Digital Object Identifier 10.1109/ACCESS.2024.DOI

Sobering Up! Gaining Focus In Sequential Model Optimization

ROHAN JOSEPH MATHEW[1], JACOB GERLACH[2], AND JOSHUA JOSEPH.[3]
[1]North Carolina State University, Raleigh, NC 27695, USA (e-mail: rjmathe2@ncsu.edu)
[2]North Carolina State University, Raleigh, NC 27695, USA (e-mail: jwgerlac@ncsu.edu)
[3]North Carolina State University, Raleigh, NC 27695, USA (e-mail: jjoseph6@ncsu.edu)

ABSTRACT Sequential model optimization (SMO) offers a method to navigate through large search spaces in a cost-effective manner enabling us to find a decent, rather than the optimal solution, with significantly less effort compared to comprehensive models that always attempt to yield the optimal solution. The performance of sequential model optimization depends on the acquisition function that determines which sample in the dataset is to be added to the model next. We explore novel acquisition functions for sequential model optimizers that lay more emphasis on selecting the "best" samples towards the end of the search rather than picking the most "interesting" sample throughout the entire search. We have developed and explored three different acquisition functions for the SMO that vary the importance assigned to the best sample and the most interesting sample while introducing new samples to the model as we progress through the SMO process. We have named them: progressive, exponential progressive, and simulated annealing-like acquisition functions. We compare the performance of SMO using each of these acquisition methods and existing acquisition functions that solely consider the most interesting samples throughout the search on software configuration datasets. We observed that SMO performs significantly better with the simulated annealing-like acquisition function when compared to the other acquisition functions on most of the software configuration datasets we ran our experiments on. The progressive and exponential progressive acquisition functions appear to perform comparably with the existing acquisition functions that only pick interesting samples.

INDEX TERMS Simulated Annealing, Configuration, Multi-objective optimization, Sequential Model-based Methods, Progressive learning, Acquisition functions

I. INTRODUCTION

THE world of computer science is filled with problems that require finding a good solution from large search spaces. While there are several machine learning models that are capable of finding the best solution, or at least near-best solutions from large and high-dimensional datasets, most of them require a lot of data, long training times, and are resource intensive. Moreover, not every application necessitates finding the best solution from the search space. For applications that do not require a high level of precision, we can settle for solutions that are good enough but not the best. Techniques like sequential model optimization come in handy in scenarios like this. Some examples in computer science that could benefit from such an approach are finding good configurations for software, finding a good set of hyperparameters from a hyperparameter space for machine learning, and finding good security settings and virtual machine types for cloud environments. [1] [5]

Sequential model optimization (SMO) is a technique that uses the evidence seen by the model so far to determine which sample to evaluate next. An acquisition function returns a score that indicates how good a sample, that has yet to be evaluated, would be to consider next. The sample for which the acquisition function yields the highest score is the one that is evaluated next. The acquisition function calculates this score by determining how similar a given sample is to the "best" subset and "rest" subset (the remaining samples that are not considered "best") in the evidence. The "best" subset is the subset of samples in the evidence that are closest to the ideal solution (heaven). SMO will be explained in more detail in the background section (Section II). [2] [3]

The acquisition function that is used for selecting the next sample clearly impacts the performance of the SMO algorithm. This is because a sample that is deemed to be the best by one acquisition function need not be considered the best by another acquisition function. Therefore, different acquisition functions might cause SMO to look at a different series of samples for the same dataset where one sample may be a more ideal solution than another. Through our work, we are trying to answer the following research questions about acquisition functions:

A. RESEARCH QUESTIONS

RQ1: Can we achieve better results from SMO if we give more importance to the current "best" sample rather than the most "interesting" sample towards the end of the SMO process?

When there are very few samples in the evidence, it makes sense to pick samples that are most challenge the current knowledge, i.e, samples that are highly and equally likely to belong to the "best" subset of samples or the "rest" subset of samples in the evidence as they are likely to be the most informative. Once we have a sufficient number of samples in the evidence (after a few steps in the SMO algorithm), we could transition to look for samples that are the best as per the "best" subset in the evidence. Our intuition behind attempting such an approach is that once we have a sufficient number of samples in the evidence, any sample that is considered to be the best as per the evidence could be good enough, and possibly better than the most ambiguous sample. It does not make sense to start off by looking for the best sample in the earlier stages of the SMO since we do not have enough samples in the evidence yet to make a claim as to whether a sample is good.

RQ2: How do we determine this current "best" sample in the later stages of SMO and how much importance should we give them compared to the most interesting sample?

We attempt to design acquisition functions that start by picking the most interesting samples and eventually give more importance to the best sample as per the evidence. We have come up with three approaches that vary the importance or weights assigned to the most interesting sample and the current best sample as the SMO process progresses:

(i) The progressive acquisition function assigns linear weights to an exploration function and an exploitation function that look for the most interesting and the current best samples respectively which are varied as the SMO process progresses according to the current distance to heaven values and the rate of change of distance to heaven of the best sample seen so far.
(ii) The exponential progressive acquisition function is the linear weighted sum of an exploration function and an exploitation function that look for the most interesting and the current best samples respectively, whose weights are varied as the SMO process progresses according to an exponential function e^{ax}.
(iii) The simulated annealing-like acquisition function which adjusts the exponent of a term in the acquisition based on the function e^{ax} (please refer to section x for more details about the simulated annealing-like acquisition function).

We use the same collection of datasets used by Nair et al. in [1] that contain different software configurations along with attributes to be optimized. The datasets in this collection are configurations for either stream processing systems or FPGA software. The sizes of these datasets vary from 195 to 2880 and their number of attributes vary from 5 to 8. Here, Nair et al. used these datasets to propose a version of SMO that uses CART [4], a fixed-point regression model as the surrogate model rather than the Gaussian Process Model (GPM) and uses a maximum mean acquisition function. We do not attempt to make any changes to the residual model in our work, we only make changes to the acquisition function.

B. CONTRIBUTIONS

This paper introduces three novel acquisition functions that can be applied during the process of Sequential Model Optimizations. These are namely:

1) **Progressive:** Controls exploration vs. exploitation bias by looking back on distance to heaven values in the evidence
2) **Simulated Annealing-Like:** Starts off looking for the interesting sample, then exponentially moves towards looking for the best sample
3) **Exponential Progressive:** Controls exploration vs. exploitation bias using an exponential schedule

These acquisition functions are evaluated across the `FLASH` configuration dataset, with a total of 12,461 configurations, performing 20 repeats of each, and ranked using the Scott Knott statistical test. Scott Knott analysis is then applied again, on top of the ranks seen across all the datasets, to understand the significance of our methods. A parallelized `Makefile` based methodology is introduced that speeds up the time to attain results.

C. STRUCTURE

The rest of the paper is structured as follows. In Section II, we explain the motivation behind our work and provide a brief overview of sequential model optimization. In Section III, we explain the acquisition functions we run SMO using and how they impact the working of the SMO algorithm. In Section IV, we present our experimental setup, the dataset we ran our experiments on, the evaluation metrics, and the statistical methods we have used to rank the acquisition functions. In Section V, we present our results and in Section VI we explain what we have inferred from those results and possible future work. We end with conclusions in Section VII.

II. BACKGROUND

A. MOTIVATION

Most software products in the market today are highly configurable. The behavior of the same piece of software could vary drastically with changes in configuration parameters. Given the ever-increasing numbers of parameters and options per parameter in software systems, it is getting harder to keep track of them and identify the best-performing configurations for a given use case. [1] [5]

A study carried out by Xu et al [5] backs this claim with the following examples: i) the number of parameters in Apache increased almost four-fold in a span of 16 years (from about 150 to 600), ii) the number of parameters in MySQL increased more than two-fold in the same time span (from about 200 to 450), and iii) the number of parameters in Hadoop increased nine-fold in a span of 8 years (from about 20 to 180). A study conducted by Han et al [6] digs into 113 real-world performance bugs sampled from the bug repositories and changelogs of three highly configurable, open-source projects: MySQL, Apache, and Firefox. This study showed that there are specific configuration options that are more likely to cause performance bugs than others. They also claim that configuration-related performance bugs are generally caused by a small subset of the set of all configuration options. An algorithm that finds a good enough configuration set (rather than the best) for a software product is likely to avoid this subset of configuration options that causes these performance bugs.

Sequential model optimization provides a way to select a good configuration from a large search space of configuration options by picking a small number of samples based on a heuristic (defined by the acquisition function). Once sampling is done, only this set of configuration options is evaluated for performance, and the best-performing configuration option is used for running the software in the future.

Since the set of configurations sampled by SMO depends on the acquisition function, it is worth exploring acquisition functions that might yield better results than the ones commonly used. This is the motivation behind our work. We attempt to come up with acquisition functions that start by searching for samples based on how interesting they are and gradually switch over to picking samples that are the best based on the evidence seen by SMO so far. We analyze how SMO performs on the software configurations datasets (described in Section IV A) using these acquisition functions and compare them to existing acquisition functions.

B. SEQUENTIAL MODEL OPTIMIZATION

Sequential model optimization (SMO) is a technique that uses the evidence seen by the model so far to determine which sample to evaluate next. An acquisition function returns a score that indicates how good a sample, that has yet to be evaluated, would be to consider next. [2]

Listed below are the steps carried out in the sequential model optimization algorithm:

(i) Randomly pick a very small number of samples and place them in the evidence. The number of samples randomly placed in the evidence initially will be referred to as $budget_0$ for the rest of this report.

(ii) Sort the samples in the evidence in the ascending order of their "distances to heaven" (means closeness to the objective, this is explained in more detail in Section IV C).

(iii) Place the first n^k samples from the evidence in the "best" subset and the remaining samples in the "rest" subset. (where $k < 1$)

(iv) Repeat the following steps (a to b) till we have reached the limit to the number of samples we can have in our evidence (budget):

  a) For each of the samples that are not in the evidence, determine how likely they are to belong to the "best" and "rest" subsets respectively using Naïve-Bayes. Let us call these likeliness values b and r respectively.

  b) The score returned by the acquisition function will be a function of these b and r values. The sample that is not in the evidence that maximizes the score will be added to the evidence.

(v) Once the number of samples in the evidence reaches the budget, return the sample in the evidence with the least distance to heaven.

The likeliness values b and r for a sample are calculated as follows using Naive-Bayes:

$$b = P(best) \prod_{i=1}^{n} P(x_i \mid best) \tag{1}$$

$$r = P(rest) \prod_{i=1}^{n} P(x_i \mid rest) \tag{2}$$

, where x_i is the i^{th} attribute value of a sample.

C. SIMULATED ANNEALING

Simulated annealing is a local search method that combines the principles of moving towards a region that is closer to the goal and randomness. At each step, the simulated annealing algorithm considers a random move. If making that random move takes it closer to the goal, then it makes that move. If not, it will make that move with a probability of, say 'p'. This probability 'p' is higher in the earlier stages of the algorithm and it gradually decreases exponentially with time, according to the function $e^{-\Delta E/T}$, where ΔE is the difference in the scores (closeness to the goal) between the current step and the next step and the T is the number of steps completed. [7]

Since the probability of picking a random sample is high in the earlier stages of simulated annealing, this gives the algorithm a chance to jump out of local minima in the earlier stages. As the algorithm reaches its final stages, the probability of picking a random sample is low. This ensures that it does not shoot past the global minimum. It makes a move only if the sample being considered takes it closer to the goal at this stage.

We use principles of simulated annealing in the acquisition functions that we propose. We suggest using acquisition functions that select samples with the most ambiguity initially so that we can jump out of potential local minima we might reach if we solely pick the samples that are the best based on the evidence. We decrease the importance given to the ambiguity and increase the importance given to the best sample as per the evidence exponentially with respect to the number of steps completed, similar to simulated annealing.

D. RELATED WORK

Nair et al [1], in their paper “Finding Faster Configurations using FLASH”, suggest using a version of SMO where CART is used as the surrogate model instead of the Gaussian Process Model for finding good software configurations. They used ‘Maximum Mean’ as the acquisition function in their work. We do not attempt to make any changes to the surrogate model in SMO, we only attempt to develop acquisition functions that become more focused in the later stages of the SMO process and evaluate how well they pick good software configurations.

In their paper “Using Bad Learners to Find Good Configurations”, Nair et al [10] show that “bad” learners that do not yield exact performance measures for software configurations can still be used to rank the performances of different software configurations.

In their paper “Transfer Learning with Bellwethers to Find Good Configurations”, Nair et al [11] show that upon choosing a suitable source to learn from, simple transfer learning techniques can match or even outperform complex transfer learning techniques in finding good configurations even with several times fewer number of iterations.

III. ALGORITHMS

A. ACQUISITION FUNCTIONS

In SMO, the sample that maximizes acquisition function is added to the evidence at each step from budget$_0$ to budget. The acquisition function is evaluated for each sample that is not in the evidence.

Our motivation lies in building acquisition functions that initially favor acquiring information about the most informative examples (exploration) and eventually shifting to acquiring information about the proposed "best" examples (exploitation).

In the following acquisition functions:

- $b =$ the likelihood of a sample belonging to "best"
- $r =$ the likelihood of a sample belonging to "rest"
- $n =$ budget $-$ budget$_0$
- $small = 1 \times 10^{-300}$ (avoids dividing by zero)

1) bonr

$$\text{bonr}(b, r) = \frac{b + r}{|b - r| + small} \tag{3}$$

bonr is maximized when both b and r are very high and b and r are similar values. This acquisition function aids in finding the sample that most challenges the current evidence.

2) b2

$$\text{b2}(b, r) = \frac{b^2}{r + small} \tag{4}$$

b2 is maximized when b is very high and r is very low. This acquisition function aids in finding the sample that is highly likely to belong to "best" with a low likelihood of belonging to "rest" given the current evidence. b2 does not simply help find what is likely to be the "best" sample given the current evidence, it uses its knowledge to avoid samples which the evidence suggest might belong to "rest".

3) Progressive

$$\begin{aligned} \text{Progressive}(b, r, y, i) = \\ \text{wantToExploit}(y, i) \cdot \text{exploit}(b, r) \\ +(1 - \text{wantToExploit}(y, i)) \cdot \text{explore}(b, r) \end{aligned} \tag{5}$$

Progressive utilizes a sum of the outputs of exploit and explore weighted by wantToExploit. wantToExploit is calculated with respect to, i, the SMO iteration from budget$_0$ to budget and, y, a list of the lowest distance to heaven values found at each iteration from 0 to i. This acquisition function changes throughout SMO to modulate bias between feature space "exploration" and evidence-based "exploitation" based on already acquired distance to heaven values.

$$\text{wantToExploit}(y, i) = \begin{cases} 0 & \text{if } i < 2 \\ 1 & \text{if } i/n \geq 0.85 \\ (|y_{i-1} - y_{i-2}| + (1 - y_{i-1}))/2 & \text{otherwise} \end{cases} \tag{6}$$

wantToExploit is calculated using the rate of change in y and the current best distance to heaven value, y_{i-1}. wantToExploit evaluates to 0 when $i < 2$ as y does not contain enough values to calculate the rate of change. This bound can be changed to $i < x$ where $x > 2$ to solely "explore" for longer during the beginning of SMO. wantToExploit evaluates to 1 when $i/n \geq 0.85$ to solely "exploit" during the last 85% of SMO. Similarly, this bound can be changed to exploit for longer at the end of SMO. We attempted several different bounds but seemed to achieve the best results on the datasets tested using these bounds.

Otherwise, wantToExplore is equal to the absolute change in distance to heaven between the two previous best distance to heaven scores, y_{i-1} and y_{i-2} averaged with the previous distance to heaven score, y_{i-1}. Since distance to heaven is bounded by $[0, 1]$, wantToExplore is also bounded by $[0, 1]$. Intuitively, this acquisition function will want to exploit most when distance to heaven is low, as we are likely closer to the global minimum, and when rate of change is high. Alternatively, when rate of change is low, we could be stuck in a local minima in which case it is favorable to bias exploration.

We experimented by using a weighted sum of all previous rate of change values as opposed to merely the most recent change in y. Weights were generated using an exponential decay function strengthen the impact of the most recent information. This approach uses more information from the evidence however we did not observe performance gains. Thus, we favored a simpler approach.

$$\text{explore}(b, r) = \text{bonr}(b, r) \tag{7}$$

We set explore to Equation 3.

$$\text{exploit}(b, r) = b \tag{8}$$

exploit is simply b, the likelihood of a sample belonging to best. When we are exploiting the model we just want to find the "best" sample given the evidence.

4) Simulated Annealing-Like

$$\text{simAnnealing}(b, r) = \left[\frac{(b+1)^{m_i} + (r+1)}{|b-r| + small}\right]_{i=0}^{n-1} \tag{9}$$

Simulated Annealing-Like is similar to Equation 3 with the primary difference being that $b+1$ on the numerator is raised to the power of m_i. This acquisition function changes with respect to, i, the SMO iteration from budget$_0$ to budget.

m increases from 1 to 2 as i increases, meaning that this metric will favor samples with higher likelihood of belonging to best more as SMO reaches later iterations. 1 is added to b on the numerator to prevent the term from decreasing beyond b when $0 < b < 1$ and $m_i > 1$.

$$m = \text{norm}\left(\left[e^{0.25 \cdot i}\right]_{i=0}^{n-1}, 1\right) \tag{10}$$

m determines the power b on the numerator is raised to. m varies with respect to, i, the SMO iteration from budget$_0$ to budget. m is an exponential function normalized between 1 and 2.

$$\text{norm}(x, \text{shift}) = \left[\frac{(x_i - \min(x))}{(\max(x) - \min(x))} + \text{shift}\right]_{i=0}^{n-1} \tag{11}$$

norm normalizes an input list, x, between 0 and 1 and increases all values by shift.

5) Exponential Progressive

$$\begin{aligned}\text{ExpProgressive}(b, r) &= m' \cdot \text{exploit}(b, r) \\ &+ (1 - m') \cdot \text{explore}(b, r)\end{aligned} \tag{12}$$

This acquisition function is a sort of hybrid between Equations 5 and 9. It shares a similar structure to Equation 5 except that wantToExploit (Equation 6) is replaced with m'. explore (Equation 7) and exploit (Equation 8) are unchanged. While Progressive factors in distance to heaven values of samples in the evidence, Exponential Progressive does not have access to this knowledge. Rather, Exponential Progressive merely increases its "want to exploit" (m') exponentially with respect to SMO iteration.

$$m' = \text{norm}\left(\left[e^{0.25 \cdot i}\right]_{i=0}^{n-1}, 0\right) \tag{13}$$

m' is almost identical to m except that exponential values are normalized between 0 and 1 rather than 1 and 2.

IV. EXPERIMENTAL METHODS

A problem that we faced while working with new acquisition functions was getting feedback on the effectiveness of our changes. To address this, and to tighten the feedback loop, we developed a system of auto-generated `Makefiles`. A `Makefile` would be generated per dataset, and would define the acquisition functions at several "budgets". Then, using the `-j` flag of `make`, we were able to run all the acquisition functions for the datasets in parallel. To test our changes across multiple datasets, a higher level `Makefile` was also defined, which further ran our functions across all datasets in parallel. This method of parallelizing 20 test runs across 11 datasets, which contained a total of 12,461 configurations, helped us achieve an average run-time of 01:30s (measured using "`time make -j 8`" on an Apple M2 Pro MacBook Pro).

A. DATA

Nair et al. [1] explore the problem of finding good configurations of a software system. Most systems have a large configuration space, which if configured incorrectly, can lead to sub-par energy and performance characteristics. We utilize the datasets explored in this paper to evaluate our methods. We have picked 11 datasets that consist of configuration spaces from stream processing systems and FPGAs. These datasets all-together consist of 12,461 configurations. The stream processing datasets consist of the objectives of maximizing throughput and minimizing latency. The FPGA datasets, on the other hand focus on minimizing factors like area, throughput, energy usage and runtime. By using fewer evaluations to find better configurations, we therefore have a way of optimizing for energy and performance without having to iterate through the entire configuration space which can be costly.

B. EXPERIMENTAL SET UP

We have used GitHub as the collaboration platform for our experiments. A GitHub organization `ase24-group` was created that houses all the repositories related to our experiments. The repository that contains our code and results can be found on https://github.com/ase24-group/ase24-project. We decided to use Python to implement our changes due to it's familiarity and simple syntax. The `data/flash` folder contains all the datasets from Nair et al. [1], that we have used in our study. The `results/stats/flash` folder consists of intermediate results from from running the following treatments over budgets 9, 15, and sqrt:

- rand: Random sampling without replacement
- progressive: SMO with Progressive acquisition function
- SimAnnealing: SMO with Simulated Annealing-Like acquisition function
- ExpProgressive: SMO with Exponential Progressive acquisition function

sqrt is a budget of $\sqrt{n}$ where n is the number of rows in the dataset.

As well as the following baselines:

- rand_p90: Sample with the minimum distance to heaven out of a random subset of 90% of the data
- base: Median distance to heaven across the entire dataset

The files here, named with the format `<dataset>.stats.txt` are generated by combining the results from multiple parallel runs of our acquisition functions. Once generated, this file is used as input to the Scott Knott statistical test. The results from the Scott Knott tests are placed into `results/sk/flash`. The Scott Knott result for a dataset is written into two files with the format: `<dataset>.sk.txt` and `<dataset>.sk.csv`. While the `txt` files present the statistical groupings more legibly, the `csv` files provide us with an easily parse-able format for future analysis. `results/plots/flash` consists of graphical plots for select acquisition functions, grouped under their dataset folder. Each plot depicts the best distance to heaven values across increasing "budgets". When possible, the plots also include lines to depict the behaviour of the underlying algorithm. The `results/plots/bargraphs` folder provides comparisons for our acquisition functions across different budgets. This helps us understand the overall effectiveness of our changes per dataset. As mentioned previously, our methods make use of dual layer `Makefiles` to parallelize test runs across all datasets. The dataset-level `Makefile` can be found in `src/Makefile`, and the auto-generated treatment-level `Makefile` for each dataset can be found inside `src/makefiles/flash`. To run our experiments, we navigate to the `src` folder and run "`make -j 8`". The `-j` flag defines the maximum number of jobs that can be run in parallel. The top-level `Makefile` then invokes the lower-level `Makefiles`, and the results are written by each process into the `results` folder, scoped by the dataset group and the dataset name. Across all our acquisition functions, the random seed was set to `31210` and the initial "budget" was set to `4`. We then evaluated each function on three different budgets: 9, 15 and the square root of the number of rows in our dataset.

C. EVALUATION METRICS

When optimizing for multiple objectives inside our dataset, we are looking for configurations that maximize or minimize for that objective. The objective columns inside a dataset are denoted by those that have a `+` or a `-` at the end of their name. For example, the headers of `SS-A.csv` are `Spout_wait, Spliters, Counters, Throughput+, Latency-.` Here, the `Throughput` and `Latency` are our objectives, and we need to find rows in the `csv` with maximum values for `Throughput` and minimum values for `Latency`. A critical metric that is used to reach our objective is called distance to heaven. Heaven is defined based on the goal we would like to achieve. If our goal is to maximize an objective, the heaven point is set to `1`, and if our goal is to minimize the objective, the heaven point is set to `0`. By finding the distance to heaven across all objectives, we are able to sort our rows for sequential model optimization into `best` and `rest` sections. Specifically, we make use of the euclidean distance when computing the distance to heaven.

$$H = \sqrt{\left(\sum_{i}^{n}(\overline{y_i} - h_i)^2\right)/n} \qquad (14)$$

In this equation, n denotes the number of objectives in our dataset. $\overline{y_i}$ is the value of our objective normalized from 0 to 1, and h_i indicates the heaven point. $h_i = 0$ for goals we are minimizing and $h_i = 1$ for goals we are maximizing.

After compiling the results of our experimental methods using "`make -j 8`", we then re-applied Scott Knott analysis on top of the ranks seen across all the datasets. This would not only help us understand which method performed better, but would also tell us if that method was significantly better than other methods. The top-level `Makefile` also consists of three new targets to help automate the ranking process. The `ranks` target displays the Scott Knott results for the treatments made across all datasets. The `ranks_plot` target displays a box plot for the ranks assigned to the treatments across all datasets, as seen in Figure 6.

D. STATISTICAL METHODS

When analyzing different experimental methods, one might assume a scoring function for each method, and sort the methods by that score. While a ranking might provide us with a rough ordering of performance, how do we know if a method is "significantly better" than other ones? In such a case, it would make sense to place our ranks into a distribution and see how much the curve is spread around the median. A curve with a sharp peak would indicate that most ranks are around the median, whereas, a curve with a soft peak would indicate otherwise. Now, when comparing two methods, we would not only take into consideration their median ranks, but also how "different" their curves are from each other. A common approach to form our curves would be to "fit" our data into a standard distribution, like a Gaussian or Binomial distribution, and fill in the parameters for that distribution. This is called parametric statistical tests [8]. But in most cases, real world data does not conform to a single distribution. A more effective method would, therefore be to perform non-parametric tests. We have decided to pick up a non-parametric test called Scott Knott [9] for our analysis. With Scott Knott, we group treatments into statistically similar groups, and each group is sorted by their median values. To determine if a treatment is statistically different from another, we perform two tests.

1) **Significance test**: To test for significance, we use the `Bootstrap` test, where we summarize the difference between two samples by comparing how often an observation is larger than a baseline observation. This test returns true if we have a lower count for the number of observations that are larger (which means that it is easier to separate the two samples) [8].

TABLE 1. Rankings for the acquisition function for different budgets across datasets according to the Scott-Knott tests

Rank	Acquisition function and budget
0	rand_p90
1	SimAnnealing_sqrt
2	SimAnnealing_9 SimAnnealing_15 b2_sqrt b2_15 ExpProgressive_sqrt b2_9 bonr_sqrt
3	progressive_sqrt
4	bonr_9 ExpProgressive_9 progressive_15 progressive_9
5	bonr_15 ExpProgressive_15
6	base rand_sqrt
7	rand_15 rand_9

2) **Effect size test**: To test for effect size, we use `CliffsDelta`, where we compare two samples through the number of entries that are larger or smaller in either sample. This test returns true if the number of differences are more than a certain effect size [8].

The goal, therefore is to separate samples that are significantly distinguishable, by more than a small effect size.

V. RESULTS

RQ1: Can we achieve better results from SMO if we give more importance to the current "best" sample rather than the most "interesting" sample towards the end of the SMO process?

Figure 6 suggests that the best performing treatment, besides rand_p90, is SimAnnealing_sqrt and Table 1 suggests the difference between this treatment and lower-ranking treatments is significant according to the Scott-Knott clustering algorithm. According to Table 1, SimAnnealing (budgets 9 and 15), b2 (budgets 9, 15, sqrt), ExpProgressive_sqrt, and bonr_sqrt all perform similarly. Overall, these results suggest that the SimAnnealing and b2 treatments outperform the other treatments tested. b2 always favors finding the most likely "best" sample that is least likely in "rest" while SimAnnealing increases the weight of the "best" likelihood later in the SMO process. This suggests that, for the datasets tested, placing more importance on the current "best" sample rather than the most "interesting" sample yields better results. Our research also suggests that transitioning from prioritizing "interesting" samples to "best" samples over the SMO process yields better results since SimAnnealing performs slightly better than b2 in our testing.

RQ2: How do we determine this current "best" sample in the later stages of SMO and how much importance (or weight) should we give them compared to the most interesting sample?

We have designed three different acquisition functions that determine the current "best" sample in their own different ways. The progressive acquisition function determines the current "best" from the $exploit$ score and assigns to it a weight of $wantToExploit$ both of which are defined in Section III A3). The exponential progressive function determines the current "best" from the $exploit$ score as well, but it uses different criteria to assign and update the weight assigned to it. This is defined in Section III A4). Equations 5 and 12 show how the weights of the current best (represented by $exploit$ in the equation) vary step-by-step in SMO.

For the simulated annealing-like acquisition function, we increase the importance assigned to the current best by increasing the power of $(b+1)$ in Equation 9.

Table 1 shows us that the simulated annealing-like acquisition function outperforms all the other acquisition functions, including the baseline acquisition functions, for a budget size of $\sqrt{n}$ (where n is the size of the dataset). Simulated annealing also outperforms the other acquisition functions we have developed for budgets 9 and 15. The exponential progressive acquisition function outperforms the exponential progressive function for budgets 9 and 15. From the results we have, we can conclude that the simulated annealing-like acquisition function follows the best approach to assign weights to the current best.

VI. DISCUSSION

A. WHAT WAS LEARNED

From Figure 6 and Table 1, we observe that one of our acquisition functions (the simulated annealing-like acquisition function) that assigns more importance to the current best outperformed all the other acquisition functions including the baseline acquisition function for a relatively larger yet small budget (where $budget = \sqrt{n}$, n is the size of the dataset). Even for very small budgets like 9 and 15, the simulated annealing-like acquisition function performs comparably to the baseline $b2$ while outperforming $bonr$. From this, we learn that it might be worth exploring other acquisition functions that vary the importance/weight they assign to the current best and the most interesting samples as the SMO process progresses rather than using an acquisition function that applies the same formula to pick a sample from start till end.

B. THREATS TO VALIDITY

The conclusions made from this study should be considered with the following issues in mind:

1) **Tuning bias:** While developing the acquisition functions and evaluating the results, we worked with the values of the variables and weights in the function, until the results felt favorable. This could indicate a form of bias that leans towards the type of dataset we're using.
2) **Conclusion validity:** We have come to conclusions to our new approaches specifically on the `FLASH` configu-

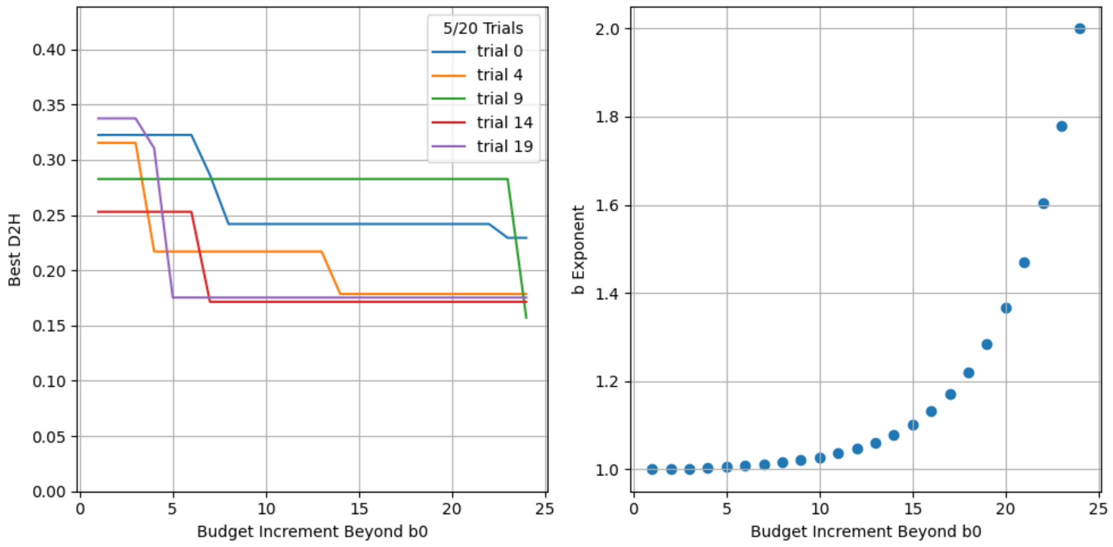

FIGURE 1. Best distance to heaven plotted over budgets for `SS-E.csv`, using the Simulated Annealing-Like acquisition function

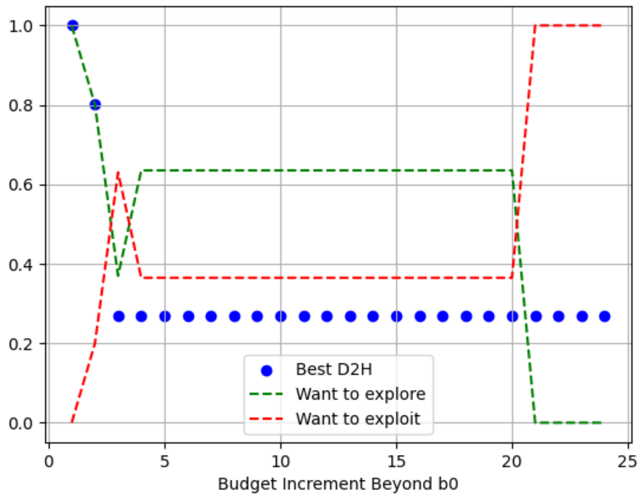

FIGURE 2. Best distance to heaven plotted over budgets for `SS-E.csv`, using the Progressive acquisition function

ration dataset. The validity of these conclusions should be vetted by looking at varied datasets.

3) **Sensitivity:** We noticed the rankings of our treatments responding differently with a small change in function parameters / inputs. While our tests indicate good performance across 12 datasets, we must take into consideration the effect of tiny values on the final result.

C. FUTURE WORK

1) Datasets

We believe applying our novel algorithms to new datasets would be the most impactful way to build from our work. We explored datasets related to software configurations, however our work is not limited to this application. There are many other cases were feature sets are plentiful and acquiring target knowledge is costly. In many of these applications, cheaply arriving at a "good enough" solution is favorable to investing for the absolute "best" solution.

2) Algorithms

Our Progressive algorithm, Equation 5, is highly flexible such that each explore, exploit, and wantToExploit component can be modified. As mentioned in Equation 6, we tried a few different ways to calculate wantToExploit and we are

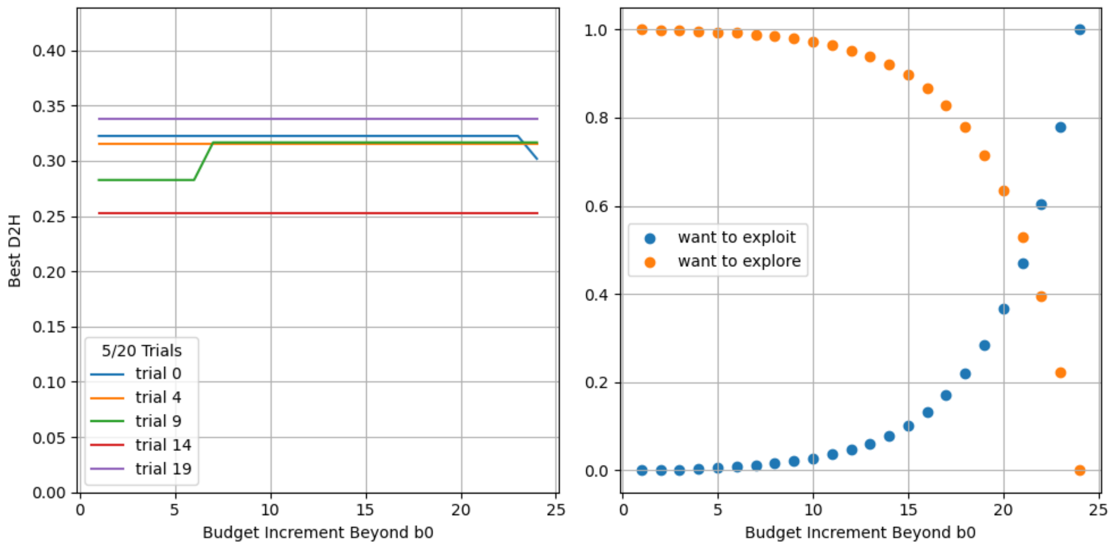

FIGURE 3. Best distance to heaven plotted over budgets for `SS-E.csv`, using the Exponential Progressive acquisition function

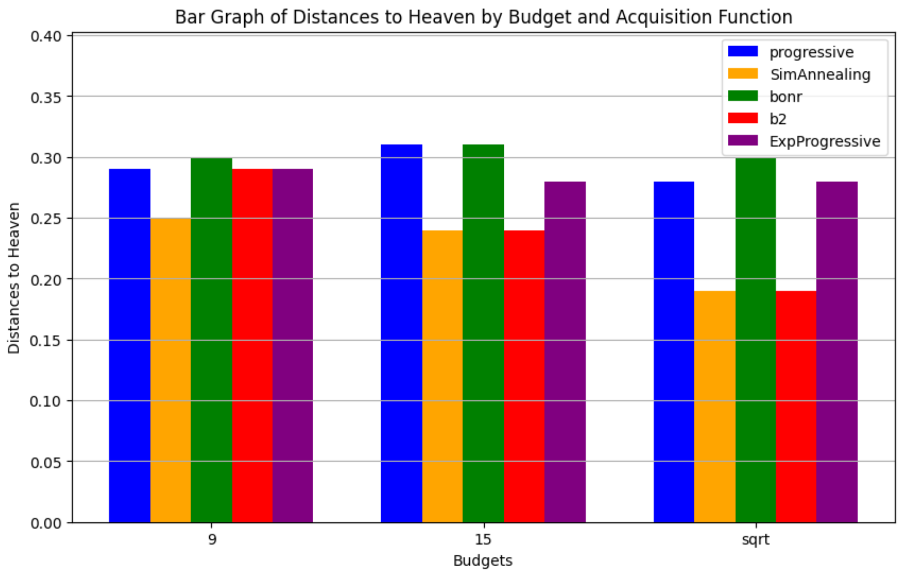

FIGURE 4. Median distance to heaven of 20 trials for multiple acquisition functions on `SS-E.csv`

curious how these, and new, modifications perform across a broad range of datasets. Similarly, other explore and exploit functions should be tested.

Concepts from Simulated Annealing can be applied in different ways to SMO beyond what we tested with Equations 9 and 12.

VII. CONCLUSION

In this report, we have explored three novel acquisition functions that can be applied during Sequential Model Optimization. The idea shared among these functions is to avoid prioritizing for "interestingness" across all iterations of model-building. Instead, we start off "drunk", being exploratory, and then transition to "sobering up", honing on the best sample at the iteration.

Across the datasets we tested, the best performing treatment was our Simulated Annealing-Like acquisition function with a sqrt budget. SimAnnealing also performed well for budgets 9 and 15 but matched the performance of b2 on all budgets. All treatments performed better than the random baselines and the median distance to heaven values. Our Progressive and Exponential Progressive acquisition functions along with bonr exhibited worse performance than SimAnnealing and b2. According to 1, Exponential Progressive

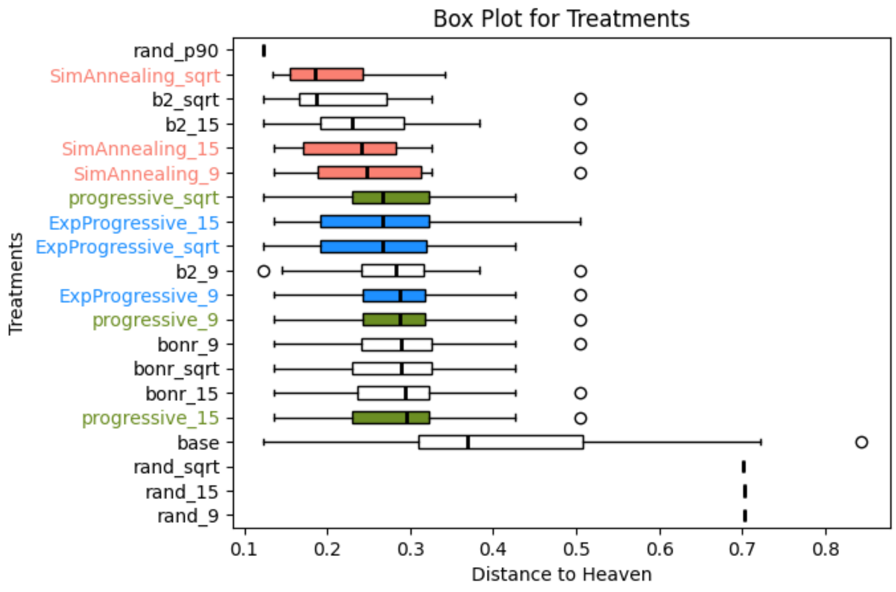

FIGURE 5. Box plot of the best distance to heaven scores, applied on `SS-E.csv`

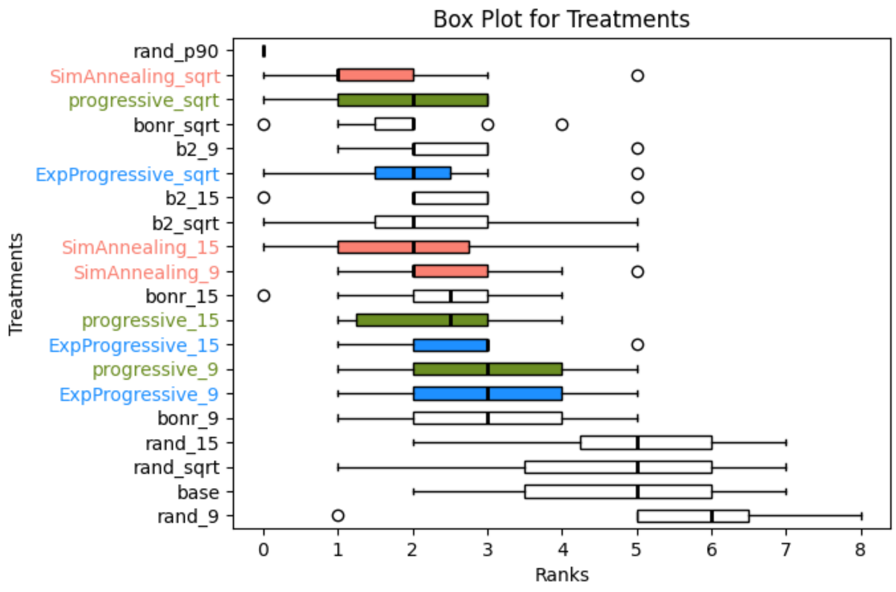

FIGURE 6. Box plot of Scott-Knott ranks across all the datasets

exhibited the same performance as bonr.

We believe the results of our research warrant further testing of our acquisition functions across a broad range of applications beyond software configuration.

REFERENCES

[1] V. Nair, Z. Yu, T. Menzies, N. Siegmund and S. Apel, "Finding Faster Configurations Using FLASH," in *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 794-811, 1 July 2020, doi: 10.1109/TSE.2018.2870895.

[2] T. Menzies, 2024, *AI can be made easy (using good SE)*, Accessed April 2024, <https://github.com/timm/lo/blob/main/docs/gate.pdf>

[3] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint*, 2010

[4] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[5] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). Association for Computing Machinery, New York, NY, USA, 307–319. https://doi.org/10.1145/2786805.2786852

[6] Xue Han and Tingting Yu. 2016. An Empirical Study on Performance Bugs for Highly Configurable Software Systems. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16). Association for Computing Machinery, New York, NY, USA, Article 23, 1–10. https://doi.org/10.1145/2961111.2962602

[7] Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson Education (US). https://ncsu.vitalsource.com/books/9780134671932

[8] T. Menzies, 2024, *Stats*, Accessed April 2024, <https://github.com/txt/aa24/blob/main/docs/04stats.md>

[9] Scott R.J., Knott M. 1974. A cluster analysis method for grouping mans in the analysis of variance. Biometrics, 30, 507-512.

[10] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using bad learners to find good configurations. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). Association for Computing Machinery, New York, NY, USA, 257–267. https://doi.org/10.1145/3106237.3106238

[11] V. Nair, R. Krishna, T. Menzies, and P. Jamshidi, "Transfer Learning with Bellwethers to find Good Configurations," arXiv:1803.03900 [cs.SE], 2018.