferred.

To further verify the performance of add-MES in high dimensional problems, we test on two real-world high dimensional experiments. One is a function that returns the distance between a goal location and two objects being pushed by a robot which has 14 parameters. We implemented the function in Box2D [32]. The other function returns the walking speed of a planar bipedal robot, with 25 parameters to tune [196]. In Fig. 6-6, we show the best-sample simple regrets achieved by add-GP-UCB and add-MES. Add-MES methods performed competitively compared to add-GP-UCB on both tasks.

6.7 Conclusion

We explore a new family of acquisition functions in BO that are guided by the maximum *value* of the function to be optimized, instead of e.g. the maximizing argument. This new approach bridges historically significant ideas in BO including probability of improvement, upper confidence bounds and entropy search. Building upon the connections we show, we establish the first regret bounds for a variant of entropy search methods and probability of improvement. Empirically, on a variety of target functions, we show that our BO methods guided by max-values are significantly faster to compute than their entropy search counterparts and perform competitively compared to other state-of-the-art BO approaches.

One important implication of this chapter are the deep connections among existing BO methods,

and how they reduced to the simple PI approach. Although perhaps less recognized in the recent literature, it was visionary of [111] to use the simple yet elegant *probability of improvement* as the data acquisition criterion in the early days. This work made use of max-values to fill in the missing piece of how this improvement needs to be measured.

Chapter 7

Bayesian Optimization With Learned Priors

Bayesian optimization (BO) adopts a Bayesian perspective and assumes that there is a prior on the function; typically, we use a Gaussian process (GP) prior. Then, the information collection strategy can rely on the prior to focus on good inputs, where the goodness is determined by an acquisition function derived from the GP prior and current observations. In past literature, it has been shown both theoretically and empirically that if the function is indeed drawn from the given prior, there are many acquisition functions that BO can use to locate the function maximizer quickly [171, 23, 187].

However, in reality, the prior we choose to use in BO often does not reflect the distribution from which the function is drawn. Hence, we sometimes have to estimate the hyper-parameters of a chosen form of the prior *on the fly* as we collect more data [169]. One popular choice is to estimate the prior parameters using empirical Bayes with, e.g., the maximum likelihood estimator [151] .

Despite the vast literature that shows many empirical Bayes approaches have well-founded theoretical guarantees such as consistency [145] and admissibility [96], it is difficult to analyze a version of BO that uses empirical Bayes because of the circular dependencies between the estimated parameters and the data acquisition strategies. The requirement to select the prior model and estimate its parameters leads to a BO version of the chicken-and-egg dilemma: the prior model selection depends on the data collected and the data collection strategy depends on having a "correct" prior. Theoretically, there is little evidence that BO with unknown parameters in the prior can work well.

Zi Wang*, Beomjoon Kim*, and Leslie Pack Kaelbling. Regret bounds for meta Bayesian optimization with an unknown Gaussian process prior. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. (* indicates equal contribution.)

Empirically, there is evidence showing it works well in some situations, but not others [122, 90], which is not surprising in light of no free lunch results [198, 83].

In this chapter, we propose a simple yet effective strategy for learning a prior in a meta-learning setting where training data on functions from the same Gaussian process prior are available. We use a variant of empirical Bayes that gives unbiased estimates for both the parameters in the prior and the posterior given observations of the function we wish to optimize. We analyze the regret bounds in two settings: (1) finite input space, and (2) compact input space in $\mathbb{R}^d$. We clarify additional assumptions on the training data and form of Gaussian processes of both settings in Sec. 7.3.1 and Sec. 7.3.2. We prove theorems that show a near-zero regret bound for variants of GP-UCB [7, 171] and *probability of improvement* (PI) [111, 187]. The regret bound decreases to a constant proportional to the observational noise as online evaluations and offline data size increase.

From a more pragmatic perspective on Bayesian optimization for important areas such as robotics, we further explore how our approach works for problems in task and motion planning domains [97], and we explain why the assumptions in our theorems make sense for these problems in Sec. 7.4. Indeed, assuming a common kernel, such as squared exponential or Matérn, is very limiting for robotic problems that involve discontinuity and non-stationarity. However, with our approach of setting the prior and posterior parameters, BO outperforms all other methods in the task and motion planning benchmark problems.

The contributions of this chapter are (1) a stand-alone BO module that takes in only a multi-task training data set as input and then actively selects inputs to efficiently optimize a new function and (2) analysis of the regret of this module. The analysis is constructive, and determines appropriate hyperparameter settings for the GP-UCB acquisition function. Thus, we make a step forward to resolving the problem that, despite being used for hyperparameter tuning, BO algorithms themselves have hyperparameters.

7.1 Problem formulation and notations

Unlike the standard BO setting, we do not assume knowledge of the mean or covariance in the GP prior, but we do assume the availability of a dataset of iid sets of potentially non-iid observations on functions sampled from the same GP prior. Then, given a new, unknown function sampled from that same distribution, we would like to find its maximizer.

More formally, we assume there exists a distribution $GP(\mu, k)$, and both the mean $\mu : \mathfrak{X} \rightarrow$

$\mathbb{R}$ and the kernel $k : \mathfrak{X} \times \mathfrak{X} \to \mathbb{R}$ are unknown. Nevertheless, we are given a dataset $\bar{D}_N = \{[(\bar{x}_{ij}, \bar{y}_{ij})]_{j=1}^{M_i}\}_{i=1}^{N}$, where $\bar{y}_{ij}$ is drawn independently from $\mathcal{N}(f_i(\bar{x}_{ij}), \sigma^2)$ and $f_i : \mathfrak{X} \to \mathbb{R}$ is drawn independently from $GP(\mu, k)$. The noise level σ is unknown as well. We will specify inputs $\bar{x}_{ij}$ in Sec. 7.3.1 and Sec. 7.3.2.

Given a new function f sampled from $GP(\mu, k)$, our goal is to maximize it by sequentially querying the function and constructing $D_T = [(x_t, y_t)]_{t=1}^{T}$, $y_t \sim \mathcal{N}(f(x_t), \sigma^2)$. We study two evaluation criteria: (1) the *best-sample simple regret* $r_T = \max_{x \in \mathfrak{X}} f(x) - \max_{t \in [T]} f(x_t)$ which indicates the value of the best query in hindsight, and (2) the *simple regret*, $R_T = \max_{x \in \mathfrak{X}} f(x) - f(\hat{x}_T^*)$ which measures how good the inferred maximizer $\hat{x}_T^*$ is.

Notation We use $\mathcal{N}(u, V)$ to denote a multivariate Gaussian distribution with mean u and variance V and use $\mathcal{W}(V, n)$ to denote a Wishart distribution with n degrees of freedom and scale matrix V. We also use $[n]$ to denote $[1, \cdots, n], \forall n \in \mathbb{Z}^+$. We overload function notation for evaluations on vectors $\boldsymbol{x} = [x_i]_{i=1}^{n}, \boldsymbol{x}' = [x_j]_{j=1}^{n'}$ by denoting the output column vector as $\mu(\boldsymbol{x}) = [\mu(x_i)]_{i=1}^{n}$, and the output matrix as $k(\boldsymbol{x}, \boldsymbol{x}') = [k(x_i, x_j')]_{i \in [n], j \in [n']}$, and we overload the kernel function $k(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x})$.

7.2 Related work

BO optimizes a black-box objective function through sequential queries. We usually assume knowledge of a Gaussian process [151] prior on the function, though other priors such as Bayesian neural networks and their variants [63, 112] are applicable too. Then, given possibly noisy observations and the prior distribution, we can do Bayesian posterior inference and construct acquisition functions [111, 134, 7] to search for the function optimizer.

However, in practice, we do not know the prior and it must be estimated. One of the most popular methods of prior estimation in BO is to optimize mean/kernel hyper-parameters by maximizing data-likelihood of the current observations [151, 75]. Another popular approach is to put a prior on the mean/kernel hyper-parameters and obtain a distribution of such hyper-parameters to adapt the model given observations [78, 169]. These methods require a predetermined form of the mean function and the kernel function. In the existing literature, mean functions are usually set to be 0 or linear and the popular kernel functions include Matérn kernels, Gaussian kernels, linear kernels [151] or additive/product combinations of the above [51, 91].

Meta BO aims to improve the optimization of a given objective function by learning from

past experiences with other similar functions. Meta BO can be viewed as a special case of transfer learning or multi-task learning. One well-studied instance of meta BO is the machine learning (ML) hyper-parameter tuning problem on a dataset, where, typically, the validation errors are the functions to optimize [55]. The key question is how to transfer the knowledge from previous experiments on other datasets to the selection of ML hyper-parameters for the current dataset.

To determine the similarity between validation error functions on different datasets, meta-features of datasets are often used [26]. With those meta-features of datasets, one can use contextual Bayesian optimization approaches [105] that operate with a probabilistic functional model on both the dataset meta-features and ML hyper-parameters [14]. Feurer et al. [57], on the other hand, used meta-features of datasets to construct a distance metric, and to sort hyper-parameters that are known to work for similar datasets according to their distances to the current dataset. The best k hyper-parameters are then used to initialize a vanilla BO algorithm. If the function meta-features are not given, one can estimate the meta-features, such as the mean and variance of all observations, using Monte Carlo methods [174], maximum likelihood estimates [202] or maximum *a posteriori* estimates [148, 147].

As an alternative to using meta-features of functions, one can construct a kernel between functions. For functions that are represented by GPs, Malkomes et al. [128] studied a "kernel kernel", a kernel for kernels, such that one can use BO with a "kernel kernel" to select which kernel to use to model or optimize an objective function [127] in a Bayesian way. However, [128] requires an initial set of kernels to select from. Instead, Golovin et al. [67] introduced a setting where the functions come in sequence and the posterior of the former function becomes the prior of the current function. Removing the assumption that functions come sequentially, Feurer et al. [56] proposed a method to learn an additive ensemble of GPs that are known to fit all of those past "training functions".

Theoretically, it has been shown that meta BO methods that use information from similar functions may result in an improvement for the cumulative regret bound [105, 164] or the simple regret bound [147] with the assumptions that the GP priors are given. If the form of the GP kernel is given and the prior mean function is 0 but the kernel hyper-parameters are unknown, it is possible to obtain a regret bound given a range of these hyper-parameters [192]. In this chapter, we prove a regret bound for meta BO where the GP prior is unknown; this means, neither the range of GP hyper-parameters nor the form of the kernel or mean function is given.

A more ambitious approach to solving meta BO is to train an end-to-end system, such as a recurrent neural network [80], that takes the history of observations as an input and outputs the next

point to evaluate [35]. Though it has been demonstrated that the method in [35] can learn to trade-off exploration and exploitation for a short horizon, it is unclear how many "training instances", in the form of observations of BO performed on similar functions, are necessary to learn the optimization strategies for any given horizon of optimization. In this chapter, we show both theoretically and empirically how the number of "training instances" in our method affects the performance of BO.

Our methods are most similar to the BOX algorithm [97], which uses evaluations of previous functions to make point estimates of a mean and covariance matrix on the values over a discrete domain. Our methods for the discrete setting (described in Sec. 7.3.1) directly improve on BOX by choosing the exploration parameters in GP-UCB more effectively. This general strategy is extended to the continuous-domain setting in Sec. 7.3.2, in which we extend a method for learning the GP prior [146] and the use the learned prior in GP-UCB and PI.

Learning how to learn, or "meta learning", has a long history in machine learning [160]. It was argued that learning how to learn is "learning the prior" [17] with "point sets" [133], a set of iid sets of potentially non-iid points. We follow this simple intuition and present a meta BO approach that learns its GP prior from the data collected on functions that are assumed to have been drawn from the same prior distribution.

Empirical Bayes [152, 96] is a standard methodology for estimating unknown parameters of a Bayesian model. Our approach is a variant of empirical Bayes. We can view our computations as the construction of a sequence of estimators for a Bayesian model. The key difference from traditional empirical Bayes methods is that we are able to prove a regret bound for a BO method that uses estimated parameters to construct priors and posteriors. In particular, we use frequentist concentration bounds to analyze Bayesian procedures, which is one way to certify empirical Bayes in statistics [168, 53].

7.3 Meta BO and its theoretical guarantees

Instead of hand-crafting the mean μ and kernel k, we estimate them using the training dataset $\bar{D}_N$. Our approach is fairly straightforward: in the offline phase, the training dataset $\bar{D}_N$ is collected and we obtain estimates of the mean function $\hat{\mu}$ and kernel $\hat{k}$; in the online phase, we treat $GP(\hat{\mu}, \hat{k})$ as the Bayesian "prior" to do Bayesian optimization. We illustrate the two phases in Fig. 7-1. In Alg. 16, we depict our algorithm, assuming the dataset $\bar{D}_N$ has been collected. We use ESTIMATE($\bar{D}_N$) to denote the "prior" estimation and INFER($D_t; \hat{\mu}, \hat{k}$) the "posterior" inference, both of

Algorithm 16 Meta Bayesian optimization

function META-BO($\bar{D}_N, f$)
    $\hat{\mu}(\cdot), \hat{k}(\cdot,\cdot) \leftarrow$ ESTIMATE($\bar{D}_N$)
    return BO($f, \hat{\mu}, \hat{k}$)
end function

function BO ($f, \hat{\mu}, \hat{k}$)
    $D_0 \leftarrow \emptyset$
    for $t = 1, \cdots, T$ **do**
        $\hat{\mu}_{t-1}(\cdot), \hat{k}_{t-1}(\cdot) \leftarrow$ INFER($D_{t-1}; \hat{\mu}, \hat{k}$)
        $\alpha_{t-1}(\cdot) \leftarrow$ACQUISITION ($\hat{\mu}_{t-1}, \hat{k}_{t-1}$)
        $x_t \leftarrow \arg\max_{x \in \mathfrak{X}} \alpha_{t-1}(x)$
        $y_t \leftarrow$ OBSERVE($f(x_t)$)
        $D_t \leftarrow D_{t-1} \cup [(x_t, y_t)]$
    end for
    return D_T
end function

which we will introduce in Sec. 7.3.1 and Sec. 7.3.2. For acquisition functions, we consider special cases of *probability of improvement* (PI) [187, 111] and *upper confidence bound* (GP-UCB) [171, 7]:

$$\alpha^{\text{PI}}_{t-1}(x) = \frac{\hat{\mu}_{t-1}(x) - \hat{f}^*}{\hat{k}_{t-1}(x)^{\frac{1}{2}}}, \quad \alpha^{\text{GP-UCB}}_{t-1}(x) = \hat{\mu}_{t-1}(x) + \zeta_t \hat{k}_{t-1}(x)^{\frac{1}{2}}.$$

Here, PI assumes additional information[1] in the form of the upper bound on function value $\hat{f}^* \geq \max_{x \in \mathfrak{X}} f(x)$. For GP-UCB, we set its hyperparameter ζ_t to be

$$\zeta_t = \frac{\left(6(N - 3 + t + 2\sqrt{t \log \frac{6}{\delta}} + 2 \log \frac{6}{\delta})/(\delta N(N-t-1))\right)^{\frac{1}{2}} + (2 \log(\frac{3}{\delta}))^{\frac{1}{2}}}{(1 - 2(\frac{1}{N-t} \log \frac{6}{\delta})^{\frac{1}{2}})^{\frac{1}{2}}},$$

where N is the size of the dataset $\bar{D}_N$ and $\delta \in (0, 1)$. With probability $1 - \delta$, the regret bound in Thm. 7.3.2 or Thm. 7.3.4 holds with these special cases of GP-UCB and PI. Under two different settings of the search space $\mathfrak{X}$, finite $\mathfrak{X}$ and compact $\mathfrak{X} \in \mathbb{R}^d$, we show how our algorithm works in detail and why it works via regret analyses on the best-sample simple regret. Finally in Sec. 7.3.3 we show how the simple regret can be bounded. **The proofs of the analyses can be found in Appendix A.**

[1] Alternatively, an upper bound $\hat{f}^*$ can be estimated adaptively [187]. Note that here we are maximizing the PI acquisition function and hence $\alpha^{\text{PI}}_{t-1}(x)$ is a negative version of what was defined in [187].

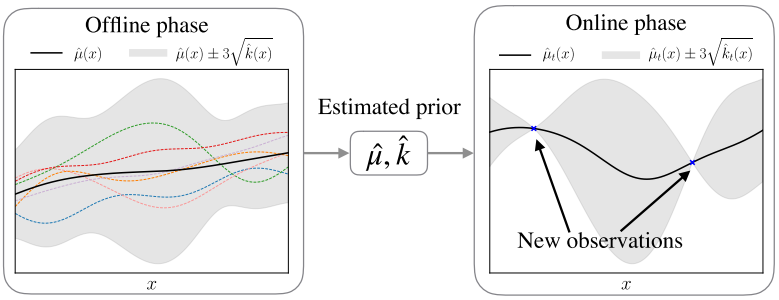


Figure 7-1: Our approach estimates the mean function $\hat{\mu}$ and kernel $\hat{k}$ from functions sampled from $GP(\mu, k)$ in the offline phase. Those sampled functions are illustrated by colored lines. In the online phase, a new function f sampled from the same $GP(\mu, k)$ is given and we can estimate its posterior mean function $\hat{\mu}_t$ and covariance function $\hat{k}_t$ which will be used for Bayesian optimization.

7.3.1 Function domain is a finite set

We first study the simplest case, where the function domain $\mathfrak{X} = [\bar{x}_j]_{j=1}^M$ is a finite set with cardinality $|\mathfrak{X}| = M \in \mathbb{Z}^+$. For convenience, we treat this set as an ordered vector of items indexed by $j \in [M]$. We collect the training dataset $\bar{D}_N = \{[(\bar{x}_j, \bar{\delta}_{ij}\bar{y}_{ij})]_{j=1}^M\}_{i=1}^N$, where $\bar{y}_{ij}$ are independently drawn from $\mathcal{N}(f_i(\bar{x}_j), \sigma^2)$, f_i are drawn independently from $GP(\mu, k)$ and $\bar{\delta}_{ij} \in \{0, 1\}$. Because the training data can be collected offline by querying the functions $\{f_i\}_{i=1}^N$ in parallel, it is not unreasonable to assume that such a dataset $\bar{D}_N$ is available. If $\bar{\delta}_{ij} = 0$, it means the (i, j)-th entry of the dataset $\bar{D}_N$ is missing, perhaps as a result of a failed experiment.

Estimating GP parameters If $\bar{\delta}_{ij} < 1$, we have missing entries in the observation matrix $\bar{Y} = [\bar{\delta}_{ij}\bar{y}_{ij}]_{i\in[N],j\in[M]} \in \mathbb{R}^{N\times M}$. Under additional assumptions specified in [31], including that $\text{rank}(Y) = r$ and the total number of valid observations $\sum_{i=1}^N \sum_{j=1}^M \bar{\delta}_{ij} \geq O(rN^{\frac{6}{5}} \log N)$, we can use matrix completion [31] to fully recover the matrix $\bar{Y}$ with high probability. In the following, we proceed by considering completed observations only.

Let the completed observation matrix be $Y = [\bar{y}_{ij}]_{i\in[N],j\in[M]}$. We use an unbiased sample mean and covariance estimator for μ and k; that is, $\hat{\mu}(\mathfrak{X}) = \frac{1}{N} Y^{\mathrm{T}} 1_N$ and $\hat{k}(\mathfrak{X}) = \frac{1}{N-1}(Y - 1_N \hat{\mu}(\mathfrak{X})^{\mathrm{T}})^{\mathrm{T}}(Y - 1_N \hat{\mu}(\mathfrak{X})^{\mathrm{T}})$, where 1_N is an N by 1 vector of ones. It is well known that $\hat{\mu}$ and $\hat{k}$ are independent [4] and

$$\hat{\mu}(\mathfrak{X}) \sim \mathcal{N}(\mu(\mathfrak{X}), \frac{1}{N}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I})), \quad \hat{k}(\mathfrak{X}) \sim \mathcal{W}(\frac{1}{N-1}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I}), N - 1).$$

Constructing estimators of the posterior Given noisy observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, we

can do Bayesian posterior inference to obtain $f \sim GP(\mu_t, k_t)$. By the GP assumption, we get

$$\mu_t(x) = \mu(x) + k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \mu(\boldsymbol{x}_t)), \quad \forall x \in \mathfrak{X} \tag{7.1}$$

$$k_t(x, x') = k(x, x') - k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1} k(\boldsymbol{x}_t, x'), \quad \forall x, x' \in \mathfrak{X}, \tag{7.2}$$

where $\boldsymbol{y}_t = [y_\tau]_{\tau=1}^T$, $\boldsymbol{x}_t = [x_\tau]_{\tau=1}^T$ [151]. The problem is that neither the posterior mean μ_t nor the covariance k_t are computable because the Bayesian prior mean μ, the kernel k and the noise parameter σ are all unknown. How to estimate μ_t and k_t without knowing those prior parameters?

We introduce the following unbiased estimators for the posterior mean and covariance,

$$\hat{\mu}_t(x) = \hat{\mu}(x) + \hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t, \boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t)), \quad \forall x \in \mathfrak{X}, \tag{7.3}$$

$$\hat{k}_t(x, x') = \frac{N-1}{N-t-1}\left(\hat{k}(x, x') - \hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t, \boldsymbol{x}_t)^{-1}\hat{k}(\boldsymbol{x}_t, x')\right), \quad \forall x, x' \in \mathfrak{X}. \tag{7.4}$$

Notice that unlike Eq. (7.1) and Eq. (7.2), our estimators $\hat{\mu}_t$ and $\hat{k}_t$ do not depend on any unknown values or an additional estimate of the noise parameter σ. In Lemma 7.3.1, we show that our estimators are indeed unbiased and we derive their concentration bounds.

Lemma 7.3.1. *Pick probability $\delta \in (0, 1)$. For any nonnegative integer $t < T$, conditioned on the observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, the estimators in Eq.* (7.3) *and Eq.* (7.4) *satisfy $\mathbb{E}[\hat{\mu}_t(\mathfrak{X})] = \mu_t(\mathfrak{X}), \mathbb{E}[\hat{k}_t(\mathfrak{X})] = k_t(\mathfrak{X}) + \sigma^2 \boldsymbol{I}$. Moreover, if the size of the training dataset satisfies $N \geq T + 2$, then for any input $x \in \mathfrak{X}$, with probability at least $1 - \delta$, both*

$$|\hat{\mu}_t(x) - \mu_t(x)|^2 < a_t(k_t(x) + \sigma^2) \;\; \textit{and} \;\; 1 - 2\sqrt{b_t} < \hat{k}_t(x)/(k_t(x) + \sigma^2) < 1 + 2\sqrt{b_t} + 2b_t$$

hold, where $a_t = \frac{4\left(N-2+t+2\sqrt{t\log(4/\delta)}+2\log(4/\delta)\right)}{\delta N(N-t-2)}$ and $b_t = \frac{1}{N-t-1}\log\frac{4}{\delta}$.

Regret bounds We show a near-zero upper bound on the best-sample simple regret of meta BO with GP-UCB and PI that uses specific parameter settings in Thm. 7.3.2. In particular, for both GP-UCB and PI, the regret bound converges to a residual whose scale depends on the noise level σ in the observations.

Theorem 7.3.2. *Assume there exists constant $c \geq \max_{x \in \mathfrak{X}} k(x)$ and a training dataset is available whose size is $N \geq 4\log\frac{6}{\delta} + T + 2$. Then, with probability at least $1 - \delta$, the best-sample simple*

regret in T iterations of meta BO with special cases of either GP-UCB or PI satisfies

$$r_T^{UCB} < \eta_T^{UCB}(N)\lambda_T, \;\; r_T^{PI} < \eta_T^{PI}(N)\lambda_T, \;\; \lambda_T^2 = O(\rho_T/T) + \sigma^2,$$

where $\eta_T^{UCB}(N) = (m+C_1)(\frac{\sqrt{1+m}}{\sqrt{1-m}}+1)$, $\eta_T^{PI}(N) = (m+C_2)(\frac{\sqrt{1+m}}{\sqrt{1-m}}+1)+C_3$, $m = O(\sqrt{\frac{1}{N-T}})$, $C_1, C_2, C_3 > 0$ *are constants, and* $\rho_T = \max_{A\in\mathfrak{X},|A|=T} \frac{1}{2}\log|\boldsymbol{I} + \sigma^{-2}k(A)|$.

This bound reflects how training instances N and BO iterations T affect the best-sample simple regret. The coefficients η_T^{UCB} and η_T^{PI} both converge to constants (more details in Appendix A, with components converging at rate $O(1/(N-T)^{\frac{1}{2}})$. The convergence of the shared term λ_T depends on ρ_T, the maximum information gain between function f and up to T observations $\boldsymbol{y}_T$. If, for example, each input has dimension $\mathbb{R}^d$ and $k(x, x') = x^{\mathrm{T}}x'$, then $\rho_T = O(d\log(T))$ [171], in which case λ_T converges to the observational noise level σ at rate $O(\sqrt{\frac{d\log(T)}{T}})$. Together, the bounds indicate that the best-sample simple regret of both our settings of GP-UCB and PI decreases to a constant proportional to noise level σ.

7.3.2 Function domain is compact

For compact $\mathfrak{X} \subset \mathbb{R}^d$, we consider the primal form of GPs. We further assume that there exist basis functions $\Phi = [\phi_s]_{s=1}^K : \mathfrak{X} \to \mathbb{R}^K$, mean parameter $\boldsymbol{u} \in \mathbb{R}^K$ and covariance parameter $\Sigma \in \mathbb{R}^{K\times K}$ such that $\mu(x) = \Phi(x)^{\mathrm{T}}\boldsymbol{u}$ and $k(x, x') = \Phi(x)^{\mathrm{T}}\Sigma\Phi(x')$. Notice that $\Phi(x) \in \mathbb{R}^K$ is a column vector and $\Phi(\boldsymbol{x}_t) \in \mathbb{R}^{K\times t}$ for any $\boldsymbol{x}_t = [x_\tau]_{\tau=1}^t$. This means, for any input $x \in \mathfrak{X}$, the observation satisfies $y \sim \mathcal{N}(f(x), \sigma^2)$, where $f = \Phi(x)^{\mathrm{T}}W \sim GP(\mu, k)$ and the linear operator $W \sim \mathcal{N}(\boldsymbol{u}, \Sigma)$ [139]. In the following analyses, we assume the basis functions Φ are given.

We assume that a training dataset $\bar{D}_N = \{[(\bar{x}_j, \bar{y}_{ij})]_{j=1}^M\}_{i=1}^N$ is given, where $\bar{x}_j \in \mathfrak{X} \subset \mathbb{R}^d$, y_{ij} are independently drawn from $\mathcal{N}(f_i(\bar{x}_j), \sigma^2)$, f_i are drawn independently from $GP(\mu, k)$ and $M \geq K$.

Estimating GP parameters Because the basis functions Φ are given, learning the mean function μ and the kernel k in the GP is equivalent to learning the mean parameter $\boldsymbol{u}$ and the covariance parameter Σ that parameterize distribution of the linear operator W. Notice that $\forall i \in [N]$,

$$\bar{\boldsymbol{y}}_i = \Phi(\bar{\boldsymbol{x}})^{\mathrm{T}}W_i + \bar{\boldsymbol{\epsilon}}_i \sim \mathcal{N}(\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{u}, \Phi(\bar{\boldsymbol{x}})^{\mathrm{T}}\Sigma\Phi(\bar{\boldsymbol{x}}) + \sigma^2\boldsymbol{I}),$$

where $\bar{\boldsymbol{y}}_i = [\bar{y}_{ij}]_{j=1}^M \in \mathbb{R}^M$, $\bar{\boldsymbol{x}} = [\bar{x}_j]_{j=1}^M \in \mathbb{R}^{M\times d}$ and $\bar{\boldsymbol{\epsilon}}_i = [\bar{\epsilon}_{ij}]_{j=1}^M \in \mathbb{R}^M$. If the matrix

$\Phi(\bar{\boldsymbol{x}}) \in \mathbb{R}^{K\times M}$ has linearly independent rows, one unbiased estimator of W_i is

$$\hat{W}_i = (\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{+}\bar{\boldsymbol{y}}_i = (\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{-1}\Phi(\bar{\boldsymbol{x}})\bar{\boldsymbol{y}}_i \sim \mathcal{N}(\boldsymbol{u}, \Sigma + \sigma^2(\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{-1}).$$

Let $\mathsf{W} = [\hat{W}_i]_{i=1}^N \in \mathbb{R}^{N\times K}$. We use the estimator $\hat{\boldsymbol{u}} = \frac{1}{N}\mathsf{W}^{\mathrm{T}}1_N$ and $\hat{\Sigma} = \frac{1}{N-1}(\mathsf{W} - 1_N\hat{\boldsymbol{u}})^{\mathrm{T}}(\mathsf{W} - 1_N\hat{\boldsymbol{u}})$ to the estimate GP parameters. Again, $\hat{\boldsymbol{u}}$ and $\hat{\Sigma}$ are independent and $\hat{\boldsymbol{u}} \sim \mathcal{N}\left(\boldsymbol{u}, \frac{1}{N}(\Sigma + \sigma^2(\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{-1})\right), \hat{\Sigma} \sim \mathcal{W}\left(\frac{1}{N-1}\left(\Sigma + \sigma^2(\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{-1}\right), N-1\right)$ [4].

Constructing estimators of the posterior We assume the total number of evaluations $T < K$. Given noisy observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, we have $\mu_t(x) = \Phi(x)^{\mathrm{T}}\boldsymbol{u}_t$ and $k_t(x, x') = \Phi(x)^{\mathrm{T}}\Sigma_t\Phi(x')$, where the posterior of $W \sim \mathcal{N}(\boldsymbol{u}_t, \Sigma_t)$ satisfies

$$\boldsymbol{u}_t = \boldsymbol{u} + \Sigma\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\Sigma\Phi(\boldsymbol{x}_t) + \sigma^2\boldsymbol{I})^{-1}(\boldsymbol{y}_t - \Phi(\boldsymbol{x}_t)^{\mathrm{T}}\boldsymbol{u}), \tag{7.5}$$

$$\Sigma_t = \Sigma - \Sigma\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\Sigma\Phi(\boldsymbol{x}_t) + \sigma^2\boldsymbol{I})^{-1}\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\Sigma. \tag{7.6}$$

Similar to the strategy used in Sec. 7.3.1, we construct an estimator for the posterior of W to be

$$\hat{\boldsymbol{u}}_t = \hat{\boldsymbol{u}} + \hat{\Sigma}\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\hat{\Sigma}\Phi(\boldsymbol{x}_t))^{-1}(\boldsymbol{y}_t - \Phi(\boldsymbol{x}_t)^{\mathrm{T}}\boldsymbol{u}), \tag{7.7}$$

$$\hat{\Sigma}_t = \frac{N-1}{N-t-1}\left(\hat{\Sigma} - \hat{\Sigma}\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\hat{\Sigma}\Phi(\boldsymbol{x}_t))^{-1}\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\hat{\Sigma}\right). \tag{7.8}$$

We can compute the conditional mean and variance of the observation on $x \in \mathfrak{X}$ to be $\hat{\mu}_t(x) = \Phi(x)^{\mathrm{T}}\hat{\boldsymbol{u}}_t$ and $\hat{k}_t(x) = \Phi(x)^{\mathrm{T}}\hat{\Sigma}_t\Phi(x)$. For convenience of notation, we define

$$\bar{\sigma}^2(x) = \sigma^2\Phi(x)^{\mathrm{T}}(\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{-1}\Phi(x).$$

Lemma 7.3.3. *Pick probability $\delta \in (0,1)$. Assume $\Phi(\bar{\boldsymbol{x}})$ has full row rank. For any nonnegative integer $t < T$, $T \le K$, conditioned on the observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, $\mathbb{E}[\hat{\mu}_t(x)] = \mu_t(x), \mathbb{E}[\hat{k}_t(x)] = k_t(x) + \bar{\sigma}^2(x)$. Moreover, if the size of the training dataset satisfies $N \ge T + 2$, then for any input $x \in \mathfrak{X}$, with probability at least $1-\delta$, both*

$$|\hat{\mu}_t(x) - \mu_t(x)|^2 < a_t(k_t(x) + \bar{\sigma}^2(x))$$

and

$$1 - 2\sqrt{b_t} < \hat{k}_t(x)/(k_t(x) + \bar{\sigma}^2(x)) < 1 + 2\sqrt{b_t} + 2b_t$$

hold, where $a_t = \frac{4\left(N-2+t+2\sqrt{t\log(4/\delta)}+2\log(4/\delta)\right)}{\delta N(N-t-2)}$ *and* $b_t = \frac{1}{N-t-1}\log\frac{4}{\delta}$.

Regret bounds Similar to the finite $\mathfrak{X}$ case, we can also show a near-zero regret bound for compact $\mathfrak{X} \in \mathbb{R}^d$. The following theorem clarifies our results. The convergence rates are the same as Thm. 7.3.2. Note that λ_T^2 converges to $\bar{\sigma}^2(\cdot)$ instead of σ^2 in Thm. 7.3.2 and $\bar{\sigma}^2(\cdot)$ is proportional to σ^2 .

Theorem 7.3.4. *Assume all the assumptions in Thm. 7.3.2 and that* $\Phi(\bar{\boldsymbol{x}})$ *has full row rank. With probability at least* $1-\delta$*, the best-sample simple regret in* T *iterations of meta BO with either GP-UCB or PI satisfies*

$$r_T^{UCB} < \eta_T^{UCB}(N)\lambda_T, \;\; r_T^{PI} < \eta_T^{PI}(N)\lambda_T, \;\; \lambda_T^2 = O(\rho_T/T) + \bar{\sigma}(x_\tau)^2,$$

where $\eta_T^{UCB}(N) = (m+C_1)(\frac{\sqrt{1+m}}{\sqrt{1-m}}+1)$, $\eta_T^{PI}(N) = (m+C_2)(\frac{\sqrt{1+m}}{\sqrt{1-m}}+1)+C_3$, $m = O(\sqrt{\frac{1}{N-T}})$, $C_1, C_2, C_3 > 0$ *are constants,* $\tau = \arg\min_{t\in[T]} k_{t-1}(x_t)$ *and* $\rho_T = \max_{A\in\mathfrak{X},|A|=T} \frac{1}{2}\log|\boldsymbol{I}+\sigma^{-2}k(A)|$.

7.3.3 Bounding the simple regret by the best-sample simple regret

Once we have the observations $D_T = \{(x_t, y_t)\}_{t=1}^T$, we can infer where the $\arg\max$ of the function is. For all the cases in which $\mathfrak{X}$ is discrete or compact and the acquisition function is GP-UCB or PI, we choose the inferred $\arg\max$ to be $\hat{x}_T^* = x_\tau$ where $\tau = \arg\max_{\tau\in[T]} y_\tau$. We show in Lemma 7.3.5 that with high probability, the difference between the simple regret R_T and the best-sample simple regret r_T is proportional to the observation noise σ.

Lemma 7.3.5. *With probability at least* $1-\delta$*,* $R_T \leq r_T + 2(2\log\frac{1}{\delta})^{\frac{1}{2}}\sigma$.

Together with the bounds on the best-sample simple regret from Thm. 7.3.2 and Thm. 7.3.4, our result shows that, with high probability, the simple regret decreases to a constant proportional to the noise level σ as the number of iterations and training functions increases.

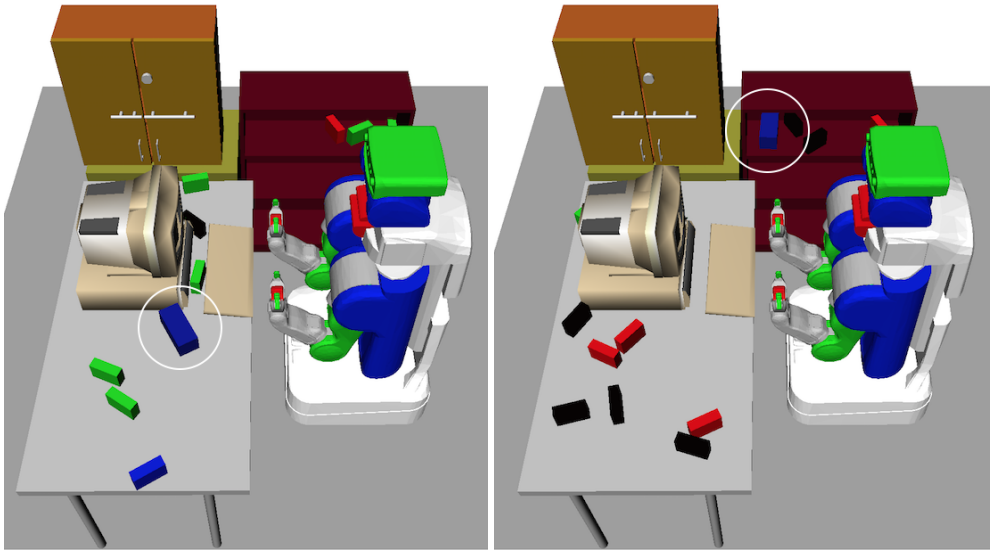


Figure 7-2: Two instances of a picking problem. A problem instance is defined by the arrangement and number of obstacles, which vary randomly across different instances. The objective is to select a grasp that can pick the blue box, marked with a circle, without violating kinematic and collision constraints. [97].

7.4 Experiments

We evaluate our algorithm in four different black-box function optimization problems, involving discrete or continuous function domains. One problem is optimizing a synthetic function in $\mathbb{R}^2$, and the rest are optimizing decision variables in robotic task and motion planning problems that were used in [97][2].

At a high level, our task and motion planning benchmarks involve computing kinematically feasible collision-free motions for picking and placing objects in a scene cluttered with obstacles. This problem has a similar setup to experimental design: the robot can "experiment" by assigning values to decision variables including grasps, base poses, and object placements until it finds a feasible plan. Given the assigned values for these variables, the robot program makes a call to a planner[3] which then attempts to find a sequence of motions that achieve these grasps and placements. We score the variable assignment based on the results of planning, assigning a very low score if the problem was infeasible and otherwise scoring based on plan length or obstacle clearance. An example problem is given in Figure 7-2.

Planning problem instances are characterized by arrangements of obstacles in the scene and the

[2]Our code is available at `https://github.com/beomjoonkim/MetaLearnBO`.

[3]We use Rapidly-exploring random tree (RRT) [118] with predefined random seed, but other choices are possible.

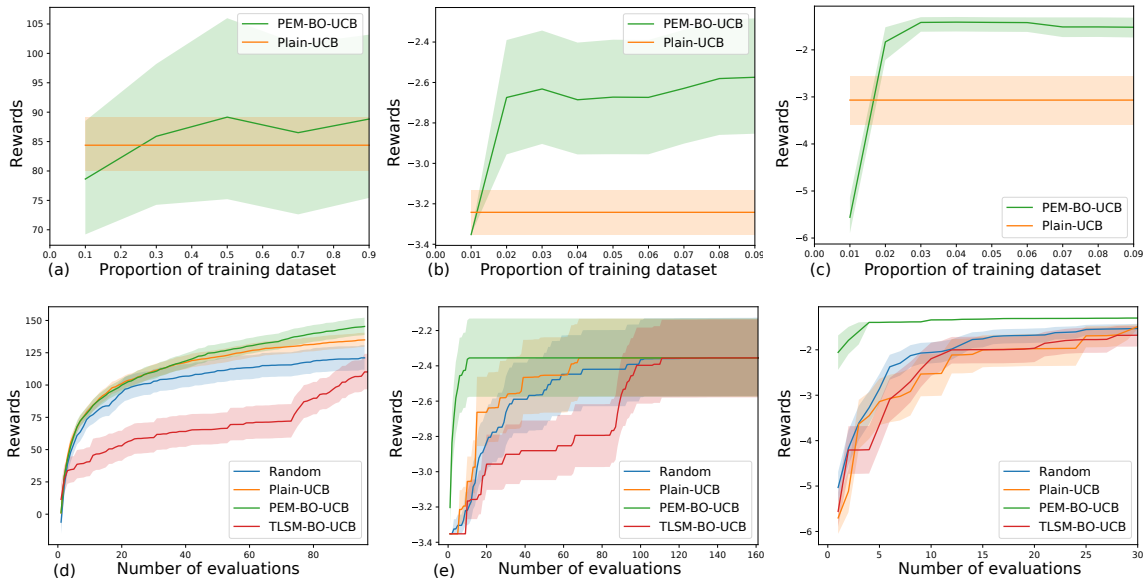


Figure 7-3: Learning curves (top) and rewards vs number of iterations (bottom) for optimizing synthetic functions sampled from a GP and two scoring functions from.

shape of the target object to be manipulated, and each problem instance defines a different score function. Our objective is to optimize the score function for a new problem instance, given sets of decision-variable and score pairs from a set of previous planning problem instances as training data.

In two robotics domains, we discretize the original function domain using samples from the past planning experience, by extracting the values of the decision variables and their scores from successful plans. This is inspired by the previous successful use of BO in a discretized domain [40] to efficiently solve an adaptive locomotion problem.

We compare our approach, called *point estimate meta Bayesian optimization* (PEM-BO), to three baseline methods. The first is a plain Bayesian optimization method that uses a kernel function to represent the covariance matrix, which we call Plain. Plain optimizes its GP hyperparameters by maximizing the data likelihood. The second is a *transfer learning sequential model-based optimization* [202] method, that, like PEM-BO, uses past function evaluations, but assumes that functions sampled from the same GP have similar response surface values. We call this method TLSM-BO. The third is random selection, which we call Random. We present the results on the UCB acquisition function here and results on the PI acquisition function are available in the appendix.

In all domains, we use the ζ_t value as specified in Sec. 7.3. For continuous domains, we use $\Phi(x) = [\cos(x^T\beta^{(i)} + \beta_0^{(i)})]_{i=1}^K$ as our basis functions. In order to train the weights $W_i, \beta^{(i)}$, and $\beta_0^{(i)}$, we represent the function $\Phi(x)^T W_i$ with a 1-hidden-layer neural network with cosine activation function and a linear output layer with function-specific weights W_i. We then train this

network on the entire dataset $\bar{D}_N$. Then, fixing $\Phi(x)$, for each set of pairs $(\bar{\boldsymbol{y}}_i, \bar{\boldsymbol{x}}_i), i = \{1 \cdots N\}$, we analytically solve the linear regression problem $\boldsymbol{y}_i \approx \Phi(\boldsymbol{x}_i)^T W_i$ as described in Sec. 7.3.2.

For Plain and TLSM-BO with UCB in our experiments, we used the same ζ_t as PEM-BO.

7.4.1 Optimizing a continuous synthetic function

In this problem, the objective is to optimize a black-box function sampled from a GP, whose domain is $\mathbb{R}^2$, given a set of evaluations of different functions from the same GP. Specifically, we consider a GP with a squared exponential kernel function. The purpose of this problem is to show that PEM-BO, which estimates mean and covariance matrix based on $\bar{D}_N$, would perform similarly to BO methods that start with an appropriate prior. We have training data from $N = 100$ functions with $M = 1000$ sample points each.

Figure 7-3(a) shows the learning curve, when we have different portions of data. The x-axis represents the percentage of the dataset used to train the basis functions, $\boldsymbol{u}$, and W from the training dataset, and the y-axis represents the best function value found after 10 evaluations on a new function. We can see that even with just ten percent of the training data points, PEM-BO performs just as well as Plain, which uses the appropriate kernel for this particular problem. Compared to PEM-BO, which can efficiently use all of the dataset, we had to limit the number of training data points for TLSM-BO to 1000, because even performing inference requires $O(NM)$ time. This leads to its noticeably worse performance than Plain and PEM-BO.

Figure 7-3(d) shows the how $\max_{t \in [T]} y_t$ evolves, where $T \in [1, 100]$. As we can see, PEM-BO using the UCB acquisition function performs similarly to Plain with the same acquisition function. TLSM-BO again suffers because we had to limit the number of training data points.

7.4.2 Optimizing a grasp

In the robot-planning problem shown in Figure 7-2, the robot has to choose a grasp for picking the target object in a cluttered scene. A planning problem instance is defined by the poses of obstacles and the target objects, which changes the feasibility of a grasp across different instances.

The reward function is the negative of the length of the picking motion if the motion is feasible, and $-k \in \mathbb{R}$ otherwise, where $-k$ is a suitably lower number than the lengths of possible trajectories. We construct the discrete set of grasps by using grasps that worked in the past planning problem instances. The original space of grasps is $\mathbb{R}^{58}$, which describes position, direction, roll, and depth of a robot gripper with respect to the object, as used in [48]. For both Plain and TLSM-BO,

we use squared exponential kernel function on this original grasp space to represent the covariance matrix. We note that this is a poor choice of kernel, because the grasp space includes angles, making it a non-vector space. These methods also choose a grasp from the discrete set. We train on dataset with $N = 1800$ previous problems, and let $M = 162$.

Figure 7-3(b) shows the learning curve with $T = 5$. The x-axis is the percentage of the dataset used for training, ranging from one percent to ten percent. Initially, when we just use one percent of the training data points, PEM-BO performs as poorly as TLSM-BO, which again, had only 1000 training data points. However, PEM-BO outperforms both TLSM-BO and Plain after that. The main reason that PEM-BO outperforms these approaches is because their prior, which is defined by the squared exponential kernel, is not suitable for this problem. PEM-BO, on the other hand, was able to avoid this problem by estimating a distribution over values at the discrete sample points that commits only to their joint normality, but not to any metric on the underlying space. These trends are also shown in Figure 7-3(e), where we plot $\max_{t \in [T]} y_t$ for $T \in [1, 100]$. PEM-BO outperforms the baselines significantly.

7.4.3 Optimizing a grasp, base pose, and placement

We now consider a more difficult task that involves both picking and placing objects in a cluttered scene. A planning problem instance is defined by the poses of obstacles and the poses and shapes of the target object to be pick and placed. The reward function is again the negative of the length of the picking motion if the motion is feasible, and $-k \in \mathbb{R}$ otherwise. For both Plain and TLSM-BO, we use three different squared exponential kernels on the original spaces of grasp, base pose, and object placement pose respectively and then add them together to define the kernel for the whole set. For this domain, $N = 1500$, and $M = 1000$.

Figure 7-3(c) shows the learning curve, when $T = 5$. The x-axis is the percentage of the dataset used for training, ranging from one percent to ten percent. Initially, when we just use one percent of the training data points, PEM-BO does not perform well. Similar to the previous domain, it then significantly outperforms both TLSM-BO and Plain after increasing the training data. This is also reflected in Figure 7-3(f), where we plot $\max_{t \in [T]} y_t$ for $T \in [1, 100]$. PEM-BO outperforms baselines. Notice that Plain and TLSM-BO perform worse than Random, as a result of making inappropriate assumptions on the form of the kernel.

7.4.4 Sensitivity to missing data

In the following, we include more experiments that we performed with PI acquisition function and matrix completion for the missing entry case in the discrete domains. The PI approach uses the maximum function value in the training dataset $\bar{D}_N$ as the target value. These results show that our approach is resilient to missing data. BO with the PI acquisition function performs similarly to UCB.

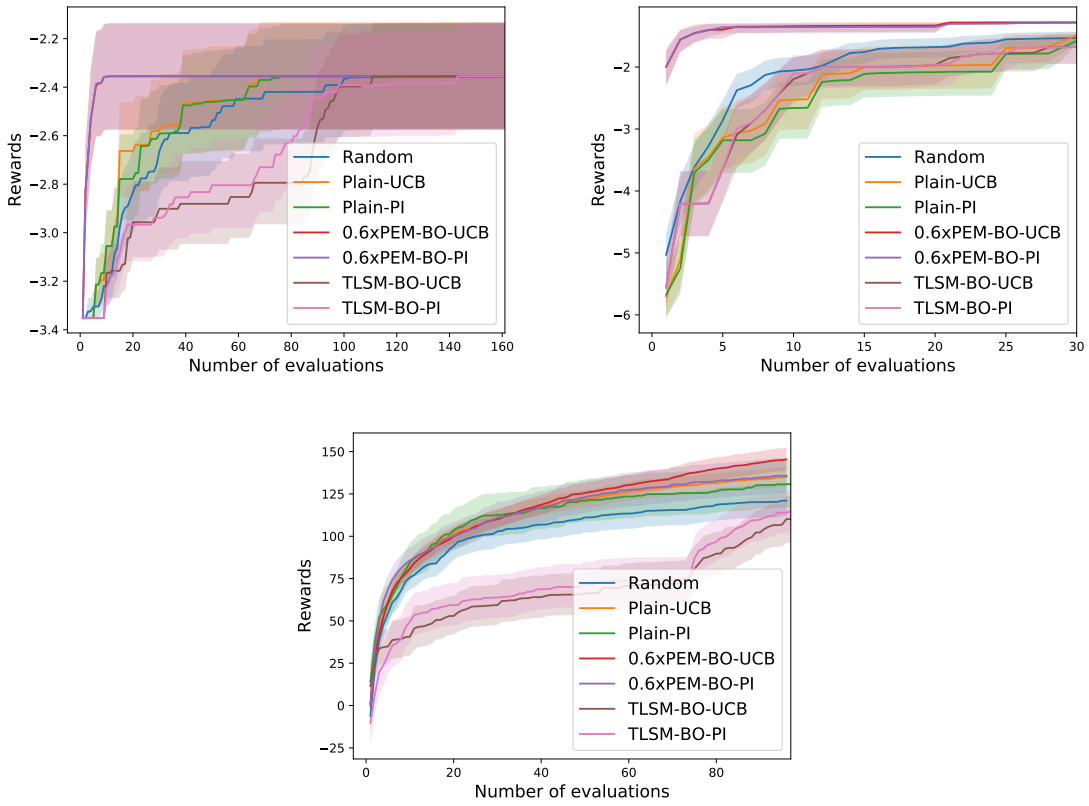


Figure 7-4: Rewards vs. Number of evals for grasp optimization, grasp, base pose, and placement optimization, and synthetic function optimization problems (from top-left to bottom). 0.6xPEM-BO refers to the case where we have 60 percent of the dataset missing.

7.5 Discussions and conclusions

In this section, we discuss related topics to our approach. Both theoreticians and practitioners may find this section useful in terms of clarifying theoretical insights and precautions.

7.5.1 Connections and differences to empirical Bayes

In classic empirical Bayes [152, 96], we estimate the unknown parameters of the Bayesian model and usually use a point estimate to proceed any Bayesian computations. One very popular approach to estimate those unknown parameters is by maximizing the data likelihood. There also exit other variants of empirical Bayes; for example, oracle Bayes, which "shows empirical Bayes in its most frequentist mode" [53].

In this chapter, we use a variant of empirical Bayes that constructs estimators for both the prior distribution and the posterior distribution. For the estimators of the posterior, we do not use a plug-in estimate like classic empirical Bayes but we construct them through Lemma. A.1.6, which establishes the unbiasedness and concentration bounds for those estimates.

7.5.2 Connections and differences to hierarchical Bayes

Hierarchical Bayes is a Bayesian hierarchical model that places priors on priors. For both of our finite $\mathfrak{X}$ case and continuous and compact $\mathfrak{X} \in \mathbb{R}^d$ case, we can write down a hierarchical Bayes model that puts a normal inverse Wishart prior on $\mu(\mathfrak{X}), k(\mathfrak{X})$ or $\boldsymbol{u}, \Sigma$.

Our approach can be viewed as a special case of the hierarchical Bayes model using point estimates to approximate the posterior. Neither our estimators nor our regret analyses depend on the prior parameters of those hierarchical Bayes models. But one may analyze the regret of BO with a better approximation from a full Bayesian perspective using hierarchical Bayes.

7.5.3 Future directions

Due to the limited space, we only give the formulation of meta BO in its simple and basic settings. Our setting restricts the evaluated inputs in the training data to follow certain norms, such as where they are and how many they are, but one may certainly extend our analyses to less restrictive scenarios.

Missing entries We did not consider any bounds in matrix completion [31] in our regret analyses, and proceeded with the assumption that there is no missing entry in the training data. But if missing data is a concern, one should definitely consider adapting bounds from [31] or use better estimators [126] that take into account missing entries when bounding the estimates.

7.5.4 Broader impact

We developed a statistically sound approach for meta BO with an unknown Gaussian process prior. We verified our approach on simulated task and motion planning problems. We showed that our approach is able to guide task and motion planning with good action recommendations, such that the resulting plans are better and faster to compute. We believe the theoretical guarantees may support better explanations for more practical BO approaches. In particular, our method can serve as a building block of artificial intelligence systems, and our analyses can be combined with the theoretical guarantees of other parts of the system to analyze an integrated system.

7.5.5 Caveats

We did not expand the experiment sections to include applications other than task and motion planning in simulation. But there are many more scenarios that this meta BO approach will be useful. For example, our finite $\mathfrak{X}$ formulation can be used to adaptively recommend advertisements, movies or songs to Internet users, by learning a mean and kernel for those discrete items.

Optimization objectives Like other bandit algorithms, our approach only treats objective functions or any metrics to be optimized as *given*. Practitioners need to be very careful about what exactly they are optimizing with our approach or other optimization algorithms. For example, maximizing number of advertisement clicks or corporation profits may not be a good metric in recommendation systems; maximizing a poorly designed reward function for robotic systems may result in unexpected hazards.

Guarantees with assumptions In real-world applications, practitioners need to be extra cautious with our algorithm. We provided detailed assumptions and analyses, that are only based those assumptions, in Section 3 and Section 4. Outside those assumptions, we do not claim that our analyses will hold in any way. For example, in robotics applications, it may not be true that the underlying reward/cost functions are actually sampled from a GP, in which case using our method may harm the physical robot; even if those objective functions are in fact from a GP, because our regret bounds only hold with high probability, meta BO may still give dangerous actions with certain probabilities (as in frequency).

In addition, please notice that we did not provide any theoretical guarantees for using basis functions trained with neural networks. We assume those basis functions are given, which is usually not the case in practice. To the best of our knowledge, proving bounds for neural networks is very

hard [95].

7.6 Conclusion

We proposed a new framework for meta BO that estimates its Gaussian process prior based on past experience with functions sampled from the same prior. We established regret bounds for our approach without the reliance on a known prior and showed its good performance on task and motion planning benchmark problems.

Chapter 8

Scaling Up Bayesian Optimization

Despite recent successes, Bayesian optimization still remains somewhat impractical, since it is typically coupled with expensive function estimators (Gaussian processes) and non-convex acquisition functions that are hard to optimize in high dimensions and sometimes expensive to evaluate. To alleviate these difficulties, recent work explored the use of random feature approximations [170, 113] and sparse Gaussian processes [132], but, while improving scalability, these methods still suffer from misestimation of confidence bounds (an essential part of the acquisition functions), and expensive or inaccurate Gaussian process (GP) hyperparameter inference. Indeed, to the best of our knowledge, Bayesian optimization is typically limited to a few thousand evaluations [113]. Yet, reliable search and estimation for complex functions in very high-dimensional spaces may well require more evaluations. With the increasing availability of parallel computing resources (e.g. running simulations in parallel), large number of function evaluations are possible if the underlying approach can leverage the parallelism. Comparing to the millions of evaluations possible (and needed) with *local* methods like stochastic gradient descent, the scalability of *global* Bayesian optimization leaves large room for desirable progress. In particular, the lack of scalable uncertainty estimates to guide the search is a major roadblock for huge-scale Bayesian optimization.

In this chapter, we propose ensemble Bayesian optimization (EBO), a global optimization method targeted to high dimensional, large scale parameter search problems whose queries are parallelizable. Such problems are abundant in hyper and control parameter optimization in machine learning and robotics [30, 169]. EBO relies on two main ideas that are implemented at multiple

Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. In *International Conference on Machine Learning (ICML)*, 2017.

Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

levels: (1) we use efficient partition-based function approximators (across both data *and* features) that simplify and accelerate search and optimization; (2) we enhance the expressive power of these approximators by using ensembles and a stochastic approach. We maintain an evolving (posterior) distribution over the (infinite) ensemble and, in each iteration, draw one member to perform search and estimation.

In particular, we use a new combination of three types of partition-based approximations: (1-2) For improved GP estimation, we propose a novel *hierarchical* additive GP model based on tile coding (a.k.a. random binning or Mondrian forest features). We learn a posterior distribution over kernel width and the additive structure; here, Gibbs sampling prevents overfitting. (3) To accelerate the sampler, which depends on the likelihood of the observations, we use an efficient, randomized block approximation of the Gram matrix based on a Mondrian process. Sampling and query selection can then be parallelized across blocks, further accelerating the algorithm.

As a whole, this combination of simple, tractable structure with ensemble learning and randomization improves efficiency, uncertainty estimates and optimization. Moreover, we show that our realization of these ideas offers an alternative explanation for global optimization heuristics that have been popular in other communities, indicating possible directions for further theoretical analysis. Our empirical results demonstrate that EBO can speed up the posterior inference by 2-3 orders of magnitude (400 times in one experiment) compared to the state-of-the-art, without sacrificing quality. Furthermore, we demonstrate the ability of EBO to handle sample-intensive hard optimization problems by applying it to real-world problems with tens of thousands of observations.

8.1 Background and Challenges

Consider a simple but high-dimensional search space $\mathcal{X} = [0, R]^D \subseteq \mathbb{R}^D$. We aim to find a maximizer $x^* \in \arg\max_{x \in \mathcal{X}} f(x)$ of a black-box function $f : \mathcal{X} \to \mathbb{R}$.

Gaussian processes. Gaussian processes (GPs) are popular priors for modeling the function f in Bayesian optimization. They define distributions over functions where any finite set of function values has a multivariate Gaussian distribution. A Gaussian process $\mathcal{GP}(\mu, \kappa)$ is fully specified by a mean function $\mu(\cdot)$ and covariance (kernel) function $\kappa(\cdot, \cdot)$. Let f be a function sampled from $\mathcal{GP}(0, \kappa)$. Given observations $\mathcal{D}_n = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^n$ where $y_t \sim \mathcal{N}(f(\boldsymbol{x}_t), \sigma)$, we obtain the

posterior mean and variance of the function as

$$\mu_n(\boldsymbol{x}) = \boldsymbol{\kappa}_n(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}_n, \tag{8.1}$$

$$\sigma_n^2(\boldsymbol{x}) = \kappa(\boldsymbol{x},\boldsymbol{x}) - \boldsymbol{\kappa}_n(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{\kappa}_n(\boldsymbol{x}) \tag{8.2}$$

via the kernel matrix $\boldsymbol{K}_n = [\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)]_{\boldsymbol{x}_i,\boldsymbol{x}_j\in\mathcal{D}_n}$ and $\boldsymbol{\kappa}_n(\boldsymbol{x}) = [\kappa(\boldsymbol{x}_i,\boldsymbol{x})]_{\boldsymbol{x}_i\in D_n}$ [151]. The log data likelihood for $\mathcal{D}_n$ is given by

$$\begin{aligned}\log p(\mathcal{D}_n) = &-\frac{1}{2}\boldsymbol{y}_n^{\mathrm{T}}(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}_n \\ &- \frac{1}{2}\log|\boldsymbol{K}_n + \sigma^2\boldsymbol{I}| - \frac{n}{2}\log 2\pi.\end{aligned} \tag{8.3}$$

While GPs provide flexible, broadly applicable function estimators, the $O(n^3)$ computation of the inverse $(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}$ and determinant $|\boldsymbol{K}_n + \sigma^2\boldsymbol{I}|$ can become major bottlenecks as n grows, for both posterior function value predictions and data likelihood estimation.

Additive structure. To reduce the complexity of the vanilla GP, we assume a latent decomposition of the input dimensions $[D] = \{1,\dots,D\}$ into disjoint subspaces, namely, $\bigcup_{m=1}^{M} A_m = [D]$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$, $i,j \in [M]$. As a result, the function f decomposes as $f(x) = \sum_{m\in[M]} f_m(x^{A_m})$ [91]. If each component f_m is drawn independently from $\mathcal{GP}(\mu^{(m)},\kappa^{(m)})$ for all $m \in [M]$, the resulting f will also be a sample from a GP: $f \sim \mathcal{GP}(\mu,\kappa)$, with $\mu(x) = \sum_{m\in[M]}\mu_m(x^{A_m})$, $\kappa(x,x') = \sum_{m\in[M]}\kappa^{(m)}(x^{A_m},x'^{A_m})$.

The additive structure reduces sample complexity and helps BO to search more efficiently and effectively since the acquisition function can be optimized component-wise. But it remains challenging to learn a good decomposition structure $\{A_m\}$. Recently, [190] proposed learning via Gibbs sampling. This sampler takes hours for merely a few hundred points, because it needs a vast number of expensive data likelihood computations.

Random features. It is possible use random features [150] to approximate the GP kernel and alleviate the $O(n^3)$ computation in Eq. (8.1) and Eq. (8.3). Let $\boldsymbol{\phi} : \mathcal{X} \mapsto \mathbb{R}^{D_R}$ be the (scaled) random feature operator and $\boldsymbol{\Phi}_n = [\boldsymbol{\phi}(\boldsymbol{x}_1),\cdots,\boldsymbol{\phi}(\boldsymbol{x}_n)]^{\mathrm{T}} \in \mathbb{R}^{n\times D_R}$. The GP posterior mean and variance can be written as

$$\mu_n(\boldsymbol{x}) = \sigma^{-2}\boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Sigma}_n\boldsymbol{\Phi}_n^{\mathrm{T}}\boldsymbol{y}_n, \tag{8.4}$$

$$\sigma_n^2(x) = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Sigma}_n\boldsymbol{\phi}(\boldsymbol{x}), \tag{8.5}$$

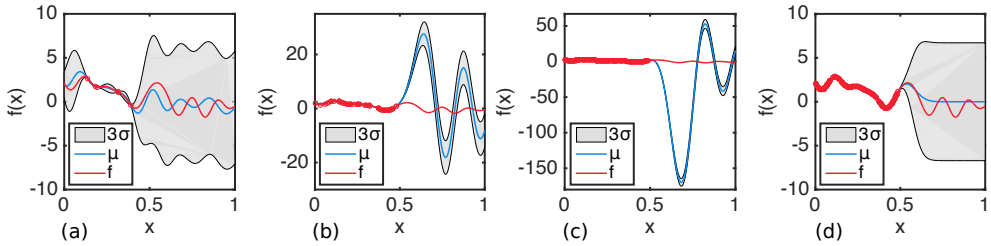


Figure 8-1: We use 1000 Fourier features to approximate a 1D GP with a squared exponential kernel. The observations are samples from a function f (red line) drawn from the GP with zero mean in the range $[-10, 0.5]$. (a) Given 100 sampled observations (red circles), the Fourier features lead to reasonable confidence bounds. (b) Given 1000 sampled observations (red circles), the quality of the variance estimates degrades. (c) With additional samples (5000 observations), the problem is exacerbated. The scale of the variance predictions relative to the mean prediction is very small. (d) For comparison, the proper predictions of the original full GP conditioned on the same 5000 observations as (c). Variance starvation becomes a serious problem for random features when the size of data is close to or larger than the size of the features.

where $\boldsymbol{\Sigma}_n = (\boldsymbol{\Phi}_n^{\mathrm{T}}\boldsymbol{\Phi}_n\sigma^{-2} + \boldsymbol{I})^{-1}$. By the Woodbury matrix identity and the matrix determinant lemma, the log data likelihood becomes

$$\log p(\mathcal{D}_n) = \frac{\sigma^{-4}}{2}\boldsymbol{y}_n^{\mathrm{T}}\boldsymbol{\Phi}_n\boldsymbol{\Sigma}_n\boldsymbol{\Phi}_n^{\mathrm{T}}\boldsymbol{y}_n \\ - \frac{1}{2}\log|\boldsymbol{\Sigma}_n^{-1}| - \frac{\sigma^{-2}}{2}\boldsymbol{y}_n^{\mathrm{T}}\boldsymbol{y}_n - \frac{n}{2}\log 2\pi\sigma^2. \tag{8.6}$$

The number of random features necessary to approximate the GP well in general increases with the number of observations [154]. Hence, for large-scale observations, we cannot expect to solely use a fixed number of features. Moreover, learning hyperparameters for random features is expensive: for Fourier features, the computation of Eq. (8.6) means re-computing the features, plus $O(D_R^3)$ for the inverse and determinant. With Mondrian features [113], we can learn the kernel width efficiently by adding more Mondrian blocks, but this procedure is not well compatible with learning additive structure, since the whole structure of the sampled Mondrian features will change. In addition, we typically need a forest of trees for a good approximation.

Tile coding. Tile coding [173, 3] is a k-hot encoding widely used in reinforcement learning as an efficient set of non-linear features. In its simplest form, tile coding is defined by k partitions, referred to as layers. An encoded data point becomes a binary vector with a non-zero entry for each bin containing the data point. There exists methods for sampling random partitions that allow to approximate various kernels, such as the 'hat' kernel [150], making tile coding well suited for our purposes.

Variance starvation. It is probably not surprising that using finite random features to learn the function *distribution* will result in a loss in accuracy [60]. For example, we observed that, while the mean predictions are preserved reasonably well around regions where we have observations, both mean and confidence bound predictions can become very bad in regions where we do not have observations, once there are more observations than features. We refer to this underestimation of variance scale compared to mean scale, illustrated in Fig. 8-1, as *variance starvation*.

8.2 Related Work

There has been a series of works addressing the three big challenges in BO: selecting batch evaluations [38, 46, 68, 190, 41], high-dimensional input spaces [193, 49, 121, 91, 190, 187], and scalability [170, 113, 132]. Although these three problems tend to co-occur, this work is the first (to the best of our knowledge) to address all three challenges jointly in one framework.

Other parts of our framework are inspired by the Mondrian forest [113], which partitions the input space via a Mondrian tree and aggregates trees into a forest. The closely related Mondrian kernels [12] use random features derived from Mondrian forests to construct a kernel. Such a kernel, in fact, approximates a Laplace kernel. In fact, Mondrian forest features can be considered a special case of the popular tile coding features widely used in reinforcement learning [173, 3]. [113] showed that, in low-dimensional settings, Mondrian forest kernels scale better than the regular GP and achieve good uncertainty estimates in many low-dimensional problems.

Besides Mondrian forests, there is a rich literature on sparse GP methods to address the scalability of GP regression [162, 167, 180, 76]. However, these methods are mostly only shown to be useful when the input dimension is low and there exist redundant data points, so that inducing points can be selected to emulate the original posterior GP well. However, data redundancy is usually not the case in high-dimensional Bayesian optimization. Recent applications of sparse GPs in BO [132] only consider experiments with less than 80 function evaluations in BO and do not show results on large scale observations. Another approach to tackle large scale GPs distributes the computation via local experts [43]. However, this is not very suitable for the acquisition function optimization needed in Bayesian optimization, since every valid prediction needs to synchronize the predictions from all the local experts. This work is also related to [72]. While [72] focuses on modeling non-stationary functions with treed partitions, our work integrates tree structures and Bayesian optimization in a novel way.

8.3 Learning Additive Kernel Structure

We take a Bayesian view on the task of learning the latent structure of the GP kernel. The decomposition of the input space $\mathcal{X}$ will be learned simultaneously with optimization as more and more data is observed. Our generative model draws mixing proportions $\theta \sim \text{DIR}(\alpha)$. Each dimension j is assigned to one out of M groups via the decomposition assignment variable $z_j \sim \text{MULTI}(\theta)$. The objective function is then $f(x) = \sum_{m=1}^{M} f_m(x^{A_m})$, where $A_m = \{j : z_j = m\}$ is the set of support dimensions for function f_m, and each f_m is drawn from a Gaussian Process. Finally, given an input x, we observe $y \sim \mathcal{N}(f(x), \sigma)$. Figure 8-2 illustrates the corresponding graphical model.

Given the observed data $\mathcal{D}_n = \{(x_t, y_t)\}_{t=1}^{n}$, we obtain a posterior distribution over possible decompositions z (and mixing proportions θ) that we will include later in the BO process:

$$p(z, \theta \mid \mathcal{D}_n; \alpha) \propto p(\mathcal{D}_n \mid z) p(z \mid \theta) p(\theta; \alpha).$$

Marginalizing over θ yields the posterior distribution of the decomposition assignment

$$\begin{aligned} p(z \mid \mathcal{D}_n; \alpha) &\propto p(\mathcal{D}_n \mid z) \int p(z \mid \theta) p(\theta; \alpha) \, \mathrm{d}\theta \\ &\propto p(\mathcal{D}_n \mid z) \frac{\Gamma(\sum_m \alpha_m)}{\Gamma(D + \sum_m \alpha_m)} \prod\nolimits_m \frac{\Gamma(|A_m| + \alpha_m)}{\Gamma(\alpha_m)} \end{aligned}$$

where $p(\mathcal{D}_n|z)$ is the data likelihood (8.3) for the additive GP given a fixed structure defined by z. We learn the posterior distribution for z via Gibbs sampling, choose the decomposition among the samples that achieves the highest data likelihood, and then proceed with BO. The Gibbs sampler repeatedly draws coordinate assignments z_j according to

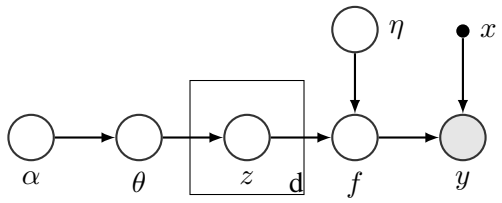
$$\begin{aligned} p(z_j = m \mid z_{\neg j}, \mathcal{D}_n;\ \alpha) &\propto p(\mathcal{D}_n \mid z) p(z_j \mid z_{\neg j}) \\ &\propto p(\mathcal{D}_n \mid z)(|A_m| + \alpha_m) \propto e^{\phi_m}, \end{aligned}$$

where

$$\begin{aligned} \phi_m = &-\frac{1}{2} \boldsymbol{y}^{\mathrm{T}} (\boldsymbol{K}_n^{(z_j=m)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} \\ &- \frac{1}{2} \log |\boldsymbol{K}_n^{(z_j=m)} + \sigma^2 \boldsymbol{I}| + \log(|A_m| + \alpha_m) \end{aligned}$$

and $\boldsymbol{K}_n^{(z_j=m)}$ is the gram matrix associated with the observations $\mathcal{D}_n$ by setting $z_j = m$. We can use

Figure 8-2: Graphical model for the structured Gaussian process; η is the hyperparameter of the GP kernel; z controls the decomposition for the input space.



the Gumbel trick to efficiently sample from this categorical distribution. Namely, we sample a vector of i.i.d standard Gumbel variables ω_i of length M, and then choose the sampled decomposition assignment $z_j = \arg\max_{i \leq M} \phi_i + \omega_i$.

With a Dirichlet process, we could make the model nonparametric and the number M of possible groups in the decomposition infinite. Given that we have a fixed number of input dimension D, we set $M = D$ in practice.

8.4 Ensemble Bayesian Optimization

Next, we describe an approach that scales Bayesian Optimization when parallel computing resources are available. We name our approach, outlined in Alg.17, *Ensemble Bayesian optimization (EBO)*. At a high level, EBO uses a (stochastic) series of Mondrian trees to partition the input space, learn the kernel parameters of a GP locally, and aggregate these parameters. Our forest hence spans across BO iterations.

In the t-th iteration of EBO in Alg. 17, we use a Mondrian process to randomly partition the search space into J parts (line 4), where J can be dependent on the size of the observations $\mathcal{D}_{t-1}$. For the j-th partition, we have a subset $\mathcal{D}_{t-1}^{j}$ of observations. From those observations, we learn a local GP with random tile coding *and* additive structure, via Gibbs sampling (line 6). For conciseness, we refer to such GPs as TileGPs. The probabilistic tile coding can be replaced by a Mondrian grid that approximates a Laplace kernel [13]. Once a TileGP is learned locally, we can run BO with the acquisition function η in each partition to generate a candidate set of points, and, from those, select a batch that is both informative (high-quality) and diverse (line 14).

Since, in each iteration, we draw an input space partition and update the kernel width and the additive structure, the algorithm may be viewed as implicitly and stochastically running BO on an ensemble of GP models. In the following, we describe our model and the procedures of Alg. 17 in detail. In the Appendix, we show an illustration how EBO optimizes a 2D function.

Algorithm 17 Ensemble Bayesian Optimization (EBO)

function EBO $(f, \mathcal{D}_0)$
  Initialize z, k
  for $t = 1, \cdots, T$ **do**
    $\{\mathcal{X}_j\}_{j=1}^{J} \leftarrow$MONDRIAN$([0, R]^D, z, k, J)$
    parfor $j = 1, \cdots, J$ **do**
      $z^j, k^j \leftarrow$ GIBBSSAMPLING$(z, k \mid \mathcal{D}_{t-1}^j)$
      $\eta_{t-1}^j(\cdot) \leftarrow$ACQUISITION $(\mathcal{D}_{t-1}^j, z^j, k^j)$
      $\{A_m\}_{m=1}^{M} \leftarrow$ DECOMPOSITION(z^j)
      for $m = 1, \cdots, M$ **do**
        $\boldsymbol{x}_{tj}^{A_m} \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{X}_j^{A_m}} \eta_{t-1}^j(\boldsymbol{x})$
      end for
    end parfor
    $z \leftarrow$ SYNC$(\{z^j\}_{j=1}^{J})$, $k \leftarrow$ SYNC$(\{k^j\}_{j=1}^{J})$
    $\{\boldsymbol{x}_{tb}\}_{b=1}^{B} \leftarrow$ FILTER $(\{\boldsymbol{x}_{tj}\}_{j=1}^{J} \mid z, k)$
    parfor $b = 1, \cdots, B$ **do**
      $y_{tb} \leftarrow f(\boldsymbol{x}_{tb})$
    end parfor
    $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{\boldsymbol{x}_{tb}, y_{tb}\}_{b=1}^{B}$
  end for
end function

8.4.1 Partitioning the input space via a Mondrian process

When faced with a "big" problem, a natural idea is to divide and conquer. For large scale Bayesian optimization, the question is how to divide without losing the valuable local information that gives good uncertainty measures. In EBO, we use a Mondrian process to divide the input space and the observed data, so that nearby data points remain together in one partition, preserving locality. Alg. 18 shows the full 'Mondrian partitioning" algorithm.

Algorithm 18 Mondrian Partitioning

function MONDRIANPARTITIONING (V, N_p, S)
  while $|V| < N_p$ **do**
    $p_j \leftarrow length(v_j) \cdot max(0, |\mathcal{D}^j| - S), \forall v_j \in V$
    if $p_j = 0, \forall j$ **then**
      break
    end if
    Sample $v_j \sim \frac{p_j}{\sum_j p_j}, v_j \in V$
    Sample a dimension $d \sim \frac{h_d^j - l_d^j}{\sum_d h_d^j - l_d^j}, d \in [D]$
    Sample cut location $u_d^j \sim U[l_d^j, h_d^j]$
    $v_{j(left)} \leftarrow [l_1^j, h_1^j] \times \cdots \times [l_d^j, u_d^j] \times \cdots \times \times [l_D^j, h_D^j]$
    $v_{j(right)} \leftarrow [l_1^j, h_1^j] \times \cdots \times [u_d^j, h_d^j] \times \cdots \times \times [l_D^j, h_D^j]$
    $V \leftarrow V \cup \{v_{j(left)}, v_{j(right)}\} \setminus v_j$
  end while
  return V
end function

In particular, we denote the maximum number of Mondrian partitions by N_p (usually the worker pool size in the experiments) and the minimum number of data points in each partition to be S. The set of partitions computed by the Mondrian tree (a.k.a. the leaves of the tree), V, is initialized to be the function domain $V = \{[0, R]^D\}$, the root of the tree. For each $v_j \in V$ described by a hyperrectangle $[l_1^j, h_1^j] \times \cdots \times [l_D^j, h_D^j]$, the length of v_j is computed to be $length(v_j) = \sum_{d=1}^{D} (h_d^j - l_d^j)$. The observations associated with v_j is $\mathcal{D}^j$. Here, for all $(x, y) \in \mathcal{D}^j$, we have $x \in [l_1^j - \epsilon, h_1^j + \epsilon] \times \cdots \times [l_D^j - \epsilon, h_D^j + \epsilon]$, where ϵ controls the how many neighboring data points to consider for the partition v_j. In our experiments, ϵ is set to be 0. Alg. 18 is different from Algorithm 1 and 2 of [113] in the stop criterion. [113] uses an exponential clock to count down the time of splitting the leaves of the tree, while we split the leaves until the number of Mondrian partitions reaches N_p or there is no partition that have more than S data points. We designed our stop criterion this way to balance the efficiency of EBO and the quality of selected points. Usually EBO is faster with larger number of partitions N_p (i.e., more parallel computing resources) and the quality of the selections are better with larger size of observations on each partition (S).

The Mondrian process uses axis-aligned cuts to divide the input space $[0, R]^D$ into a set of partitions $\{\mathcal{X}_j\}_{j=0}^{J}$ where $\cup_j \mathcal{X}_j = [0, R]^D$ and $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset, \forall i \neq j$. Each partition $\mathcal{X}_j$ can be conveniently described by a hyperrectangle $[l_1^j, h_1^j] \times \cdots \times [l_D^j, h_D^j]$, which facilitates the efficient

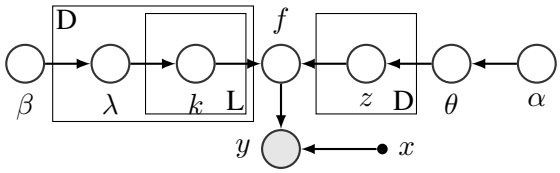


Figure 8-3: The graphical model for TileGP, a GP with additive and tile kernel partitioning structure. The parameter λ controls the rate for the number of cuts k of the tilings (inverse of the kernel bandwidth); the parameter z controls the additive decomposition of the input feature space.

use of tile coding and Mondrian grids in a TileGP. In the next section, we define a TileGP and introduce how its parameters are learned.

8.4.2 Learning a local TileGP via Gibbs sampling

For the j-th hyperrectangle partition $\mathcal{X}_j = [l_1^j, h_1^j] \times \cdots \times [l_D^j, h_D^j]$, we use a TileGP to model the function f locally. We use the acronym "TileGP" to denote the Gaussian process model that uses additive kernels, with each component represented by tilings. We show the details of the generative model for TileGP in Alg. 19 and the graphical model in Fig. 8-3 with fixed hyper-parameters α, β_0, β_1. The main difference to the additive GP model used in [190] is that TileGP constructs a hierarchical model for the random features (and hence, the kernels), while [190] do not consider the kernel parameters to be part of the generative model. The random features are based on tile coding or Mondrian grids, with the number of cuts generated by D Poisson processes on $[l_d^j, h_d^j]$ for each dimension $d = 1, \cdots, D$. On the i-th layer of the tilings, tile coding samples the offset δ from a uniform distribution $U[0, \frac{h_d^j - l_d^j}{k_{di}}]$ and places the cuts uniformly starting at $\delta + l_d^j$. The Mondrian grid samples k_{di} cut locations uniformly randomly from $[l_d^j, h_d^j]$. Because of the data partition, we always have more features than observations, which can alleviate the variance starvation problem described in Section 8.1.

We can use Gibbs sampling to efficiently learn the cut parameter k and decomposition parameter z by marginalizing out λ and θ. Notice that both k and z take discrete values; hence, unlike other continuous GP parameterizations, we only need to sample discrete variables for Gibbs sampling.

Algorithm 19 Generative model for TileGP

Draw mixing proportions $\theta \sim \textsc{Dir}(\alpha)$
for $d = 1, \cdots, D$ **do**
    Draw additive decomposition $z_d \sim \textsc{Multi}(\theta)$
    Draw Poisson rate parameter $\lambda_d \sim \textsc{Gamma}(\beta_0, \beta_1)$
    for $i = 1, \cdots, L$ **do**
        Draw number of cuts $k_{di} \sim \textsc{Poisson}(\lambda_d(h_d^j - l_d^j))$
        $\begin{cases} \text{Draw offset } \delta \sim U[0, \frac{h_d^j - l_d^j}{k_{di}}] & \text{Tile Coding} \\ \text{Draw cut locations } \boldsymbol{b} \sim U[l_d^j, h_d^j] & \text{Mondrian Grids} \end{cases}$
    end for
end for
Construct the feature projection $\boldsymbol{\phi}$ and the kernel $\kappa = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi}$ from z and sampled tiles
Draw function $f \sim \mathcal{GP}(0, \kappa)$
Given input $\boldsymbol{x}$, draw function value $y \sim \mathcal{N}(f(\boldsymbol{x}), \sigma)$

Given the observations $\mathcal{D}_{t-1}$ in the j-th hyperrectangle partition, the posterior distribution of the (local) parameters λ, k, z, θ is

$$\begin{aligned} &p(\lambda, k, z, \theta \mid \mathcal{D}_{t-1}; \alpha, \beta) \\ &\quad \propto p(\mathcal{D}_{t-1} \mid z, k) p(z \mid \theta) p(k \mid \lambda) p(\theta; \alpha) p(\lambda; \beta). \end{aligned}$$

Marginalizing over the Poisson rate parameter λ and the mixing proportion θ gives

$$\begin{aligned} &p(k, z \mid \mathcal{D}_{t-1}; \alpha, \beta) \\ &\propto p(\mathcal{D}_{t-1} | z, k) \int p(z|\theta) p(\theta; \alpha) \,\mathrm{d}\theta \int p(k|\lambda) p(\lambda; \beta) \,\mathrm{d}\lambda \\ &\propto p(\mathcal{D}_{t-1} \mid z, k) \prod\nolimits_m \frac{\Gamma(|A_m| + \alpha_m)}{\Gamma(\alpha_m)} \\ &\qquad \times \prod\nolimits_d \frac{\Gamma(\beta_1 + |k_d|)}{(\prod_{i=1}^L k_{di}!)(\beta_0 + L)^{\beta_1 + |k_d|}} \end{aligned}$$

where $|k_d| = \sum_{i=1}^L k_{di}$. Hence, we only need to sample k and z when learning the hyperparameters of the TileGP kernel. For each dimension d, we sample the group assignment z_d according to

$$\begin{aligned} &p(z_d = m \mid \mathcal{D}_{t-1}, k, z_{\neg d}; \alpha) \propto p(\mathcal{D}_{t-1} \mid z, k) p(z_d \mid z_{\neg d}) \\ &\quad \propto p(\mathcal{D}_{t-1} \mid z, k)(|A_m| + \alpha_m). \end{aligned} \tag{8.7}$$

We sample the number of cuts k_{di} for each dimension d and each layer i from the posterior

$$\begin{aligned} p(k_{di} \mid \mathcal{D}_{t-1}, k_{\neg di}, z; \beta) &\propto p(\mathcal{D}_{t-1} \mid z, k) p(k_{di} \mid k_{\neg di}) \\ &\propto \frac{p(\mathcal{D}_n \mid z, k)\Gamma(\beta_1 + |k_d|)}{(\beta_0 + L)^{k_{di}} k_{di}!}. \end{aligned} \tag{8.8}$$

If distributed computing is available, each hyperrectangle partition of the input space is assigned a worker to manage all the computations within this partition. On each worker, we use the above Gibbs sampling method to learn the additive structure and kernel bandwidth jointly. Conditioned on the observations associated with the partition on the worker, we use the learned posterior TileGP to select the most promising input point in this partition, and eventually send this candidate input point back to the main process together with the learned decomposition parameter z and the cut parameter k. In the next section, we introduce the acquisition function we used in each worker and how to filter the recommended candidates from all the partitions.

8.4.3 Acquisition functions

In this chapter, we mainly focus on parameter search problems where the objective function is designed by an expert and the global optimum or an upper bound on the function is known. While any BO acquisition functions can be used within the EBO framework, we use an acquisition function from [187] to exploit the knowledge of the upper bound. Let f^* be such an upper bound, i.e., $\forall x \in \mathcal{X}, f^* \geq f(x)$. Given the observations $\mathcal{D}_{t-1}^j$ associated with the j-th partition of the input space, we minimize the acquisition function $\eta_{t-1}^j(x) = \frac{f^* - \mu_{t-1}^j(x)}{\sigma_{t-1}^j(x)}$. Since the kernel is additive, we can optimize $\eta_{t-1}^j(\cdot)$ separately for each additive component. Namely, for the m-th component of the additive structure, we optimize $\eta_{t-1}^j(\cdot)$ only on the active dimensions A_m. This resembles a block coordinate descent, and greatly facilitates the optimization of the acquisition function.

8.4.4 Filtering, budget allocation and batched BO

In the EBO algorithm, we first use a batch of workers to learn the local GPs and recommend potential good candidate points from the local information. Then we aggregate the information of all the workers, and use a filter to select the points to evaluate from the set of points recommended by all the workers based on the aggregated information on the function.

There are two important details here: (1) how many points to recommend from each local worker (budget allocation); and (2) how to select a batch of points from the Mondrian partition on

each worker (filtering). Usually in the beginning of the iterations, we do not have a lot of Mondrian partitions (since we stop splitting a partition once it reaches a minimum number of data points). It is very likely that the number of partitions J is smaller than the size of the batch. Hence, we need to allocate the budget of recommendations from each worker properly and use batched BO for each Mondrian partition.

Budget allocation In our current version of EBO, we did the budget allocation using a heuristic, where we would like to generate at least $2B$ recommendations from all the workers, and each worker gets the budget proportional to a score, the sum of the Mondrian partition volume (volume of the domain of the partition) and the best function value of the partition.

Filtering. Once we have a proposed query point from each partition, we select B of them according to the scoring function $\xi(X) = \log\det K_X - \sum_{b=1}^{B} \eta(\boldsymbol{x}_b)$ where $X = \{\boldsymbol{x}_b\}_{b=1}^{B}$. We use the log determinant term to force diversity and η to maintain quality. We maximize this function greedily. In some cases, the number of partitions J can be smaller than the batch size B. In this case, one may either use just J candidates, or use batch BO on each partition. We use the latter detailed below.

Batched BO For batched BO, we also use a heuristic where the points achieving the top n acquisition function values are always included and the other ones come from random points selected in that partition. For the optimization of the acquisition function over each block of dimensions, we sample 1000 points in the low dimensional space associated with the additive component and minimize the acquisition function via L-BFGS-B starting from the point that gives the best acquisition value. We add the optimized $\arg\min$ to the 1000 points and sort them according to their acquisition values, and then select the top n random ones, and combine with the sorted selections from other additive components. Other batched BO methods can also be used and can potentially improve upon our results.

8.4.5 Efficient data likelihood computation and parameter synchronization

For the random features, we use tile coding due to its sparsity and efficiency. Since non-zero features can be found and computed by binning, the computational cost for encoding a data point scales linearly with dimensions and number of layers. The resulting representation is sparse and convenient to use. Additionally, the number of non-zero features is quite small, which allows us to efficiently compute a sparse Cholesky decomposition of the inner product (Gram matrix) or the outer product of the data. This allows us to efficiently compute the data likelihoods.

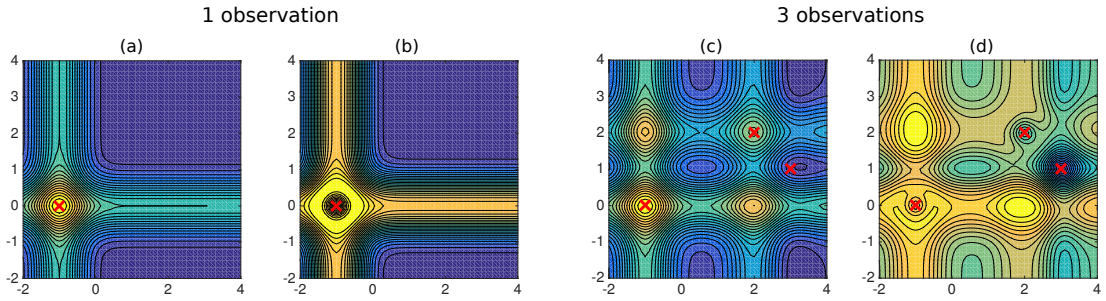


Figure 8-4: Posterior mean function (a, c) and GP-UCB acquisition function (b, d) for an additive GP in 2D. The maxima of the posterior mean and acquisition function are at the points resulting from an exchange of coordinates between "good" observed points (-1,0) and (2,2).

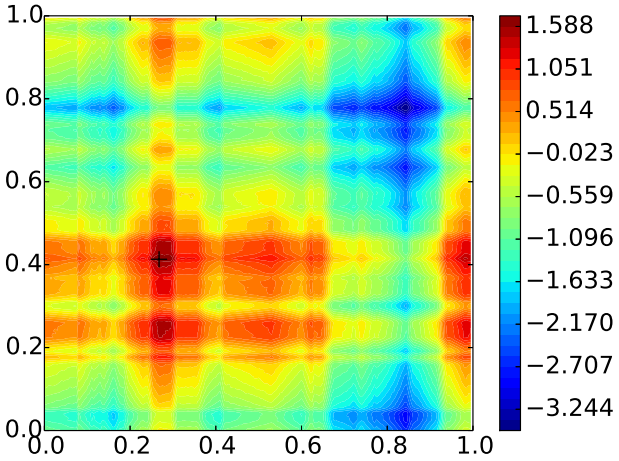


Figure 8-5: The 2D additive function we optimized in Fig. 8-6. The global maximum is marked with "+".

In each iteration t, after the batch workers return the learned decomposition indicator z^b and the number of tiles $k^b, b \in [B]$, we synchronize these two parameters (line 13 of Alg. 17). For the number of tiles k, we set k_d to be the rounded mean of $\{k_d^b\}_{b=1}^B$ for each dimension $d \in [D]$. For the decomposition indicator, we use correlation clustering to cluster the input dimensions.

8.4.6 An Illustration of EBO

We give an illustration of the proposed EBO algorithm on a 2D function shown in Fig. 8-5. This function is a sample from a 2D TileGP, where the decomposition parameter is $z = [0, 1]$, the cut parameter is (inverse bandwidth) $k = [10, 10]$, and the noise parameter is $\sigma = 0.01$.

The global maximum of this function is at $(0.27, 0.41)$. In this example, EBO is configured to have at least 20 data points on each partition, at most 50 Mondrian partitions, and 100 layers of

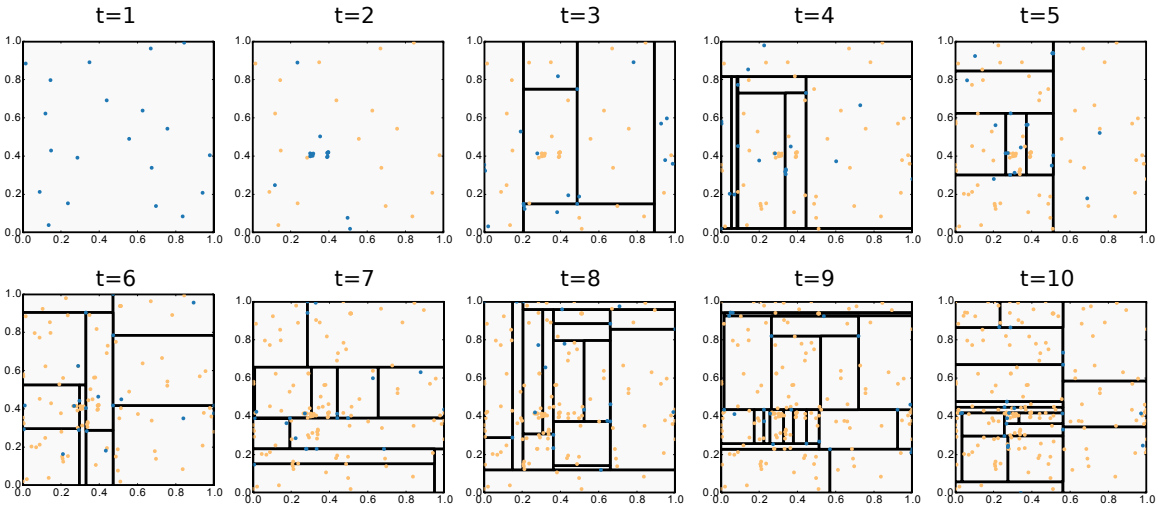


Figure 8-6: An example of 10 iterations of EBO on a 2D toy example plotted in Fig. 8-5. The selections in each iteration are blue and the existing observations orange. EBO quickly locates the region of the global optimum while still allocating budget to explore regions that appear promising (e.g. around the local optimum $(1.0, 0.4)$).

tiles to approximate the Laplace kernel. We run EBO for 10 iterations with 20 queries each batch. The results are shown in Fig. 8-6. In the first iteration, EBO has no information about the function; hence it spreads the 10 queries (blue dots) "evenly" in the input domain to collect information. In the 2nd iteration, based on the evaluations on the selected points (yellow dots), EBO chooses to query batch points (blue dots) that have high acquisition values, which appear to be around the global optimum and some other high valued regions. As the number of evaluations exceeds 20, the minimum number of data points on each partition, EBO partitions the input space with a Mondrian process in the following iterations. Notice that each iteration draws a different partition (shown as the black lines) from the Mondrian process so that the results will not "over-fit" to one partition setting and the computation can remain efficient. In each partition, EBO runs the Gibbs sampling inference algorithm to fit a local TileGP and uses batched BO select a few candidates. Then EBO uses a filter to decide the final batch of candidate queries (blue dots) among all the recommended ones from each partition as described in Sec. 8.4.4.

8.4.7 Relations to Mondrian kernels, random binning and additive Laplace kernels

Our model described in Section 8.4.2 can use Mondrian grids or (our version of) tile coding to achieve efficient parameter inference for the decomposition z and the number of cuts k (inverse of kernel bandwidth). Tile coding and Mondrian grids are also closely related to Mondrian Features

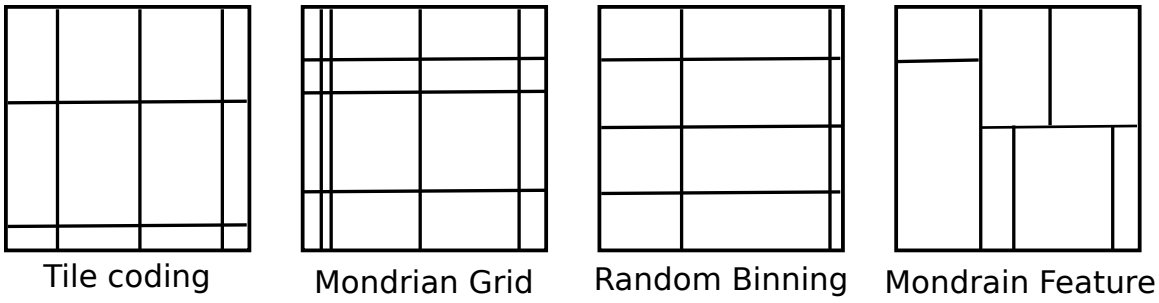


Figure 8-7: Illustrations of (our version of) tile coding, Mondrian Grid, random binning and Mondrian feature.

and Random Binning: All of the four kinds of random features attempt to find a sparse random feature representation for the raw input $\boldsymbol{x}$ based on the partition of the space with the help of layers. But there are some subtle differences.

We illustrate the differences between one layer of the features constructed by tile coding, Mondrian grid, Mondrian features and random binning in Fig. 8-7. For each layer of (our version of) tile coding, we sample a positive integer k (number of cuts) from a Poisson distribution parameterized by λR, and then set the offset to be a constant uniformly randomly sampled from $[0, \frac{R}{k}]$. For each layer of the Mondrian grid, the number of cuts k is sampled tile in coding, but instead of using an offset and uniform cuts, we put the cuts at locations independently uniformly randomly from $[0, R]$. Random binning does not sample k cuts but samples the distance δ between neighboring cuts by drawing $\delta \sim \textsc{Gamma}(2, \lambda R)$. Then, it samples the offset from $[0, \delta]$ and finally places the cuts. All of the above-mentioned three types of random features can work individually for each dimension and then combine the cuts from all dimensions. The Mondrian feature (Mondrian forest features to be exact), contrast, partitions the space jointly for all dimensions. More details of Mondrian features can be found in [113, 12]. For all of these four types of random features and for each layer of the total L layers, the kernel is $\kappa_L(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{L}\sum_{l=1}^{L} \chi_l(\boldsymbol{x}, \boldsymbol{x}')$ where

$$\chi_l(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 1 & \boldsymbol{x} \text{ and } \boldsymbol{x}' \text{ are in the same cell on the layer } l \\ 0 & \text{otherwise} \end{cases} \tag{8.9}$$

For the case where the kernel has M additive components, we simply use the tiling for each decomposition and normalize by LM instead of L. More precisely, we have $\kappa_L(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{LM}\sum_{m=1}^{M}\sum_{l=1}^{M} \chi_l(\boldsymbol{x}^{A_m}, \boldsymbol{x}'^{A_m})$.

Mondrian grids, Mondrian features and random binning all converge to the Laplace kernel as the number of layers L goes to infinity. The tile coding kernel, however, does not approximate a

Laplace kernel. Our model with Mondrian grids approximates an additive Laplace kernel:

Lemma 8.4.1. *Let the random variable $k_{di} \sim \text{POISSON}(\lambda_d R)$ be the number of cuts in the Mondrian grids of TileGP for dimension $d \in [D]$ and layer $i \in [L]$. The TileGP kernel κ_L satisfies $\lim_{L\to\infty} \kappa_L(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{M}\sum_{m=1}^{M} e^{\lambda_d R|\boldsymbol{x}^{A_m} - \boldsymbol{x}'^{A_m}|}$, where $\{A_m\}_{m=1}^{M}$ is the additive decomposition.*

Proof. When constructing the Mondrian grid for each layer and each dimension, one can think of the process of getting another cut as a Poisson point process on the interval $[0, R]$, where the time between two consecutive cuts is modeled as an exponential random variable. Similar to Proposition 1 in [12], we have

$$\lim_{L\to\infty} \kappa_L^{(m)}(\boldsymbol{x}^{A_m}, \boldsymbol{x}'^{A_m}) = \mathbb{E}[\text{no cut between } \boldsymbol{x}_d \text{ and } \boldsymbol{x}'_d, \forall d \in A_m] = e^{-\lambda_d R|\boldsymbol{x}^{A_m} - \boldsymbol{x}'^{A_m}|}.$$

By the additivity of the kernel, we have $\lim_{L\to\infty} \kappa_L(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{M}\sum_{m=1}^{M} e^{\lambda_d R|\boldsymbol{x}^{A_m} - \boldsymbol{x}'^{A_m}|}$. □

[12] showed that in practice, the Mondrian kernel constructed from Mondrian features may perform slightly better than random binning in certain cases. Although it would be possible to use a Mondrian partition for each layer of tile coding, we only consider uniform, grid based binning with random offests because this allows the non-zero features to be computed more efficiently ($O(1)$ instead of $O(\log k)$). Note that as more dimensions are discretized in this manner, the number of features grows exponentially. However, the number of non-zero entries can be independently controlled, allowing to create sparse representations that remain computationally tractable.

8.4.8 Connections to evolutionary algorithms

Next, we make some observations that connect our randomized ensemble BO to ideas for global optimization heuristics that have successfully been used in other communities. In particular, these connections offer an explanation from a BO perspective and may aid further theoretical analysis.

Evolutionary algorithms [10] maintain an ensemble of "good" candidate solutions (called chromosomes) and, from those, generate new query points via a number of operations. These methods too, implicitly, need to balance exploration with local search in areas known to have high function values. Hence, there are local operations (mutations) for generating new points, such as random perturbations or local descent methods, and global operations. While it is relatively straightforward to draw connections between those local operations and optimization methods used in machine learning, we here focus on global exploration.

A popular global operation is *crossover*: given two "good" points $x, y \in \mathbb{R}^D$, this operation outputs a new point z whose coordinates are a combination of the coordinates of x and y, i.e., $z_i \in \{x_i, y_i\}$ for all $i \in [D]$. In fact, this operation is analogous to BO with a (randomized) additive kernel: the crossover strategy implicity corresponds to the assumption that high function values can be achieved by combining coordinates from points with high function values. For comparison, consider an additive kernel $\kappa(x, x') = \sum_{m=1}^{M} \kappa^{(m)}(x^{A_m}, x'^{A_m})$ and $f(x) = \sum_{m=1}^{M} f^m(x^{A_m})$. Since each sub-kernel $\kappa^{(m)}$ is "blind" to the dimensions in the complement of A_m, any point x' that is close to an observed high-value point x in the dimensions A_m will receive a high value $f^m(x)$, independent of the other dimensions, and, as a result, looks like a "good" candidate.

We illustrate this reasoning with a 2D toy example. Figure 8-4 shows the posterior mean prediction and GP-UCB criterion $\hat{f}(x) + 0.1\sigma(x)$ for an additive kernel with $A_1 = \{1\}$, $A_2 = \{2\}$ and $\kappa^m(x_m, y_m) = \exp(-2(x_m - y_m)^2)$. High values of the observed points generalize along the dimensions "ignored" by the sub-kernels. After two good observations $(-1, 0)$ and $(2, 2)$, the "crossover" points $(-1, 2)$ and $(2, 0)$ are local maxima of GP-UCB and the posterior mean.

In real data, we do not know the best fitting underlying grouping structure of the coordinates. Hence, crossover does a random search over such partitions by performing random coordinate combinations, whereas our adaptive BO approach maintains a posterior distribution over partitions that adapts to the data.

8.5 Experiments

First, we verify the effectiveness of using our Gibbs sampling algorithm to learn the additive structure of the unknown function. We then empirically show the scalability of EBO and its effectiveness of using random adaptive Mondrian partitions, and finally evaluate EBO on two real-world problems.[1]

8.5.1 Effectiveness of Decomposition Learning

We first probe the effectiveness of using the Gibbs sampling method described in Section 8.3 to learn the decomposition of the input space.

Recovering Decompositions First, we sample test functions from a known additive Gaussian Process prior with zero-mean and isotropic Gaussian kernel with bandwidth $= 0.1$ and scale $= 5$

[1]Our code is publicly available at `https://github.com/zi-w/Ensemble-Bayesian-Optimization`.

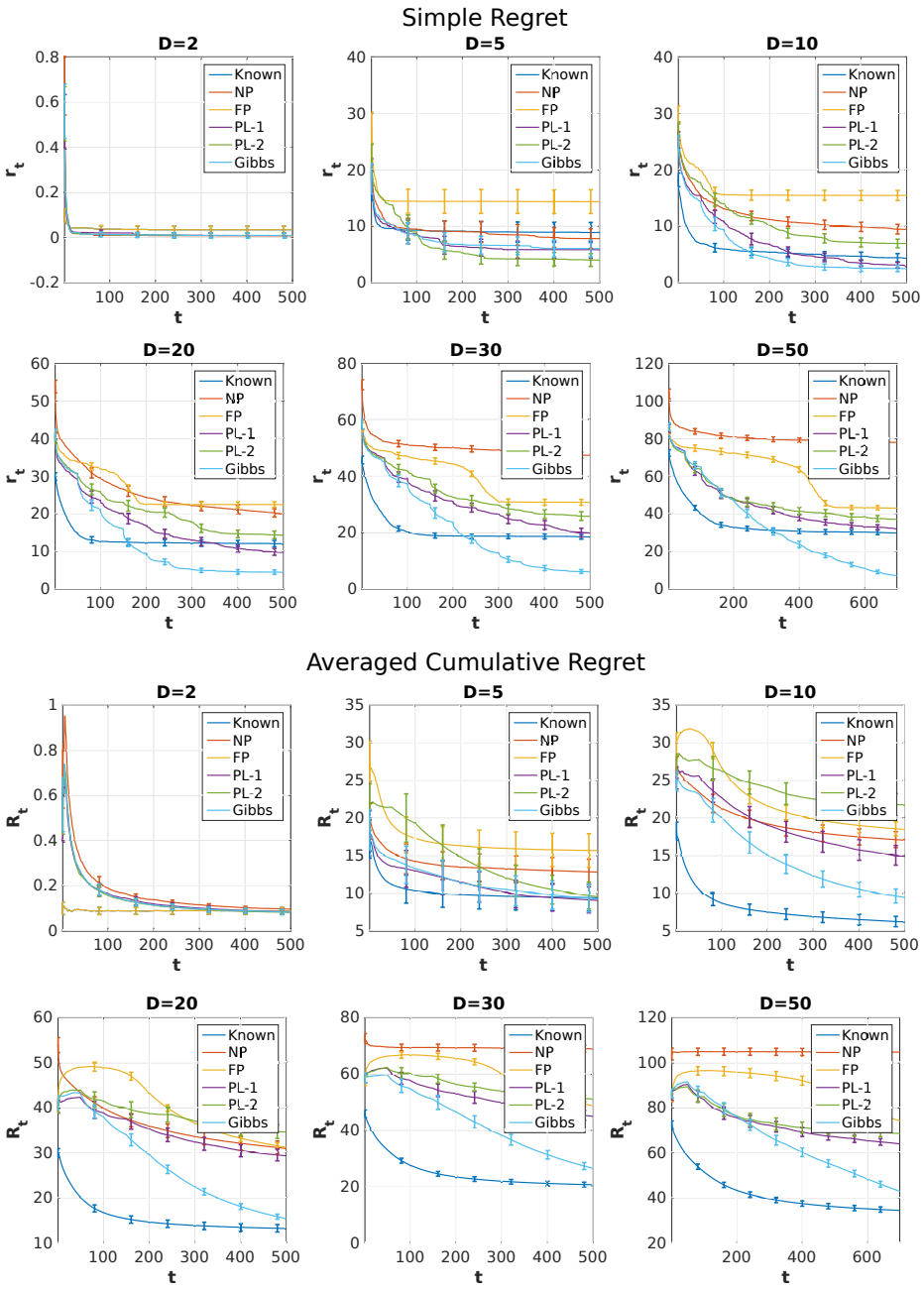


Figure 8-8: The simple regrets (r_t) and the averaged cumulative regrets (R_t) and for `Known` (ground truth partition is given), `Gibbs` (using Gibbs sampling to learn the partition), `PL-1` (randomly sample the same number of partitions sampled by `Gibbs` and select the one with highest data likelihood), PL-2 (randomly sample 5 partitions and select the one with highest data likelihood), `FP` (fully partitioned, each group with one dimension) and `NP` (no partition) on 10, 20, 50 dimensional functions. `Gibbs` achieved comparable results to `Known`. Comparing `PL-1` and `PL-2` we can see that sampling more partitions did help to find a better partition. But a more principled way of learning partition using `Gibbs` can achieve much better performance than `PL-1` and `PL-2`.

Table 8.1: Empirical posterior of any two dimensions correctly being grouped together by Gibbs sampling.

D \ N	50	150	250	450
5	0.81 ± 0.28	0.91 ± 0.19	1.00 ± 0.03	1.00 ± 0.00
10	0.21 ± 0.13	0.54 ± 0.25	0.68 ± 0.25	0.93 ± 0.15
20	0.06 ± 0.06	0.11 ± 0.08	0.20 ± 0.12	0.71 ± 0.22
50	0.02 ± 0.03	0.02 ± 0.02	0.03 ± 0.03	0.06 ± 0.04
100	0.01 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	0.02 ± 0.02

Table 8.2: Empirical posterior of any two dimensions correctly being separated by Gibbs sampling.

D \ N	50	150	250	450
2	0.30 ± 0.46	0.30 ± 0.46	0.90 ± 0.30	1.00 ± 0.00
5	0.87 ± 0.17	0.80 ± 0.27	0.60 ± 0.32	0.50 ± 0.34
10	0.88 ± 0.05	0.89 ± 0.06	0.89 ± 0.07	0.94 ± 0.07
20	0.94 ± 0.02	0.94 ± 0.02	0.94 ± 0.02	0.97 ± 0.02
50	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.01	0.98 ± 0.01
100	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00

for each function component. For $D = 2, 5, 10, 20, 50, 100$ input dimensions, we randomly sample decomposition settings that have at least two groups in the decomposition and at most 3 dimensions in each group.

We set the burn-in period to be 50 iterations, and the total number of iterations for Gibbs sampling to be 100. In Tables 8.1 and 8.2, we show two quantities that are closely related to the learned empirical posterior of the decompositions with different numbers of randomly sampled observed data points (N). Table 8.1 shows the probability of two dimensions being correctly grouped together by Gibbs sampling in each iteration of Gibbs sampling after the burn-in period, namely, $(\sum_{i<j\leq D} \mathbb{1}_{z_i^g \equiv z_j^g \wedge z_i \equiv z_j})/(\sum_{i<j\leq D} \mathbb{1}_{z_i \equiv z_j})$. Table 8.2 reports the probability of two dimensions being correctly separated in each iteration of Gibbs sampling after the burn-in period, namely, $(\sum_{i<j\leq D} \mathbb{1}_{z_i^g \neq z_j^g \wedge z_i \neq z_j})/(\sum_{i<j\leq D} \mathbb{1}_{z_i \neq z_j})$. The results show that the more data we observe, the more accurate the learned decompositions are. They also suggest that the Gibbs sampling procedure can converge to the ground truth decomposition with enough data for relatively small numbers of dimensions. The higher the dimension, the more data we need to recover the true decomposition.

Sensitivity Analysis for α Empirically, we found that the quality of the learned decomposi-

Table 8.3: Rand Index of the decompositions learned by Gibbs sampling for different values of α.

α \ N	50	150	250	350	450
0.2	0.87811 ± 0.019002	0.90126 ± 0.022394	0.95284 ± 0.047111	0.98811 ± 0.02602	0.98811 ± 0.026322
0.5	0.88211 ± 0.019893	0.90305 ± 0.024574	0.95295 ± 0.046232	0.98947 ± 0.025872	0.99505 ± 0.013881
1	0.88211 ± 0.016947	0.90326 ± 0.024935	0.95305 ± 0.043878	0.98558 ± 0.034779	0.98053 ± 0.035843
2	0.88084 ± 0.016972	0.9 ± 0.023489	0.95463 ± 0.042968	0.97989 ± 0.038818	0.98832 ± 0.023592
5	0.88337 ± 0.015784	0.90158 ± 0.02203	0.96126 ± 0.037045	0.98716 ± 0.030949	0.99316 ± 0.015491

tions is not very sensitive to the scale of α (see Table 8.3), because the log data likelihood plays a much more important role than $\log(|A_m| + \alpha)$ when α is less than the total number of dimensions. The reported results correspond to alpha = 1 for all the partitions.

Effectiveness of Learning Decompositions for Bayesian Optimization To verify the effectiveness of the learned decomposition for Bayesian optimization, we tested on 2, 10, 20 and 50 dimensional functions sampled from a zero-mean Add-GP with randomly sampled decomposition settings (at least two groups, at most 3 dimensions in each group) and isotropic Gaussian kernel with bandwidth $= 0.1$ and scale $= 5$. Each experiment was repeated 50 times. An example of a 2-dimensional function component is shown in Fig. 8-9. Because of the numerous local maxima, it is very challenging to achieve the global optimum even for 2 dimensions, let alone maximizing an additive sum of them, only by observing their sum.

For Add-GP-UCB, we used $\beta_t^{(m)} = |A_m| \log 2t$ for lower dimensions ($D = 2, 5, 10$), and $\beta_t^{(m)} = |A_m| \log 2t/5$ for higher dimensions ($D = 20, 30, 50$). We show averaged cumulative regret and simple regret in Fig. 8-8. We compare Add-GP-UCB with known additive structure (`Known`), no partitions (`NP`), fully partitioned with one dimension for each group (`FP`) and the following methods of learning the decomposition: Gibbs sampling (`Gibbs`), randomly sampling the same number of decompositions sampled by `Gibbs` and select the one with the highest data likelihood (`PL-1`), randomly sampling 5 decompositions and selecting the one with the highest data likelihood (`PL-2`). For the latter two learning methods are referred to as "partial learning" in [91]. The learning of the decomposition is done every 50 iterations. Fig. 8-10 shows the improvement of learning decompositions with `Gibbs` over optimizing without partitions (NP). For $D = 20, 30$, it is quite obvious that when a new partition is learned from the newly observed data (e.g. at iteration 100 and 150), the simple regret gets a boost.

Overall, the results show that `Gibbs` outperforms both of the partial learning methods, and for higher dimensions, `Gibbs` is sometimes even better than `Known`. Interestingly, similar results can

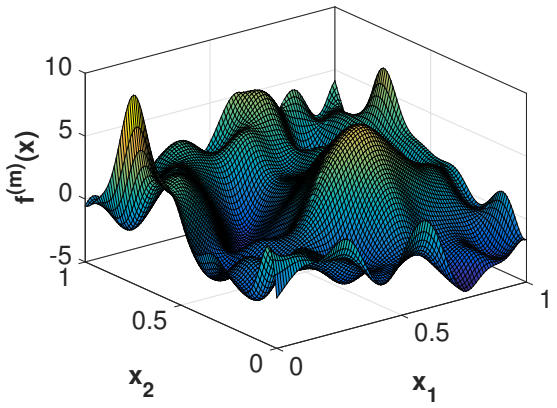


Figure 8-9: An example of a 2 dimensional function component of the synthetic function.

be found in Fig. 3 (c) of [91], where different decompositions than the ground truth may give better simple regret. We conjecture that this is because `Gibbs` is able to explore more than `Known`, for two reasons:

1. Empirically, `Gibbs` changes the decompositions across iterations, especially in the beginning. With fluctuating partitions, even exploitation leads to moving around, because the supposedly "good" points are influenced by the partition. The result is an implicit "exploration" effect that is absent with a fixed partition.

2. `Gibbs` sometimes merges "true" parts into larger parts. The parameter β_t in UCB depends on the size of the part, $|A_m|(\log 2t)/5$ (as in [91]). Larger parts hence lead to larger β_t and hence more exploration.

Of course, more exploration is not always better, but `Gibbs` was able to find a good balance between exploration and exploitation, which leads to better performance. Our preliminary experiments indicate that one solution to ensure that the ground truth decomposition produces the best result is to tune β_t. Hyperparameter selection (such as choosing β_t) for BO is, however, very challenging and an active topic of research (e.g. [191]).

Next, we test the decomposition learning algorithm on a real-world function, which returns the distance between a designated goal location and two objects being pushed by two robot hands, whose trajectory is determined by 14 parameters specifying the location, rotation, velocity, moving direction etc. This function is implemented with a physics engine, the Box2D simulator [32]. We use add-GP-UCB with different ways of setting the additive structure to tune the parameters for the robot hand so as to push the object closer to the goal. The regrets are shown in Fig. 8-11. We observe

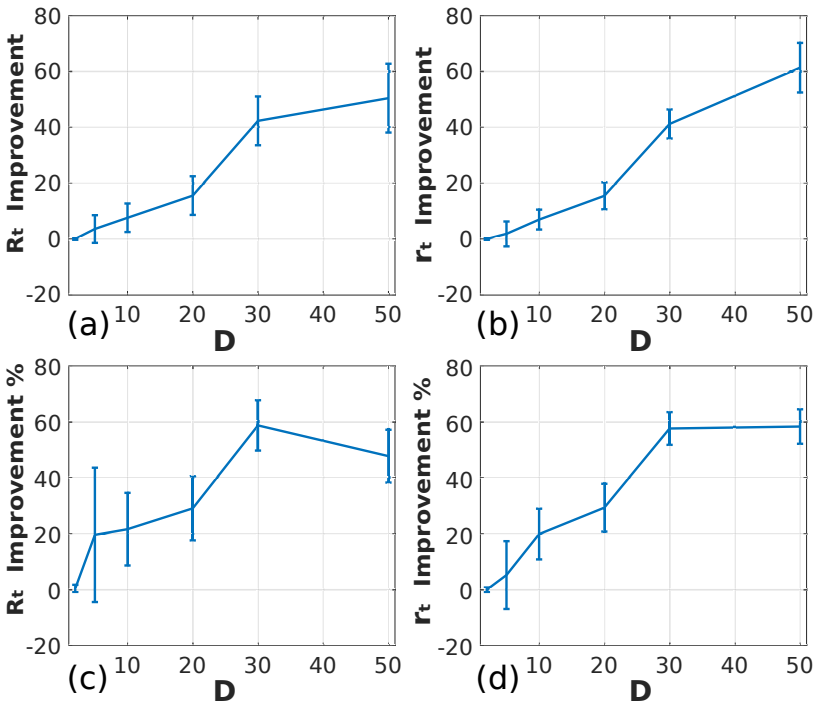


Figure 8-10: Improvement made by learning the decomposition with `Gibbs` over optimizing without partitions (NP). (a) averaged cumulative regret; (b) simple regret. (c) averaged cumulative regret normalized by function maximum; (d) simple regret normalized by function maximum. Using decompositions learned by `Gibbs` continues to outperform BO without `Gibbs`.

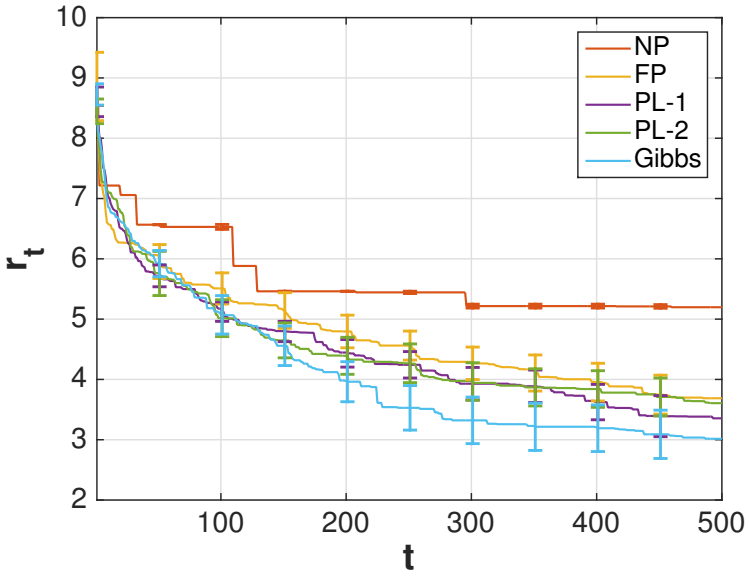


Figure 8-11: Simple regret of tuning the 14 parameters for a robot pushing task. Learning decompositions with `Gibbs` is more effective than partial learning (`PL-1`, `PL-2`), no partitions (`NP`), or fully partitioned (`FP`). Learning decompositions with `Gibbs` helps BO to find a better point for this tuning task.

that the performance of learning the decomposition with `Gibbs` dominates all existing alternatives including partial learning. Since the function we tested here is composed of the distance to two objects, there could be some underlying additive structure for this function in certain regions of the input space, e.g. when the two robots hands are relatively distant from each other so that one of the hands only impacts one of the objects. Hence, it is possible for `Gibbs` to learn a good underlying additive structure and perform effective BO with the structures it learned.

8.5.2 Scalability of EBO

We compare EBO with the approach described in Section 8.3, abbreviated as Structural Kernel Learning (SKL). EBO can make use of parallel resources for Gibbs sampling, while SKL cannot. Because the kernel learning part is the computationally dominating factor of large scale BO, we compare the time each method needs to run 10 iterations of Gibbs sampling with 100 to 50000 observations in 20 dimensions. For EBO, the maximum number of Mondrian partitions is set to be 1000 and the minimum number of data points in each Mondrian partition is 100. The function that we used to test was generated from a fully partitioned 20 dimensional GP with an additive Laplace kernel ($|A_m| = 1, \forall m$).

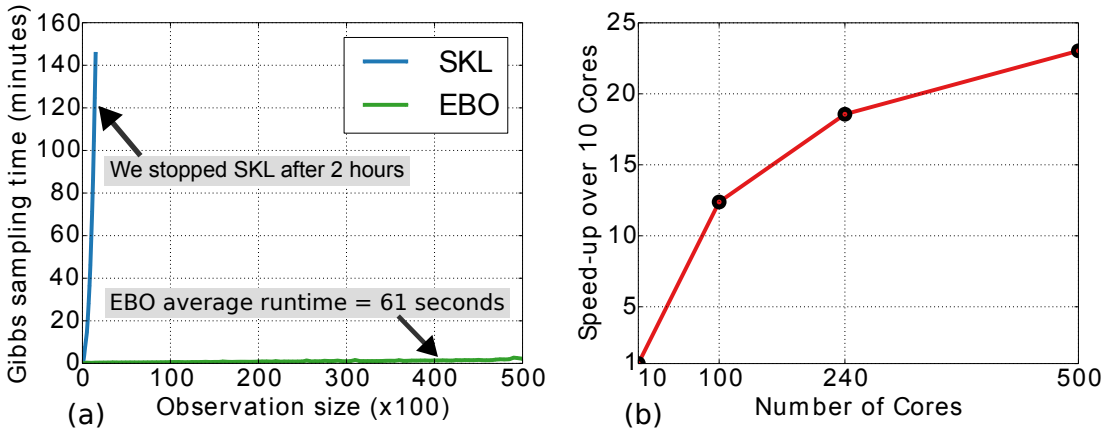


Figure 8-12: (a) Timing for the Gibbs sampler of EBO and SKL. EBO is significantly faster than SKL when the observation size N is relatively large. (b) Speed-up of EBO with 100, 240, 500 cores over EBO with 10 cores on 30,000 observations. Running EBO with 240 cores is almost 20 times faster than with 10 cores.

We show the timing results for the Gibbs samplers in Fig. 8-12(a), where EBO uses 240 cores via the Batch Service of Microsoft Azure. Due to a time limit we imposed, we did not finish SKL for more than 1500 observations. EBO runs more than 390 times faster than SKL when the observation size is 1500. Comparing the quality of learned parameter z for the additive structure, SKL has a Rand Index of 96.3% and EBO has a Rand Index of 96.8%, which are similar. In Fig. 8-12(b), we show speed-ups for different number of cores. EBO with 500 cores is not significantly faster than with 240 cores because EBO runs synchronized parallelization, whose runtime is decided by the slowest core. It is often the case that most of the cores have finished while the program is waiting for the slowest 1 or 2 cores to finish.

8.5.3 Effectiveness of EBO

Optimizing synthetic functions We verify the effectiveness of using ensemble models for BO on 4 functions randomly sampled from a 50-dimensional GP with an additive Laplace kernel. The lengthscale parameter of the Laplace kernel is set to be 0.1, variance scale to be 1 and active dimensions are around 1 to 4. Namely, the kernel we used is $\kappa(x, x') = \sum_{i=1}^{M} \kappa^{(m)}(x^{A_m}, x'^{A_m})$ where $\kappa^{(m)}(x^{A_m}, x'^{A_m}) = e^{\frac{|x^{A_m} - x'^{A_m}|}{0.1}}, \forall m$. The domain of the function is $[0, 1]^{50}$. We implemented the BO-SVI and BO-Add-SVI using the same acquisition function and batch selection strategy as EBO but with SVI-GP [76] and SVI-GP with additive kernels instead of TileGPs. We used the SVI-GP implemented in [71] and defined the additive Laplace kernel according to the priors of the tested functions. For both BO-SVI and BO-Add-SVI, we used 100 batchsize, 200 inducing points and the

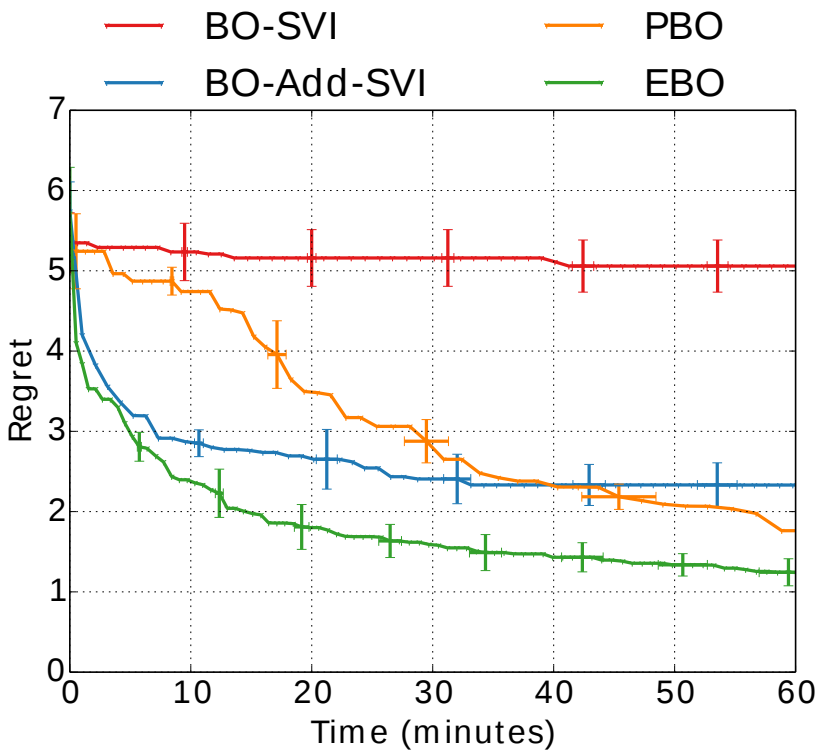


Figure 8-13: Averaged results of the regret of BO-SVI, BO-Add-SVI, PBO and EBO on 4 different functions drawn from a 50D GP with an additive Laplace kernel. BO-SVI has the highest regret for all functions. Using an additive GP within SVI (BO-Add-SVI) significantly improves over the full kernel. In general, EBO finds a good point much faster than the other methods.

parameters were optimized for 100 iterations. For EBO, we set the minimum size of data points on each Mondrian partition to be 100. We set the maximum number of Mondrian partitions to be 1000 for both EBO and PBO.

In each iteration, each algorithm evaluates a batch of parameters of size B in parallel. We denote $\tilde{r}_t = \max_{x \in \mathcal{X}} f(x) - \max_{b \in [B]} f(x_{t,b})$ as the immediate regret obtained by the batch at iteration t, and $r_T = \min_{t \leq T} \tilde{r}_t$ as the regret, which captures the minimum gap between the best point found and the global optimum of the black-box function f.

We compare BO using SVI [76] (BO-SVI), BO using SVI with an additive GP (BO-Add-SVI) and a distributed version of BO with a fixed partition (PBO) against EBO with a randomly sampled partition in each iteration. PBO has the same 1000 Mondrian partitions in all the iterations while EBO can have at most 1000 Mondrian partitions. BO-SVI uses a Laplace isotropic kernel without any additive structure, while BO-Add-SVI, PBO, EBO all use the known prior. Our experimental results in Fig. 8-13 shows that EBO is able to find a good point much faster than BO-SVI and BO-Add-SVI; and, randomization and the ensemble of partitions matters: EBO is much better than PBO. Note that the evaluations of the test functions are negligible, so the timing results in Fig. 8-13 reflect the actual runtime of each method.

Optimizing control parameters for robot pushing We follow [190] and test our approach, EBO, on a 14 dimensional control parameter tuning problem for robot pushing.

We implemented the simulation of pushing two objects with two robot hands in the Box2D physics engine [32]. The 14 parameters specifies the location and rotation of the robot hands, pushing speed, moving direction and pushing time. The lower limit of these parameters is $[-5, -5, -10, -10, 2, 0, -5, -5, -$ and the upper limit is $[5, 5, 10, 10, 30, 2\pi, 5, 5, 10, 10, 30, 2\pi, 5, 5]$. Let the initial positions of the objects be s_{i0}, s_{i1} and the ending positions be s_{e0}, s_{e1}. We use s_{g0} and s_{g1} to denote the goal locations for the two objects. The reward is defined to be $r = \|s_{g0} - s_{i0}\| + \|s_{g1} - s_{i1}\| - \|s_{g0} - s_{e0}\| - \|s_{g1} - s_{e1}\|$, namely, the progress made towards pushing the objects to the goal.

We compare EBO, BO-SVI, BO-Add-SVI and CEM [176] with the same 10^4 random observations and repeat each experiment 10 times. All the methods choose a batch of 100 parameters to evaluate at each iteration. CEM uses the top 30% of the 10^4 initial observations to fit its initial Gaussian distribution. At the end of each iteration in CEM, 30% of the new observations with top values were used to fit the new distribution. For all the BO based methods, we use the maximum value of the reward function in the acquisition function. The standard deviation of the observation noise in the GP models is set to be 0.1. We set EBO to have Modrian partitions with fewer than

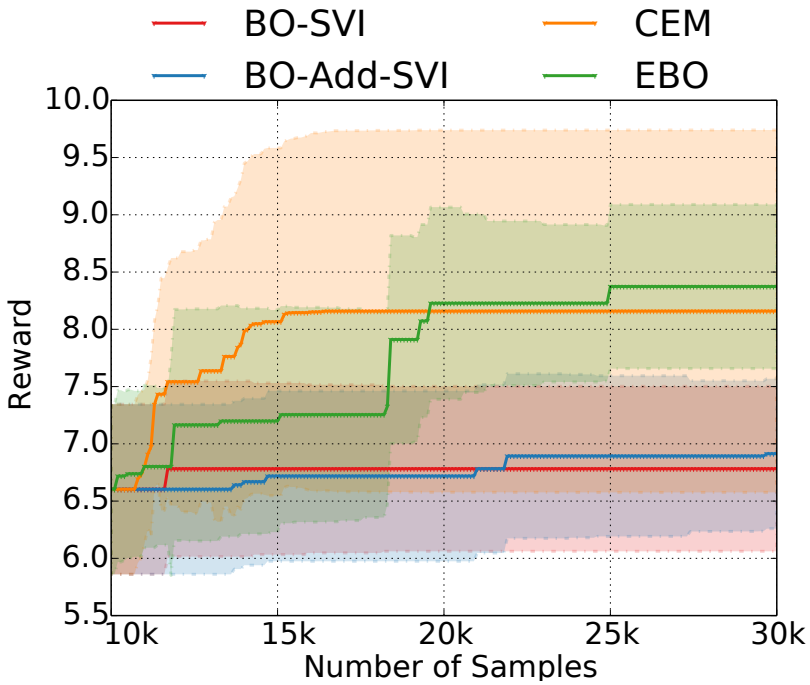


Figure 8-14: Comparing BO-SVI, BO-Add-SVI, CEM and EBO on a control parameter tuning task with 14 parameters.

150 data points and constrain EBO to have no more than 200 Mondrian partitions. In EBO, we set the hyper parameters $\alpha = 1.0$, $\beta = [5.0, 5.0]$, and the Mondrian observation offset $\epsilon = 0.05$. In BO-SVI, we used 100 batchsize in SVI, 200 inducing points and 500 iterations to optimize the data likelihood with 0.1 step rate and 0.9 momentum. BO-Add-SVI used the same parameters as BO-SVI, except that BO-Add-SVI uses 3 outer loops to randomly select the decomposition parameter z and in each loop, it uses an inner loop of 50 iterations to maximize the data likelihood over the kernel parameters. The batch BO strategy used in BO-SVI and BO-Add-SVI is identical to the one used in each Mondrian partition of EBO.

We run all the methods for 200 iterations, where each iteration has a batch size of 100. In total, each method obtains 2×10^4 data points in addition to the 10^4 initializations. We plot the median of the best rewards achieved by CEM and EBO at each iteration in Fig. 8-14.

Overall CEM and EBO performed comparably and much better than the sparse GP methods (BO-SVI and BO-Add-SVI). We noticed that among all the experiments, CEM achieved a maximum reward of 10.19 while EBO achieved 9.50. However, EBO behaved slightly better and more stable than CEM as reflected by the standard deviation on the rewards.

Optimizing rover trajectories

To further explore the performance of our method, we consider a trajectory optimization task

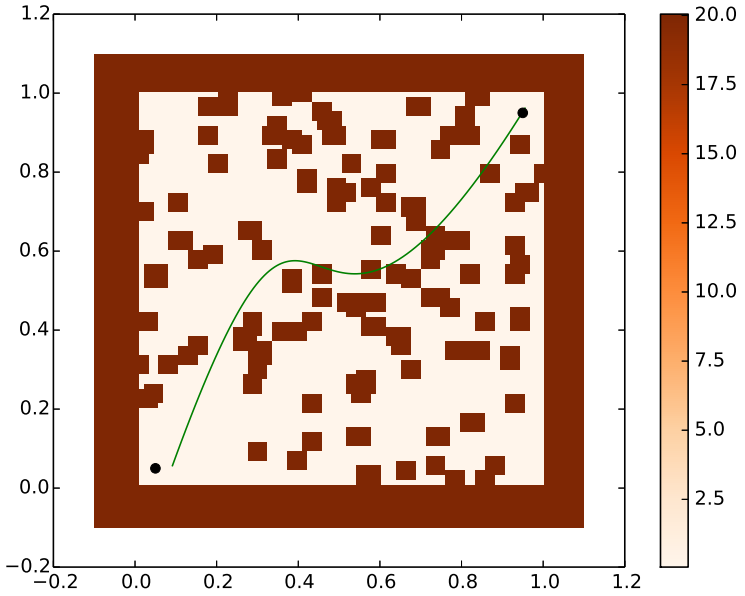


Figure 8-15: An example trajectory found by EBO.

in 2D, meant to emulate a rover navigation task. We describe a problem instance by defining a start position s and a goal position g as well as a cost function over the state space. Trajectories are described by a set of points on which a BSpline is to be fitted. By integrating the cost function over a given trajectory, we can compute the trajectory cost $c(\boldsymbol{x})$ of a given trajectory solution $\boldsymbol{x} \in [0, 1]^{60}$. We define the reward of this problem to be $f(\boldsymbol{x}) = c(\boldsymbol{x}) + \lambda(\|\boldsymbol{x}_{0,1} - s\|_1 + \|\boldsymbol{x}_{59,60} - g\|_1) + b$. This reward function is non smooth, discontinuous, and concave over the first two and last two dimensions of the input. These 4 dimensions represent the start and goal position of the trajectory.

We illustrate the problem in Fig. 8-15 with an example trajectory found by EBO. We set the trajectory cost to be -20.0 for any collision, λ to be -10.0 and the constant $b = 5.0$. This reward function is non smooth, discontinuous, and concave over the first two and last two dimensions of the input. These 4 dimensions represent the start and goal position of the trajectory. We maximize the reward function f over the points on the trajectory. All the methods choose a batch of 500 trajectories to evaluate. Each method is initialized with 10^4 trajectories randomly uniformly selected from $[0, 1]^{60}$ and their reward function values. We again compare EBO with BO-SVI, BO-Add-SVI and CEM [176]. All the methods choose a batch of 500 trajectories to evaluate. Each method is initialized with 10^4 trajectories randomly uniformly selected from $[0, 1]^{60}$ and their reward function values. The initializations are the same for each method, and we repeat the experiments 5 times.

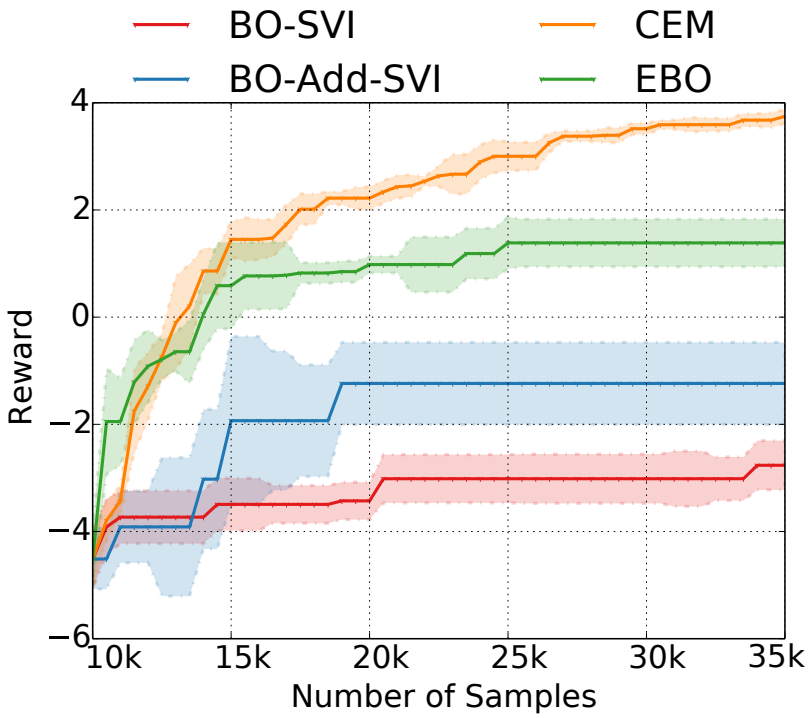


Figure 8-16: Comparing BO-SVI, BO-Add-SVI, CEM and EBO on a 60 dimensional trajectory optimization task.

CEM uses the top 30% of the 10^4 initial observations to fit its initial Gaussian distribution. At the end of each iteration in CEM, 30% of the new observations with top values were used to fit the new distribution. For all the BO based methods, we use the maximum value of the reward function, 5.0, in the acquisition function. The standard deviation of the observation noise in the GP models is set to be 0.01. We set EBO to attempt to have Modrian partitions with fewer than 100 data points, with a hard constraint of no more than 1000 Mondrian partitions. In EBO, we set the hyper parameters $\alpha = 1.0$, $\beta = [2.0, 5.0]$, and the Mondrian observation offset $\epsilon = 0.01$. In BO-SVI, we used 100 batchsize in SVI, 200 inducing points and 500 iterations to optimize the data likelihood with 0.1 step rate and 0.9 momentum. BO-Add-SVI used the same parameters as BO-SVI, except that BO-Add-SVI uses 3 outer loops to randomly select the decomposition parameter z and in each loop, it uses an inner loop of 50 iterations to maximize the data likelihood over the kernel parameters. The batch BO strategy used in BO-SVI and BO-Add-SVI is identical to the one used in each Mondrian partition of EBO.

The results in Fig. 8-16 showed that CEM was able to achieve better results than the BO methods on these functions, while EBO was still much better than the BO alternatives using SVI.

Verifying the acquisition function As introduced in Section 8.4.3, we used a different acqui-

sition function optimization technique from [91, 187]. In [91, 187], the authors used the fact that each additive component is by itself a GP. Hence, they did posterior inference on each additive component and Bayesian optimization independently from other additive components. In this work, we use the full GP with the additive kernel to derive its acquisition function and optimize it with a block coordinate optimization procedure, where the blocks are selected according to the decomposition of the input dimensions. One reason we did this instead of following [91, 187] is that we observed the over-estimation of variance for each additive component if inferred independently from others. We conjecture that this over-estimation could result in an invalid regret bound for Add-GP-UCB [91]. Nevertheless, we found that using the block coordinate optimization for the acquisition function on the full GP is actually very helpful. In Figure. 8-17, we compare the acquisition function we described in Section 8.4.3 (denoted as BlockOpt) with Add-GP-UCB [91], Add-MES-R and Add-MES-G [187] on the same experiment described in the first experiment of Section 6.5 of [187], averaging over 20 functions. Notice that we used the maximum value of the function as part of our acquisition function in our approach (BlockOpt). Add-GP-UCB, ADD-MES-R and ADD-MES-G cannot use this max-value information even if they have access to it, because then they don't have a strategy to deal with "credit assignment", which assigns the maximum value to each additive component. We found that BlockOpt is able to find a solution as well as or even better than the best of the three competing approaches.

8.6 Discussion

8.6.1 Failure modes of EBO

EBO is a general framework for running large scale batched BO in high-dimensional spaces. Admittedly, we made some compromises in our design and implementation to scale up BO to a degree that conventional BO approaches cannot deal with. In the following, we list some limitations and aspects that we can improve in EBO in our future work.

- EBO partitions the space into smaller regions $\{[l_j, h_j]\}_{j=1}^{J}$ and only uses the observations within $[l_j - \epsilon, h_j + \epsilon]$ to do inference and Bayesian optimization. It is hard to determine the value of ϵ. If ϵ is large, we may have high computational cost for the operations within each region. But if ϵ is very small, we found that some selected BO points are on the boundaries of the regions, partially because of the large uncertainty on the boundaries. We used $\epsilon = 0$ in our experiments, but the results can be improved with a more appropriate ϵ.

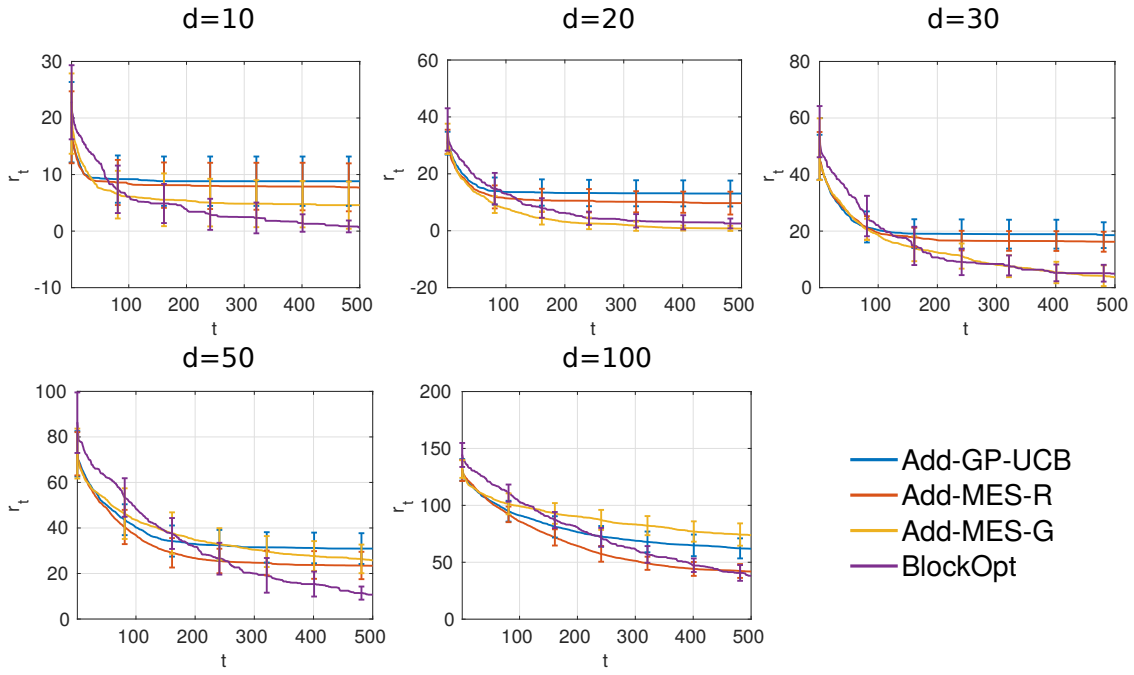


Figure 8-17: Comparing different acquisition functions for BO with an additive GP. Our strategy, BlockOpt, achieves comparable or better results than other methods.

- Because of the additive structure, we need to optimize the acquisition function for each additive component. As a result, EBO has increased computational cost when there are more than 50 additive components, and it becomes harder for EBO to optimize functions more than a few hundred dimensions. One solution is to combine the additive structure with a low dimensional projection approach [193]. We can also simply run block coordinate descent on the acquisition function, but it is harder to ensure that the acquisition function is fully optimized.

8.6.2 Importance of avoiding variance starvation

Neural networks have been applied in many applications and received success for tasks including regression and classification. While researchers are still working on the theoretical understanding, one hyoothesis is that neural networks "overfit" [205]. Due to the similarity between the test and training set in the reported experiments in, for example, the computer vision community, overfitting may seem to be less of a problem. However, in active learning (e.g. Bayesian optimization), we do not have a "test set". We require the model to generalize well across the search space, and using the classic neural network may be detrimental to the data selection process, because of variance starvation (see Section 2). Gaussian processes, on the contrary, are good at estimating confidence bounds and avoid overfitting. However, the scaling of Gaussian processes is hard in general. We

would like to reinforce the awareness about the importance of estimating confidence of the model predictions on new queries, i.e., avoiding variance starvation.

8.6.3 Future directions

Possible future directions include analyzing theoretically what should be the best input space partition strategy, batch worker budget distribution strategy, better ways of predicting variance in a principled way (not necessarily GP), better ways of doing small scale BO and how to adapt it to large scale BO. Moreover, add-GP is only one way of reducing the function space, and there could be others suitable ones too.

8.7 Conclusion

Many black box function optimization problems are intrinsically high-dimensional and may require a huge number of observations in order to be optimized well. In this chapter, we propose a novel framework, ensemble Bayesian optimization, to tackle the problem of scaling Bayesian optimization to both large numbers of observations and high dimensions. To achieve this, we propose a new framework that jointly integrates randomized partitions at various levels: our method is a stochastic method over a randomized, adaptive ensemble of partitions of the input data space; for each part, we use an ensemble of TileGPs, a new GP model we propose based on tile coding and additive structure. We also developed an efficient Gibbs sampling approach to learn the latent variables. Moreover, our method automatically generates batch queries. We empirically demonstrate the effectiveness and scalability of our method on high dimensional parameter search tasks with tens of thousands of observation data.

Chapter 9

Conclusion

> *People who value Tao all turn to books. But books are nothing more than words. Words have value; what is of value in words is meaning. Meaning has something it is pursuing, but the thing that it is pursuing cannot be put into words and handed down. The world values words and hands down books but, though the world values them, I do not think them worth valuing. What the world takes to be values is not real value.*
>
> Chuang Tzu

In this thesis, we have mainly explored robot learning from two perspectives: learning useful models for planning and active data acquisition with Bayesian optimization. We have identified and explored solutions to several critical problems in model learning, including

- identifying what environment parameters are relevant to a skill;
- identifying what conditions should hold before using a skill;
- how to sample skill parameters that satisfy those conditions;
- how to handle non-Gaussian transition models in continuous state-action systems.

We have shown that embedding strong prior knowledge in learning approaches is a key to reducing sample complexity and making robot learning feasible. The forms of the prior knowledge can be expressed in various ways, including structures of the model, compositionality of the planning problem, Bayesian prior, etc.

The other important factor in our approaches is Bayesian optimization which guides autonomous agents to acquire data in a principled way. We have presented methods and theories for

- designing query selection criteria based on Gaussian processes;
- estimating the Gaussian process prior via meta-learning;
- scaling up to high-dimensional inputs and large-scale observations.

On one hand, these new methods provide valuable insights for active data acquisition; while on the other hand, we also demonstrate the possibility of building in strong priors that is meta-learned instead of hand designed.

Given the complexity of learning for robotics tasks, especially for long-horizon problems, finding good ways to build in strong priors will continue to be one of the most important subject of study. This thesis is only a starting point.

Appendix A

Omitted Proofs from Chapter 7

A.1 Proofs for Section 7.3.1

Recall that we assume $\mathfrak{X}$ is a finite set. The posterior given observations D_t is $GP(\mu_t, k_t)$ where

$$\mu_t(x) = \mu(x) + k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \mu(\boldsymbol{x}_t)), \;\; \forall x \in \mathfrak{X}$$
$$k_t(x, x') = k(x, x') - k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1} k(\boldsymbol{x}_t, x'), \;\; \forall x, x' \in \mathfrak{X}.$$

We use the following estimators to approximate μ_t, k_t:

$$\hat{\mu}_t(x) = \hat{\mu}(x) + \hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t, \boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t)), \;\; \forall x \in \mathfrak{X}, \tag{A.1}$$
$$\hat{k}_t(x, x') = \frac{N-1}{N-t-1}\left(\hat{k}(x, x') - \hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t, \boldsymbol{x}_t)^{-1}\hat{k}(\boldsymbol{x}_t, x')\right), \;\; \forall x, x' \in \mathfrak{X}. \tag{A.2}$$

We will prove a bound on the best-sample simple regret $r_T = \max_{x \in \mathfrak{X}} f(x) - \max_{t \in [T]} f(x_t)$. The evaluated inputs $\boldsymbol{x}_t = [x_\tau]_\tau^t$ are selected either by a special case of GP-UCB using the acquisition function

$$\alpha_{t-1}^{\text{GP-UCB}}(x) = \hat{\mu}_{t-1}(x) + \zeta_t \hat{k}_{t-1}(x)^{\frac{1}{2}}, \tag{A.3}$$
$$\zeta_t = \frac{\left(6(N - 3 + t + 2\sqrt{t \log \frac{6}{\delta}} + 2\log\frac{6}{\delta})/(\delta N(N-t-1))\right)^{\frac{1}{2}} + (2\log(\frac{3}{\delta}))^{\frac{1}{2}}}{(1 - 2(\frac{1}{N-t}\log\frac{6}{\delta})^{\frac{1}{2}})^{\frac{1}{2}}}, \delta \in (0, 1) \tag{A.4}$$

or by a special case of PI using the acquisition function

$$\alpha_{t-1}^{\mathrm{PI}}(x) = \frac{\hat{\mu}_{t-1}(x) - \hat{f}^*}{\hat{k}_{t-1}(x)^{\frac{1}{2}}}.$$

This special case of PI assumes additional information of the upper bound on function value $\hat{f}^* \geq \max_{x \in \mathfrak{X}} f(x)$.

Corollary A.1.1 ([171]). *Let $\delta_0 \in (0,1)$. For any Gaussian variable $x \sim \mathcal{N}(\mu, \sigma^2), x \in \mathbb{R}$,*

$$\Pr[x - \mu \leq \zeta_0 \sigma] \geq 1 - \delta_0,\ \Pr[x - \mu \geq -\zeta_0 \sigma] \geq 1 - \delta_0$$

where $\zeta_0 = (2\log(\frac{1}{2\delta_0}))^{\frac{1}{2}}$.

Proof. Let $z = \frac{\mu - x}{\sigma} \sim \mathcal{N}(0,1)$. We have

$$\begin{aligned}
\Pr[z > \zeta_0] &= \int_{\zeta_0}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-z^2/2}\,\mathrm{d}z \\
&= \int_{\zeta_0}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-(z-\zeta_0)^2/2 - \zeta_0^2/2 - z\zeta_0}\,\mathrm{d}z \\
&\leq e^{-\zeta_0^2/2} \int_{\zeta_0}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-(z-\zeta_0)^2/2}\,\mathrm{d}z \\
&= \frac{1}{2} e^{-\zeta_0^2/2}.
\end{aligned}$$

Similarly, $\Pr[z < -\zeta_0] \leq \frac{1}{2} e^{-\zeta_0^2/2}$. We reach the conclusion by rearranging the constants. □

Lemma A.1.2. *Assume $X_1, \cdots, X_n \in \mathbb{R}^m$ are sampled i.i.d. from $\mathcal{N}(u, V)$. Suppose we estimate the sample mean to be $\hat{u} = \frac{1}{n} X^T 1_n$ and the sample covariance to be $\hat{V} = \frac{1}{n-1}(X - 1_n \hat{u}^T)^T (X - 1_n \hat{u}^T)$ where $X = [X_i]_{i=1}^n \in \mathbb{R}^{n \times m}$. Then, $\hat{u}$ and $\hat{V}$ are independent, and*

$$\hat{u} \sim \mathcal{N}(u, \frac{1}{n} V),\ \ \hat{V} \sim \mathcal{W}(\frac{1}{n-1} V, n-1).$$

Lemma A.1.2 is a combination of Theorem 3.3.2 and Corollary 7.2.3 of [4]. Interested readers can find the proof of Lemma A.1.2 in [4]. Corollary A.1.3 directly follows Lemma A.1.2.

Corollary A.1.3. *$\hat{\mu}$ and $\hat{k}$ are independent and*

$$\hat{\mu}(\mathfrak{X}) \sim \mathcal{N}(\mu(\mathfrak{X}), \frac{1}{N}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I}),\ \ \hat{k}(\mathfrak{X}) \sim \mathcal{W}(\frac{1}{N-1}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I}), N-1).$$

Corollary A.1.4. *For any $X \sim \mathcal{W}(v, n), v \in \mathbb{R}$ and $b > 0$, we have*

$$\Pr[\frac{X}{vn} \geq 1 + 2\sqrt{b} + 2b] \leq e^{-bn}, \;\; \Pr[\frac{X}{vn} \leq 1 - 2\sqrt{b}] \leq e^{-bn}.$$

Proof. Let X be a random variable such that $X \sim \mathcal{W}(v, n)$. So $\frac{X}{v}$ is distributed according to a chi-squared distribution with n degrees of freedom; namely, $\frac{X}{v} \sim \chi^2(n)$. By Lemma 1 in [116], we have

$$\Pr[\frac{X}{v} - n \geq 2\sqrt{na} + 2a] \leq e^{-a}, \;\; \Pr[\frac{X}{v} - n \leq -2\sqrt{na}] \leq e^{-a}.$$

As a result, if $a = bn$,

$$\Pr[\frac{X}{vn} \geq 1 + 2\sqrt{b} + 2b] \leq e^{-bn}, \;\; \Pr[\frac{X}{vn} \leq 1 - 2\sqrt{b}] \leq e^{-bn}.$$

□

Lemma A.1.5. *Let $X \in \mathbb{R}^d$ be a sample from $\mathcal{N}(w, V)$ and define $Z = (X - w)^T V^{-1}(X - w)$. Then, we have $Z \sim \chi^2(d)$. With probability at least $1 - \delta_0$, $Z < d + 2\sqrt{d \log \frac{1}{\delta_0}} + 2 \log \frac{1}{\delta_0}$.*

Proof. By [166], $Z \sim \chi^2(d)$. The bound on Z follows Lemma 1 in [116]. □

Lemma A.1.6. *Pick $\delta_1 \in (0, 1)$ and $\delta_2 \in (0, 1)$. For any fixed non-negative integer $t < T$, conditioned on the observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, our estimators $\hat{\mu}_t$ and $\hat{k}_t$ satisfy*

$$\mathbb{E}[\hat{\mu}_t(\mathfrak{X})] = \mu_t(\mathfrak{X}), \;\; \mathbb{E}[\hat{k}_t(\mathfrak{X})] = k_t(\mathfrak{X}) + \sigma^2 \boldsymbol{I}.$$

Suppose $N \geq T + 2$. Then, for any fixed inputs $x, z \in \mathfrak{X}$,

$$\Pr\left[\hat{\mu}_t(x) - \mu_t(x) < \iota_t \sqrt{(k_t(x) + \sigma^2)} \wedge \hat{\mu}_t(z) - \mu_t(z) > -\iota_t \sqrt{(k_t(z) + \sigma^2)}\right] \geq 1 - \delta_1, \quad \text{(A.5)}$$

$$\Pr[\frac{\hat{k}_t(x)}{k_t(x) + \sigma^2} < 1 + 2\sqrt{b_t} + 2b_t] \geq 1 - \delta_2, \;\; \Pr[\frac{\hat{k}_t(x)}{k_t(x) + \sigma^2} > 1 - 2\sqrt{b_t}] \geq 1 - \delta_2. \quad \text{(A.6)}$$

where $\iota_t = \sqrt{\frac{2\left(N-2+t+2\sqrt{t \log \frac{2}{\delta_1}} + 2 \log \frac{2}{\delta_1}\right)}{\delta_1 N(N-t-2)}}$ and $b_t = \frac{1}{N-t-1} \log \frac{1}{\delta_2}$.

Proof. By assumption, all rows of the observation $Y = [\bar{y}_{ij}]_{i \in [N], j \in [M]}$ are sampled i.i.d. from

$\mathcal{N}(\mu(\mathfrak{X}), k(\mathfrak{X}) + \sigma^2 \boldsymbol{I})$. By Corollary A.1.3,

$$\hat{\mu}(\mathfrak{X}) \sim \mathcal{N}(\mu, \frac{1}{N}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I})), \;\; \hat{k}(\mathfrak{X}) \sim \mathcal{W}(\frac{1}{N-1}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I}), N-1).$$

By Proposition 8.7 in [52], we have

$$\hat{k}(x, x') - \hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t, \boldsymbol{x}_t)^{-1}\hat{k}(\boldsymbol{x}_t, x') \sim \mathcal{W}(\frac{1}{N-1}(k_t(x, x') + \sigma^2 \mathbb{1}_{x=x'}), N-t-1).$$

Hence, the estimate $\hat{k}_t$ satisfy

$$\hat{k}_t(x) \sim \mathcal{W}(\frac{1}{N-t-1}(k_t(x) + \sigma^2), N-t-1) \tag{A.7}$$

Clearly, $\mathbb{E}[\hat{k}_t(x)] = k_t(x) + \sigma^2$. Now it is easy to show Eq. (A.6). By Corollary A.1.4, for any fixed $t \in [T] \cup 0$ and x, $\forall \frac{1}{4} \geq b_t > 0$,

$$\begin{aligned} &\Pr[\frac{\hat{k}_t(x)}{k_t(x) + \sigma^2} \geq 1 + 2\sqrt{b_t} + 2b_t] \leq e^{-b_t(N-t-1)}, \\ &\Pr[\frac{\hat{k}_t(x)}{k_t(x) + \sigma^2} \leq 1 - 2\sqrt{b_t}] \leq e^{-b_t(N-t-1)}. \end{aligned} \tag{A.8}$$

where $b_t = \frac{1}{N-t-1} \log \frac{1}{\delta_2} > 0$ and $\delta_2 \in (0, 1)$. Thus, we have shown Eq. (A.6).

We next prove the second half of the results for $\hat{\mu_t}$ in Eq. (A.5). We use the shorthand $S = \frac{1}{N-1}(k(\mathfrak{X}) + \sigma^2 \boldsymbol{I})$. By definition of the Wishart distributions in [52] (Definition 8.1), there exist random vectors $X_1, \cdots, X_{N-1} \in \mathbb{R}^M$ sampled iid from $\mathcal{N}(0, S), \forall i = 1, \cdots, N-1$, and $\hat{k}(\mathfrak{X}) = \sum_{i=1}^{n-1} X_i X_i^{\mathrm{T}}$. We denote $X \in \mathbb{R}^{(N-1) \times M}$ as a matrix whose i-th row is X_i. Clearly, $\hat{k}(\mathfrak{X}) = X^{\mathrm{T}} X$ and $\hat{k}(\mathfrak{X}_a, \mathfrak{X}_b) = X_{\cdot,a}^{\mathrm{T}} X_{\cdot,b}, \forall a, b \subseteq [M]$. Let the indices of $\boldsymbol{x}_t$ in $\mathfrak{X}$ be $\Theta_t \subseteq [M]$ and the index of x in $\mathfrak{X}$ be $\theta \in [M]$. Thus we have $\boldsymbol{x}_t = \mathfrak{X}_{\Theta_t}$ and $x = \mathfrak{X}_\theta$.

Conditional on $\hat{\mu}(\boldsymbol{x}_t)$ and $X_{\cdot,\Theta_t}$, the term $\hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))$ is a weighted sum of independent Gaussian variables, because $X_{\cdot,\theta}^{\mathrm{T}}$ consists of independent Gaussian variables and $\hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t)) = X_{\cdot,\theta}^{\mathrm{T}} P$ where $P = X_{\cdot,\Theta_t} \left(X_{\cdot,\Theta_t}^{\mathrm{T}} X_{\cdot,\Theta_t} \right)^{-1} (\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))$. Recall that $X_i \sim \mathcal{N}(0, S)$; hence, we have

$$X_{\cdot,\theta} \mid X_{\cdot,\Theta_t} \sim \mathcal{N}(X_{\cdot,\Theta_t} S_{\Theta_t}^{-1} S_{\Theta_t,\theta}, \boldsymbol{I}_{N-1} \otimes S_{\theta|\Theta_t}),$$

where $S_{\theta|\Theta_t} = S_\theta - S_{\theta,\Theta_t} S_{\Theta_t}^{-1} S_{\theta,\Theta_t}^{\mathrm{T}}$. As a result, the Gaussian variable $X_{\cdot,\theta}^{\mathrm{T}} P$ has mean

$$\mathbb{E}[X_{\cdot,\theta}^{\mathrm{T}} P \mid \hat{\mu}(\boldsymbol{x}_t), X_{\cdot,\Theta_t}] = S_{\theta,\Theta_t} S_{\Theta_t}^{-1} (\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))$$

and variance

$$\mathbb{V}[X_{\cdot,\theta}^{\mathrm{T}} P \mid \hat{\mu}(\boldsymbol{x}_t), X_{\cdot,\Theta_t}] = (\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))^{\mathrm{T}} \hat{k}(\boldsymbol{x}_t)^{-1} (\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t)) S_{\theta|\Theta_t}.$$

By independence between $\hat{k}(\mathfrak{X})$ and $\hat{\mu}(\mathfrak{X})$ shown in Corollary A.1.3, we can show that $\hat{k}(x, \boldsymbol{x}_t)$ and $\hat{\mu}(x)$ are independent conditional on $\hat{\mu}(\boldsymbol{x}_t)$ and $\hat{k}(\boldsymbol{x}_t)$, by noting that

$$\begin{aligned}
&p(\hat{\mu}(\mathfrak{X}), \hat{k}(\mathfrak{X})) = p(\hat{\mu}(\mathfrak{X}))p(\hat{k}(\mathfrak{X})) \\
\Rightarrow &p(\hat{\mu}(\boldsymbol{x}_t \cup \{x\}), \hat{k}(\boldsymbol{x}_t \cup \{x\})) = p(\hat{\mu}(\boldsymbol{x}_t \cup \{x\}))p(\hat{k}(\boldsymbol{x}_t \cup \{x\})) \\
\Rightarrow &p(\hat{\mu}(\boldsymbol{x}_t \cup \{x\}), \hat{k}(\boldsymbol{x}_t \cup \{x\})) = p(\hat{\mu}(\boldsymbol{x}_t \cup \{x\}) \mid \hat{k}(\boldsymbol{x}_t))p(\hat{k}(\boldsymbol{x}_t \cup \{x\}) \mid \hat{\mu}(\boldsymbol{x}_t)) \\
\Rightarrow &p(\hat{\mu}(x), \hat{k}(x), \hat{k}(x, \boldsymbol{x}_t) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)) = p(\hat{\mu}(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t))p(\hat{k}(x), \hat{k}(x, \boldsymbol{x}_t) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)) \\
\Rightarrow &p(\hat{\mu}(x), \hat{k}(x, \boldsymbol{x}_t) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)) = p(\hat{\mu}(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)))p(\hat{k}(x, \boldsymbol{x}_t) \mid \hat{\mu}(\boldsymbol{x}_t)), \hat{k}(\boldsymbol{x}_t)).
\end{aligned}$$

Hence, $\hat{\mu}(x)$ and $X_{\cdot,\theta}^{\mathrm{T}} P = \hat{k}(x, \boldsymbol{x}_t)\hat{k}(\boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))$ are independent conditional on $\hat{\mu}(\boldsymbol{x}_t)$ and $\hat{k}(\boldsymbol{x}_t)$. Moreover, $X_{\cdot,\theta}^{\mathrm{T}} P$ is dependent on $X_{\cdot,\Theta_t}$ only through $\hat{k}(\boldsymbol{x}_t) = X_{\cdot,\Theta_t}^{\mathrm{T}} X_{\cdot,\Theta_t}$; hence, we have

$$\hat{\mu}_t(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t) \sim \mathcal{N}(\bar{\mu}, \bar{S}), \tag{A.9}$$

By linearity of expectation and the Bienaymé formula,

$$\begin{aligned}
\bar{\mu} &= \mathbb{E}[\hat{\mu}(x) \mid \hat{\mu}(\boldsymbol{x}_t)] + k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t)) && \text{(A.10)} \\
&= \mu(x) + k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \mu(\boldsymbol{x}_t)) \\
&= \mu_t(x), \\
\bar{S} &= \mathbb{V}[\hat{\mu}(x) \mid \hat{\mu}(\boldsymbol{x}_t)] + \frac{(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))^{\mathrm{T}} \hat{k}(\boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))(k_t(x) + \sigma^2)}{n-1}, && \text{(A.11)} \\
&= \frac{k_t(x) + \sigma^2}{N} + \frac{(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))^{\mathrm{T}} \hat{k}(\boldsymbol{x}_t)^{-1}(\boldsymbol{y}_t - \hat{\mu}(\boldsymbol{x}_t))(k_t(x) + \sigma^2)}{N-1}.
\end{aligned}$$

In Eq. (A.10) and Eq. (A.11), we use the conditional Gaussian distribution for $\hat{\mu}(x)$ as follows

$$\hat{\mu}(x) \mid \hat{\mu}(\boldsymbol{x}_t) \sim \mathcal{N}(\mu(x) + k(x, \boldsymbol{x}_t)(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\hat{\mu}(\boldsymbol{x}_t) - \mu(\boldsymbol{x}_t)), \frac{k_t(x) + \sigma^2}{N}).$$

By the law of total expectation,

$$\mathbb{E}[\hat{\mu}_t(x)] = \mathbb{E}\left[\mathbb{E}[\hat{\mu}_t(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)]\right] = \mu_t(x). \tag{A.12}$$

By the law of total variance,

$$\begin{aligned}
\mathbb{V}[\hat{\mu}_t(x)] &= \mathbb{E}\left[\mathbb{V}[\hat{\mu}_t(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)]\right] + \mathbb{V}\left[\mathbb{E}[\hat{\mu}_t(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)]\right] \\
&= \mathbb{E}\left[\bar{S}\right] + \mathbb{V}\left[\bar{\mu}\right] \\
&= \frac{\left(N - 2 + (\boldsymbol{y}_t - \mu(\boldsymbol{x}_t))^{\mathrm{T}}(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \mu(\boldsymbol{x}_t))\right)(k_t(x) + \sigma^2)}{N(N - t - 2)} \\
&= \frac{(N - 2 + K_{\boldsymbol{x}_t, \boldsymbol{y}_t})(k_t(x) + \sigma^2)}{N(N - t - 2)}.
\end{aligned}$$

where $K_{\boldsymbol{x}_t, \boldsymbol{y}_t} = (\boldsymbol{y}_t - \mu(\boldsymbol{x}_t))^{\mathrm{T}}(k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y}_t - \mu(\boldsymbol{x}_t))$.

Notice that $\hat{\mu}_t(x) \mid \hat{\mu}(\boldsymbol{x}_t), \hat{k}(\boldsymbol{x}_t)$ in Eq. (A.9) is a normal distribution centered at $\mu_t(x)$ regardless of the conditional distribution. So the distribution of $\hat{\mu}_t(x)$ must be symmetric with a center at $\mu_t(x)$. Hence, applying Chebyshev's inequality, we have

$$\begin{aligned}
\Pr\left[\hat{\mu}_t(x) - \mu_t(x) < \sqrt{\frac{(N - 2 + K_{\boldsymbol{x}_t, \boldsymbol{y}_t})(k_t(x) + \sigma^2)}{2\delta_1' N(N - t - 2)}}\right] &\geq 1 - \delta_1', \\
\Pr\left[\hat{\mu}_t(x) - \mu_t(x) > -\sqrt{\frac{(N - 2 + K_{\boldsymbol{x}_t, \boldsymbol{y}_t})(k_t(x) + \sigma^2)}{2\delta_1' N(N - t - 2)}}\right] &\geq 1 - \delta_1'.
\end{aligned}$$

Notice that the randomness of $K_{\boldsymbol{x}_t, \boldsymbol{y}_t}$ is from $\boldsymbol{y}_t$ and $\boldsymbol{y}_t \sim \mathcal{N}(\mu(\boldsymbol{x}_t), k(\boldsymbol{x}_t) + \sigma^2 \boldsymbol{I})$. So we can further bound $K_{\boldsymbol{x}_t, \boldsymbol{y}_t} \leq t + 2\sqrt{t \log \frac{1}{\delta_1''}} + 2 \log \frac{1}{\delta_1''}$ with probability at most δ_1'' by Corollary A.1.4. Hence, if we set $\delta_1' = \frac{\delta_1}{4}$ and $\delta_1'' = \frac{\delta_1}{2}$, with probability at least $1 - \delta_1$, we have

$$\hat{\mu}_t(x) - \mu_t(x) < \iota_t \sqrt{(k_t(x) + \sigma^2)} \wedge \hat{\mu}_t(z) - \mu_t(z) > -\iota_t \sqrt{(k_t(z) + \sigma^2)},$$

for fixed inputs x, x'.

Combining this result and the results in Eq. (A.7), Eq. (A.8), Eq. (A.12), we proved the lemma.

□

Lemma A.1.7 (Lemma 7.3.1 in Chapter 7)**.** *Pick probability $\delta \in (0,1)$. For any nonnegative integer $t < T$, conditioned on the observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^t$, the estimators in Eq.* (A.1) *and Eq.* (A.2) *satisfy $\mathbb{E}[\hat{\mu}_t(\mathfrak{X})] = \mu_t(\mathfrak{X}), \mathbb{E}[\hat{k}_t(\mathfrak{X})] = k_t(\mathfrak{X}) + \sigma^2 \boldsymbol{I}$. Moreover, if the size of the training dataset satisfy $N \geq T + 2$, then for any input $x \in \mathfrak{X}$, with probability at least $1 - \delta$, both*

$$|\hat{\mu}_t(x) - \mu_t(x)|^2 < a_t(k_t(x) + \sigma^2) \;\; \textit{and} \;\; 1 - 2\sqrt{b_t} < \hat{k}_t(x)/(k_t(x) + \sigma^2) < 1 + 2\sqrt{b_t} + 2b_t$$

hold, where $a_t = \frac{4\left(N-2+t+2\sqrt{t\log(4/\delta)}+2\log(4/\delta)\right)}{\delta N(N-t-2)}$ and $b_t = \frac{1}{N-t-1}\log\frac{4}{\delta}$.

Proof. By a union bound on Eq. (A.8) of Lemma A.1.6, we have

$$\Pr\left[1 - 2\sqrt{b_t} < \hat{k}_t(x)/(k_t(x) + \sigma^2) < 1 + 2\sqrt{b_t} + 2b_t\right] \geq 1 - 2e^{-b_t(N-t-1)}$$

where $b_t = \frac{1}{N-t-1}\log\frac{1}{\delta_2} > 0$ and $\delta_2 \in (0,1)$. By Lemma A.1.6, we also have

$$\Pr\left[\hat{\mu}_t(x) - \mu_t(x) < \iota_t\sqrt{(k_t(x) + \sigma^2)} \wedge \hat{\mu}_t(z) - \mu_t(z) > -\iota_t\sqrt{(k_t(z) + \sigma^2)}\right] \geq 1 - \delta_1,$$

where $\iota_t = \sqrt{\frac{2\left(N-2+t+2\sqrt{t\log\frac{2}{\delta_1}}+2\log\frac{2}{\delta_1}\right)}{\delta_1 N(N-t-2)}}$. We get the conclusion of this lemma by setting $a_t = \iota_t, \delta_1 = \delta_2 = \frac{\delta}{2}$, and $z = x$. □

Corollary A.1.8 (Corollary of Bernoulli's inequality)**.** *For any $0 \leq x \leq c$ and $a > 0$, we have $x \leq \frac{c\log(1+\frac{ax}{c})}{\log(1+a)}$.*

Proof. By Bernoulli's inequality, $(1+a)^{\frac{x}{c}} \leq 1 + \frac{ax}{c}$. Because $\log(1+a) > 0$, by rearranging, we have $x \leq \frac{c\log(1+\frac{ax}{c})}{\log(1+a)}$. □

Lemma A.1.9. *For any $0 \leq x \leq c$ and $a > 0$, we have $\sqrt{x} < \sqrt{x+a} - \frac{a}{2\sqrt{c+a}}$.*

Proof. Numerically, for any $n \geq 1$, $\frac{1}{\sqrt{n}} < 2\sqrt{n} - 2\sqrt{n-1}$ [195]. Let $n = \frac{x}{a} + 1$. Then, we have

$$\begin{aligned}
\frac{1}{\sqrt{\frac{x}{a}+1}} &< 2\sqrt{\frac{x}{a}+1} - 2\sqrt{\frac{x}{a}} \\
\frac{a}{\sqrt{a+c}} < \frac{a}{\sqrt{a+x}} &< 2\sqrt{x+a} - 2\sqrt{x} \\
\sqrt{x} &< \sqrt{x+a} - \frac{a}{2\sqrt{a+c}}.
\end{aligned}$$

□

Lemma A.1.10 (Lemma 5.3 of [171]). *Let $\boldsymbol{x}_T = [x_t]_{t=1}^T \subseteq \mathfrak{X}$. The mutual information between the function values $f(\boldsymbol{x}_T)$ and their observations $\boldsymbol{y}_T = [y_t]_{t=1}^T$ satisfy*

$$I(f(\boldsymbol{x}_T); \boldsymbol{y}_T) = \frac{1}{2}\log\det(\boldsymbol{I} + \sigma^{-2}k(\boldsymbol{x}_t)) = \frac{1}{2}\sum_{t=1}^{k}\log(1+\sigma^{-2}k_{t-1}(x_t)).$$

Theorem A.1.11. *Assume there exist constant $c \geq \max_{x\in\mathfrak{X}} k(x)$ and a training dataset is available whose size is $N \geq 4\log\frac{6}{\delta} + T + 2$. Define*

$$\iota_{t-1} = \sqrt{\frac{6\left(N-3+t+2\sqrt{t\log\frac{6}{\delta}}+2\log\frac{6}{\delta}\right)}{\delta N(N-t-1)}},\ b_{t-1} = \frac{1}{N-t}\log\frac{6}{\delta},\ \textit{for any } t\in[T],$$

and $\rho_T = \max\limits_{A\in\mathfrak{X},|A|=T}\frac{1}{2}\log|\boldsymbol{I}+\sigma^{-2}k(A)|$. Then, with probability at least $1-\delta$, the best-sample simple regret in T iterations of meta BO with GP-UCB that uses Eq. (A.4) *as its hyperparameter satisfies*

$$r_T^{\textit{GP-UCB}} \leq \eta^{\textit{GP-UCB}}\sqrt{\frac{2c\rho_T}{T\log(1+c\sigma^{-2})}+\sigma^2} - \frac{(2\log(\frac{3}{\delta}))^{\frac{1}{2}}\sigma^2}{\sqrt{c+\sigma^2}},$$

where $\eta^{\textit{GP-UCB}} = (\frac{\iota_{T-1}+(2\log(\frac{3}{\delta}))^{\frac{1}{2}}}{\sqrt{1-2\sqrt{b_{T-1}}}}\sqrt{1+2\sqrt{b_{T-1}}+2b_{T-1}}+\iota_{T-1}+(2\log(\frac{3}{\delta}))^{\frac{1}{2}})$.

With probability at least $1-\delta$, the best-sample simple regret in T iterations of meta BO with PI that uses $\hat{f}^* \geq \max_{x\in\mathfrak{X}} f(x)$ as its target value satisfies

$$r_T^{\textit{PI}} < \eta^{\textit{PI}}\sqrt{\frac{2c\rho_T}{T\log(1+c\sigma^{-2})}+\sigma^2} - \frac{(2\log(\frac{3}{2\delta}))^{\frac{1}{2}}\sigma^2}{2\sqrt{c+\sigma^2}},$$

where $\eta^{\textit{PI}} = (\frac{\hat{f}^*-\mu_{\tau-1}(x_*)}{\sqrt{k_{\tau-1}(x_*)+\sigma^2}}+\iota_{\tau-1})\sqrt{\frac{1+2b_{\tau-1}^{\frac{1}{2}}+2b_{\tau-1}}{1-2b_{\tau-1}^{\frac{1}{2}}}}+\iota_{\tau-1}+(2\log(\frac{3}{2\delta}))^{\frac{1}{2}}$, $\tau = \arg\min_{t\in[T]} k_{t-1}(x_t)$.

Proof. We first show the regret bound for GP-UCB with our estimators of prior and posterior. All of the probabilities mentioned in the proofs need to be interpreted in a frequentist manner. Let $\tau = \arg\min_{t\in[T]} k_{t-1}(x_t)$. By Corollary A.1.1, with probability at least $1-\frac{\delta}{3}$,

$$\begin{aligned}
r_T^{\text{GP-UCB}} &= f^* - \max_{t\in[T]} f(x_t)\\
&\leq f^* - f(x_\tau)\\
&\leq f^* - \mu_{\tau-1}(x_\tau) + \mu_{\tau-1}(x_\tau) - f(x_\tau)\\
&\leq \mu_{\tau-1}(x_*) + \zeta'\sqrt{k_{\tau-1}(x_*)} - \mu_{\tau-1}(x_\tau) + \zeta'\sqrt{k_{\tau-1}(x_\tau)},
\end{aligned}$$

where $\zeta' = (2\log(\frac{3}{\delta}))^{\frac{1}{2}}$. By Lemma A.1.6, with probability at least $1 - \frac{\delta}{3}$,

$$\mu_{\tau-1}(x_*) - \mu_{\tau-1}(x_\tau) < \hat{\mu}_{\tau-1}(x_*) - \hat{\mu}_{\tau-1}(x_\tau) + \iota_{\tau-1}\sqrt{k_{\tau-1}(x_*) + \sigma^2} + \iota_{\tau-1}\sqrt{k_{\tau-1}(x_\tau) + \sigma^2},$$

where $\iota_t = \sqrt{\frac{6\left(N-2+t+2\sqrt{t\log\frac{6}{\delta}}+2\log\frac{6}{\delta}\right)}{\delta N(N-t-2)}} \leq \iota_{T-1}$.

Lemma A.1.6 and Lemma A.1.9 also show that with probability at least $1 - \frac{\delta}{6}$, we have

$$\sqrt{k_{\tau-1}(x_*)} \leq \sqrt{k_{\tau-1}(x_*) + \sigma^2} - \frac{\sigma^2}{2\sqrt{c+\sigma^2}} < \sqrt{\frac{\hat{k}_{\tau-1}(x_*)}{1 - 2\sqrt{b_{\tau-1}}}} - \frac{\sigma^2}{2\sqrt{c+\sigma^2}}$$

where $b_t = \frac{1}{N-t-1}\log\frac{6}{\delta} \leq b_{T-1} \in (0, \frac{1}{4})$. Notice that because of the input selection strategy of GP-UCB with $\zeta_t = \frac{\iota_{t-1}+\zeta'}{\sqrt{1-2\sqrt{b_{t-1}}}}$, the following inequality holds with probability at least $1 - \frac{\delta}{6}$,

$$\begin{aligned}\hat{\mu}_{\tau-1}(x_*) + (\iota_{t-1} + \zeta')\sqrt{k_{\tau-1}(x_*) + \sigma^2} &\leq \hat{\mu}_{\tau-1}(x_*) + \zeta_t\sqrt{\hat{k}_{\tau-1}(x_*)} \\ &\leq \hat{\mu}_{\tau-1}(x_\tau) + \zeta_t\sqrt{\hat{k}_{\tau-1}(x_\tau)}.\end{aligned}$$

Hence, with probability at least $1 - \delta$,

$$\begin{aligned}r_T^{\text{GP-UCB}} &\leq \mu_{\tau-1}(x_*) + \zeta'\sqrt{k_{\tau-1}(x_*) + \sigma^2} - \mu_{\tau-1}(x_\tau) + \zeta'\sqrt{k_{\tau-1}(x_\tau) + \sigma^2} - \frac{\zeta'\sigma^2}{\sqrt{c+\sigma^2}} \\ &< \hat{\mu}_{\tau-1}(x_*) - \hat{\mu}_{\tau-1}(x_\tau) + (\iota_{t-1} + \zeta')(\sqrt{k_{\tau-1}(x_*) + \sigma^2} + \sqrt{k_{\tau-1}(x_\tau) + \sigma^2}) - \frac{\zeta'\sigma^2}{\sqrt{c+\sigma^2}} \\ &\leq \zeta_t\sqrt{\hat{k}_{\tau-1}(x_\tau)} + (\iota_{t-1} + \zeta')\sqrt{k_{\tau-1}(x_\tau) + \sigma^2} - \frac{\zeta'\sigma^2}{\sqrt{c+\sigma^2}} \\ &< (\zeta_t\sqrt{1 + 2\sqrt{b_{t-1}} + 2b_{t-1}} + \iota_{t-1} + \zeta')\sqrt{k_{\tau-1}(x_\tau) + \sigma^2} - \frac{\zeta'\sigma^2}{\sqrt{c+\sigma^2}} \\ &< \eta^{\text{GP-UCB}}\sqrt{k_{\tau-1}(x_\tau) + \sigma^2} - \frac{\zeta'\sigma^2}{\sqrt{c+\sigma^2}},\end{aligned}$$

where $\eta^{\text{GP-UCB}} = (\frac{\iota_{T-1}+\zeta'}{\sqrt{1-2\sqrt{b_{T-1}}}}\sqrt{1 + 2\sqrt{b_{T-1}} + 2b_{T-1}} + \iota_{T-1} + \zeta')$. By Corollary A.1.8 and the

fact that $\tau = \arg\min_{t\in[T]} k_{t-1}(x_t)$, we have

$$\begin{aligned} k_{\tau-1}(x_\tau) &\leq \frac{1}{T}\sum_{t=1}^{T} k_{t-1}(x_t) \\ &\leq \frac{1}{T}\sum_{t=1}^{T} \frac{c\log(1+\frac{c\sigma^{-2}k_{t-1}(x_t)}{c})}{\log(1+c\sigma^{-2})} \\ &= \frac{c}{T\log(1+c\sigma^{-2})}\sum_{t=1}^{T}\log(1+\sigma^{-2}k_{t-1}(x_t)). \end{aligned}$$

Notice that here Corollary A.1.8 applies because $0 \leq k_{\tau-1}(x_\tau) \leq c$.

By Lemma A.1.10, $I(f(\boldsymbol{x}_T); \boldsymbol{y}_T) = \frac{1}{2}\sum_{t=1}^{T}\log(1+\sigma^{-2}k_{t-1}(x_t)) \leq \rho_T$, so

$$k_{\tau-1}(x_\tau) \leq \frac{2c\rho_T}{T\log(1+c\sigma^{-2})},$$

which implies

$$r_T^{\text{GP-UCB}} < \eta\sqrt{\frac{2c\rho_T}{T\log(1+c\sigma^{-2})}+\sigma^2} - \frac{\zeta'\sigma^2}{\sqrt{c+\sigma^2}}.$$

Next, we show the proof for a special case of PI with $\hat{f}^*$, an upper bound on f, as its target value. Again, by Corollary A.1.1, with probability at least $1-\frac{\delta}{3}$,

$$\begin{aligned} r_T^{\text{PI}} &= \hat{f}^* - \max_{t\in[T]} f(x_t) \\ &\leq \hat{f}^* - f(x_\tau) \\ &\leq \hat{f}^* - \mu_{\tau-1}(x_\tau) + \mu_{\tau-1}(x_\tau) - f(x_\tau) \\ &\leq \hat{f}^* - \mu_{\tau-1}(x_\tau) + \zeta'\sqrt{k_{\tau-1}(x_\tau)}, \end{aligned}$$

where $\zeta' = (2\log(\frac{3}{2\delta}))^{\frac{1}{2}}$ and $\tau = \arg\min_{t\in[T]} k_{t-1}(x_t)$. By Lemma A.1.6 and the selection

strategy of PI, with probability at least $1-\frac{2\delta}{3}$,

$$\begin{aligned}
\hat{f}^* - \mu_{\tau-1}(x_\tau) &< \hat{f}^* - \hat{\mu}_{\tau-1}(x_\tau) + \iota_{\tau-1}\sqrt{k_{\tau-1}(x_\tau)+\sigma^2} \\
&\leq \frac{\hat{f}^* - \hat{\mu}_{\tau-1}(x_*)}{\sqrt{\hat{k}_{\tau-1}(x_*)}}\sqrt{\hat{k}_{\tau-1}(x_\tau)} + \iota_{\tau-1}\sqrt{k_{\tau-1}(x_\tau)+\sigma^2} \\
&\leq \frac{\hat{f}^* - \mu_{\tau-1}(x_*) + \iota_{\tau-1}\sqrt{k_{\tau-1}(x_*)+\sigma^2}}{\sqrt{\hat{k}_{\tau-1}(x_*)}}\sqrt{\hat{k}_{\tau-1}(x_\tau)} + \iota_{\tau-1}\sqrt{k_{\tau-1}(x_\tau)+\sigma^2} \\
&< \left((\frac{\hat{f}^* - \mu_{\tau-1}(x_*)}{\sqrt{k_{\tau-1}(x_*)+\sigma^2}} + \iota_{\tau-1}) \sqrt{\frac{1+2b_{\tau-1}^{\frac{1}{2}}+2b_{\tau-1}}{1-2b_{\tau-1}^{\frac{1}{2}}}} + \iota_{\tau-1} \right) \sqrt{k_{\tau-1}(x_\tau)+\sigma^2}.
\end{aligned}$$

Hence, with probability at least $1-\delta$, the best-sample simple regret of PI satisfy

$$r_T^{\text{PI}} < \eta^{\text{PI}} \sqrt{\frac{2c\rho_T}{T\log(1+c\sigma^{-2})} + \sigma^2} - \frac{\zeta'\sigma^2}{2\sqrt{c+\sigma^2}},$$

where $\eta^{\text{PI}} = (\frac{\hat{f}^* - \mu_{\tau-1}(x_*)}{\sqrt{k_{\tau-1}(x_*)+\sigma^2}} + \iota_{\tau-1}) \sqrt{\frac{1+2b_{\tau-1}^{\frac{1}{2}}+2b_{\tau-1}}{1-2b_{\tau-1}^{\frac{1}{2}}}} + \iota_{\tau-1} + \zeta'$. □

Theorem A.1.12 (Theorem 7.3.2 in Chapter 7)**.** *Assume there exist constant $c \geq \max_{x\in\mathfrak{X}} k(x)$ and a training dataset is available whose size is $N \geq 4\log\frac{6}{\delta} + T + 2$. Then, with probability at least $1-\delta$, the best-sample simple regret in T iterations of meta BO with special cases of either GP-UCB or PI satisfies*

$$r_T^{UCB} < \eta_T^{UCB}(N)\lambda_T, \;\; r_T^{PI} < \eta_T^{PI}(N)\lambda_T, \;\; \lambda_T^2 = O(\rho_T/T) + \sigma^2,$$

where $\eta_T^{UCB}(N) = (m+C_1)(\frac{\sqrt{1+m}}{\sqrt{1-m}}+1)$, $\eta_T^{PI}(N) = (m+C_2)(\frac{\sqrt{1+m}}{\sqrt{1-m}}+1)+C_3$, $m = O(\sqrt{\frac{1}{N-T}})$, $C_1, C_2, C_3 > 0$ are constants, and $\rho_T = \max\limits_{A\in\mathfrak{X},|A|=T} \frac{1}{2}\log|\boldsymbol{I}+\sigma^{-2}k(A)|$.

Proof. This theorem is a condensed version of Thm. A.1.11 with big O notations. □

A.2 Proofs for Section 7.3.2

Recall that we assume $\mathfrak{X}$ is a compact set which is a subset of $\mathbb{R}^d$. We only considers a special case of GPs that assumes $f(x) = \Phi(x)^{\mathrm{T}}W$, $W \sim \mathcal{N}(\boldsymbol{u}, \Sigma)$ and the basis functions $\Phi(x) \in \mathbb{R}^K$ are

given. The mean function and kernel are defined as

$$\mu(x) = \Phi(x)^{\mathrm{T}}\boldsymbol{u} \;\text{ and }\; k(x) = \Phi(x)^{\mathrm{T}}\Sigma\Phi(x).$$

Given noisy observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^{t}, t \leq K$, we have

$$\mu_t(x) = \Phi(x)^{\mathrm{T}}\boldsymbol{u}_t \;\text{ and }\; k_t(x, x') = \Phi(x)^{\mathrm{T}}\Sigma_t\Phi(x'),$$

where the posterior of $W \sim \mathcal{N}(\boldsymbol{u}_t, \Sigma_t)$ satisfies

$$\begin{aligned}
\boldsymbol{u}_t &= \boldsymbol{u} + \Sigma\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\Sigma\Phi(\boldsymbol{x}_t) + \sigma^2\boldsymbol{I})^{-1}(\boldsymbol{y}_t - \Phi(\boldsymbol{x}_t)^{\mathrm{T}}\boldsymbol{u}),\\
\Sigma_t &= \Sigma - \Sigma\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\Sigma\Phi(\boldsymbol{x}_t) + \sigma^2\boldsymbol{I})^{-1}\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\Sigma.
\end{aligned}$$

Our estimators for $\boldsymbol{u}_t$ and Σ_t are

$$\begin{aligned}
\hat{\boldsymbol{u}}_t &= \hat{\boldsymbol{u}} + \hat{\Sigma}\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\hat{\Sigma}\Phi(\boldsymbol{x}_t))^{-1}(\boldsymbol{y}_t - \Phi(\boldsymbol{x}_t)^{\mathrm{T}}\boldsymbol{u}),\\
\hat{\Sigma}_t &= \frac{N-1}{N-t-1}\left(\hat{\Sigma} - \hat{\Sigma}\Phi(\boldsymbol{x}_t)(\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\hat{\Sigma}\Phi(\boldsymbol{x}_t))^{-1}\Phi(\boldsymbol{x}_t)^{\mathrm{T}}\hat{\Sigma}\right).
\end{aligned}$$

We can compute the approximated conditional mean and variance of the observation on $x \in \mathfrak{X}$ to be

$$\hat{\mu}_t(x) = \Phi(x)^{\mathrm{T}}\hat{\boldsymbol{u}}_t \;\text{ and }\; \hat{k}_t(x) = \Phi(x)^{\mathrm{T}}\hat{\Sigma}_t\Phi(x).$$

Again, we prove a bound on the best-sample simple regret $r_T = \max_{x\in\mathfrak{X}} f(x) - \max_{t\in[T]} f(x_t)$. The evaluated inputs $\boldsymbol{x}_t = [x_\tau]_\tau^t$ are selected either by a special case of GP-UCB using the acquisition function

$$\begin{aligned}
&\alpha_{t-1}^{\text{GP-UCB}}(x) = \hat{\mu}_{t-1}(x) + \zeta_t\hat{k}_{t-1}(x)^{\frac{1}{2}}, \;\text{ with}\\
&\zeta_t = \frac{\left(6(N-3+t+2\sqrt{t\log\frac{6}{\delta}} + 2\log\frac{6}{\delta})/(\delta N(N-t-1))\right)^{\frac{1}{2}} + (2\log(\frac{3}{\delta}))^{\frac{1}{2}}}{(1-2(\frac{1}{N-t}\log\frac{6}{\delta})^{\frac{1}{2}})^{\frac{1}{2}}}, \delta \in (0,1),
\end{aligned}$$

or by a special case of PI using the acquisition function

$$\alpha_{t-1}^{\text{PI}}(x) = \frac{\hat{\mu}_{t-1}(x) - \hat{f}^*}{\hat{k}_{t-1}(x)^{\frac{1}{2}}}.$$

This special case of PI assumes additional information of the upper bound on function value $\hat{f}^* \geq \max_{x \in \mathfrak{X}} f(x)$.

For convenience of the notations, we define $\bar{\sigma}^2(x) = \sigma^2 \Phi(x)^{\mathrm{T}} (\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^{\mathrm{T}})^{-1} \Phi(x)$.

Corollary A.2.1 combines Lemma A.1.2 and basic properties of the Wishart distribution [52].

Corollary A.2.1. *Assume the matrix $\Phi(\bar{\boldsymbol{x}}) \in \mathbb{R}^{K \times M}$ has linearly independent rows. Then, $\hat{\boldsymbol{u}}$ and $\hat{\Sigma}$ are independent and*

$$\hat{\boldsymbol{u}} \sim \mathcal{N}\left(\boldsymbol{u}, \frac{1}{N}(\Sigma + \sigma^2(\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^T)^{-1})\right), \hat{\Sigma} \sim \mathcal{W}\left(\frac{1}{N-1}\left(\Sigma + \sigma^2(\Phi(\bar{\boldsymbol{x}})\Phi(\bar{\boldsymbol{x}})^T)^{-1}\right), N-1\right).$$

For finite set of inputs $\boldsymbol{x} \subset \mathfrak{X}$, $\hat{\mu}(\boldsymbol{x})$ and $\hat{k}(\boldsymbol{x})$ are also independent; they satisfy

$$\hat{\mu}(\boldsymbol{x}) \sim \mathcal{N}\left(\mu, \frac{1}{N}(k(\boldsymbol{x}) + \bar{\sigma}^2(\boldsymbol{x}))\right), \hat{k}(\boldsymbol{x}) \sim \mathcal{W}\left(\frac{1}{N-1}\left(k(\boldsymbol{x}) + \bar{\sigma}^2(\boldsymbol{x})\right), N-1\right).$$

The proofs of Lemma 7.3.3 and Theorem 7.3.4 in Chapter 7 directly follow Corollary A.2.1 and proofs of Lemma A.1.6, Theorem A.1.11 in this appendix.

A.3 Proofs for Section 7.3.3

We show that the simple regret with $\hat{x}_T^* = x_\tau, \tau = \arg\max_{t \in [T]} y_t$ is very close to the best-sample simple regret.

Lemma A.3.1. *With probability at least $1 - \delta$, $R_T - r_T \leq 2(2 \log \frac{1}{\delta})^{\frac{1}{2}} \sigma$.*

Proof. Let $\tau' = \arg\max_{t \in [T]} f(x_t)$ and $\tau = \arg\max_{t \in [T]} y_t$. Note that $y_\tau \geq y_{\tau'}$. By Corollary A.1.1, with probability at least $1 - \delta$, $f(x_\tau) + C\sigma \geq y_\tau \geq y_{\tau'} \geq f(x_{\tau'}) - C\sigma$, where $C = (2 \log \frac{1}{\delta})^{\frac{1}{2}}$. Hence $R_T - r_T = f(x_{\tau'}) - f(x_\tau) \leq 2C\sigma$. □

Bibliography

[1] Raja Hafiz Affandi, Emily Fox, Ryan Adams, and Ben Taskar. Learning the parameters of determinantal point process kernels. In *International Conference on Machine Learning (ICML)*, 2014.

[2] Philip E Agre and David Chapman. Pengi: An implementation of a theory of activity. In *AAAI Conference on Artificial Intelligence*, volume 87, pages 286–272, 1987.

[3] James S Albus et al. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227, 1975.

[4] Theodore Wilbur Anderson. *An Introduction to Multivariate Statistical Analysis*. Wiley New York, 1958.

[5] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning (ICML)*, 2017.

[6] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[7] Peter Auer. Using confidence bounds for exploitation-exploration tradeoffs. *Journal of Machine Learning Research*, 3:397–422, 2002.

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

[9] Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013.

[10] Thomas Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[11] Leemon C. Baird, III and A. Harry Klopf. Reinforcement learning with high-dimensional continuous actions. Technical report, Wright Laboratory, Wright Patterson Air Force Base, 1993.

[12] Matej Balog, Balaji Lakshminarayanan, Zoubin Ghahramani, Daniel M Roy, and Yee Whye Teh. The Mondrian kernel. In *Uncertainty in Artificial Intelligence (UAI)*, 2016.

[13] Matej Balog and Yee Whye Teh. The Mondrian process for machine learning. *arXiv preprint arXiv:1507.05181*, 2015.

[14] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *International Conference on Machine Learning (ICML)*, 2013.

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

[16] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *ArXiv e-prints*, June 2018.

[17] J Baxter. A Bayesian /information theoretic model of bias learning. In *Conference on Learning Theory (COLT)*, New York, New York, USA, 1996.

[18] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[19] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[20] Scott S Benson. Learning action models for reactive autonomous agents. Technical report, Stanford University, Stanford, CA, USA, 1997.

[21] Dimitri P Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 1995.

[22] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022, 2003.

[23] Ilija Bogunovic, Jonathan Scarlett, Andreas Krause, and Volkan Cevher. Truncated variance reduction: A unified approach to Bayesian optimization and level-set estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[25] Sascha Brandi, Oliver Kroemer, and Jan Peters. Generalizing pouring actions between objects using warped parameters. In *Humanoids*, 2014.

[26] Pavel Brazdil, João Gama, and Bob Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *European Conference on Machine Learning (ECML)*, 1994.

[27] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical Report TR-2009-023, University of British Columbia, 2009.

[28] Brent Bryan, Robert C Nichol, Christopher R Genovese, Jeff Schneider, Christopher J Miller, and Larry Wasserman. Active learning for identifying function threshold boundaries. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2006.

[29] Adam D Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12:2879–2904, 2011.

[30] Roberto Calandra. *Bayesian Modeling for Optimization and Control in Robotics*. PhD thesis, Technische Universität, 2017.

[31] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717, 2009.

[32] Erin Catto. Box2D, a 2D physics engine for games. `http://box2d.org`, 2011.

[33] Augustin Louis Cauchy. *Cours d'analyse de l'Ecole Royale Polytechnique*, volume 1. Imprimerie Royale, 1821.

[34] Jianfei Chen, Jun Zhu, Zi Wang, Xun Zheng, and Bo Zhang. Scalable inference for logistic-normal topic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[35] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning (ICML)*, 2017.

[36] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[37] Sachin Chitta, Ioan Sucan, and Steve Cousins. MoveIt! *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

[38] Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013.

[39] Erwin Coumans and Yunfei Bai. Pybullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[40] A. Cully, J. Clune, D. Tarapore, and J. Mouret. Robots that adapt like animals. *Nature*, 2015.

[41] Erik A Daxberger and Bryan Kian Hsiang Low. Distributed batch Gaussian process optimization. In *International Conference on Machine Learning (ICML)*, 2017.

[42] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[43] Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian processes. In *International Conference on Machine Learning (ICML)*, 2015.

[44] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Model-based reinforcement learning with continuous states and actions. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 19–24, 2008.

[45] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7):1508–1524, 2009.

[46] Thomas Desautels, Andreas Krause, and Joel W Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 2014.

[47] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2010.

[48] Rosen Diankov and James Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, 2008.

[49] Josip Djolonga, Andreas Krause, and Volkan Cevher. High-dimensional Gaussian process bandits. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[50] Gary L Drescher. *Made-up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press, 1991.

[51] David K Duvenaud, Hannes Nickisch, and Carl E Rasmussen. Additive Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[52] M. L. Eaton. *Multivariate Statistics: A Vector Space Approach*. Beachwood, Ohio, USA: Institute of Mathematical Statistics, 2007.

[53] Bradley Efron. Bayes, oracle Bayes, and empirical Bayes. *Statistical Science*, 2019.

[54] Peter Englert and Marc Toussaint. Combined optimization and reinforcement learning for manipulation skills. In *Robotics: Science and Systems Conference (RSS)*, volume 2016, 2016.

[55] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[56] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for Bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018.

[57] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *AAAI Conference on Artificial Intelligence*, 2015.

[58] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[59] Ronald Aylmer Fisher. *The Genetical Theory of Natural Selection: A Complete Variorum Edition*. Oxford University Press, 1930.

[60] Malcolm R Forster. Notice: No free lunches for anyone, Bayesians included. *Department of Philosophy, University of Wisconsin–Madison Madison, USA*, 2005.

[61] Dean P Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1-2), 1999.

[62] Marcus Frean and Phillip Boyle. Using Gaussian processes to optimize expensive functions. In *Australasian Joint Conference on Artificial Intelligence*, pages 258–267. Springer, 2008.

[63] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, 2016.

[64] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sample-based methods for factored task and motion planning. In *Robotics: Science and Systems Conference (RSS)*, 2017.

[65] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research*, 37(13-14):1796–1825, 2018.

[66] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 8(5–6):359–483, 2015.

[67] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Elliot Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

[68] Javier González, Zhenwen Dai, Philipp Hennig, and Neil D Lawrence. Batch Bayesian optimization via local penalization. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[69] Michael A Goodrich, Alan C Schultz, et al. Human–robot interaction: A survey. *Foundations and Trends in Human–Computer Interaction*, 1(3):203–275, 2008.

[70] Alkis Gotovos, Nathalie Casati, Gregory Hitz, and Andreas Krause. Active learning for level set estimation. In *International Conference on Artificial Intelligence (IJCAI)*, 2013.

[71] GPy. GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

[72] Robert B Gramacy and Herbert K H Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

[73] V. Gullapalli, J. A. Franklin, and H. Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems*, 14(1):13–24, 1994.

[74] Raja Hafiz Affandi, Emily B Fox, and Ben Taskar. Approximate inference in continuous determinantal point processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[75] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012.

[76] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence (UAI)*, 2013.

[77] Tucker Hermans, Fuxin Li, James M Rehg, and Aaron F Bobick. Learning contact locations for pushing and orienting unknown objects. In *Humanoids*, 2013.

[78] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[79] José Miguel Hernández-Lobato, James Requeima, Edward O Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *International Conference on Machine Learning (ICML)*, 2017.

[80] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[81] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and others. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7310–7311, 2017.

[82] Vu Anh Huynh, Sertac Karaman, and Emilio Frazzoli. An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research*, 35(4):305–333, 2016.

[83] Christian Igel and Marc Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3(4):313–322, 2005.

[84] Hunor S Jakab and Lehel Csató. Reinforcement learning with guided policy search using Gaussian processes. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2012.

[85] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[86] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[87] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[88] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Learning composable models of parameterized skills. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[89] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Learning composable models of primitive actions. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[90] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[91] Kirthevasan Kandasamy, Jeff Schneider, and Barnabas Poczos. High dimensional Bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning (ICML)*, 2015.

[92] Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, D. Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *International Conference on Machine Learning (ICML)*, 2017.

[93] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Active learning with Gaussian processes for object categorization. In *International Conference on Computer Vision (ICCV)*. IEEE, 2007.

[94] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[95] Kenji Kawaguchi, Bo Xie, Vikas Verma, and Le Song. Deep semi-random features for nonlinear function approximation. In *AAAI Conference on Artificial Intelligence*, 2017.

[96] Robert W Keener. *Theoretical Statistics: Topics for a Core Course*. Springer, 2011.

[97] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[98] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[99] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Guiding Search in Continuous State-action Spaces by Learning an Action Sampler from Off-target Search Experience. In *AAAI Conference on Artificial Intelligence*, 2018.

[100] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 2019.

[101] George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal or Artificial Intelligence Research*, 2018.

[102] Marek Kopicki. *Prediction Learning in Robotic Manipulation*. PhD thesis, University of Birmingham, 2010.

[103] Marek Kopicki, Jeremy Wyatt, and Rustam Stolkin. Prediction learning in robotic pushing manipulation. In *International Conference on Advanced Robotics*. IEEE, 2009.

[104] Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Mörwald, and Jeremy Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[105] Andreas Krause and Cheng S Ong. Contextual Gaussian process bandit optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[106] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[107] Oliver Kroemer and Gaurav Sukhatme. Meta-level priors for learning manipulation skills with sparse features. In *International Symposium on Experimental Robotics (ISER)*, 2016.

[108] Oliver Kroemer and Gaurav S Sukhatme. Learning spatial preconditions of manipulation skills using random forests. In *Humanoids*, 2016.

[109] James J. Kuffner, Jr. and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[110] Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2–3), 2012.

[111] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 86(1):97–106, 1964.

[112] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[113] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests for large-scale regression when uncertainty matters. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[114] Rémi Lam, Matthias Poloczek, Peter Frazier, and Karen E Willcox. Advances in Bayesian optimization with applications in aerospace engineering. In *2018 AIAA Non-Deterministic Approaches Conference*, 2018.

[115] Tobias Lang and Marc Toussaint. Planning with noisy probabilistic relational rules. *Journal or Artificial Intelligence Research*, 39:1–49, 2010.

[116] Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pages 1302–1338, 2000.

[117] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept., Iowa State University, 1998.

[118] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2000.

[119] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

[120] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.

[121] Chun-Liang Li, Kirthevasan Kandasamy, Barnabás Póczos, and Jeff Schneider. High dimensional Bayesian optimization via restricted projection pursuit models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[122] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *International Conference on Learning Representations (ICLR)*, 2016.

[123] Michael Lederman Littman. *Algorithms for Sequential Decision Making*. Brown University Providence, RI, 1996.

[124] Daniel J Lizotte, Russell Greiner, and Dale Schuurmans. An experimental methodology for response surface optimization methods. *Journal of Global Optimization*, 53(4):699–736, 2012.

[125] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. Automatic gait optimization with Gaussian process regression. In *International Conference on Artificial Intelligence (IJCAI)*, 2007.

[126] Karim Lounici et al. High-dimensional covariance matrix estimation with missing observations. *Bernoulli*, 20(3):1029–1058, 2014.

[127] Gustavo Malkomes and Roman Garnett. Towards automated Bayesian optimization. In *ICML AutoML Workshop*, 2017.

[128] Gustavo Malkomes, Charles Schaff, and Roman Garnett. Bayesian optimization for automated model selection. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[129] Christopher R Mansley, Ari Weinstein, and Michael L Littman. Sample-based planning for continuous action Markov decision processes. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

[130] Ofir Marom and Benjamin Rosman. Zero-shot transfer with deictic object-oriented representation in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[131] Pascal Massart. *Concentration Inequalities and Model Selection*, volume 6. Springer, 2007.

[132] Mitchell McIntire, Daniel Ratner, and Stefano Ermon. Sparse Gaussian processes for Bayesian optimization. In *Uncertainty in Artificial Intelligence (UAI)*, 2016.

[133] T P Minka and R W Picard. Learning how to learn is learning with point sets. Technical report, MIT Media Lab, 1997.

[134] J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, 1974.

[135] Ankur Moitra and Gregory Valiant. Settling the polynomial learnability of mixtures of Gaussians. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

[136] Teodor Mihai Moldovan, Sergey Levine, Michael I Jordan, and Pieter Abbeel. Optimism-driven exploration for nonlinear systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[137] Kira Mourão. Learning probabilistic planning operators from noisy observations. In *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*, 2014.

[138] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Uncertainty in Artificial Intelligence (UAI)*, pages 614–623, 2012.

[139] R.M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics 118. Springer, 1996.

[140] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: A survey. *Cognitive Processing*, 12(4):319–340, 2011.

[141] Barry D. Nichols. Continuous action-space reinforcement learning methods applied to the minimum-time swing-up of the acrobot. In *IEEE International Conference on Systems, Man, and Cybernetics*, 2015.

[142] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[143] Zherong Pan, Chonhyon Park, and Dinesh Manocha. Robot motion planning for pouring liquids. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[144] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, pages 309–352, 2007.

[145] Sonia Petrone, Judith Rousseau, and Catia Scricciolo. Bayes and empirical Bayes: Do they merge? *Biometrika*, 101(2):285–302, 2014.

[146] John C Platt, Christopher JC Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a Gaussian process prior for automatically generating music playlists. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2002.

[147] Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[148] Matthias Poloczek, Jialei Wang, and Peter I Frazier. Warm starting Bayesian optimization. In *Winter Simulation Conference (WSC)*, 2016.

[149] D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007, 1997.

[150] Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[151] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[152] Herbert Robbins. An empirical Bayes approach to statistics. In *Third Berkeley Symp. Math. Statist. Probab.*, 1956.

[153] Axel Rottmann and Wolfram Burgard. Adaptive autonomous control using online value iteration with Gaussian processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[154] Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[155] Walter Rudin. *Fourier Analysis on Groups*. John Wiley & Sons, 2011.

[156] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[157] Claude Sammut and Geoffrey I Webb. *Encyclopedia of Machine Learning*. Springer Science & Business Media, 2011.

[158] Connor Schenck and Dieter Fox. Visual closed-loop control for pouring liquids. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[159] Connor Schenck, Jonathan Tompson, Dieter Fox, and Sergey Levine. Learning robotic manipulation of granular media. In *Conference on Robot Learning (CoRL)*, 2017.

[160] J Schmidhuber. On learning how to learn learning strategies. Technical report, FKI-198-94 (revised), 1995.

[161] Jens Schreiter, Duy Nguyen-Tuong, Mona Eberts, Bastian Bischoff, Heiner Markert, and Marc Toussaint. Safe exploration for active learning with Gaussian processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 133–149. Springer, 2015.

[162] Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics 9*, 2003.

[163] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[164] Alistair Shilton, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Regret bounds for transfer learning in Bayesian optimisation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[165] David Slepian. The one-sided barrier problem for Gaussian noise. *Bell System Technical Journal*, 41(2):463–501, 1962.

[166] Mlnoru Slotani. Tolerance regions for a multivariate normal population. *Annals of the Institute of Statistical Mathematics*, 16(1):135–153, 1964.

[167] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2006.

[168] Suzanne Sniekers, Aad van der Vaart, et al. Adaptive Bayesian credible sets in regression with a Gaussian process prior. *Electronic Journal of Statistics*, 9(2):2475–2527, 2015.

[169] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[170] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning (ICML)*, 2015.

[171] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

[172] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[173] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[174] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[175] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

[176] István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, 2006.

[177] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[178] Minija Tamosiunaite, Bojan Nemec, Aleš Ude, and Florentin Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59(11), 2011.

[179] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[180] Michalis K Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[181] Michalis K Titsias and Miguel Lázaro-Gredilla. Variational heteroscedastic Gaussian process regression. In *International Conference on Machine Learning (ICML)*, 2011.

[182] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite Markov decision processes with Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[183] Hado van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

[184] Jarno Vanhatalo, Jaakko Riihimäki, Jouni Hartikainen, Pasi Jylänki, Ville Tolvanen, and Aki Vehtari. Gpstuff: Bayesian modeling with Gaussian processes. *Journal of Machine Learning Research*, 14:1175–1179, 2013.

[185] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Active model learning and diverse action sampling for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[186] Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[187] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning (ICML)*, 2017.

[188] Zi Wang, Stefanie Jegelka, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Focused model-learning and planning for non-Gaussian continuous state-action systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[189] Zi Wang, Beomjoon Kim, and Leslie Pack Kaelbling. Regret bounds for meta Bayesian optimization with an unknown Gaussian process prior. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[190] Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. In *International Conference on Machine Learning (ICML)*, 2017.

[191] Zi Wang, Bolei Zhou, and Stefanie Jegelka. Optimization as estimation with Gaussian processes in bandit settings. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[192] Ziyu Wang and Nando de Freitas. Theoretical analysis of Bayesian optimisation with unknown Gaussian process hyper-parameters. In *NIPS workshop on Bayesian Optimization*, 2014.

[193] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal or Artificial Intelligence Research*, 55:361–387, 2016.

[194] Ari Weinstein and Michael L Littman. Bandit-based planning and learning in continuous-action Markov decision processes. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[195] Eric W. Weisstein. Square root inequality. MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/SquareRootInequality.html`, 1999-2018.

[196] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*, volume 28. CRC Press, 2007.

[197] Dominik Wied and Rafael Weißbach. Consistency of the kernel density estimator: A survey. *Statistical Papers*, 53(1):1–21, 2012.

[198] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[199] Victoria Xia, Zi Wang, Kelsey Allen, Tom Silver, and Leslie Pack Kaelbling. Learning sparse relational transition models. In *International Conference on Learning Representations (ICLR)*, 2019.

[200] Akihiko Yamaguchi and Christopher G Atkeson. Differential dynamic programming for graph-structured dynamical systems: Generalization of pouring behavior with different skills. In *Humanoids*, 2016.

[201] Timothy Yee, Viliam Lisy, and Michael Bowling. Monte carlo tree search in continuous action spaces with execution uncertainty. In *International Conference on Artificial Intelligence (IJCAI)*, 2016.

[202] Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

[203] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[204] Chao Yuan and Claus Neubauer. Variational mixture of Gaussian process experts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2009.

[205] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017.