Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

BANDIT-BASED MULTI-START STRATEGIES FOR GLOBAL CONTINUOUS OPTIMIZATION

Phillip Guo

University of Maryland
7030 Preinkert Drive
Prince Frederick Hall
College Park, MD 20742, USA

Michael C. Fu

Robert H. Smith Business School
Institute for Systems Research
University of Maryland
College Park, MD 20742, USA

ABSTRACT

Global continuous optimization problems are often characterized by the existence of multiple local optima. For minimization problems, to avoid settling in suboptimal local minima, optimization algorithms can start multiple instances of gradient descent in different initial positions, known as a multi-start strategy. One key aspect in a multi-start strategy is the allocation of gradient descent steps as resources to promising instances. We propose new strategies for allocating computational resources, developed for parallel computing but applicable in single-processor optimization. Specifically, we formulate multi-start as a Multi-Armed Bandit (MAB) problem, viewing different instances to be searched as different arms to be pulled. We present reward models that make multi-start compatible with existing MAB and Ranking and Selection (R&S) procedures for allocating gradient descent steps. We conduct simulation experiments on synthetic functions in multiple dimensions and find that our allocation strategies outperform other strategies in the literature for deterministic and stochastic functions.

1 INTRODUCTION

Global optimization refers to the problem of finding the set of parameters that optimally achieves an objective, often maximizing or minimizing a numerical objective function (Horst et al. 2000). Optimization problems are present everywhere in engineering, logistics, finance, and many other fields. In many applications, the objective function in question is continuous, has many local optima, and involves black box models. As a consequence, black box optimization has an objective function where direct analytical gradient information is unavailable, which makes several optimization strategies infeasible or much more expensive. Non-convex optimization problems can have many local extrema but only one global optimum, and finding this global optimum could be nearly impossible and require an unbounded number of function evaluations (Rinnooy Kan and Timmer 1989).

When local search algorithms such as gradient descent are applied to non-convex optimization, they can often get stuck in these local extrema with a low probability of escaping them, so upon reaching a local minimum the algorithm may conclude that no direction will improve the minimum and will end (György and Kocsis 2011). To illustrate, Figure 1 is highly non-convex with one global minimum at $x = .5$ and several local minima at other x values. If a gradient descent search algorithm was started at any position except for near $x = .5$, it would descend to a suboptimal local minimum.

The problem of getting stuck in local optima is compounded for optimization scenarios with expensive objective functions in a high-dimensional input parameter space. For example, Monte Carlo simulations to test the effectiveness of a set of parameters can be computationally expensive, and in machine learning large datasets are evaluated for each iteration with many parameters that have to be optimized. Even search algorithms that are guaranteed to find the global optima given an infinite amount of time (e.g., random

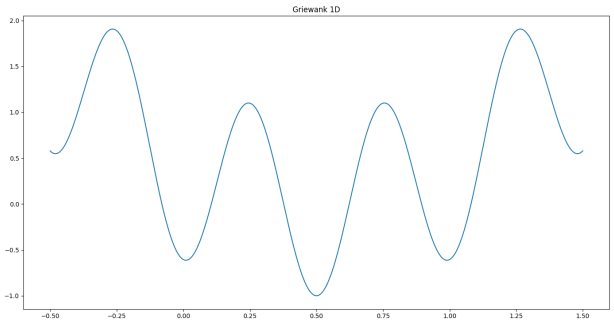

Figure 1: Example of a non-convex function, Griewank function in one dimension.

restart gradient descent) may converge arbitrarily slowly and therefore are computationally infeasible for such expensive problems.

One approach to avoiding being trapped in local optima is starting multiple instances of local search at different starting parameters, which is known as multi-start (Martí 2003). Since local search algorithms such as gradient descent require many evaluations of the potentially expensive objective function, it can become prohibitively expensive to evaluate every started instance of local search to its termination. Therefore, it is useful to efficiently allocate computational resources to the instances, taking into account exploration of instances that have not had a chance to explore too much and exploitation of instances that have a chance of reaching a near-optimal local extremum.

This view of the problem leads us to propose a Multi-Armed Bandit (MAB) approach, with the arms being instances that must be searched. Since our goal is to find the best optima possible, the process of finding the arm/instance that will produce the best optima becomes a Ranking and Selection (R&S) problem (Hong et al. 2021). We seek to iteratively choose instances to carry out steps of a local search algorithm with R&S procedures that can rank instances.

Multi-start is not a traditional MAB problem: implementation necessitates determining a suitable reward structure, which is not obvious. Furthermore, the state of each instance changes as instances descend and near a local minimum: each gradient descent step will likely decrease the function value less than the previous step. Therefore, almost any notion of reward would likely be neither independent nor identically distributed. In this work, we develop "reward models" that take as input the points visited by each instance and output a measurement of reward for MAB approaches to use, while accounting for inherent changes in the instance state.

From the many procedures in the MAB/R&S literature that allocate resources to different arms, we utilize the procedures of Upper Confidence Bound (UCB) (Auer et al. 2002) and Optimal Computing Budget Allocation (OCBA) (Chen et al. 2000), two of the most successful approaches in the literature. Previous research on multi-start has considered allocations that have similar exploration-exploitation goals as MAB approaches (György and Kocsis 2011), but to the best of our knowledge none have introduced reward models that directly translate multi-start into a MAB problem.

The rest of the paper is structured as follows. Section 2 provides the problem setting of multi-start optimization. Section 3 describes the local optimization algorithms and MAB/R&S procedures we use, and our methods of transforming multi-start into a MAB or R&S problem in order to yield our multi-start strategies. In Section 4, we conduct experiments to test the effectiveness of our multi-start strategies vs. others on several synthetic optimization scenarios. Section 5 discusses the results, finding which strategies performed best in each tested scenario. Finally, we conclude and discuss possible future research in Section 6.

2 PROBLEM SETTING

We consider the goal of minimizing a black box objective function f that can be non-convex and has multiple local minima. Formally, we consider $f\colon [0,1]^d \mapsto \mathbb{R}$. Then, we try to find

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}).$$

Since the objective function is a black box, the gradient cannot be calculated analytically: we numerically approximate the gradient with methods described in Section 3.1, which requires multiple evaluations of the objective function. Since the objective function may be extremely expensive to evaluate, we consider the total number of function evaluations used in gradient approximations as the budget. Therefore, our goal is to find the best minimum within a set number of function evaluations.

We assume that optimization is carried out using local search multi-start gradient descent, in which multiple instances of gradient descent are started in different locations. Each instance keeps track of all of the points it has reached so far.

For multi-start, multiple search instances are initialized at different locations. An example is provided in Figure 2, which presents three multi-start instances descending the function from different starting points. The green, red, and cyan curves show the path that each instance takes, with the large colored dots representing the starting points of each instance. As can be seen, one instance descends to the global minimum while the other two descend to local minima: without multi-start, there would be a high chance that the global minimum was never reached.

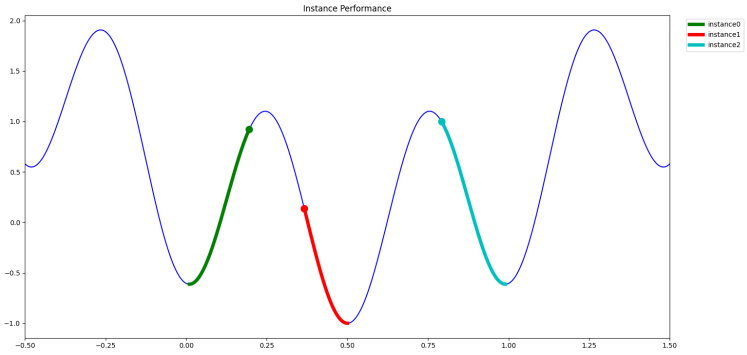

Figure 2: Example of multi-start (three instances) gradient descent on the 1-D Griewank function.

Multi-start strategies consider rounds in which an allocation is computed and certain instances perform a batch of gradient descent steps. In this research, whenever an instance is allocated one descent, the next point for the instance is calculated by one step of gradient descent from the instance's previous point (more details in Section 3.1).

Many applications use parallel computing in their optimization, including the training of machine learning models (Cybenko 2017). We consider the parallel computing scheme to be one in which the task of synchronous computations can be split up between multiple processors. We aim for our strategies to be compatible with a given number of parallel processors ρ fewer than the number of instances k we start, where each processor performs a batch of gradient descents for one instance only. Therefore, in every round, we select ρ out of k instances to carry out gradient descent steps. Assuming that every processor is computationally equal, we perform the same number of descent steps on each processor/instance (the batch size is constant).

We consider two versions of multi-start strategies: constant multi-start and expanding multi-start. Expanding multi-start works the same way as constant multi-start but without a set number of instances. Instead, the algorithm starts with one instance, and in every successive round of allocations a new instance is spawned by generating a random position in $[0,1]^d$ and performing a small number of initial gradient

descent steps. An allocation is then created for the set of instances including the newly created instance. The number of instances grows until the computing budget is exhausted. The expanding version is described in György and Kocsis (2011) as a multi-start strategy with infinitely many search instances.

We propose several strategies which, given each instance's behavior (the points they have previously visited), return an allocation specifying which instances receive descents that round.

3 METHODS

In multi-start optimization, multiple instances of a local search algorithm are started at different positions and are explored. Gradient descent is the local search algorithm considered in this work. Strategies are used to allocate computational resources/gradient evaluations to the different instances depending on their previous performance (the points the instance has visited during gradient descent).

3.1 Gradient Descent

We utilize the gradient descent method of Simultaneous Perturbation Stochastic Approximation (SPSA) (Spall 1992), where the gradient $\nabla f =< \nabla f_1, \nabla f_2, ..., \nabla f_d >$ is stochastically approximated with two function evaluations. SPSA is used for its efficiency in higher dimensions, only requiring two function evaluations independent of the number of dimensions. First, a d-dimension vector of mean-zero random variables $\boldsymbol{\Delta} =< \Delta_1, \Delta_2, ..., \Delta_d >$ is generated: in this research, following rules in Spall (1998), Δ_i are i.i.d. symmetric Bernoulli, i.e., ± 1 with equal probability. The ith component of the estimator for ∇f at position $\boldsymbol{X}_t$ is given by

$$\hat{\nabla}_i f(\boldsymbol{X}_t) = \frac{f(\boldsymbol{X}_t + c_t \cdot \boldsymbol{\Delta}) - f(\boldsymbol{X}_t - c_t \cdot \boldsymbol{\Delta})}{2 c_t \Delta_i}, \quad i = 1, ..., d,$$

where c_t is the finite difference perturbation. Since the numerator is the same for all i, only two function evaluations are necessary (Spall 1992).

Once the gradient is calculated, the next position is given by

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + a_t \cdot \hat{\nabla} f(\boldsymbol{X}_t),$$

where a_t is the step size. Following Spall (1998), we use the sequences $\{a_t\}$ and $\{c_t\}$:

$$a_t = a/(A + t + 1)^{\alpha}, \;\; c_t = c/(t+1)^{\gamma},$$

with the values of the parameters A, α, γ, a, c given in Section 4.2.

3.2 Multi-Start Allocation Strategies

We introduce new allocation strategies based on MAB and R&S procedures and compare them to several multi-start allocation strategies from the literature: MetaMax(K), MetaMax(∞), Uniform, and Explore-Exploit Uniform allocation strategies from György and Kocsis (2011).

3.3 Multi-Armed Bandit and Ranking and Selection

The strategies we introduce in this paper are formulated around reducing the choice of instances to a MAB or R&S problem and applying allocation from MAB/R&S algorithms. These allocation algorithms rank arms relative to each other by some score function, which allows us to select the top arms to exploit in parallel computing.

MAB problems model sequential decision making in which agents choose arms to pull that will yield rewards, with the rewards having unknown distribution. The goal of MAB approaches is to determine the best strategy of pulling arms to maximize overall reward. MAB strategies consist of exploring all arms enough to discover potentially more rewards, while exploiting the arms that are likely to yield high rewards (Auer et al. 2002).

R&S seeks to find the best choice among a set of stochastic alternatives, where the mean performance of each alternative (akin to MAB reward) is sampled from an unknown probability distribution. The primary difference between R&S and MAB is that R&S focuses more on exploration, not needing to additionally exploit multiple high-performing alternatives (Hong et al. 2021).

In order to balance exploration and exploitation, we apply two approaches: Upper Confidence Bound (UCB) and Optimal Computing Budget Allocation (OCBA). UCB is an MAB procedure that seeks to minimize the regret from choosing an arm at a stage (Auer et al. 2002), and OCBA is an R&S procedure that seeks to maximize the probability that the selected arm at the end of the sampling is the correct/optimal arm (Chen et al. 2000). As inputs, UCB and OCBA require an estimate of the expected rewards and variances on the reward distributions of each instance, which can be calculated in different ways (see Section 3.4). For k arms corresponding to k instances, denote the k-dimensional vector of expected rewards for each arm by $\boldsymbol{R} =< R_1, R_2, ..., R_k >$, a k-dimensional vector of the number of samples each arm has received so far $\boldsymbol{N} =< N_1, N_2, ..., N_k >$, a k-dimensional vector of the standard errors in rewards $\boldsymbol{\sigma} =< \sigma_1, ..., \sigma_k >$.

UCB works by assigning a value to each arm. Using a tunable exploration factor constant c, the UCB value for arm i is given by Equation (1), and the arms with the highest UCB values are selected to be pulled.

$$\text{UCB}_i = R_i + c \cdot \sigma_i \cdot \sqrt{\frac{\ln(\sum_{j=1}^{k} N_j)}{N_i}} \tag{1}$$

We combine the OCBA allocation with a "most-starving" algorithm. OCBA calculates the optimal number of samples for each arm $\{\tilde{N}_i\}$, following Equations (2) through (4). Then, each arm is ranked by how "starving" it is, which is the difference between the new optimal allocation of samples and the number of samples it has allocated thus far $\{N_i\}$ given in Equation (5). The arms that are the most starving, i.e., have the highest OCBA-SV$_i$ values, are selected to be pulled.

$$\hat{i} := \text{argmax}_i \, R_i, \quad \delta_i = R_i - R_{\hat{i}} \;\; \forall i \neq \hat{i} \tag{2}$$

$$\frac{\tilde{N}_i}{\tilde{N}_j} = \left(\frac{\sigma_i / \delta_i}{\sigma_j / \delta_j} \right)^2 \;\; \forall i, j \neq \hat{i} \tag{3}$$

$$\tilde{N}_{\hat{i}} = \sigma_{\hat{i}} \sqrt{\sum_{j \neq \hat{i}} \frac{\tilde{N}_j^2}{\sigma_j^2}}, \quad \sum_{i=1}^{k} \tilde{N}_i = \sum_{i=1}^{k} N_i + 1 \tag{4}$$

$$\text{OCBA-SV}_i = \tilde{N}_i - N_i \tag{5}$$

3.4 Reward Models

MAB and R&S approaches can be used to model multi-start optimization by viewing the different local search instances as arms to allocate gradient descent resources. However, these approaches require a reward distribution, so we must formulate some quantity of reward that measures how well each instance is performing.

We introduce two categories of reward models that allow UCB and OCBA to be used by outputting expected reward and variance in the reward. They take in as input all of the points each instance has visited so far. The pth point visited by instance i is denoted as $\boldsymbol{I}(i, p)$. Reward models calculate expected reward R_i and variance in reward σ_i^2, which can be used in the UCB and OCBA formulas in Section 3.3.

3.4.1 Restless Model

The first of our models uses Restless Bandit approaches (Whittle 1988), which generalize multi-armed bandit problems to settings where the reward distributions of arms are not assumed to be independent and identically distributed (i.i.d.).

We consider the amount that the instance's optima improve by in one iteration divided by the step size in that iteration as its reward: i.e., after one iteration of finding the gradient and stepping the position, the change in the function value divided by a_t (SPSA step size, see Section 3.1) is the reward sampled. It is interesting to note that this approach resembles Expected Improvement (EI) methods from Bayesian Optimization.

The reward samples are not i.i.d., because as the local search progresses near an extremum, it is expected that the function changes less with each iteration, so the expected reward is lower.

To account for the changing reward distributions, we turn to the Restless Bandit (RB) literature. In the RB model, the reward distributions of each arm may change over time, so older reward samples are likely to be less helpful than newer samples at predicting the current shape of the reward function. RB approaches use some way to lessen the weight of older reward samples, including weighting the importance of previous rewards with an exponentially decreasing discount factor (Raj and Kalyani 2017) or considering only the past few rewards (Trovo et al. 2020). We denote this model as "Restless".

We alter the expected reward and reward standard error formulas to discount older reward samples with an exponentially decreasing discount factor $\lambda \in (0,1]$, as well as only consider a sliding window of the last τ values. Then, these new values are plugged in for UCB and OCBA.

Our formulas are based on Garivier and Moulines (2008). They take instance history $\boldsymbol{I}(i)$ for instance i, the total number of points sampled by $\boldsymbol{I}(i)$ as p_i, discount factor λ, and sliding window τ. For the tth point sampled, the reward sample is

$$\frac{f(\boldsymbol{I}(i,t+1)) - f(\boldsymbol{I}(i,t))}{a_t}.$$

The expected reward is a weighted mean of the previous rewards, and the standard error is the square root of a weighted variance of those differences.

$$M(i) = \sum_{s=1}^{\tau} \lambda^s$$

$$R_i = \frac{1}{M(i)} \sum_{s=1}^{\tau} \lambda^s \cdot \frac{f(\boldsymbol{I}(i,p_i-s+1)) - f(\boldsymbol{I}(i,p_i-s))}{a_{p_i-s}}$$

$$\sigma_i^2 = \frac{1}{M(i)} \sum_{s=1}^{\tau} \lambda^s \cdot \left(\frac{f(\boldsymbol{I}(i,p_i-s+1)) - f(\boldsymbol{I}(i,p_i-s))}{a_{p_i-s}} - R_i \right)^2$$

3.4.2 Traditional Model

In our second model, we consider the function value of every point reached by the instance as a reward sample. The expected reward is the function value of last point reached by the instance, and the variance is the sum of squares of deviations from the mean function value for all of the points reached by the instance. We denote this model as "Traditional", or shortened to "Trad" (the two names are interchangeably used throughout this paper).

$$R_i = \boldsymbol{I}(i,p_i)$$

$$\sigma_i^2 = \frac{1}{p_i} \sum_{s=1}^{p_i} (f(\boldsymbol{I}(i,s)) - \bar{f}_i)^2, \quad \bar{f}_i = \frac{1}{p_i} \sum_{s=1}^{p_i} f(\boldsymbol{I}(i,s))$$

3.4.3 Strategies

We test each combination of a reward model (Restless or Traditional), an allocation method (UCB or OCBA), and the constant and expanding variations as an allocation strategy. In total, we present 8 new strategies: Restless OCBA Constant, Restless OCBA Expanding, Restless UCB Constant, Restless UCB Expanding, Traditional OCBA Constant, Traditional OCBA Expanding, Traditional UCB Constant, and Traditional UCB Expanding.

3.5 Other Allocation Strategies

For purposes of comparison, we implement other allocation strategies from the literature.

The Uniform allocation strategy is to allocate the same number of function evaluations to each instance, regardless of performance. In every round, each instance is given one batch of descents. We test a constant and expanding version of Uniform.

The Explore-Exploit Uniform allocation strategy performs a Uniform allocation for the first half of the total allocation budget, and then exploits only the instances which have reached the lowest minima so far for the second half of the allocation budget. We shorten the name to EEUniform. We only test a constant version and not an expanding version for Explore-Exploit Uniform because for the expanding version, in the latter half of allocation, new instances would not be explored at all.

The MetaMax allocation strategy was introduced in György and Kocsis (2011). It calculates all of the instances that could possibly be optimal within a confidence level using a convex hull algorithm. It has a constant version, MetaMax(K), and an expanding version, MetaMax(∞). In the paper, the researchers demonstrate that MetaMax has superior performance to a variety of other allocation strategies, so if our strategies can outperform MetaMax, they would likely outperform other strategies in the literature.

The MetaMax strategies cannot allocate to a set number of instances; they select a variable number of instances in every round (often only 2), so there might be more or fewer selected instances than the fixed number of processors in any given round. Therefore, the MetaMax strategies are not easily generalized to the parallel computing case. However, we still compare to MetaMax because many optimization applications cannot make use of parallel computing, and we want to determine if our strategies will perform comparatively well in these typical applications.

4 NUMERICAL EXPERIMENTS AND RESULTS

4.1 Objective Functions

We want to compare the effectiveness of our strategies vs. others in the literature. In order to conduct experiments with many iterations, we test our strategies by attempting to minimize synthetic functions, for both deterministic and stochastic objective functions.

We use rescaled versions of the Ackley, Griewank, and Rastrigin synthetic functions (Zhu and Kwong 2010), which all have many local minima: traditional gradient descent would very likely get stuck in a suboptimal local minimum. Each function accepts input with an arbitrary number of dimensions as a vector $\boldsymbol{x} =< x_1, x_2, ..., x_d >$. We rescale each function to be active in the domain of $[0,1]^d$ for all i (since we initialize all of our instances independently and uniformly randomly in the domain of $[0,1]^d$). The functions are shown in Figure 3.

We also test noisy versions of our functions to compare our strategies with stochastic objective functions. We add Gaussian noise to the output of our functions, with standard deviation depending on the range of the function on the domain of $[0,1]^d$. We use a standard deviation of .25 for Ackley, .05 for Griewank, and 1.0 for Rastrigin.

4.2 Simultaneous Perturbation Stochastic Approximation

We set the values of SPSA according to Spall (1998) and György and Kocsis (2011): $A = 60, \alpha = .602, \gamma = .101$, and a, c depending on the specific case. In our experiments, we test both stochastic and deterministic functions with $x_i \in [0,1]$ for each parameter x_i in X: in deterministic functions we use $c = .00001$, since we want to simulate more direct gradient descent that can get stuck in local minima easily, and in stochastic functions we use $c = .2$ from Spall (1998). We use different a for different numbers of dimensions.

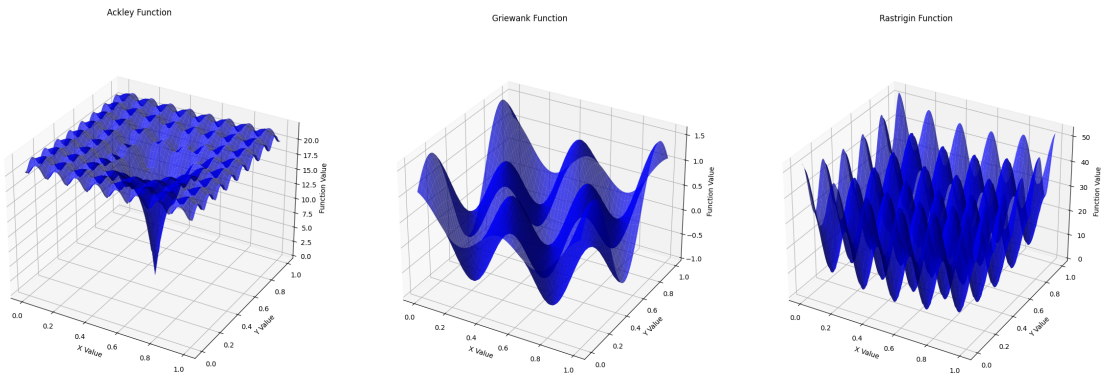

Figure 3: Rescaled versions of the Ackley, Griewank, and Rastrigin functions, respectively, shown on a domain of $[0,1]^d$.

4.3 Illustrative Allocation Examples

We implement the algorithms described in Section 3. We present two examples for visualization purposes: one of our strategies, the Restless OCBA Constant strategy in Figure 4, and an expanding strategy, MetaMax(∞) in Figure 5. On the left of each figure is a visualization of multiple instances descending the 2D Griewank function, on the upper right is a graph of the function evaluations allocated to each instance, and on the bottom right is a graph of the current discovered minimum as function evaluations are allocated.

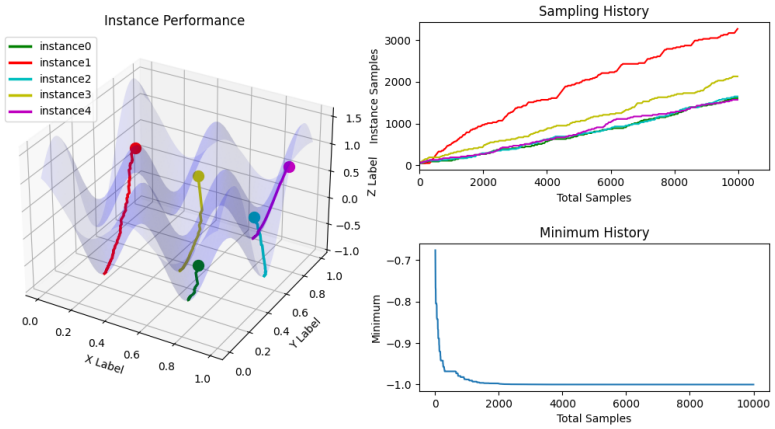

Figure 4: 5 instances descending the 2D Griewank function according to Restless OCBA Constant allocation.

4.4 Error Curves

We ran a series of tests with shared parameters to determine the relative performance of all of the strategies. We created an average error curve, with error being the difference between the minimum reached so far and the actual global minimum of the function. The error was plotted against the number of function samples allocated so far. Error curves for each allocation strategy were averaged over 2500 trials of multi-start descent, with each trial having different starting positions for each instance.

4.4.1 Parameters

We test the strategies in 2, 5, 10, and 20 dimensions. Each number of dimensions has different parameters. The number of instances selected in each round, ρ, is 2 for 2 dimensions, 3 for 5 dimensions, and 5 for

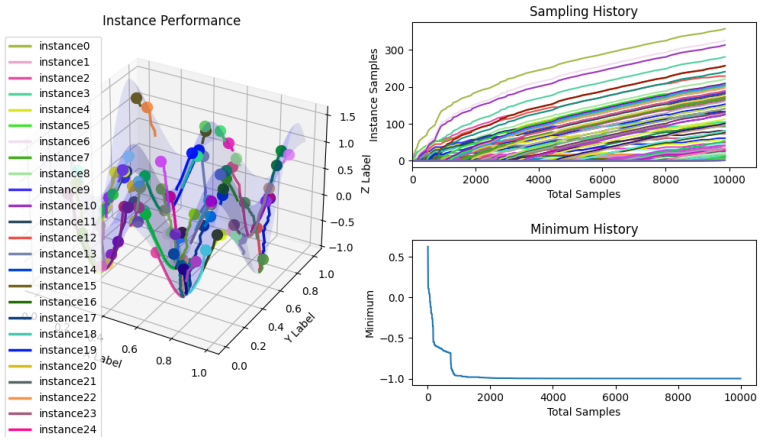

Figure 5: The MetaMax(∞) Strategy descending the 2D Griewank function.

10 and 20 dimensions. The number of instances started for constant strategies, k, is 5 for 2 dimensions, 10 for 5 dimensions, and 100 for 10 and 20 dimensions.

Every test shares certain parameters. In every trial, our maximum computing budget of functions is 10000: the given function is evaluated 10000 times in the trial. In every round, each selected instance is given 10 gradient descent steps. Every instance receives 10 gradient descent steps at the start to ensure a minimum level of exploration. We set the discount factor λ from Section 3.4 to .9 and the sliding window τ to 15.

We test both the deterministic and the stochastic versions of our optimization functions described in Section 4.1. We set the perturbation size for SPSA to be $c = 1 * 10^{-5}$ in the deterministic case since we want to simulate descending more direct gradient descent, and $c = .2$ in the stochastic case according to guidelines in Spall (1998). The SPSA perturbation size constant, a, is 0.01 for 2 and 5 dimensions, and 0.1 for 10 and 20 dimensions. We present an example of an error curve, for the deterministic (not noisy) Ackley function in 5 dimensions, in Figure 6.

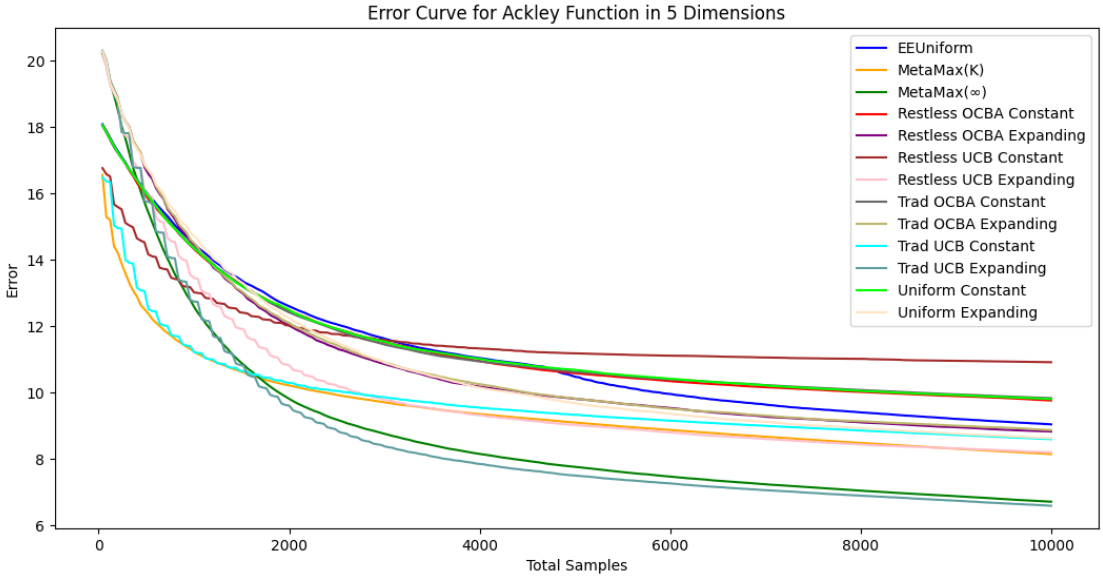

Figure 6: Error curve for the deterministic Ackley function in 5 dimensions.

4.5 Table Comparisons

In this section, we quantify the performance of each tested strategy by aggregating their performances across the different synthetic functions. We stratify the results by the different numbers of dimensions (2, 5, 10, 20) and deterministic/stochastic functions.

To aggregate the relative performance of each tested strategy, we consider the average final minimum reached by each strategy after the 10000 evaluation computation budget is reached, averaged over 2500 trials. We normalize the minima across the different strategies on a range of $0-1$. Then, we sum the normalized minima over the three optimization functions for a combined error term between 0 and 3.

For ranking our strategies, since we want smaller minima to be ranked higher, we subtract the error term from 3 to get a final score term such that higher scores correspond to better performance and scores are on a range of $0-3$. The scores are given in Figures 7 and 8.

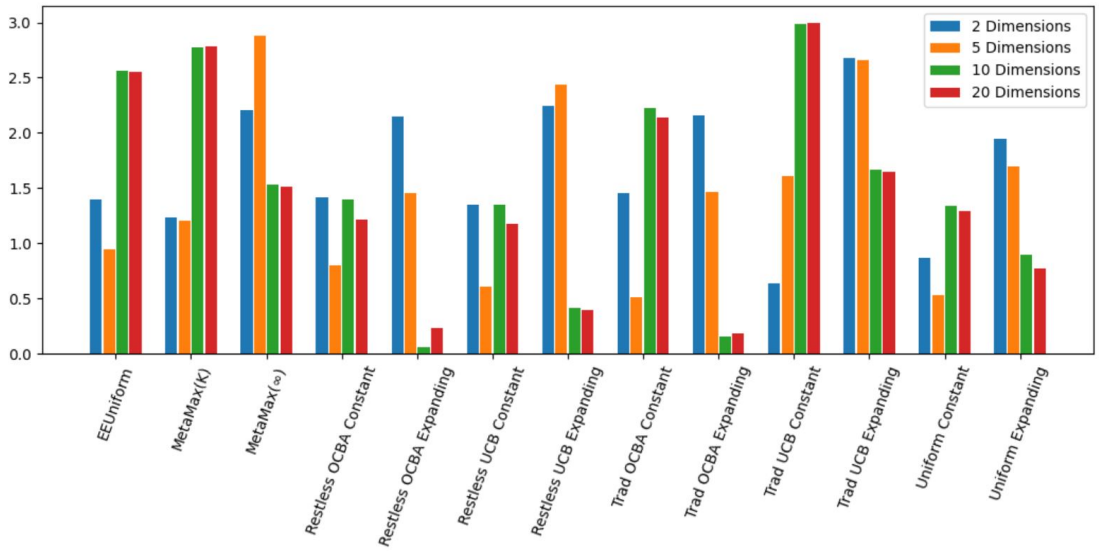

Figure 7: Bar graph showing each strategy's score per dimension, for the deterministic functions.

4.6 Deterministic Objective Functions

We use Figure 7 to compare each strategy in the deterministic function case. The best performing strategies, which have the highest scores, for lower dimensions (2 and 5) are MetaMax(∞), Restless UCB Expanding, and Traditional UCB Expanding. For higher dimensions (10 and 20), we find that Traditional UCB Constant is the best performing strategy, while MetaMax(K) and Explore-Exploit Uniform (both constant) perform almost as well. Interestingly, we find that for every strategy with a constant and an expanding version, the expanding version outperforms the constant version in 2 and 5 dimensions, and constant outperforms expanding for 10 and 20 dimensions. We explore possible reasons in Section 5.1.

4.7 Stochastic Objective Functions

We use Figure 8 in the stochastic function case. For low dimensions, Traditional UCB Expanding is the best performing strategy, while for high dimensions Traditional UCB Constant, MetaMax(K), and Explore-Exploit Uniform perform well. We note a similar distinction between constant vs. expanding strategies performing well in high vs. low dimensions.

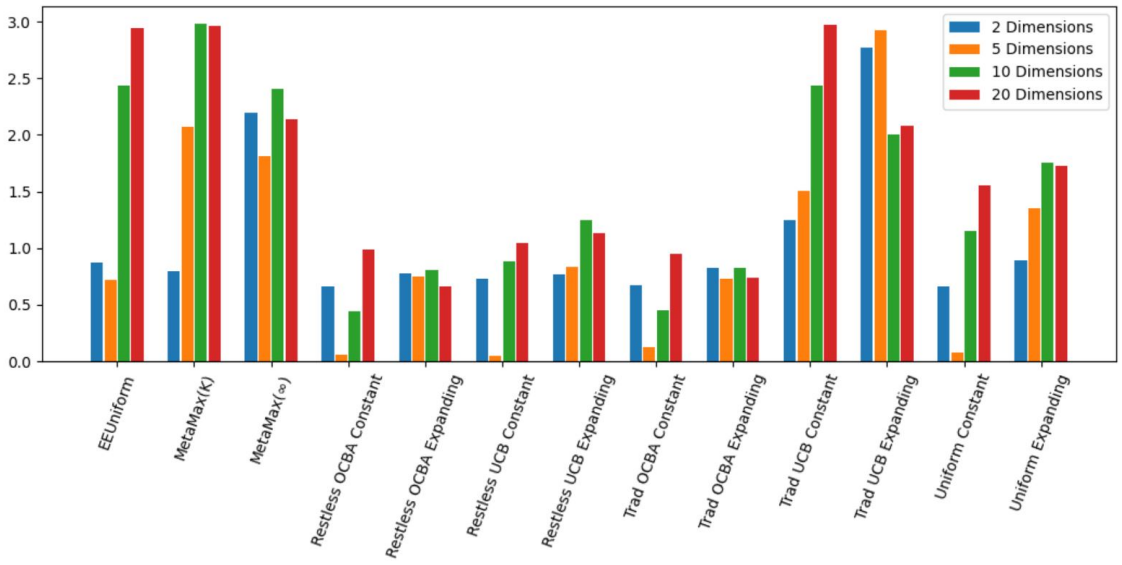

Figure 8: Bar graph showing each strategy's score per dimension, for the noisy/stochastic functions.

5 DISCUSSION

In this section, we discuss the implications of our results.

5.1 Constant vs. Expanding Strategies

Our results suggest that expanding strategies perform better than constant strategies in lower dimensions, and constant strategies outperform expanding in higher dimensions.

This makes sense because in low dimensions, there are fewer local minima, so the new instances that are spawned by expanding variations are more likely to start in a region where the global maximum will be reached. However, in high dimensions, there are many more local minima and new instances are less likely to spawn in advantageous positions that would eventually descend to a global minimum. Thus, more instances being created does not necessarily help - it would be better to exploit only a few instances.

Therefore, for low dimensions we recommend the expanding strategies, while for high dimensions we recommend a constant strategy.

5.2 Recommendations

In our experiments, MetaMax(∞) outperforms Traditional UCB Expanding in 5 dimensions deterministic, and MetaMax(K) outperforms Traditional UCB Constant in 10 dimensions stochastic. In every other case, the Traditional UCB Constant and Expanding strategies outperform every other strategy in both the deterministic and stochastic functions. Therefore, we recommend the Traditional UCB Constant strategy for high dimensions and Traditional UCB Expanding for low dimensions, and especially so for parallel computing applications since the MetaMax strategies which are the second-best are not easily adaptable.

6 CONCLUSIONS AND FUTURE RESEARCH

We developed, implemented, and tested several multi-start allocation strategies to determine which are optimal in different scenarios. The strategies we introduced in this paper are designed for parallel computing where multiple processors can perform gradient descents in parallel, but they are also applicable in typical single processor optimization.

We find that our strategies of Traditional UCB Constant and Traditional UCB Expanding perform the best among all of the parallel computing-compatible multi-start allocation strategies we tested in every test case. When additionally considering MetaMax strategies not designed for parallel computing, our strategies are the best in all but two cases, and the second-best behind MetaMax(K) or MetaMax(∞) in the two exceptions.

Potential future research includes establishing theoretical results for the performance of our strategies; testing our successful strategies in real-world optimization scenarios; developing a combination of constant and expanding strategies, possibly a hybrid method of expanding up until some maximum bound of instances; testing alternative finite difference schemes to SPSA; and investigating connections with Bayesian EI approaches.

ACKNOWLEDGMENTS

This work was supported in part by the Air Force Office of Scientific Research under Grant FA95502010211.

REFERENCES

Auer, P., N. Cesa-Bianchi, and P. Fischer. 2002. "Finite-time Analysis of the Multiarmed Bandit Problem". *Machine Learning* 47(2):235–256.

Chen, C.-H., J. Lin, E. Yücesan, and S. E. Chick. 2000, July. "Simulation Budget Allocation for Further Enhancing the Efficiency of Ordinal Optimization". *Discrete Event Dynamic Systems* 10(3):251–270.

Cybenko, G. 2017, May. "Parallel Computing for Machine Learning in Social Network Analysis". In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1464–1471.

Garivier, A., and E. Moulines. 2008, May. "On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems". *arXiv:0805.3415 [math, stat]*. arXiv: 0805.3415.

György, A., and L. Kocsis. 2011, July. "Efficient Multi-Start Strategies for Local Search Algorithms". *Journal of Artificial Intelligence Research* 41:407–444. arXiv: 1401.3894.

Hong, L. J., W. Fan, and J. Luo. 2021, September. "Review on Ranking and Selection: A New Perspective". *Frontiers of Engineering Management* 8(3):321–343. arXiv: 2008.00249.

Horst, R., P. M. Pardalos, and N. V. Thoai. 2000, December. *Introduction to Global Optimization*. Springer.

Martí, R. 2003. "Multi-Start Methods". In *Handbook of Metaheuristics*, edited by F. Glover and G. A. Kochenberger, International Series in Operations Research & Management Science, 355–368. Boston, MA: Springer US.

Raj, V., and S. Kalyani. 2017, July. "Taming Non-stationary Bandits: A Bayesian Approach". *arXiv:1707.09727 [cs, stat]*. arXiv: 1707.09727.

Rinnooy Kan, A. H. G., and G. T. Timmer. 1989, January. "Chapter IX Global Optimization". In *Handbooks in Operations Research and Management Science*, Volume 1 of *Optimization*, 631–662. Elsevier.

Spall, J. 1992, March. "Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation". *IEEE Transactions on Automatic Control* 37(3):332–341.

Spall, J. 1998, July. "Implementation of the Simultaneous Perturbation Algorithm for Stochastic Optimization". *IEEE Transactions on Aerospace and Electronic Systems* 34(3):817–823.

Trovo, F., S. Paladino, M. Restelli, and N. Gatti. 2020, May. "Sliding-Window Thompson Sampling for Non-Stationary Settings". *Journal of Artificial Intelligence Research* 68:311–364.

Whittle, P. 1988. "Restless Bandits: Activity Allocation in a Changing World". *Journal of Applied Probability* 25:287–298.

Zhu, G., and S. Kwong. 2010, December. "Gbest-Guided Artificial Bee Colony Algorithm for Numerical Function Optimization". *Applied Mathematics and Computation* 217(7):3166–3173.

AUTHOR BIOGRAPHIES

PHILLIP GUO is an undergraduate at the University of Maryland, majoring in computer science. He conducted this research as part of his high school Senior Research Project at the Math, Science, and Computer Science Magnet Program in Montgomery Blair HS. His research interests include optimization and machine learning. His e-mail address is philliphguo@gmail.com.

MICHAEL C. FU holds the Smith Chair of Management Science in the Decision, Operations, and Information Technologies department of the Robert H. Smith School of Business, University of Maryland. He served as Program Chair for the 2011 Winter Simulation Conference. His e-mail address is mfu@umd.edu, and his Web page is https://www.rhsmith.umd.edu/directory/michael-fu.