Multi Agent Bayesian Optimization

by

Shyam Sundar Nambiraja

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2024 by the
Graduate Supervisory Committee:

Giulia Pedrielli, Chair
Dimitri Bertsekas
Nakul Gopalan

ARIZONA STATE UNIVERSITY

May 2024

ABSTRACT

Efficiently solving global optimization problems remains a pervasive challenge across diverse domains, characterized by complex, high-dimensional search spaces with non-convexity and noise. Most of the approaches in the Bayesian optimization literature have highlighted the computational complexity involved when scaling to high dimensions. Non myopic approximations over a finite horizon has been adopted in recent years by modeling the problem as a partially observable Markov Decision Process (MDP). Another promising direction is the partitioning of the input domain into sub regions facilitating local modeling of the input space. This localized approach helps prioritize regions of interest, which is particularly crucial in high dimensions. However, very few literature exist which leverage agent based modeling of the problem domain along with the aforementioned methodologies.

This work explores the synergistic integration of Bayesian Optimization and Reinforcement Learning by proposing a Multi Agent Rollout formulation of the global optimization problem. Multi Agent Bayesian Optimization (MABO) partitions the input domain among a finite set of agents enabling distributed modeling of the input space. In addition to selecting candidate samples from their respective sub regions, these agents also influence each other in partitioning the sub regions. Consequently, a portion of the function is optimized by these agents which prioritize candidate samples that don't undermine exploration in favor of a single step greedy exploitation. This work highlights the efficacy of the algorithm on a range of complex synthetic test functions.

ACKNOWLEDGMENTS

Firstly, I would like to express my utmost gratitude to my mentor and the committee chair Dr. Giulia Pedrielli, my committee members Dr. Dimitri Bertsekas and Dr. Nakul Gopalan for providing me the opportunity to work on some of the most interesting ideas throughout my master's graduate program which culminated into this thesis. They played a significant role in building my foundations needed to pursue this thesis and have played a huge role in guiding me towards various directions, eventually supporting me throughout the process and help narrow down to areas which culminated into the work presented in this thesis.

Most of the work presented in this thesis wouuld not have been possible without the support of my fellow colleagues Tanmay Bhaskar Khandait and Surdeep Chotaliya. They have played an integral part in shaping up this thesis in its current form by having necessary discussions and feedbacks from time to time.

The two years of my master's degree would not have been possible without the support of my friends that I have made here. I would like to especially thank my roommates for putting up with me throughout the year. From having late night discussions and board games to exploring the skylines and trails in and around Arizona, they have carved a special place for these two years eternally etched in memory.

Most importantly, I would like to express my sincere gratitude to my parents for the uncountable number of sacrifices that they have made to help me pursue my interests in all walks of life. Whatever I do, whenever I do will always be driven towards making them proud.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

The quest for efficiently solving global optimization problems has been a longstanding challenge across various domains, ranging from engineering design and finance to healthcare and logistics. Black-box optimization represents a class of challenging optimization problems where the objective function is treated as an opaque entity, accessible only through evaluations at specific input points. Unlike traditional optimization scenarios where analytical expressions or gradients are available, in black-box optimization, the function's internal structure or properties are unknown or deliberately concealed. This characteristic renders the optimization process akin to navigating a maze blindfolded, relying solely on the outcomes of function evaluations to guide search efforts.

Imagine you are tasked with optimizing the supply chain network for a global manufacturing company. The supply chain involves various interconnected processes, such as sourcing raw materials, manufacturing products, and distributing them to customers. However, the exact details of the supply chain, including transportation routes, inventory levels, and production capacities, are complex and not fully known.

In this scenario, the entire supply chain network acts as a black box. You can't directly observe or control every aspect of the supply chain; instead, you have limited visibility into certain key performance indicators, such as delivery times, production costs, and inventory levels. Your goal is to optimize these indicators to improve overall efficiency, reduce costs, and enhance customer satisfaction.

Using black-box optimization techniques, you explore different strategies and

configurations for the supply chain network. For example, you might experiment with different transportation routes, adjust inventory management policies, or optimize production schedules. Each change you make represents a decision in the optimization process.

However, since you can't directly observe the inner workings of the supply chain, you rely on feedback from performance metrics to guide your decisions. You monitor how changes in one part of the supply chain affect overall performance and adjust your strategy accordingly. Over time, through iterative experimentation and adjustment, you aim to find the optimal configuration of the supply chain network that maximizes efficiency and profitability.

The inherent challenges of black-box optimization stem from the lack of explicit knowledge about the objective function, presenting obstacles in efficiently exploring the search space and converging to optimal solutions. Strategies to address these challenges range from direct search methods, such as evolutionary algorithms and pattern search, to model-based approaches like Bayesian optimization and surrogate modeling.

Despite its inherent difficulties, black-box optimization remains a crucial tool for solving real-world problems where explicit knowledge of the objective function is unavailable or impractical to obtain. Advances in algorithmic techniques and computational resources continue to drive progress in this field, unlocking new possibilities for optimizing complex systems and processes.

Traditional optimization techniques often struggle to navigate complex, high-dimensional search spaces characterized by non-convexity, discontinuities, and noise. Bayesian optimization (BO) has been one of the most powerful paradigms and in recent years, reinforcement learning (RL) has emerged as a promising approach for

tackling such challenges. Both approaches have demonstrated remarkable success across a variety of domains, from hyperparameter tuning in deep learning to robotic control and autonomous systems.

Imagine using Bayesian optimization to optimize a global supply chain network, aiming to enhance efficiency and reduce costs. Bayesian optimization starts with an initial configuration of the supply chain network, including parameters like transportation routes, inventory levels, and production schedules. The objective function represents the performance of the supply chain network, incorporating metrics such as delivery times, production costs, and inventory levels. However, this function is complex and not directly observable. Bayesian optimization explores the space of possible configurations by selecting new sets of parameters to evaluate. These selections are guided by a probabilistic model of the objective function, which provides insights into promising regions of the parameter space. Each selected configuration is evaluated using simulations or real-world experiments to assess its performance according to the objective function. The performance of each evaluated configuration provides feedback to the Bayesian optimization algorithm. This feedback is used to update a probabilistic model and refine the search for optimal configurations. Through iterative evaluation and feedback, Bayesian optimization gradually identifies configurations that maximize supply chain performance, achieving a balance between exploration of new configurations and exploitation of known promising regions. Eventually, Bayesian optimization converges to an optimal or near-optimal configuration of the supply chain network, maximizing efficiency and minimizing costs while satisfying operational constraints.

Whereas, reinforcement learning (RL) treats the optimization process as a sequential decision-making problem. The supply chain network learns optimal behaviors

through interactions with the environment, receiving feedback based on actions taken. Feedback is received in the form of rewards or penalties based on the performance of the supply chain network. These rewards guide the learning process, reinforcing behaviors that lead to desirable outcomes. RL agent continuously explores different actions to discover optimal behaviors while exploiting learned knowledge to maximize rewards. Requires modeling the dynamics of the supply chain environment, including state representations, action spaces, and reward functions. RL algorithms must learn complex decision-making policies from high-dimensional and dynamic data. RL converges when the RL agent has learned an effective policy that maximizes cumulative rewards over time. Convergence may take a significant number of iterations and depends on factors such as exploration strategy and the complexity of the environment.

Despite their individual successes, both Bayesian optimization and reinforcement learning possess distinct strengths and limitations. Bayesian optimization, rooted in probabilistic modeling and surrogate-based optimization, has gained popularity for its ability to effectively explore and exploit unknown objective functions while providing principled uncertainty estimates. However, its application to global optimization problems is often hindered by scalability issues and the curse of dimensionality, especially in settings where the objective function is computationally expensive to evaluate.

On the other hand, reinforcement learning offers a principled framework for learning optimal decision-making policies by interacting with an environment and maximizing cumulative rewards. While RL methods have shown remarkable success in sequential decision-making tasks and control problems, their application to global optimization is limited by sample inefficiency, sensitivity to hyper parameters, and challenges in handling noisy or sparse rewards.

The blending of Bayesian optimization and reinforcement learning presents a compelling opportunity to address these limitations and develop more effective strategies for global optimization. Combining the principled exploration-exploitation trade-offs of Bayesian optimization with the adaptive learning and decision-making capabilities of reinforcement learning is actively being explored a lot in recent years.

1.1 Research Questions

This thesis seeks to answer the following research questions

- **RQ1.** Can the utilization of non-myopic acquisition functions enhance the performance of Bayesian optimization by providing better point estimates?
- **RQ2.** Does partitioning the input space prove to be effective in high dimensional optimization problems by systematically narrowing down the search space into regions of interest?

1.2 Contributions

This thesis presents an approach inspired by Partition-based methods but introduces a unique twist by integrating a multi-agent methodology. This method utilizes a class of Approximate Dynamic Programming technique known as Multi-agent Rollout. The inherent collaborative nature of this approach allows agents to act independently yet in a coordinated manner. By combining partitioning with these agents, the input space is divided based on the achieved lookahead, and actions taken by the agents.

This work introduces a Multi-agent formulation of Rollout for Bayesian optimiza-

tion using non-myopic acquisition functions. The key contributions of this paper are as follows:

- Given the high-dimensional nature of many real-world functions, a multi-agent algorithm that focuses on specific regions of the unknown function by partitioning the search space and distributing them among the agents is proposed. These agents provide point estimates from their assigned "active" sub-regions, facilitating faster convergence compared to vanilla Bayesian optimization.
- The inherent collaborative nature of Multi-agent Rollout allows us to converge more rapidly towards the optimum. However, considering the multimodal nature and uncertainty of real-world functions, a slight modification is proposed to the traditional Multi-agent Rollout formulation. An agent tasked with tracking the "inactive" sub-regions is introduced, enhancing our approach's robustness to uncertainties and multimodality.

The proposed algorithm can be found here - MABO

1.3 Structure

The organisation of Thesis is as follows. Chapter 2 gives an exhaustive overview of the research literature that tackle problems in this domain along with a comparison of the methods. Chapter 3 gives a brief overview of the background of the thesis and introduces some key concepts that will be used extensively. This is followed by the formulation of the algorithm in Chapter 4 along with the notations that will be extensively used to describe the algorithm. Finally, Chapter 5 shows the results of the algorithm tested on synthetic functions.

Chapter 2

LITERATURE REVIEW

2.1 High Dimensional Bayesian Optimization

High dimensional black box optimization problems are significantly challenging due to multiple reasons. (1) Curse of Dimensionality - As the dimensionality of the search space increases, the number of possible combinations of parameters or inputs grows exponentially. This makes it difficult to explore the entire search space efficiently. (2) Computational Complexity - Modeling GP models which are inherently non parametric becomes challenging as the search space increases. hyper parameters of the likelihood function to be maximized from a (d + 1) dimensional vector, evaluating the objective function for each combination of parameters can be time-consuming or require a large amount of computational resources. (3) Lack of Gradient Information - In black box optimization, the objective function is treated as a black box, meaning that you have no knowledge of its analytical form or gradient information. This makes it challenging to apply gradient-based optimization techniques, which are effective in low-dimensional spaces. (4) Local Optima - High-dimensional spaces often contain many local optima, which are sub optimal solutions that can trap optimization algorithms. Finding the global optimum becomes more difficult in such cases. (5) Noisy or Stochastic Objectives - In many practical scenarios, the objective function is noisy or stochastic, meaning that evaluations may have some level of uncertainty or randomness.

Projection based approaches Projection based approaches perform the search

in lower dimensional manifolds, assuming a low effective dimensional space exists that contains all the behavioral information of the original high dimensional function. Some of the noteworthy approaches in this category are Random Embedding BO, Hashing enhanced subspace Bayesian Optimization by Munteanu, Nayebi, and Poloczek 2019, subspace identification Bayesian optimization (SIBO) algorithm. These approaches exhibit significant sensitivity to the embedded sub-space dimension which often needs to be specified. Literature exist that try to automate the selection of the embedding dimension. A lot of real world problems applications exist where the existence of a low dimensional manifold does not hold.

Statistical learning based approaches The low effective dimensionality assumption of the above approach is restrictive because all the input variable might contribute to the objective function. So, Statistical learning based approaches rely on the structural properties of the black box function in question. Additive structure assumes that the objective function is a sum of functions of small, disjoint groups of dimensions. Additive Gaussian processes differ in: (1) how the components are learned (2) how the additive Gaussian process hyper parameters are estimated, and (3) how the additive Gaussian process is used to take samples. Success depends on correct choice of decompositions that must accurately mimic the inter-dimensional dependencies.

Variable selection approaches Variable selection takes a different approach where it operates under the assumption that not all dimensions are important to reach the optimal. SAASBO proposed by Eriksson and Jankowiak 2021 assumes that the dimensions have a hierarchy of relevance. They introduce a novel sparsity inducing SAAS prior. The level of sparsity is controlled by a scalar. As a result, most of the dimensions are 'turned off' in accordance with the principle of automatic relevance

determination. However, the cost of inference scales cubically with the number of function evaluations. Thus, SAASBO is not expected to scale beyond small sampling budgets. SAASBO perform variable selection implicitly but has high computational cost of inference. Song et al. 2022 applies MCTS to iteratively partition all variables into important and unimportant ones, and perform BO only for those important variables. Again, it relies on the assumption of low effective dimensionality, and might not work well if the percentage of valid variables is high.

Trust region based approaches To optimize high-dimensional functions without relying on assumptions regarding their structure or dimensionality, a localized modeling approach of the objective function can be adopted. This method iteratively refines the objective function by cumulatively optimizing it within localized regions. TuRBO put forth in Eriksson et al. 2020 represents a prominent example within this category of approaches, employing "Trust Regions" that form hyper rectangles centered around the best solution encountered thus far. These regions serve as repositories for local probabilistic models, facilitating the exploration of diverse search trajectories capable of swiftly uncovering optimal objective values. Notably, the local surrogate models enable heterogeneous modeling of the objective function and mitigate the risk of excessive exploration. To complement the local modeling strategy, TuRBO incorporates a global bandit strategy, which intelligently distributes samples across confidence regions, thereby achieving a balance between exploration and exploitation. However, a limitation of TuRBO lies in the independent learning of trust regions without data sharing, potentially resulting in inefficiencies for computationally expensive problems.

In contrast, Mathesen et al. 2021 employs a surrogate model to strategically generate restart locations for local searches. Through dynamic allocation of the number of function evaluations and restarts for local searches based on observed

performance, SOAR optimizes its search process effectively. This adaptive approach ensures efficient utilization of computational resources while achieving optimal solutions

2.1.1 Partitioning Based Approaches

Exact optimization methods, such as Partitioning methods like branch and bound, have been employed extensively in various domains. However, they often assume constant correlation functions within Gaussian processes across the input space. Partitioning the input space into distinct regions, where each region employs a unique correlation function or set of parameters, proves beneficial when the data generating process exhibits differential rates of variation across different parts of the input space. This approach allows for a more precise modeling of the local data behavior. For instance, in scenarios where data displays varying levels of smoothness or variability across different regions of the input space, employing distinct correlation functions for each partition enables more accurate capture of these localized characteristics.

Incorporating reinforcement learning techniques facilitates optimal sequential sampling of the input space, thereby guiding the branching decisions effectively. Contrasting with local modeling methods like TuRBO, LA-MCTS (Limited Area Monte Carlo Tree Search) by Wang, Fonseca, and Tian 2022 dynamically exploits and explores promising regions concerning samples through Monte Carlo Tree Search, continuously refining learned boundaries with new data points. LA-MCTS identifies promising regions by iteratively partitioning the space. On the other hand, PART-X by Pedrielli et al. 2021 leverages local Gaussian process estimations to adaptively branch and sample within the input space. While LA-MCTS establishes nonlinear boundaries, PART-X employs axis-aligned branches. Additionally, LA-MCTS employs

the Upper Confidence Bound (UCB) method for selections, while PART-X utilizes heuristics to identify level sets in a black-box function.

2.1.2 Lookahead Approaches

While most acquisition functions are myopic, considering only the next function evaluation, non-myopic ones consider the impact of multiple future evaluations. One of the prominent approach is put forth by Lee et al. 2020 Rollout Acquisition Functions: Non-myopic acquisition functions are typically computed through rollout, which involves simulating multiple steps of BO. These rollout acquisition functions are defined as high-dimensional integrals, making them computationally expensive to compute and optimize. Efficient Computation: The paper proposes a combination of quasi-Monte Carlo methods, common random numbers, and control variates to significantly reduce the computational burden of computing rollout acquisition functions. These techniques aim to improve the efficiency of evaluating and optimizing these functions. Policy-Search Approach: The authors formulate a policy-search based approach that eliminates the need to directly optimize the rollout acquisition function. Instead, the focus is on optimizing a policy that implicitly determines the rollout decisions. Qualitative Behavior of Rollout Policies: The paper discusses the qualitative behavior of rollout policies, particularly in the context of multi-modal objectives and model error. Understanding how rollout policies behave in various scenarios is crucial for their practical applicability.

Jiang et al. 2020 proposes a joint Optimization of Decision Variables. Instead of solving nested optimization problems within a multi-step scenario tree, the paper proposes jointly optimizing all decision variables in the full tree simultaneously,

termed as a "one-shot" approach. This method enhances computational tractability and demonstrates superior performance compared to existing methods on various benchmarks. Efficient Conditioning of Gaussian Processes (GPs): Traditional methods for conditioning a GP on additional data involve rank-1 updates to the Cholesky decomposition of the input covariance matrix. The paper introduces a related approach called "multi-step fast fantasies," which efficiently constructs fantasy models for GP models representing the full look ahead tree. This approach includes novel linear algebra methods for updating GPyTorch's caches for posterior inference in each step. The paper discusses related methods, including a recent development enabling gradient-based optimization of two-step expected improvement (EI). However, these methods often rely on assumptions about the maximizers of second-stage value functions and may involve non-adaptive or computationally expensive procedures.

2.1.3 Multi Agent Approaches

Single-agent approach may struggle to explore the search space effectively or exploit discovered regions efficiently. By distributing the workload across multiple agents, each responsible for exploring different parts of the search space, a multi-agent approach enables parallelization and enhanced exploration-exploitation trade-offs.

Furthermore, in real-world applications such as source seeking or complex optimization problems, a single-agent approach may be insufficient to capture the diversity of objectives or adequately handle uncertainty. Multi-agent systems offer inherent flexibility and adaptability, allowing for diverse strategies and collaborative decision-making. Additionally, in scenarios where resources or expertise are distributed across different agents, a multi-agent approach facilitates decentralized coordination and

communication, leveraging the strengths of individual agents while mitigating their limitations.

Entropy Search (ES) put forth by Ma et al. 2023 has gained attention for its ability to select query points that maximize the mutual information about the maximum of the black-box function. However, one of the primary challenges of ES lies in its computational complexity, often requiring costly approximation techniques. Efforts have been made to address the scalability issue of ES, especially concerning its exponential complexity with respect to the number of agents involved. Two main approaches have been explored in the literature for selecting batches of query points for agents: sequential query calculation and batch query calculation. While sequential query calculation assigns explicit roles of exploration/exploitation to each agent, batch query calculation implicitly handles collaboration through its probabilistic model. Recent advancements have focused on enhancing the efficiency and scalability of ES. For instance, gradient-based multi-agent ES algorithms have been proposed to reduce the exponential cost to polynomial cost. However, these approaches still involve heavy computations, particularly due to the large number of matrix inversions required.

To mitigate computational burdens, some studies have leveraged normal distributions to approximate the distribution of the function maximum and calculate its entropy, resulting in closed-form expressions for acquisition functions. Additionally, central coordinators have been employed to facilitate communication among agents, receiving observations, updating GP models, and calculating queries based on mutual information maximization.Trade-offs between time and batch dimensions have also been explored, with strategies such as Upper Confidence Bound (UCB) balancing the proximity of query points to maximize exploration while maintaining spatial separation among them.

Existing baseline multi-agent BO algorithms have been compared in recent literature, including GPUCB with pure exploitation, Gaussian Process Batch Upper Confidence Bound, EI with Monte-Carlo sampling, EI with stochastic policies, and Parallel Thompson sampling. Empirical evaluations on various test functions have revealed insights into the performance of these algorithms, with some showing superiority in certain tasks or domains.

Krishnamoorthy and Paulson 2023 framework adds a penalty term to traditional BO acquisition functions to account for coupling between the the agents without data sharing. They derive a suitable form for this penalty term using alternating directions method of multipliers (ADMM), which enables the local decision making problems to be solved in parallel. While this approach relies on a central coordinator that talks to these local models through the penalty term, the approach presented in this thesis uses a class of Approximate Dynamic programming approach called Multi agent Rollout proposed by Bertsekas 2020. The inherent coordinating nature exhibited by the agents in this approach enables these agents to act independently. Coupled with partitioning, these agents branch the input space based on the look ahead achieved by decomposing the actions taken among the agents.

Chapter 3

BACKGROUND

3.1 Black Box Models

Let's consider a black box function $f : \mathbb{R}^n \to \mathbb{R}$, where $\mathbb{R}^n$ represents the input space and $\mathbb{R}$ represents the output space.

The black box model can be represented as:

$$y = f(\mathbf{x}) + \epsilon$$

where:

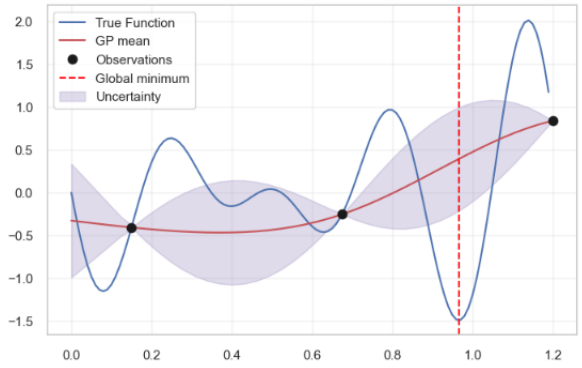
- y is the observed output,
- $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is the input vector,
- $f(\cdot)$ is the unknown function representing the black box,
- ϵ is the noise term representing any uncertainty or error in the observations.

In Bayesian optimization, we seek to find the optimal input $\mathbf{x}^*$ that maximizes (or minimizes) the objective function $f(\cdot)$:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$$

subject to any constraints imposed on the input space.

3.2 Bayesian Optimization



Let's consider the following problem:

$$min_{x \in X} \quad f(x) \tag{3.1}$$

The function $f : \mathbb{D} \rightarrow \mathbb{R}$ is unknown and can be any continuous real valued function like the True function depicted above and a set of parameters that map to this function. f is not convex or linear or any known structure to exploit using existing efficient techniques to optimize. Also, we only observe $f(x)$ and not it's derivatives so we can't use gradient based methods. We only have a sample set of observations as shown to provide an estimate of the true objective function. Our goal is to find a point $x^* \in X$ with the lowest objective value by sequentially estimating points $\{x_k\}_{k=1}^{N}$ over a budget of N function evaluations.

Bayesian optimization is among a popular class of sample efficient sequential estimation methods. Bayesian optimization leverages Bayes' theorem, which updates prior beliefs with observed data to derive a posterior distribution. Unlike conventional supervised learning methods that yield precise parameter values, Bayesian optimization provides a probability distribution over potential parameter values. This distribution

is initially informed by a set of observations, and as new observations are obtained, the distribution is updated iteratively, leading to a more refined estimation.

The process begins with a prior distribution based on initial observations, depicted in the figure by the shaded purple area, representing uncertainty. As additional observations are sampled, the prior distribution is updated using Bayes' conditional probability, gradually reducing uncertainty and converging towards the true underlying function.

$$P(w|y, \mathbf{X}) = P(w)\frac{P(y|\mathbf{X}, w)}{P(y|\mathbf{X})} \tag{3.2}$$

Equation 3.2 facilitates this update, where the numerator represents the likelihood multiplied by the prior, and the denominator is the marginal likelihood. By integrating over possible parameter values weighted by their posterior distributions, the predictive distribution $P(f|x, y, \mathbf{X})$ is obtained.

$$P(f|x, y, \mathbf{X}) = \int_w p(f|x, w)p(w|y, \mathbf{X})dw \tag{3.3}$$

To make the integration tractable, a Gaussian prior distribution is often chosen. Additionally, utilizing a Gaussian prior results in a predictive distribution that is also Gaussian. Consequently, the mean of this predictive distribution serves as the predicted observation, while the variance captures uncertainty, providing a reliable surrogate function for further optimization.

3.2.1 Surrogate Function

Gaussian process serves as a surrogate function that fits a probabilistic model to the candidate observations. It is non parametric and provides a flexible model of continuous functions. Since we have only sample observations, which can take many possible admissible functions, Gaussian processes calculate a probability distribution of all these admissible functions

First, we assume a GP prior with a mean μ and covariance function K as $f \sim GP(\mu, K)$. This will serve as an infinite dimensional multivariate Gaussian distribution that acts as the prior distribution of the space of functions possible using their mean and covariances. The kernel function $k(x, x^{'})$ correlates the pairs of nearby points with hyperparameters to get a smooth function approximation. Several gradient based optimizers are used to tune these hyperparameters. For a given set of sample observations, we can write the data as $D = (x_i, y_i)i = 1..k$ where y_i is the observation with a Gaussian noise.

Calculating the posterior distribution is tractable since we used a GP prior. The predictions are described by the mean and covariances in the diagonal of the covariance matrix since the resulting posterior distribution is also Gaussian. Time complexity for fitting the GP prior is $O(n^3)$ where n is the number of function evaluations. Now that we have a predictive distribution, we need to choose the next observation by efficiently searching the hyper parameter space. This is done using an Acquisition function.

3.2.2 Acquisition Function

Acquisition functions help us choose the next observation by estimating the likelihood that an observation is worth evaluating with the objective function. Since we have used a GP, we can use this information to estimate how worthy the observation is. Some of the well known acquisition functions in the literature are:

- Probability of improvement
- Expected Improvement
- Upper Confidence bound

This thesis extensively uses Expected improvement. It is calculated by computing the probability of improvement over the current best solution at a given point, weighted by the magnitude of that improvement. In other words, it quantifies the expected amount of improvement in the objective function that can be obtained by sampling at a certain point. EI strikes a good balance between exploration (by increasing variance) by looking at regions of high uncertainty and exploitation (by increasing the mean) by choosing the point where the surrogate model achieves high objective. It is defined as:

$$EI(x) = \mathbb{E}[max(f(x) - f(x'), 0) \mid x, D] \tag{3.4}$$

where $f(x)$ is the minimal possible value observed so far and if we find a sample x' that happens to have $f(x')$ less than $f(x)$ then we consider that as an improvement. Eventually, we need to optimize this function to get the sample with the maximum expected improvement. This inner optimization is relatively easier to compute because it computes a scalar value which can be optimized using gradient based methods and is typically $O(d^2)$ where d is the dimension of the search space.

Broyden–Fletcher–Goldfarb–Shannon is the most common numerical optimization technique chosen to optimize EI.

As it can be seen, the EI and other commonly used acquisition functions are greedy and only give the next observation that results in a better realization of the objective function and are oblivious to the remaining function evaluations. Multiple steps of these are needed to find the global optimum and it is prone to be stuck at a local minimum. A better approach would be to simulate multiple BO realisations to choose the next observation by planning out a sequence of observations in a non myopic manner thereby resulting in a faster convergence to global optimum.

3.3 Approximate Dynamic Programming And Non-Myopic Acquisition Functions

Bayesian optimization discussed so far utilizes the Acquisition function in a greedy manner, in the sense that they only look at the immediate reward. Lookahead approaches enable us to estimate the long term reward of the sample points. BO with lookahead can be formulated as an instance of a finite N-stage DP, where N denotes the number of stages or steps considered. State Space and Policy: At each stage k, the state space is defined, and a policy π is a sequence of rules mapping the state space to the action space. The policy π aims to maximize the cumulative reward. Reward Function: The reward function at each stage is denoted by r_k, and the end-stage reward is denoted by r_{N+1}. A discounted expected cumulative reward over the N-step horizon under a given policy is calculated, considering a discount factor . The objective is to find the optimal policy $\pi*$ that maximizes the cumulative reward.

However, DP formulation suffers from computational burden and the curse of

dimensionality, especially due to the uncountable state and action space. To address the computational challenges, an approximate dynamic programming approach called Rollout is proposed in Bertsekas 2022c. Rollout involves approximating the reward-to-go functions using heuristic base policies, which are more computationally efficient. The rolling horizon approach limits the number of stages through which the approximate reward is calculated, reducing the dimensionality. These are calculated using heuristic base policies, and the process involves iterating through different stages and updating policies accordingly.

3.3.1 Rollout

In the context of BO, acquisition functions take the role of the heuristic base policy that can be utilized to compute the reward-to-go functions. Non myopic acquisition function model the exploration exploitation trade off by considering immediate and future rewards. To encapsulate the exploration-exploitation dilemma in BO we can formulate it as a finite horizon Markov Decision Process (MDP). The MDP framework takes the form $< T, S, A, P, R >$, where $T = 0, 1...h-1, h < \infty$. S is the state space, A is the action space, $P(s' \mid s, a)$ is the transition probability of moving from state s to s' taking an action a and $R(s, a, s')$ is the reward of moving from state s to s' by taking the action a.

In the context of BO, with a GP prior over data D, we model h steps of BO as an MDP. The state space is all possible sets of data reachable from a current dataset D_t. Action is choosing an input u from the domain. Transition probability from dataset D_t to D_{t+1} by choosing an action u_{t+1} is the probability of choosing $z_{u_{k+1}}$ from the

posterior distribution at u_{k+1}. In other words, for a given prior distribution b_0, an observation u_{k+1} and posterior b_k, the state transitions according to:

$$b_{k+1} = B_k(b_k, u_{k+1}, z_{u_{k+1}}) \tag{3.5}$$

Reward is defined using EI where z_t^* is the minimum observed value given the dataset D_t.

$$R(D_t, u_{k+1}, D_{t+1}) = max((z_t^* - z_{t+1}), 0) \tag{3.6}$$

For horizon (h = 1) this is simply maximising the immediate reward. Whereas a non myopic policy is an optimal policy for h horizon MDP. The expected total reward of this MDP is therefore given by:

$$Q_h^\pi(D_k) = \mathbb{E}[\sum_{t=k}^{k+h-1} (z_t^* - z_{t+1})] \tag{3.7}$$

A DP formulation is infeasible because we require the knowledge of the Q factors recursively done through DP to obtain the optimal solution. Moreover the space of posterior distribution is infinite-dimensional even for the Gaussian case by using mean and covariance to reduce it to a finite set.

Since, for h $>$ 2, this is expensive to compute, we can use Rollout to get sub optimal policies. We denote the base policy as $\pi = (\pi_0, \pi_1, ...\pi_{h-1})$. At each decision we are maximising the base acquisition function given the current state provided by the data D_t. Using this base policy, a non myopic acquisition function is given by rolling out this base policy π for h horizon in a forward fashion from any state. The expected reward after h horizon is given by:

$$\Lambda_h(u_{k+1}) = \mathbb{E}[Q_h^\pi(D_k \cup (u_{k+1}, z_{k+1})] \tag{3.8}$$

where z_{k+1} is a noisy observation from the posterior. Rollout being sequentially consistent and sequentially improving, the expected reward of π should be better than the base acquisition function, provided the GP prior is correctly specified.

To get a reasonable estimate of the reward, Monte Carlo simulation is clearly needed, where majority of the computation burden lies and is directly dependent on the total number of samples. Approaches used in Lee et al. 2020 have come up with methods to reduce this computation by integrating over the h horizons. The reason being since the posterior is a normal distribution the expectation of rolling out EI is approximately equal to mapping the h dimensional vector to the posterior and averaging. Empirical evidences show that standard MC converges at the rate of $\sigma/\sqrt{N}$. Any increase in precision warrants two orders of magnitude more samples.

3.3.2 Multi Agent Rollout

A natural extension in parallel and distributed settings would create scenarios where we can sample multiple observations simultaneously before receiving feedback on their outcomes. But handling multiple simultaneous decisions can pose significant computational challenges due to the exponential growth in the observation space. Moreover, we need to take into account the communication overhead that such an approach entails. To address this issue, a multi-agent methodology, that extends the traditional rollout algorithm to scenarios involving multiple agents making decisions in collaboration has been proposed. The multi-agent rollout methodology proposed by Bertsekas 2020 aims to approximate the value or reward of a decision by simulating future trajectories of the system. Unlike the standard rollout algorithm, where a single agent makes decisions at each stage, multi-agent rollout involves multiple agents acting sequentially, with each agent making its own decision based on partial knowledge of preceding agents' choices. Although, the number of controls and the computational requirements grow rapidly with the number of agents, we reduce the

computational cost per stage by trading off control space complexity with state space complexity. The reformulated problem involves breaking down the collective decision into individual component decisions. This reduces the complexity of the control space while introducing additional layers of states in the state space.

To integrate the provided multi-agent rollout methodology into the context of Bayesian optimization (BO), we can adapt the sequential decision-making process to represent how agents interact within the BO framework:

Let M denote the total number of agents involved in the decision-making process. Define an ordered sequence of agents as $\{A_1, A_2, ..., A_M\}$, where each A_i represents an agent. At each stage t of the Bayesian optimization process:

1. Sequential Decision Making: - Starting with A_1, each agent A_i selects a candidate point $x_i(t)$ based on its partial knowledge of preceding agents' choices and the current state of the surrogate function.

2. Reward estimation: - The selected candidate points $\{x_1(t), x_2(t), ..., x_M(t)\}$ are evaluated using the posterior mean to obtain an estimate of their corresponding function values $\{f_1(t), f_2(t), ..., f_M(t)\}$. The regret is calculated using the function value estimate to yield the rewards $\{r_1(t), r_2(t), ..., r_M(t)\}$

3. Update: - The Bayesian optimization framework updates its belief about the objective function based on the new observations $\{(x_1(t), f_1(t)), (x_2(t), f_2(t)), ..., (x_M(t), f_M(t))\}$ that have the highest cumulative reward. This way the candidate points are chosen in such a way that decomposes the reward among the agents and thereby the candidate points chosen gets closer to the optimum.

Sequential Decision Making:

1) $x_i(t) = \text{Agent Decision}(A_i, t, \{x_j(t)\}_{j=1}^{i-1}, \{f_j(t)\}_{j=1}^{i-1}) \quad \text{for } i = 1, 2, ..., M$

2) Evaluate $f(x_i(t)) \quad \text{for } i = 1, 2, ..., M$

3) Calculate the rewards $r_i(t) \quad \text{for } i = 1, 2, ..., M$

4) Update Gaussian process model$\{(x_i(t), f_i(t))\}_{i=1}^{M}$

We will adapt this methodology to Bayesian optimization in Chapter 4.

Chapter 4

MULTI AGENT BAYESIAN OPTIMIZATION USING SEARCH SPACE PARTITIONING

This chapter discusses the MABO algorithm, a multi agent partitioning based algorithm the relies on local Gaussian processes for generating surrogates over the subset of the input space and a non myopic choice of the samples to update the local GPs. The major contribution of this algorithm is:

1. Using local surrogates to model the input space and yield point estimates within those sub regions. The local modelling enables us to distribute the choice of candidate samples taken to update the local GPs.

2. A branching criteria based on the existence of agents in the sub regions at any point in time enables us to partition the input space and narrow down to regions of interest.

3. A look ahead approach that renders a non myopic choice of candidate samples by simulating the local modeling resulting from the agents' partitioning for finite horizons

4.1 Multi Agent Rollout For Bayesian Optimization

In Multi Agent Rollout, an action or control consists of or can be decomposed into multiple components, with a separable control constraint structure, and each component at each stage is chosen by a separate decision entity referred to as "agent" Bertsekas 2021. In MARO, the action from the original decision problem $\mathbf{x}$ is decomposed across

a number of agents. As a result, the evolution of the algorithm together with the cost function are now defined over the cross-product of the agent-level actions. When the problem structure lands itself to the definition of agents, MARO can lead to more efficient solutions algorithms compared to traditional Rollout that, as mentioned in the literature review, as already tested in Bayesian optimization contexts Bertsekas 2021, 2022a, 2022b. There are two categories of MARL agent learning scheme: independent learning scheme and action-dependent learning scheme Li et al. 2023. Within a fully *independent learning scheme* framework, agents make action decisions independently with their own object function or policies Li et al. 2023. A *dependent learning scheme*, on the other hand, allows agents to have a protocol to communicate and make decisions based on other agents' decision. This is also the original definition of Multi-Agent Rollout proposed in Bertsekas 2021 and further discussed in Bertsekas 2022a, 2022b. MABO follows the dependent learning scheme for coordinating feasibility and scope of agents.

In the MARO context, the optimization problem we are trying to solve can be formulated as:

$$\min_{\mathbf{x}\in X}\left[\min_{i}\left(f(x_i)\right)\right] \tag{4.1}$$

Where X is the input space, and the decision variable is encoded as $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$, where the component x_i is an agent feasible solution, and $x_i \in X_i : \cup_i X_i = X, \cap_i X_i = \emptyset$. MABO returns as solution the location associated with the agent that achieves the lowest cost. More specifically, we define a decision vector at algorithm iterate k that can be decomposed according to a separable input structure, i.e., $\mathbf{x}_k = (\mathbf{x}_k^1, \ldots, \mathbf{x}_k^m)$,

where $\mathbf{x}_k^i \in X_k^l, l = 1, \ldots, m$ Bertsekas 2021. Thus the domain of the original problem can be expressed through the product Bertsekas 2021:

$$X_k = X_k^1 \times \cdots \times X_k^m.$$

MABO does not only sample points as it progresses through iterates, rather it sequentially defines and evaluates *agent configurations* that induce a solution. Specifically, given a partition with m subregions, i.e., $\cap_i X_k^i = \emptyset, \cup_i X_k^i = X$, each agent will be assigned a subregion X_k^i, a data set within the subregion D_k^i, and a surrogate model Y_k^i trained with the subregion data set D_k^i. These three elements define the agent configuration object at the k-th iteration, c_k^i, and a configuration is the collection of all agents configurations, namely c_k. At each iteration k, the i-th agent can modify the configuration based on the information from agents $j = 1, \ldots, i-1$ and their prediction on the remainder of the agents configurations $j = i+1, \ldots, m$. Given a configuration, the MABO incumbent solution is:

$$\hat{x}_k^* = \arg\min_i \left(\min_{x^i \in D_k^i} f\left(x^i\right) \right). \tag{4.2}$$

We distinguish two types of agent. Agents $i = 1, \ldots, m-1$ are *active agents* and they can at each iteration actively change the subregion of interest and sample new locations, agent m is an *inactive agent*. The inactive agent is responsible to collect all the subregions that the $m-1$ active agents disregard as a result of their update. The inactive agent can hence have associated multiple subregions and while it maintains the locations and the surrogate model no new sampling is performed by this agent. Subregions contained by the inactive agents can receive new locations if an active agent decides to jump into an inactive region. At iteration k, we define the state of MABO S_k is the set of agents and the corresponding tuple $S_k = (X_k^i, D_k^i, Y_k^i)_{i=1}^m$, considering this state, the algorithm generates number N^C of alternative configurations. To do so,

we sequentially solve the following equations starting at agent $i = 1$:

$$\tilde{c}_k^i \in \arg\min_{C^i} F^a(\tilde{c}_k^1, \ldots, c_k^i, \ldots, \hat{c}_k^m) \; i = 1, \ldots, m. \tag{4.3}$$

Where C^i represents the set of feasible configurations for the i-th agent, and F^a is a cost function that the agent uses to sample the configuration component. In fact, each agent can change its assigned subregion X_k^i, and, given the subregion, it will sample novel locations following an acquisition function implied by a surrogate model built with the locations already assigned to X_k^i that have been sampled from previous iterations. The additional points are added to those currently present within the generated X_k^i. The sample points are then added to the data set D_k^i, and, upon evaluation of the locations, each agent can update a surrogate model Y_k^i. Such generation is performed for a user defined number of times, N^c. Once the configurations are generated, the rollout method is called to evaluate the competing configurations and move MABO to the *best* configuration.

At iteration k, the rollout algorithm, is used to evaluate the candidate configurations, namely $c_k^g, g = 1, \ldots, N^c$. To do so it applies a base-heuristic to simulate the evolution of each configuration for an horizon of length h, namely, $(\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_{k-1}, \tilde{\mathbf{c}}_k, \hat{\mathbf{c}}_{k+1}, \ldots, \hat{\mathbf{c}}_{k+h})$.

It then chooses the configuration that minimizes the approximate Q-factor (here F^{RO}) over all the simulated solutions solutions:

$$\tilde{\mathbf{c}}_k \in \arg\min_{\mathbf{c}_k^g} F^{RO}\left(\tilde{\mathbf{c}}_1^g, \ldots, \tilde{\mathbf{c}}_{k-1}^g, \mathbf{c}_k^g, \hat{\mathbf{c}}_{k+1}^g, \ldots, \hat{\mathbf{c}}_{k+h}^g\right). \tag{4.4}$$

It then repeats with $(\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_{k-1})$ replaced by $(\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_k)$. After K iterations the rollout algorithm produces the complete sequence of configurations

$$\tilde{\mathbf{c}} = (\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_N)$$

which is called the *rollout solution*. The fundamental result underlying the rollout algorithm is that under certain assumptions, we have *cost improvement*, i.e.,

$$F(\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_N) \leq F(\hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_N), \tag{4.5}$$

where $(\hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_N)$ is the configuration produced by the base heuristic.

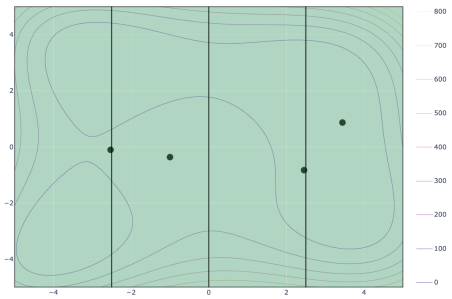


Figure 1. MABO iteration $k = 1$, number of active agents $m = 4$, the inactive agent is associated with the empty set.

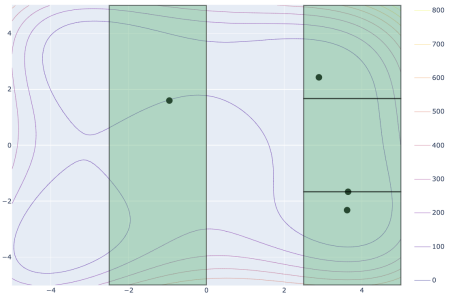


Figure 2. BO iteration $k = 2$, number of active agents $m = 4$, the inactive agent includes the subregions "abandoned" by the active agents.

MABO full step example (Figures 1-2). Figures 1-2 show an example of configuration update. Specifically, Figure 1 shows that at iteration $k = 1$, there are 4 subregions (active agents), each with an associated dataset $D_1^i, i = 1, \ldots, 4$ (dots). Each agent builds its own surrogate $Y_1^i, i = 1, \ldots, 4$ using the respective datasets. Figure 2 shows the configuration obtained after a full MABO iteration, for $k = 2$, where we see two subregions in gray that are now associated with the inactive agent. We observe that the iterate has produced a new set of subregions and, as a result of the the sampling decision made by the agents, updated data sets $D_2^i, i = 1, \ldots, 4$ and surrogates $Y_2^i, i = 1, \ldots, 4$. The detailed process for the generation of a configuration will be detailed in Section 4.1.1 (Step 1).

4.1.1 Multi Agent Bayesian Optimization Algorithm

In this section, we provide the algorithmic details associated with our MABO. We repeat this procedure for K iterations, i.e., until the available function evaluation budget is exhausted. Upon termination MABO returns the *rollout solution* as explained in Section 4.1.

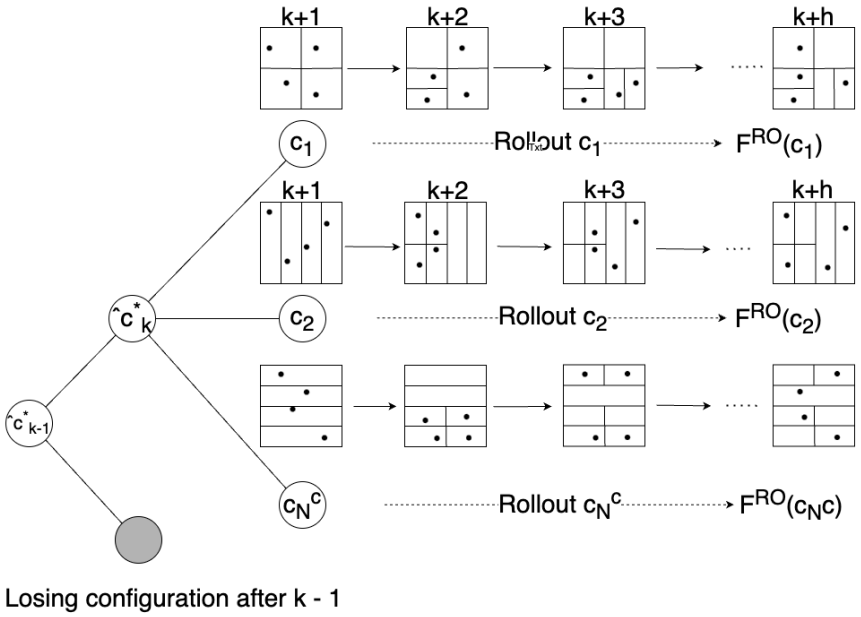


Figure 3. MABO progression.

Figure 3 provides an overview of the key components of the algorithm that we further detail below. MABO evolves using two key functions (Figure 3): *(i)* the configuration generator responsible for producing the tuple $c_k = (X_k^i, D_k^i, Y_k^i)_{i=1}^m$; and *(ii)* the configuration rollout responsible for the simulation (h-step) and evaluation of the generated configurations at each iteration. While several surrogates and acquisition functions can be used by these two functions, MABO adopts a Gaussian process to produce the needed predictions for the function value in each of the subregions

Below, we provide the MABO algorithmic steps that make up the algorithm.

Step 0. Initialization: Provide the feasible region $\mathbf{X}$, the number of active agents $m-1$, the total budget K; Define the number of configurations per iteration N^c; The rollout simulation horizon h; the inactive agent is assigned the empty set, i.e., $X_0^m \leftarrow \emptyset$. Initialize the active agents considering:

(i) An agent is defined by the tuple (X_k^i, D_k^i, Y_k^i);

(ii) Each active agent a_j is assigned to only one sub region X_k^i and each sub region X_k^i contains only one agent a_j.

Step 1. Configuration Generation: A configuration is defined as a tuple $(X_k^i, D_k^i, Y_k^i)_{i=1}^m$. Hence, a configuration update starts from the first agent and updates the tuple by:

(1) letting the agent choose whether to stay in its currently assigned region or jump into any of the successive agents region;

(2) if the agent jumps it will branch the target region according to a partitioning scheme;

(3) the agent samples a location in the target region thus updating the data set and the surrogate.

We encode such update step for the i-th agent as:

$$X_g^i \leftarrow \ \mathbb{P}^a\left(\{X_k^i, D_k^i, Y_k^i\}_{i=1}^m\right) \tag{4.6}$$

The i-th agent generates the related configuration component fixing the tuples associated to agents $1, \ldots, i-1$, and may change X_k^i: *(i) Jump to a subsequent agent*: agent i has a positive probability to jump into any agent region $l = i+1, \ldots, m-1$ or jumping into the *inactive* region (m); *(ii)* once a jump has been determined, the

agent can probabilistically branch the target subregion.

To manage the partitioning of the space, we adopt a tree encoding (see Figure 6).

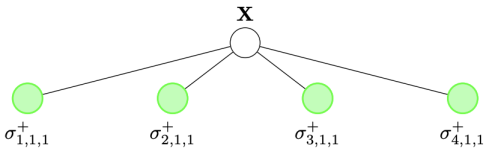


Figure 4. BO iteration k = 1, number of active agents $m-1=4$.

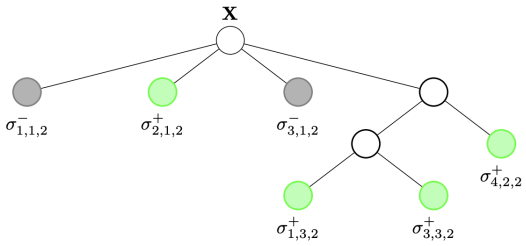


Figure 5. BO iteration k = 2, number of active agents $m-1=4$, the inactive agent has two subregions.

Figure 6. An example of input space partitioning at iteration $k=1$ and $k=2$ into sub regions σ^{γ}_{eft}. The indices e, f refer to the leaf e located at the f-th level of the tree in the t^{th} iteration. Each leaf is a sub region that is either depicted in green or $\gamma=+$ to denote *active* agents, *inactive* regions depicted in grey or $\gamma=-$ are all assigned to the inactive agent, i.e., X^m_k can be a disconnected set.

More specifically, the probability to jump to a candidate region σ^+_{eft} from the agent-associated subregion σ^+_{eft} is proportional to the maximum Expected Improvement Jones, Schonlau, and Welch 1998 calculated in each region as:

$$EI^*_{\cdot} = \max_{\mathbf{x}\in\sigma^{\gamma}_{\cdot}} \mathrm{EI}(\mathbf{x})$$

$$EI^*_{\cdot} = \max\left(\left[f^* - \hat{Y}^{\gamma}_{\cdot}(\mathbf{x})\right]\Phi\left(\frac{f^* - \hat{Y}^{\gamma}_{\cdot}(\mathbf{x})}{\hat{s}^{\gamma}_{\cdot}(\mathbf{x})}\right) + \hat{s}^{\gamma}_{\cdot}(\mathbf{x})\,\phi\left(\frac{f^* - \hat{Y}^{\gamma}_{\cdot}(\mathbf{x})}{\hat{s}^{\gamma}_{\cdot}(\mathbf{x})}\right), 0\right). \quad (4.7)$$

$$p_{\cdot} \propto \frac{EI^*_{\cdot}}{\sum_i EI^*_i}$$

where f^*_i is the best function value sampled so far across the entire input space. We get a higher EI when the mean of the Gaussian process is high at $\mathbf{x}$ or when there is a lot of uncertainty ($\hat{s}^{\gamma}_{\cdot}(\mathbf{x}) > 1$), and $p_{\cdot} \propto \frac{EI^*_{\cdot}}{\sum_i EI^*_i}$ is the jump probability to subregion $(\cdot)$. If an agent jumps to a target region, then it will branch it by choosing uniform at

random a dimension d and a cutting point. Specifically if agent i jumps to the region X_k^j given the bounds $[l_j, u_j]^d$, that contains agent j, then

$$D_k^i = \{\mathbf{x} \in D_k^i : x_d \leq \frac{l_j + u_j}{2}\}$$
$$D_k^{j'} = \{\mathbf{x} \in D_k^j : x_d > \frac{l_j + u_j}{2}\}$$

such that $D_k^j = D_k^i \cup D_k^{j'}$ and $D_k^i \cup D_k^j = \emptyset$ where $i, j = 1, 2, \ldots, m-1$. If agent i jumps to an *inactive* region (m), then $D_k^i \leftarrow D_k^m$. Upon branching the agent makes a sampling decision based on the region-Expected improvement, i.e., the agents adds a location to the selected region according to Jones, Schonlau, and Welch 1998:

$$\mathbf{x}_g^i \in \arg\max_{\mathbf{x} \in \sigma_{eft}^\gamma} \mathrm{EI}(\mathbf{x})$$
$$\mathbf{x}_g^i = \max\left(\left[f_i^* - \hat{Y}_{eft}^\gamma(\mathbf{x})\right]\Phi\left(\frac{f_i^* - \hat{Y}_{eft}^\gamma(\mathbf{x})}{\hat{s}_{eft}^\gamma(\mathbf{x})}\right) + \hat{s}_{eft}^\gamma(\mathbf{x})\,\phi\left(\frac{f_i^* - \hat{Y}_{eft}^\gamma(\mathbf{x})}{\hat{s}_{eft}^\gamma(\mathbf{x})}\right), 0\right). \tag{4.8}$$

where f_i^* is the best function value sampled so far in subregion σ_{eft}^γ. In case of a rollout step, the predicted value $\hat{Y}_{eft}^\gamma(\mathbf{x})$ is used in place of the actual function value. Once the location has been sampled, the true function is evaluated and the following updates are performed by the agent to complete the configuration generation:

$$D_g^i \leftarrow D_g^i \cup \mathbf{x}_g^i : D_g^i = \left\{x \in \cup_l D_k^l : x \in X_g^i\right\}, \tag{4.9}$$
$$Y_g^i \leftarrow GP|X_g^i, D_g^i. \tag{4.10}$$

Once all agents have their assigned sub region and the local Gaussian process model, and N^c configurations have been generated, the rollout is needed to evaluate each configuration.

Step 2. Configurations Rollout: The agent configurations are rolled out for an horizon of length h. A rollout step is nothing but a simulation of the steps encountered

so far. So, for each configuration resulting from Step 1, simulates Step 1-Step 2 of the algorithm using a base heuristic in place of the exact function evaluations. A graphic representation of this process is depicted in Figure 3 along with the progression of each of the N^c configurations.

At each step of the rollout, the configurations are sequentially updated i.e., region partitions are updated and sample points selected from these new regions, based on the base heuristic which in our case is the Expected Improvement (Jones, Schonlau, and Welch 1998) (eqn. (4.8)). The evolution of the agents configurations for the rollout horizon follows:

$$X^i_{k+r} \leftarrow \mathbb{P}^{RO}\left(\left\{X^i_{k+r-1}, D^i_{k+r-1}, Y^i_{k+r-1}\right\}_{i=1}^{m}\right), r = 1, \ldots, h \tag{4.11}$$

$$D^i_{k+r} \leftarrow \left\{x : x \in X^i_{k+r}, x \in \cup_l D^l_{k+r-1}\right\} \cup x^i_{k+r-1} : x^i_{k+r-1} \in \arg\max_{x \in X^i_{k+r} \setminus D^i_{k+r}} EI\left(x\right), \quad r = 1, \ldots, h \tag{4.12}$$

$$Y^i_{k+r} \leftarrow GP | X^i_{k+r}, D^i_{k+r}, r = 1, \ldots, h \tag{4.13}$$

In eqn. (4.11), $\mathbb{P}^{RO}$ represents the rollout partitioning scheme. In the implementation, the i-th agent receives as input the subregions from the other agents and follows one of two mechanisms we discussed in Step 1 i.e., the probability to jump to any agent region $l = i+1, \ldots, m$ and the probability to jump to *inactive* region $(m+1)$ given by eqn. (4.7).The jump probability can be calculated using one of two ways: *(i)* using the Gaussian process model Y^i_{k+r} in the respective sub regions or *(ii)* using Y^i_{k+r} from region $(X^i_{k+r} \cup X^j_{k+r})$, which is the common region between agent associated regions X^i_{k+r} and X^j_{k+r} where $i, j = 1, 2, \ldots, m-1$. The two mechanisms are introduced as a means to manipulate the exploration and exploitation trade off using the agent movements. A jump probability following *(i)* might make the agents movements

greedy whereas following *(ii)* might make the agents more explorative by using a model that is trained on a larger sub region. For instance, if we want to calculate the jump probability between $\sigma^+_{1,3,2}$ and $\sigma^+_{4,2,2}$, then following *(ii)* would utilise the Gaussian process model built using D^i in region $\sigma^+_{4,1,2}$.

Example of tree update simulation step. An instance of the partitioning scheme is depicted in Figure 6. Agent $i = 1$ assigned to subregion $\sigma_{1,1,2}$, following $\mathbb{P}^{RO}$, it jumps to subregion $\sigma^+_{4,1,2}$ thus splitting the subregion into $\sigma_{1,2,2}$ and $\sigma^+_{4,2,2}$ with agent and $i = 4$. Next, agent $i = 2$, stays in its assigned subregion $\sigma_{2,1,2}$. And agent 3 jumps to subregion $\sigma_{1,2,2}$ to split it into $\sigma^+_{1,3,2}$ and $\sigma^+_{3,3,2}$. Finally, agent 4 stays in its subregion $\sigma^+_{4,2,2}$. Consequently, the inactive agent a' *collects* the subregions $\sigma^-_{1,1,2}$ and $\sigma^-_{3,1,2}$. During each jump, the agent level updates to the dataset and GP follow eqn. (4.12)-(4.13).

Given the uncertainty of the approach N^{RO} independent replications of the rollout are performed for each configuration to increase it's stability and the approximate Q-factor (F^{RO} in eqn. (4.4)) is updated as follows:

$$f^{\mathrm{RO}}_{g,\ell} = \min_{i=1,\ldots,m} \left(q^g_{1,\ell}\left(\widehat{S}^1_{k+h,\ell}\right), q^g_{2,\ell}\left(\widehat{S}^2_{k+h,\ell}\right), \ldots, q^g_{m,\ell}\left(\widehat{S}^m_{k+h,\ell}\right)\right)$$

$$g = 1, \ldots, N^c, \ell = 1, \ldots, N^{\mathrm{RO}}$$

$$F^{\mathrm{RO}}_g = \frac{1}{N^{\mathrm{RO}}} \sum_{\ell} f^{\mathrm{RO}}_{g,\ell}. \tag{4.14}$$

where $\left(\widehat{S}^i_{k+h}\right)$ represents the state of the i-th agent at the end of the rollout horizon and the cost component q^g_i is the minimum predicted function value by the i-th agent. Based on the approximate Q-factor, we can choose the configuration c_{k+1} as follows:

$$c_{k+1} \leftarrow \arg\min_g F^{RO}_g, \tag{4.15}$$

$$\hat{x}^*_{k+1} \leftarrow \arg\min_{x \in \cup_i D^i_K} \hat{Y}(x). \tag{4.16}$$

This renders the other configurations in iteration k as *losing* configuration depicted in gray in Figure 3. From the configuration c_{k+1} , the sampling decision $\hat{x}^*_{k+1}$ is obtained from all the agents. The locations sampled are a result of predicted values $\hat{Y}(x)$ as opposed to actual function evaluations.

Update the iteration index, $k \leftarrow k + 1$.

Step 3. Stopping Condition if $k == N$, STOP, return the best so far $\hat{x}^*_K \arg\min_{x \in \cup_i D^i_{k+1}} f(x)$. Else Go to Step 1.

The complete algorithm is shown in Algorithm 1. The Algorithm 2 is a recursive implementation of the Steps 1 and 2 for horizon of length h. It should be noted that in the implementation, the reassignment might include a temporary classification scheme with an additional *crowded* region classification as shown in Algorithm 3. This classification is just a minor implementation detail that needs less attention.

Algorithm 1 Multi Agent BO

Input: Input space $\mathcal{X} \subset \mathbb{R}^d$, Number of agents m, initialization budget n_0, Total budget K, Number of Configurations N^c, Number of rollout replications NRO, number of cuts per dimension per sub region $B \to f(M)$, Branching operator $\mathcal{C} : X \to (X_i)_i : \cup_i X_i = X, \cap_i X_i = \emptyset$, sub region classified as active $(+)$, inactive $(-)$ $\gamma \in \{+, -\}$, Rollout horizon h, Number of samples per subregion s.
Set iteration index $k \leftarrow 1$
Initialize the sets $\Theta^{k+}, \Theta^{k-} = \emptyset, \Theta^k \leftarrow \mathcal{X}$
Output: Sets Θ^{k+} with Agents $(a)_{i=1}^{m-1}$ assigned to sub regions in set Θ^{k+} and set of inactive sub regions Θ^{k-} assigned to agent a^m

Get the initial set of observations $D_0 = \{x_{n_0}, y_{n_0}\}$ from $\mathcal{X}$
A configurations is defined as a tuple $\{X_k^i, D_k^i, Y_k^i\}_{i=1}^m$
while $k \leq N$ **do**
    Generate/Update configurations
    $X_g^i \leftarrow \mathbb{P}^a\left(\{X_k^i, D_k^i, Y_k^i\}_{i=1}^m\right)$
    $(c_g)_{g=1}^{N^c} \leftarrow$ Generate configurations $C(d, m!, s)$ (Step 1)
    for $c_k \in c_g$ **do**
        Rollout configuration $c_k \leftarrow \{X_k^i, D_k^i, Y_k^i\}_{i=1}^m$
        `Replicate rollout` N^{RO} `times`
        for $l = 1$ **to** N^{RO} **do**
            $f_{g,l}^{RO} \leftarrow$ **ConfigurationRollout**(c_k, h) (Step 2)
        end for
        Average h-step cost
        $F_g^{\mathrm{RO}} = \frac{1}{N^{\mathrm{RO}}} \sum_\ell f_{g,\ell}^{\mathrm{RO}}$
    end for
    Choose the configuration with min cost
    $c^* \leftarrow \arg\min_g F_g^{RO}$
    $x^* \leftarrow \arg\min_{x \in \cup_i D_K^i} Y(x)$
    $k \leftarrow k + 1$
end while

Algorithm 2 Configuration Rollout

Input: Configuration c^k, Rollout horizon h
Output: Sets $\hat{\Theta}^{k+}$ with Agents $(a)_{i=1}^{m-1}$ assigned to it and set of inactive sub regions $\hat{\Theta}^{k-}$ assigned to agent a^m

while $h \geq 0$ **do**
    Generate/Update configurations
    $X_g^i \leftarrow \mathbb{P}^{\mathbb{RO}}\left(\{X_k^i, D_k^i, Y_k^i\}_{i=1}^m\right)$
    Rollout configuration $c_k \leftarrow \{X_k^i, D_k^i, Y_k^i\}_{i=1}^m$
    $f_{g,l}^{RO} \leftarrow$ **ConfigurationRollout**$(c_k, h-1)$
end while

Algorithm 3 Reassign

Input: Set of active sub regions $\sigma_{i,j,k}^{\beta}$, $p. \propto \frac{EI_i^*}{\sum_i EI_i^*}$ of all sub regions to decide on reassignment
Output: New status of the sub region $\gamma = (+, -, ++)$

Get the region with the minimum h-step cost based on jump probability p
$\sigma_{e,f,t}^{*,\gamma} \leftarrow \arg\max_{\sigma \in \Theta^k} \quad p(\sigma_{e,f,t}^{\gamma})$
Check if the region with min cost is same as the current active region
if $\sigma_{e,f,t}^{\beta} \neq \sigma_{e,f,t}^{*,\gamma}$ **then**
    $\beta = -$
    if $\gamma = +$ **then**
        $\gamma = ++$
    else if $\gamma = -$ **then**
        $\gamma = +$
    end if
end if
Return γ

Chapter 5

EXPERIMENTS ON SYNTHETIC TEST FUNCTIONS

5.1 Results

Objective of the study: In this chapter, we put forth the results of a set of experiments on synthetic test function to understand the performance of the algorithm in low dimensions and establish a baseline. The objective is to understand the impact of the rollout horizon, the number of agents and the number of configurations, which are all user-defined parameters, on the performance of the algorithm. The impact of the multi agent nature of the algorithm coupled with the rollout approach is evaluated and studied to understand its ability to scale to higher dimensions. The performance of the algorithm is compared with Bayesian Optimization against these non-convex highly multi-modal test functions whose global minimum is known apriori.

Experimental Settings: We consider the difference between the best observed function value and the true optimum value $|\bar{f}^K - f^*|$ averaged across 25 macro-replications. The mean along with the standard error of the metric are recorded.

We compare the results of BO and MABO against the following synthetic test functions:

- **Shubert** The Shubert function is defined as a product of sums, where each term in the product represents a sum of cosine functions. This structure contributes to the function's multi-modal nature, as the interaction between the individual

cosine terms creates a complex landscape with multiple local optima.

$$f(\mathbf{x}) = \prod_{i=1}^{n} \left(\sum_{j=1}^{5} j \cdot \cos\left((j+1) \cdot x_i + j\right) \right) \tag{5.1}$$

which has feasible region $[-10, 10]^d$, has many local minima and one global minima at $(-7.0835, 4.8580)$;

- **Rastrigin** has a feasible region $[-2.5, 3]^d$, with several local minima and one global minimum at $\mathbf{0}$;

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left(x_i^2 - A\cos(2\pi x_i) \right) \tag{5.2}$$

- **Schwefel** 4-d and 10-d function has a feasible region $[-500, 500]^d$, with several local minima and one global minimum at $\mathbf{0}$;

$$f(\mathbf{x}) = -\sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \tag{5.3}$$

- **Langermann** 6-d and 10-d function has a feasible region $[0, 10]^d$ with many local minima and one global minimum with a function value (-5.16)

$$f(\mathbf{x}) = -\sum_{i=1}^{m} c_i \exp\left(-\sum_{j=1}^{n} \left((x_j - a_{ij})^2 \right) \right) \tag{5.4}$$

Key characteristics of these functions :

Multimodality: These functions exhibit multiple local optima, challenging optimization algorithms to navigate the search space effectively. Oscillatory Behavior: The function features rapid oscillations, posing difficulties for gradient-based optimization methods. Scalability: As the dimensionality increases, the complexity of the function grows exponentially, making it a stringent test for high-dimensional optimization techniques. Similar to many benchmark functions, the Rastrigin function is continuous but non-smooth, with sharp, narrow peaks and valleys in its landscape. The function

contains periodic components due to the cosine term, leading to oscillations in its landscape The Schwefel function exhibits non-separability, meaning that the variables are interdependent and optimizing each variable individually may not lead to optimal solutions. Scaling: The function's landscape can be steep and rugged, with scaling issues becoming more prominent as the dimensionality of the problem increases. This can pose challenges for optimization algorithms, particularly those that rely on gradient information. The Schwefel function has a search space that extends to both positive and negative infinity for each variable, making it an unbounded function. The Langermann function is continuous but non-smooth. This non-smoothness adds to the complexity of optimization problems based on this function. The function involves exponential and trigonometric operations, introducing non-linear behavior that complicates the optimization process. The locations of the function's peaks can be defined arbitrarily, allowing for flexibility in generating different instances of the function

5.2 Experiment Parameters

All the experiments were performed with an initial budget n_0 equivalent to 10-d where d is the dimension of the test function. These samples were taken using Latin Hypercube Sampling for its ability to divide each input parameter's range into equally probable intervals. It ensures that each interval along each dimension is represented by exactly one sample, thereby providing a more uniform coverage of the parameter space compared to simple random sampling. If the sub region is not branchable during configuration rollout then the parent region is used to perform the rollout routine. Number of cuts $B = f(m)$ is a function of the number of agents. To ensure fairness,

all agents are assigned to sub regions that have equal Hypervolume. Across the experiments the number of configurations N^c, the rollout horizon h and the number of agents m are modified. All the experiments were executed on Sol - High Performance Computing infrastructure at ASU with 64-core node and 2GB of RAM per core.

5.3 Performance Analysis

In this section, we verify the hypothesis concerning the performance of MAroBO. In fact, we want to verify the following: *(H1)* Increasing the number of configurations improves the performance of MABO; *(H2)* The number of agents and the rollout horizon length have a positive impact on the performance of MABO; *(H3)* MABO is more robust to higher dimensions than BO.

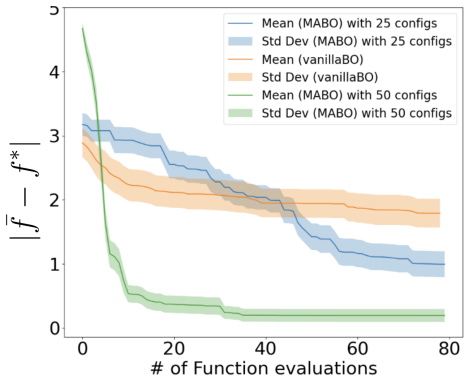


Figure 7. 2-d Langermann with $N^c = 25, N^c = 50$.

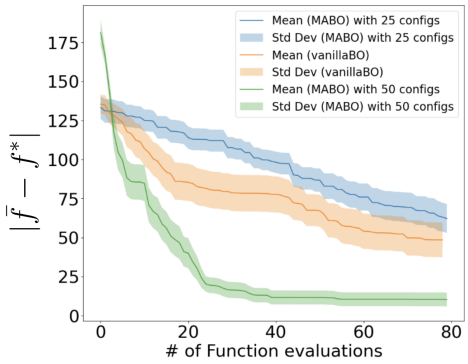


Figure 8. 2-d Schubert with $N^c = 25, N^c = 50$

Experiment 1: Performance in low dimensional settings for varying configurations. Table 1 shows the results of different 2d functions. We see here the effect of the number of configurations. We expect that the higher the number of configurations evaluated the better the solution. The choice of number of configurations

Table 1. Results for low dimensional experiments. Across the experiments $m = 4$, $h = 4$.

Function	d	N^c	$\lvert f^K_{\text{MABO}} - f^*\rvert \pm \text{stdErr}$	$\lvert f^K_{\text{BO}} - f^*\rvert \pm \text{stdErr}$	K
Shubert	2	25	62.20±6.56	48.50±6.05	78
Shubert	2	50	10.27±4.39	48.50±6.05	78
Langermann	2	25	0.83±0.17	1.62±0.22	78
Langermann	2	50	0.03±0.09	1.62±0.22	78

is arbitrary, although the count of configurations can be attributed to the dimension of the input domain. This is confirmed by the empirical results in all the cases Figure **??** shows how MABO Shubert has better performance when the configurations are doubled from 25 to 50. Similarly, Langermann in Figure 7 exhibits better performance when the evaluations per iteration is doubled from 25 to 50. While Langermann eventually surpasses as we get more samples, it clearly shows that 25 configurations is insufficient.

Table 2. Experiments to assess the impact of more agents and longer look ahead horizon. All the functions were evaluated with $N^c = 100$.

Function	d	m	h	$\lvert f^K_{\text{MABO}} - f^*\rvert \pm \text{stdErr}$	$\lvert f^K_{\text{BO}} - f^*\rvert \pm \text{stdErr}$	K
Langermann	6	4	4	4.09±0.03	4.50±0.05	100
Langermann	6	4	8	3.76±0.06	4.50±0.05	100
Langermann	6	8	4	4.02±0.001	4.50±0.05	100
Schwefel	4	4	2	440.04±38.18	518.85±36.22	100
Schwefel	4	8	2	464.15±24.15	518.85±36.22	100
Schwefel	4	8	6	516.13±27.76	518.85±36.22	100
Rastrigin	6	4	2	2.34±0.24	2.42±0.25	80
Rastrigin	6	4	6	2.04±0.20	2.42±0.25	80

Experiment 2: Performance for varying number of agents and higher dimensions. MABO was run with $N^c = 100$ configurations across all the cases and we notice that it is always superior to BO (Table 2). We can also observe how doubling the number of agents does not seem to have a remarkable effect (Figure 9). We also

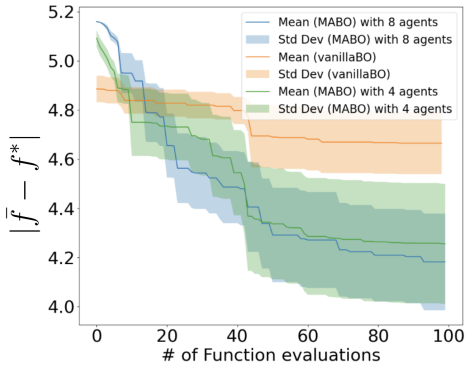


Figure 9. 6-d Langermann with m = 4 vs m = 8.

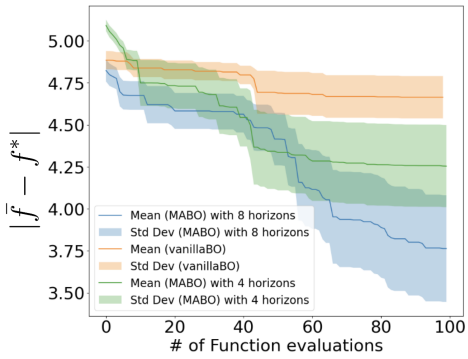


Figure 10. 6-d Langermann with h = 4 vs h = 8

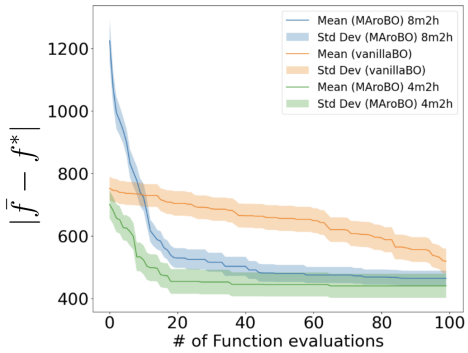


Figure 11. 4-d Schwefel with m = 4 vs m = 8.

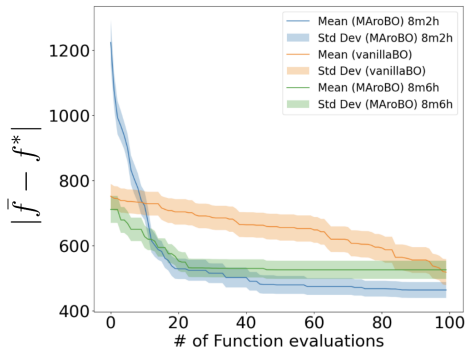


Figure 12. 4-d Schwefel with h = 2 vs h = 6

observed an important variability in the quality of the solutions which may be brought back to the low number of rollout replications N^{RO}, which varied from $5 - 10$ across all the experiments. Similarly, observing Schwefel Figure 11, we find that doubling the agents doesn't necessarily have a remarkable impact. Analysing the results by increasing the rollout horizons reveal that Langermann (Figure 10) shows positive impact when doubling the rollout horizon. Similarly, observing the Rastrigin (Figure 13) results, we notice how increasing the rollout horizon has a positive impact, but this finding is not confirmed under the Schwefel (Figure 12) which calls for a further experimental fraction and a deeper understanding of the impact of the interaction between the agents and the horizon.

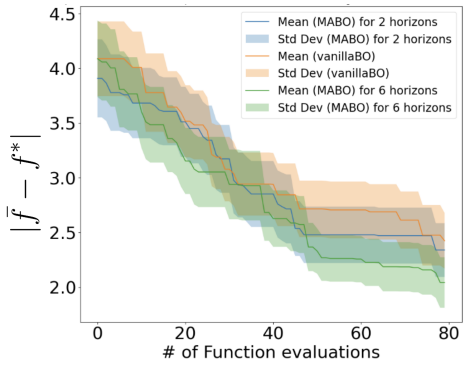


Figure 13. 6-d Rastrigin with h = 2 vs h = 6.

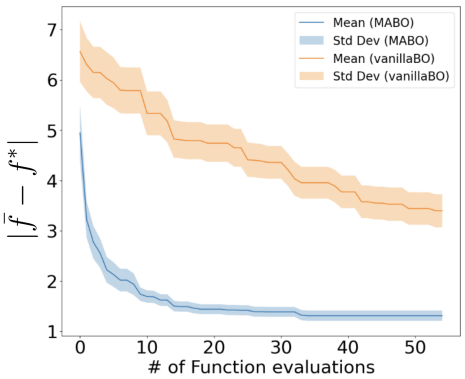


Figure 14. 10-d Rastrigin with m = 4 and h = 4

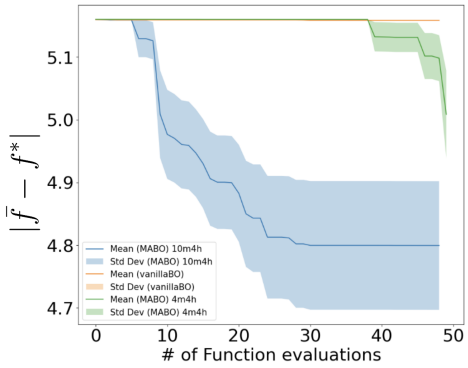


Figure 15. 10-d Langermann with m = 10 vs m = 4.

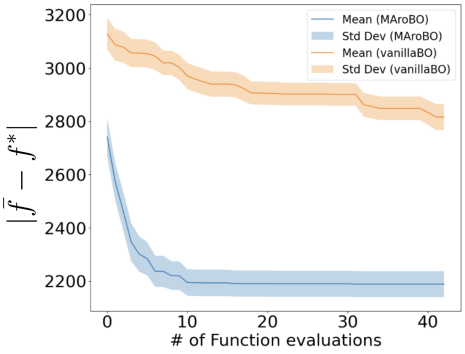


Figure 16. 10-d Schwefel with m = 4 and h = 4

Table 3. Experiments on higher dimensions. Langermann used $N^c = 500$, while Rastrigin and Schwefel were evaluated under $N^C = 600$.

Function	d	m	h	$\lvert f^K_{\text{MABO}} - f^*\rvert \pm \text{stdErr}$	$\lvert f^K_{\text{BO}} - f^*\rvert \pm \text{stdErr}$	K
Langermann	10	4	4	4.95±0.0002	4.99±0.0002	47
Langermann	10	10	4	4.64±0.01	4.99±0.0002	47
Rastrigin	10	4	4	1.31±0.09	3.39±0.31	53
Schwefel	10	4	4	2188.67±48.44	2816.05±42.49	42

Experiment 3: Effect of increased dimensionality. Table 3 reports the results for the higher dimensional cases. First off we observe the detrimental effect that the increased dimensionality has on the performance of the algorithms. We

observed that the algorithm is relatively more explorative than BO by distributing the sampling among agents. Figure 15 shows an exemplar behavior where BO has a very hard time initially to identify improving areas and MABO with its agents succeeds in moving towards improved values. Results from more replications should ensure its consistency. Further experiments on more complex functions needs to be done to ensure the consistency of MABO. Similarly, Rastrigin and Schwefel results are shown in Figure 14 and Figure 16 where MABO still outperforms the vanilla BO significantly. Configurations needed would indeed be large for high dimensional functions so an empirical study of different combinations of parameters - number of agents, configurations evaluated, lookahead horizon, MC iterations needs to be done to establish the efficacy of MABO.

Chapter 6

CONCLUSION

With this chapter, we come to the conclusion of the thesis. The thesis started by proposing the two research questions:

- *RQ1. Can the utilization of non-myopic acquisition functions enhance the performance of Bayesian optimization by providing better point estimates?*

  Response: MABO proposed in Chapter 4 builds on the foundations presented in Chapter 2. MABO indeed can result in better convergence to the global minimum/maximum. Chapter 5 results establish this fact with the results from Rastrigin and Langermann function . The consistency of which needs to be further explored and experimented going forward.

- *RQ2. Does partitioning the input space prove to be effective in high dimensional optimization problems by systematically narrowing down the search space into regions of interest?*

  Response: Chapter 4 shows how the combination of Multi agent rollout with a partitioning scheme can narrow down to the region of interest better than vanilla BO. Experimental results with 10-d Langermann, Schwefel and Rastrigin funtion in Chapter 5 reveal that a distributed modeling of the input space via region partitioning make a positive impact that is worth exploring going forward.

This thesis has proposed MABO for solving global optimization problems. The preliminary results reveal that a version of MABO is consistently dominating Bayesian Optimization, thus granting further analysis. Future work will particularly be focused on increased agent exploration and computational efficiency of the rollout loop.

REFERENCES

Bertsekas, Dimitri. 2020. *Multiagent Rollout Algorithms and Reinforcement Learning.* arXiv: 1910.00120 [cs.LG].

———. 2021. *Rollout, policy iteration, and distributed reinforcement learning.* Athena Scientific.

———. 2022a. *Lessons from AlphaZero for optimal, model predictive, and adaptive control.* Athena Scientific.

———. 2022b. "Newton's method for reinforcement learning and model predictive control." *Results in Control and Optimization* 7:100121.

———. 2022c. *Rollout Algorithms and Approximate Dynamic Programming for Bayesian Optimization and Sequential Estimation.* arXiv: 2212.07998 [cs.AI].

Eriksson, David, and Martin Jankowiak. 2021. *High-Dimensional Bayesian Optimization with Sparse Axis-Aligned Subspaces.* arXiv: 2103.00349 [cs.LG].

Eriksson, David, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. 2020. *Scalable Global Optimization via Local Bayesian Optimization.* arXiv: 1910.01739 [cs.LG].

Jiang, Shali, Daniel R. Jiang, Maximilian Balandat, Brian Karrer, Jacob R. Gardner, and Roman Garnett. 2020. *Efficient Nonmyopic Bayesian Optimization via One-Shot Multi-Step Trees.* arXiv: 2006.15779 [cs.LG].

Jones, Donald R, Matthias Schonlau, and William J Welch. 1998. "Efficient global optimization of expensive black-box functions." *Journal of Global optimization* 13:455–492.

Krishnamoorthy, Dinesh, and Joel A. Paulson. 2023. *Multi-agent Black-box Optimization using a Bayesian Approach to Alternating Direction Method of Multipliers.* arXiv: 2303.14414 [math.OC].

Lee, Eric Hans, David Eriksson, Bolong Cheng, Michael McCourt, and David Bindel. 2020. *Efficient Rollout Strategies for Bayesian Optimization.* arXiv: 2002.10539 [cs.LG].

Li, Chuming, Jie Liu, Yinmin Zhang, Yuhong Wei, Yazhe Niu, Yaodong Yang, Yu Liu, and Wanli Ouyang. 2023. "ACE: cooperative multi-agent Q-learning with

bidirectional action-dependency." In *Proceedings of the AAAI Conference on Artificial Intelligence,* 37:8536–8544. 7.

Ma, Haitong, Tianpeng Zhang, Yixuan Wu, Flavio P. Calmon, and Na Li. 2023. *Gaussian Max-Value Entropy Search for Multi-Agent Bayesian Optimization.* arXiv: 2303.05694 [cs.LG].

Mathesen, Logan, Giulia Pedrielli, Szu Hui Ng, and Zelda B. Zabinsky. 2021. "Stochastic optimization with adaptive restart: a framework for integrated local and global learning." *J. of Global Optimization* (USA) 79, no. 1 (January): 87–110. https://doi.org/10.1007/s10898-020-00937-5.

Munteanu, Alex, Amin Nayebi, and Matthias Poloczek. 2019. "A Framework for Bayesian Optimization in Embedded Subspaces." In *Proceedings of the 36th International Conference on Machine Learning, (ICML).* Accepted for publication. The code is available at https://github.com/aminnayebi/HesBO.

Pedrielli, Giulia, Tanmay Khandait, Surdeep Chotaliya, Quinn Thibeault, Hao Huang, Mauricio Castillo-Effen, and Georgios Fainekos. 2021. *Part-X: A Family of Stochastic Algorithms for Search-Based Test Generation with Probabilistic Guarantees.* arXiv: 2110.10729 [cs.LG].

Song, Lei, Ke Xue, Xiaobin Huang, and Chao Qian. 2022. *Monte Carlo Tree Search based Variable Selection for High Dimensional Bayesian Optimization.* arXiv: 2210.01628 [cs.LG].

Wang, Linnan, Rodrigo Fonseca, and Yuandong Tian. 2022. *Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search.* arXiv: 2007.00708 [cs.LG].