VEER: A Fast and Disagreement-Free Multi-objective Configuration Optimizer

Kewen Peng
North Carolina State University
Raleigh, USA
kpeng@ncsu.edu

Christian Kaltenecker
Saarland University
Germany
kaltenecc@cs.uni-saarland.de

Norbert Siegmund
Leipzig University
Germany
norbert.siegmund@informatik.uni-leipzig.de

Sven Apel
Saarland University
Germany
apel@cs.uni-saarland.de

Tim Menzies
North Carolina State University
Raleigh, USA
timm@ieee.org

ABSTRACT
Many software systems can be tuned for multiple objectives (e.g., faster runtime, less required memory, less network traffic or energy consumption, etc.). Optimizers built for different objectives suffer from "model disagreement"; i.e., they have different (or even opposite) insights and tactics on how to optimize a system.

Model disagreement is rampant (at least for configuration problems). Yet prior to this paper, it has barely been explored.

This paper shows that model disagreement can be mitigated via VEER, a one-dimensional approximation to the N-objective space. Since it is exploring a simpler goal space, VEER runs very fast (for eleven configuration problems). Even for our largest problem (with tens of thousands of possible configurations), VEER finds as good or better optimizations with zero model disagreements, three orders of magnitude faster (since its one-dimensional output no longer needs the sorting procedure).

Based on the above, we recommend VEER as a very fast method to solve complex configuration problems, while at the same time avoiding model disagreement. For replication purposes, all our code is on-line: https://github.com/anonymous12138/multiobj.

1 INTRODUCTION
One of the recent successes of AI for software engineering (SE) is automated configuration [20]. Software comes with many options as well as various objectives, and exploring all these configuration options for multiple objectives can be tedious, time consuming, and even error-prone (when done manually). Much recent work shows that AI tools can dramatically improve this procedure; for instance, regression tree learners can report what subset of the configuration options are most influential to achieving better performance [7, 12]. Further, when it takes a long time to run the system with all possible configuration settings, an incremental AI tool can reflect on the model learned so far to recommend what is the next most informative configuration to try [27]. This way, the time required to run enough configurations to effectively optimize software can be substantially reduced (e.g., as shown by the experiments of this paper, after running less than 100 configurations, we can optimize systems with nearly 90,000 configurations).

When business users ask “what has been learned from these models?”, we need interpretable models to offer clear advice on how to best configure a system [6, 29, 33]. But when software systems have multiple objectives (e.g., faster transaction response time, fewer memory requirements, decreased network traffic, decreased energy consumption, etc.), that advice can clash. We call this the model disagreement problem: While one model shows some configuration options to be useful to achieve one objective, another model might argue that such options are actually detrimental for another objective. Table 1 shows examples of model disagreement.

Looking at the literature, we find very little discussion on model disagreement. That is, model disagreement may be a long-standing, but previously undetected, problem. This begs the questions “why has this problem not been reported before?” Perhaps other researchers were content to stop after generating multiple solutions (e.g., 10,000 solutions across the frontier of best solutions). In our work, however, we have been studying interpretation tools that offer clear advice on how to best configure a system. Hence, we prefer not to confuse users with a long list of candidate solutions. Instead, we prefer rule-based summaries such as those seen in Table 1.

This paper tests the following conjecture. If multiple objectives complicate interpretations, then one possible solution is to:

1) Reduce multi-dimensional objective space to a lower-dimensional space.
2) Reason in that reduced space.

Table 1: Examples of model disagreement found by tools described later in this paper. In project SS-F, we mark the two rules chunk < 0.14 and chunk > 0.05 as disagreement because, although their ranges are not necessarily exclusive, they do point to the opposite optimizing directions.

| Project | Model 1 | Model 2 |
|---------|---------|---------|
| SS-E    | Minimize runtime | Minimize CPU load |
|         | columnTiling = True | columnTiling = False |
|         | goodQuality = True | goodQuality = False |
|         | AutoAltRef = False | AutoAltRef = True |
| SS-F    | Minimize latency | Maximize throughput |
|         | max_spout > 0.55 | max_spout < 0.05 |
|         | chunk < 0.14 | chunk > 0.05 |
| SS-G    | Minimize runtime | Minimize CPU load |
|         | compressionZqap = False | compressionZqap = True |
|         | compressionLzma = True | compressionLzma = False |
|         | processorCount > 0.17 | processorCount < 0.17 |
We propose a tactic called VEER, which is applied as follows: (a) lay out all the configuration options as points in an N-dimensional objective space, then (b) rank the best point as “1”. After that, we “VEER” to the nearest best point P to rank it as “2”; and so on. The ranks found by VEERING across all the objectives are then used as a single-dimensional objective space. Configuration recommendations are then found by reasoning over this simpler space.

Overall, the contributions of this paper are:

- We verify the existence of the model disagreement problem in multi-objective software configuration.
- We show that model disagreement is not a simple problem that can be easily solved via some simplistic weighting mechanisms (e.g., a naive multi-regression approach).
- We propose a novel tactic to resolve the model disagreement problem and generate confusion-free model interpretations.
- We show that, since VEER is exploring a simpler goal space, it runs very fast (up to 1,000 times faster while at the same time recommending configuration solutions that are as good as or better than the prior state-of-the-art).

2 RELATED WORK

Configurable software systems come with numerous options such that users can customize the system for their varying requirements. However, once a configuration space becomes large, users can easily get confused by possible interactions among configuration options with distinct impacts on diverse objectives. For example, many software systems have poorly chosen defaults [14, 35]. Hence, it is useful to seek better configurations. Unfortunately, understanding the configuration space of software systems with large configuration spaces can be challenging [22, 38], and exploring more than just a handful of configurations is usually infeasible due to long benchmarking time [40].

When manual methods fail, automatic tools can be of assistance. In the case of configuration, those automatic tools are usually assessed with respect to the objectives of the system. Many prior works have demonstrated effectiveness in optimizing configurations for single-objective systems [13, 31]. In this paper, where we focus on systems with more than one objectives, we add one more assessment criteria: transparency. A recent 2021 report by the Gartner group\(^1\) states

> Responsible AI governance (and) transparency (our emphasis) ... is the most valuable differentiators in this market, and every listed vendor is making progress in these areas.

A model with conflicting interpretations on different objectives is considered more of opaque than transparent. Hence, we seek means to resolve those conflicting and confusing interpretations when it comes to optimizing a software system for multiple objectives.

We are concerned with transparency since, as we show in Figure 1, ML methods that chase different goals might make different recommendations. Therefore, even if each of the machine learners is interpretable, it remains possible that the internals of different machine learners are disagreeing or conflicting with each other:

---

\(^1\)http://tiny.cc/gartners21

For example, one model might offer an insight that increasing certain option \(X_1\) can optimize an objective \(G_1\), while another model believes increasing such option will harm another objective \(G_2\). In Table 1, we present some examples of such disagreement observed from dataset explored in this paper.

Looking through the literature, we can find very little on how to handle model disagreement. The closest thing we have found to our work is the Clafer visualizer [2] environment. In that Clafer, when there is a trade-off among multiple objectives in an optimization problem, the users are asked to make the decision on the trade-off. The problem with human-centered approaches such as the Clafer visualizer is that a repeated empirical result illustrates that humans are very poor oracles for what best improves a project (e.g., in Devanbu’s ICSE’16 study [10] on 500+ developers at Microsoft, even when developers work on the same project, they mostly make conflicting and/or incorrect conclusions about what factors most affect software quality; see also Shrikanth ICSE-SEIP’20 [30]). Hence we seek methods that remove as much as possible those competing recommendations.

Regarding as much as possible, sometimes objectives are inherently apposed in their recommendation direction, due to the semantics of the domain. If this effect has the majority case, then the method of this paper would be doomed to fail. That said, the novel result of this paper is that at least for the data sets studied here, the objectives are not inherently apposed. Since we can generate a disagreement-free model to provide solutions that perform as well or better as those seen produced by other methods, this result should prompt much future work exploring emergent simplicity in multi-objective reasoning in SE.

3 DETAILS, DEFINITIONS AND ALGORITHMS

This section refines the motivation of this paper by defining the configuration optimization problem and reviewing prior works associated with the problem.

Note that Table 2 shows the data sets used in our experiments.

3.1 Configurable Software System (CSS)

In general, a configurable software system (CSS) contains a set of valid configurations \(C\). Let \(c_i \in C\) represent the \(i\)th valid configuration and \(c_{i,j}\) represent the \(j\)th option in that configuration. In our case studies, the option \(c_{i,j}\) is either a numerical parameter or a Boolean value. A numerical parameter (i.e., page size) has multiple different numerical values and Boolean values are indicating a certain option as enabled or disabled. A configurable software system also contains a set of performance measures \(Y\) (i.e., response time, energy consumption, etc.), where \(y_{i,k} \in Y\) represents the \(k\)th performance value of the \(i\)th configuration in \(C\). Since we consider configurable systems with more than one performance measure, \(k\) is always greater than 1. The configuration space \(C\) is referred to as an independent variable while the performance measure space \(Y\) is referred to as a dependent variable (i.e., depends on \(C\)). Theoretically, to find an optimal configuration for a software system, we learn the relationship between \(C\) and \(Y\) by approximating the function \(f : C \mapsto Y\) that maps configurations onto the performance (objective) space by \(f(c_{i,0} \ldots c_{i,j}) = (y_{i,0} \ldots y_{i,k})\). In real-world practice, the evaluations required to approximate the function \(f\),
Figure 1: An overview of the configuration optimization problem. Prior research focused on the performance of the model, as shown in the bounded region. What has been missed is the evaluation on the inside rationales of the surrogate models. Prior model-based optimizers only assess the quality of selected solutions (with the gray block being neglected). As shown later in §3.4, models trained on different objective can often disagree on how to optimize a configuration.

Table 2: Configurable software systems explored in this paper. The abbreviations of systems are ordered by the total number of valid configurations |C|. The “B/N” in the third column indicates the number of binary options and numerical options.

| Name         | Abbr. | Domain               | #Options (B/N) | |C| | Performance Measures                                                                 | #Objectives |
|--------------|-------|----------------------|----------------|---|-----------------|----------------|---------------|
| HSQLDB       | SS-A  | SQL database         | 15/0           | 864 | run time, energy, cpu load | 3               |
| MariaDB      | SS-B  | SQL database         | 7/3            | 972 | run time, cpu load | 2               |
| wc-5d-c5     | SS-C  | streaming process system | 0/5       | 1 080 | throughput, latency | 2               |
| VP8          | SS-D  | video encoder        | 7/4            | 2 736 | run time, energy, cpu load | 3               |
| VP9          | SS-E  | video encoder        | 9/3            | 3 008 | run time, cpu load | 2               |
| rs-6d-c3     | SS-F  | streaming process system | 0/6       | 3 839 | throughput, latency | 2               |
| Izip         | SS-G  | compression tool      | 9/3            | 5 184 | run time, cpu load | 2               |
| x264         | SS-H  | video encoder        | 17/0           | 4 608 | run time, cpu load | 2               |
| MongoDB      | SS-I  | No-SQL database      | 13/2           | 6 840 | run time, cpu load | 2               |
| LLVM         | SS-J  | compiler             | 16/0           | 65 536 | run time, cpu load | 2               |
| ExaStencils  | SS-K  | Stencil code generator | 4/6          | 86 058 | run time, cpu load | 2               |

In contrast, continuous domination defines that a configuration $c_1$ is dominant over $c_2$ iff:

$$loss(c_1, c_2) < loss(c_2, c_1)$$

where the loss function is defined as:

$$loss(c_1, c_2) = \sum_{j=1}^{n} -e^{(y_{1,j} - y_{2,j})} \times \frac{1}{n}$$

where $y_{i,j}$ is min-max normalized

Note that in both equations above, the default setting is that the lower performance measure $y$ is preferred. By definitions, a best (optimal) configuration is one that is not dominated by any other configuration, denoted as a non-dominated solution. And the set containing all non-dominated solutions is called Pareto frontier set. In short, the goal in a multi-objective optimization problem is to find as many optimal (non-dominated) solutions as possible while minimizing the evaluation cost (using fewer measurements).

To compute the non-dominated solution set, the non-dominated sorting process is required, which has a runtime complexity of
The advantages of SMBO are straightforward: Given the current knowledge learned about the problem space, where should the procedure explore next? The advantages of SMBO over other traditional MOEA approaches (e.g., NSGA-II [9], SPEA2 [41], MOEA/D [39]) are:

- SMBO explores the unknown part of configuration space sequentially based on knowledge already gained from the optimization so far. This results in much fewer evaluations required to achieve the termination criteria (e.g., only 70 samples needed to explore a space of nearly 80,000 configurations, while traditional genetic algorithms require much more evaluations\(^3\)).
- SMBO contains a set of surrogate models, on which the optimization is performed. Each model is fitted for a unique objective. After the termination, the surrogate models can provide human-comprehensible insights on how to achieve better performance for different objectives.

It is undeniable that traditional multi-objective optimizers still have their values, especially when the valid search space is vast and the evaluation of solutions is inexpensive. Unfortunately, the problems explored in this paper do not fall into this category. Configurable software systems can often contain constraints among configuration options, which can reduce the valid configuration space vastly [20]. For example, the system SS-H in Table 2 has 17 binary options yet with only 4,608 valid configurations in total. That is, the ratio of valid solutions in the whole search space is 4,608/\(2^{17}\) = 3.5\%. Under such circumstance, it is believed that guidelines should be adopted to improve cost efficiency of sampling [28]. Therefore, we believe that SMBO is a more suitable approach for our problem case.

3.3 Finding Interpretable Models (with FLASH)

Many approaches have been proposed using the SMBO framework. In terms of transparency, Nair et al.’s FLASH system [27], is somewhat unique in that it offers a succinct summary of the learned model [34]. As we shall see, this directly addresses one problem (model transparency) but introduces another (model disagreement).

First proposed by Nair et al., FLASH can achieve on-par performance while overcoming the shortcomings of prior SMBO methods: Nair et al. showed that FLASH can handle models orders of magnitude faster than a prior state-of-the-art methods based on Gaussian process models [27]. Nair et al. found that FLASH takes \(10^2\) evaluations—much less than the \(10^4\) evaluations required by other optimizers. Such improvement is notable since it makes FLASH much more scalable to large-scale systems with a vast search space. For example, the data used to certify this paper required 6 calendar months to collect (running on a multi-core CPU farm). While that data is necessary to certify a new algorithm (like VEER), once that algorithm is fielded, it needs to respect the practical difficulties associated with data collection.

While Gaussian process models (GPM) are often used [12] in SMBO, Nair et al. found that, for optimizing configurable software, GPM scales very poorly to larger dimensional data [27]. Nair et al. found that a faster, and the more scalable, system can be implemented using regression trees. In FLASH, each objective is modeled as a separate Classification and Regression Tree (CART) model. They found that even if regression trees are somewhat incorrect about their predictions, those approximate predictions can still be used to rank different candidate configurations [26]. FLASH is implemented following the general SMBO framework as described in Algorithm 1, where the surrogate models \(M\) are CART learners.

There are several reasons we chose FLASH:

- Nair et al. showed that FLASH can handle models orders of magnitude faster than a prior state-of-the-art methods based on Gaussian process models [42].

---

\(^3\)Holland’s advice [16] for genetic algorithms (such as NSGA-II and MOEA/D) is that 100 individuals need to be evolved over 100 generations, i.e., \(10^4\) evaluations in all.

---

**Algorithm 1: SMBO (e.g., FLASH)**

```
Data: \(C\) contains all candidate samples; performance function \(f\)

\(\text{maps configurations to the corresponding performance values; initialized surrogate models } M \text{ contain multiple models, one per objective; } S\) denote the stopping criterion; \(S\) denotes the non-dominated solutions found so far.

Result: Optimized models \(M\), evaluated configurations \(C_{\text{train}}\)

\textbf{begin}
\begin{itemize}
  \item \(C_{\text{train}} \leftarrow \text{Random}(C)\) // initialize training samples
  \item \(C_{\text{train}} \leftarrow \text{SelectConfig}(M, f, C)\)
  \item \(M \leftarrow \text{FitModel}(C_{\text{train}}, f)\)
  \item \(S \leftarrow \text{NDSorting}(C_{\text{train}})\) // as defined in Eq. 1 or Eq. 2
\end{itemize}

\textbf{while} \(\text{budget} \geq 0\) \textbf{do}
\begin{itemize}
  \item \(C_{\text{new}} \leftarrow \text{SelectConfig}(M, f, C)\)
  \item \(M \leftarrow \text{FitModel}(C_{\text{new}}, f)\)
  \item \(S \leftarrow \text{NDSorting}(C_{\text{train}})\) // as defined in Eq. 1 or Eq. 2
\end{itemize}

\textbf{if} \(S\)\textit{isUpdated} \textbf{then}
\begin{itemize}
  \item \(\text{continue}\)
\end{itemize}
\textbf{else}
\begin{itemize}
  \item \(\text{budget} \leftarrow \text{budget} - 1\)
  \item \(i + 1\)
\end{itemize}
\textbf{return} \(M, C_{\text{train}}\)
```
FLASH makes its conclusions after very few samples to the domain (just a few dozen).
- Due to the small size of the sample space, then the decisions used by FLASH generate very small models (one per objective). Hence, FLASH can produce the human-readable models needed for the AI transparency issues discussed in §2

3.4 Model Disagreement Problem and FLASH
For our purposes, FLASH is both a success and a failure. Firstly, it fixed the scalability issues of GPM. At the same time, it turns out that model disagreement is rampant in the models generated by FLASH. For example, all the examples of disagree in Table 1 were generated by FLASH.

To understand the root cause of such model disagreement, we attempted to visualize the inside rationales of the two surrogate models. As shown in Fig. 2, the “optimal” solutions selected by the two models are rather different, which is totally reasonable and expected, given that they are optimizing for different objectives. However, it is noteworthy that the final “optimal” solutions yielded by FLASH (which is selected by the non-dominated sorting procedure) share no similarity with either of the two solution sets. That is to say, the interpretations generated by the two models alone are not “final”, and if users merely rely on such interpretations to locate optimal configurations, they are more likely to obtain sub-optimal solutions that are distant to the ones yielded by FLASH. Since the non-dominated sorting procedure is a non-parametric process from which we cannot extract interpretations, we need an additional model that can mimic the performance of the sorting procedure meanwhile allowing us to obtain comprehensible insights.

4 VEER: A CONFUSION-FREE MULTI-OBJECTIVE OPTIMIZER
As a response to our insights in §3.4, we implement VEER based on the following design choices:

- To ensure our model provides final interpretations, we design a heuristic to reduce the multi-dimension objective space into a single-dimensional space. This will enable us to provide interpretations that take into account the overall performance across multiple objectives.
- To ensure our interpretation is confusion-free, we use one single-output model as the new surrogate model. This way, we avoid the dilemma that the same candidate solution (configuration) gets ranked differently by different learners (or different outputs from one multi-output model).
- To conduct a fair comparison with FLASH, and to obtain rule-based interpretations, we choose to use CART as the new surrogate model to optimize on the synthetic single-dimension space. In future deployment, VEER is applicable to any interpretable models such as Linear Regression and Naive Bayes.

An overview of the VEER framework is shown in Fig. 3 (which also includes the experimental rig used in this paper). The component ZIGZAG, as illustrated in Fig. 4, is the core heuristic used in VEER to generate the single-dimension objective space, and its implementation varies for different definitions of domination. In this paper, we choose to use the heuristic implemented with continuous domination because we believe it can better reflect and preserve the domination relationship among solutions in our setting: As one can observe from the two examples in Fig. 4, when using binary domination, point $c$ is assigned a higher rank than point $e$. However, if we only look at these two solutions, neither of them can dominate each other, thus, assigning them different ranks seems less reasonable. Such tricky situations can be avoided in the continuous domination scenario because points are ranked precisely according to their distance toward the “heaven” point (where both objectives are optimized).

VEER uses CART as the final surrogate model motivated by effectiveness and interpretability considerations. As previously shown by FLASH, an optimizer model built with CART can achieve comparable and sometimes superior performance in multi-objective optimization as compared to GPM. One reason that CART scales better than GPM in data of larger dimensionality is that CART does not presume the “smoothness” of the data space: models like GPM make an assumption to the data space that configurations closer to each other have similar performance. Such assumption can usually be invalid because a seemingly small change in configuration options might actually represent a crucial shift in configuration strategy (i.e., the choice of data structure often has a substantial impact on the performance of a storage-oriented system). Our second consideration is interpretability. Gigerenzer pointed out that tree-structured models express great rationality and interpretability, making it easier for users to obtain actionable insights about the data [11]. A recent survey about practitioners’ beliefs about visual explanations of defect prediction models also shows that CART is favoured by practitioners for its interpretability [19]. The result of the survey reported that among 6 methods of offering visual explanations, CART is ranked as the 1st tier in terms of insightfulness and quality of the generated visual explanations.

![Figure 2: Visualization of the model disagreement problem in FLASH [27]. The internal rationales of the surrogate models are as presented in Table. 1. ND sort= “non-dominated” sort. The x and y axis represent the two performance objectives (min-max scaled) in the dataset SS-E in Table 2. The star at the bottom-left corner indicates the ideal optimum.](image-url)
Figure 4: ZIGZAG: candidate configurations are ranked according to their ability of dominating other configurations across the configuration space. Starting at the best objective (bottom left, which is ranked #0), VEER zigzags around objective space looking for the next best unvisited objective. Configurations that cannot dominate or be dominated by each other are assigned the same rank.

5 RESEARCH QUESTIONS

This section will illustrate our research questions and our strategy to address these questions.

To systematically evaluate its merits, we compared VEER with one of the most recent state-of-the-art configuration optimizer, FLASH. Moreover, to verify whether the model disagreement problem can be tamed via a simple and naive approach, we also implemented two variants of FLASH to serve as the baseline.

Our research questions are geared towards assessing the performance of VEER regarding 3 aspects: (a) effectiveness of configuration solutions generated from the VEER model, (b) interpretability of the VEER model, and the (c) execution time of the VEER model. More specifically, we ask the following research questions:

RQ1: Does there exist a model disagreement problem for multi-objective optimization of CSS?

Before doing anything else, we need to first motivate the investigation of this paper. In this research question, we ask whether the standard method used in the multi-objective configuration problem results in conflicting suggestions on how to best improve different objectives. To measure the level of such conflicts, we used a rank correlation measurement called Kendall’s $\tau$ test to assess whether learners trained on different objectives rank the candidate configurations in different orders.

RQ2: Is the disagreement problem “linearly solvable”?

All the technology proposed in this paper is superfluous if an existing alternate method can handle the problem of interpreting multi-objective optimization results. Therefore, we attempt to replace FLASH with two variants of SMBO methods, using (a) the weighted sum equation and one single-output regression learner, or (b) a multi-output regression learner. The two benchmark methods are referred to as SingleWeight and MultiOut respectively. Our experiments will show that both methods have significant shortcomings.

RQ3: Can VEER resolve the model disagreement problem while maintaining on-par performance with benchmark methods?

Here, we test whether VEER can obtain on-par performance compared to the state-of-the-art algorithms. To measure that, we evaluate not only the quality of the returned solutions, but also the robustness of the model. Measurements used to evaluate the merits of VEER and other benchmark methods are elaborated in §6.3

RQ4: Can VEER reduce the execution time?

We explore other positive side effects brought by VEER. One of the most apparent improvements of VEER is that since multiple learners within the optimizer are replaced by a single learner, the

Figure 3: An overview of the VEER framework, and the evaluation rig used in our experimentation.
VEER: A Fast and Disagreement-Free Multi-objective Configuration Optimizer

6 EXPERIMENTAL RIG

To answer our research questions, our experiment compares the performance of VEER against other benchmark methods.

As depicted in Fig. 3, all the configuration optimizers in our experiment will divide the configuration space into 2 sets: the training pool, and the holdout set (we split them 50% to 50%). Each optimizer sequentially samples along the training pool to add the selected next most informative configuration item into the training set (and this set is a subset of the training pool). The selected configurations and the corresponding performance measures are then used to train the machine learners within the optimizer. Then, the performance of each optimizer is evaluated using the holdout set. Apart from that, VEER has one additional step, which is to use the ZIGZAG process as illustrated in Fig. 4 to train a hyper-space model using random samples chosen from the rest of the training pool. Note that, since this step does not require to access the actual performance measures Y of the additionally chosen samples C, it will not increase the measurement cost in the real-world application. Finally, to assess the stability and reliability of our approach in a statistical manner, our experimentation will randomly choose 50% of the configuration space as the holdout set. To reduce the effect of the random seeds, the whole process is repeated for 100 times.

6.1 Data

To empirically evaluate the effectiveness of our approach, we use datasets collected from different configurable software systems. Each dataset contains the whole population of all valid configurations of that system and the performance measures of each configuration (by “all”, we mean all combinations given the selected configuration options). Table 2 describes the nature of each dataset. We selected the datasets based on the following criteria: (1) different sizes to examine the scalability and robustness, (2) different domains to improve external validity, and (3) different application domains (client-server and desktop) to cover different performance aspects (i.e., run time of compressing a video vs. run time to perform a set of actions on a database). Among all the datasets, SS-C and SS-F are datasets used in FLASH. Others (SS-A, SS-B, SS-D, SS-E, and SS-G to SS-K) are datasets recently collected by us. For datasets that are collected by us, we applied the following process: To reduce measurement noise, we executed all measurements in isolation (i.e., no other tasks are performed) on machines with minimal Debian 9 or Debian 10 installations and repeated each measurement 3–5 times. While the standard deviation of a performance measure exceeded 10%, we repeated the measurement of the configuration.

6.2 Baselines

To assess our approach comprehensively, we included several SMBO methods as benchmark methods in our experiment. At first, we collected some existing open-source SMBO methods, such as FLASH [27], HyperOpt [4] (using TPE [3] as surrogate models) and SMAC [18]. Unfortunately, we found that both HyperOpt and SMAC do not support customized constraints on the search space. As illustrated in §3.2, such constraints are rather crucial as they filter out over 99% of invalid configurations [20]. Hence, we had to implement our own versions of the SingleWeight and MultiOut algorithms described in §5. The implementation follows the SMBO framework as described in prior works [3, 5, 27], denoted as SingleWeight and MultiOut. The major difference among these benchmark methods is the output format of surrogate models (either single-output or multi-output) and option of using either weighted sum or non-dominated sorting to select the final solutions. The implementation of all benchmark methods is available in our online repository.

6.3 Performance Criteria

First of all, to assess effectiveness of our approach against benchmark models, we choose to measure the quality of the solutions set returned by each model. A solution set contains configurations that a model believes to have optimal performance among all configurations. Given the definitions of binary and continuous domination, a solution set can contain more than one configuration. To measure the quality of the solution set, this paper uses generational distance [36] (GD) as the indicator. GD computes the average distance between the solution set returned by a model and the actual optimal solution set. There are other indicator such as inverted generational distance [8] (IGD) and hypervolume [17] (HV). However, prior research suggests GD as a more suitable metric to uniformly reflect the overall quality of the solution set [1].

Secondly, to assess the interpretability of a model, in this paper we are specifically assessing the level of disagreement among multiple learners within a multi-objective model. We argue that if interpretations extracted from different learners are conflicting or disagreeing with each other, the model will fail to provide stakeholders with unequivocal insights that are truly informatively or actionable. In that spirit, we use a rank correlation test, Kendall’s τ test [21], to measure the extent of disagreements among learners built on different objectives. Kendall’s τ test is a non-parametric statistical test which can be used to measure the ordinal association between 2 lists of measured variables (in this paper, 2 objective values on the same configurations). According to the definition from Kendall correlation, we first categorize any pair of configurations by their performance measure into 2 kinds: discordant pairs and concordant pairs. Let (Aᵢ, Bᵢ) and (Aⱼ, Bⱼ) denote a pair of configurations, represented by the performance measures in objective A and objective B. This pair is discordant if the sorted order of (Aᵢ, Aⱼ) and (Bᵢ, Bⱼ) agrees: one configuration has better performance than the other in both objectives. Otherwise, the pair is discordant. After that we compute the Kendall coefficient τ using the following equation:

\[
τ = \frac{(P - Q)}{(P + Q)}
\]

where P is the number of concordant pairs, Q the number of discordant pairs. In general, the Kendall correlation is high when the 2

\footnote{And the source code for that implementation can be found in the reproduction package mentioned in our abstract}

\footnote{We define a concordant pair in tasks of more than 2 objectives in a similar manner: one configuration has better performance than the other in all objectives. This is not originally defined by Kendall, but we believe it is a proper extension.}
variables are ranked similarly, and the correlation is low when the 2 variables are ranked differently. More specifically in our case study, a positive $\tau$ coefficient means a relatively similar ranking among different objectives, which indicates less disagreement among different learners in a multi-objective model; A $\tau$ coefficient near 0 means the 2 lists of ranks assigned by different learners have no correlation at all; A negative coefficient means the learners rank configurations in somehow opposite order.

Finally, to assess computational complexity of our approach, we measure the execution time of applying each model on the holdout data to generate a solution set. All the above analyses and measurements were executed on a 64-bit Windows 10 machine with a 2.2 GHz 4-core Intel Core i5 processor and 8 GB of RAM.

6.4 Statistical Analysis

To make comparisons among all algorithms on a single project, we use a non-parametric significance test and a non-parametric effect size test. Specifically, we use the Scott-Knott test [25] that sorts the list of treatments (in this paper, VEER and baselines) by their median scores. After the sorting, it then splits the list into two sub-lists. The objective for such a split is to maximize the expected value of differences $E(\Delta)$ in the observed performances before and after division [37]:

$$E(\Delta) = \left| \frac{|l_1|}{|l|} \right| \text{abs}(E(l_1) - E(l))^2 + \left| \frac{|l_2|}{|l|} \right| \text{abs}(E(l_2) - E(l))^2$$

where $|l_1|$ means the size of list $l_1$. The Scott-Knott test assigns ranks to each result set; the higher the rank, the better the result. Two results will be ranked the same if the difference between the distributions is not significant. In this expression, Cliff’s Delta estimates the probability that a value in list $A$ is greater than a value in list $B$, minus the reverse probability [24]. A division passes this hypothesis test if it is not a “small” effect ($\Delta \geq 0.147$). This hypothesis test and its effect size are supported by Hess and Kromery [15].

7 RESULTS

7.1 RQ1

Does there exist a model disagreement problem for multi-objective optimization of CSS?

Table 3 checks for the existence of this problem by showing the rank correlation of FLASH, which is measured by Kendall’s $\tau$ coefficient.

As shown in Table 3, there are a few systems where the rank correlation is relatively high (SS-I and SS-K). This could be because the objectives in those systems are not so conflicting. In such cases, feature interactions learned from different learners are likely to be homogeneous since essentially these learners can substitute each other without harming the model performance. On the other hand, for most of the case studies in this paper, we do observe that configurations are ranked in a rather opposite way by learners trained on different objectives.

In summary, we answer RQ1 as follows:

Table 3: RQ1 and RQ2 result: Median values of the Kendall’s $\tau$ coefficient. Higher coefficients are better. In each dataset, the lowest score(s) are highlighted.

|         | FLASH | SingleWeight | MultiOut | VEER |
|---------|-------|--------------|----------|------|
| SS-A    | 0.47  | 1.00         | 0.82     | 1.00 |
| SS-B    | 0.16  | 1.00         | 0.13     | 1.00 |
| SS-C    | 0.60  | 1.00         | 0.65     | 1.00 |
| SS-D    | -0.17 | 1.00         | -0.07    | 1.00 |
| SS-E    | -0.56 | 1.00         | -0.54    | 1.00 |
| SS-F    | -0.30 | 1.00         | -0.30    | 1.00 |
| SS-G    | -0.47 | 1.00         | -0.51    | 1.00 |
| SS-H    | -0.69 | 1.00         | -0.69    | 1.00 |
| SS-I    | 0.73  | 1.00         | 0.73     | 1.00 |
| SS-J    | -0.08 | 1.00         | -0.17    | 1.00 |
| SS-K    | 0.87  | 1.00         | 0.88     | 1.00 |

Answer 1: In our case studies, we can assert that multi-objective configurable software systems often have extensive interpretability problems, in terms of model disagreement among multiple learners.

7.2 RQ2

Is the disagreement problem “linearly solvable”?

This section compares the results of FLASH (which works on each objective separately) to MultiOut and SingleWeight (which work on some linear combinations of the objectives).

For MultiOut, we show in Table 3 that we can increase the rank correlation by replacing the multiple single-output CART learners with a single multi-output CART learner. This could be a good sign implying that in some cases FLASH can be easily improved in the model disagreement problem via making a small mutation on its original implementation. However, this kind of improvement does have its upper bound, given the disagreement among different learners still exists pervasively among all datasets.

As for SingleWeight, because we beforehand transformed multi-objective space into a single objective via a weighted sum function, the optimizer now requires one surrogate model. This totally resolved the disagreement problem by reducing the arity of output. However, as later reported in Table 4, this approach can sometimes compromise the performance significantly. We conjecture that it could be because the optimal solutions are not uniformly distributed or the optimal solutions reside in a non-convex region which leads the weighted sum approach to fail.

In summary, we answer RQ2 as follows:

Answer 2: Neither SingleWeight nor MultiOut can fix the disagreement issue while not risking to comprise the performance. That is to say, this problem is not ”linearly solvable”.

7.3 RQ3

Can VEER resolve the model disagreement problem while maintaining on-par performance with benchmark methods?
First of all, we need to clarify that we will not use the rank correlation as the major indicator to evaluate the merit of our approach. The reason is, when the model used to generate the solution set is a single-output learner, there is naturally no disagreement at all, which always guarantees a perfect correlation (Kendall’s \( \tau \) coefficient = 1). This result is totally to be expected by us since VEER is designed purposefully to resolve the disagreement problem.

Therefore, we need to evaluate whether VEER compromises the performance as compared to FLASH. In Table 4, we report the generational distance (GD) of solution sets provided by each optimizer. We used a non-parametric effect size test to determine if the difference between the two performance measures is statistically significant. As shown by the table, in most cases VEER can achieve comparable performance. That is, after combining knowledge from multiple learners into one learner, VEER has experienced no (or trivial) information loss. Additionally, as Fig. 5 shows, VEER maintains the same level of robustness compared to other benchmark methods as indicated by the variance of the distributions.

In summary, we answer RQ3 as follows:

Table 4: RQ3 result: Median values of generational distance (GD) for all 4 methods. Each row highlights the GD value(s) that are statistically significantly worst by more than a small effect size (as determined by statistical tests of §6.4).

|        | FLASH | SingleWeight | MultiOut | VEER |
|--------|-------|--------------|----------|------|
| SS-A   | 0.007 | 0.011        | 0.014    | 0.013|
| SS-B   | 0.125 | 0.171        | 0.100    | 0.119|
| SS-C   | 0.020 | 0.021        | 0.019    | 0.022|
| SS-D   | 0.060 | 0.059        | 0.061    | 0.058|
| SS-E   | 0.131 | 0.135        | 0.144    | 0.136|
| SS-F   | 0.098 | 0.082        | 0.113    | 0.113|
| SS-G   | 0.014 | 0.296        | 0.016    | 0.014|
| SS-H   | 0.013 | 0.013        | 0.016    | 0.013|
| SS-I   | 0.034 | 0.038        | 0.034    | 0.042|
| SS-J   | 0.386 | 0.437        | 0.384    | 0.362|
| SS-K   | 0.299 | 0.306        | 0.304    | 0.298|

In summary, we answer RQ4 as follows:

Figure 6: RQ4 result: The inverse runtime ratio using FLASH’s runtime as the benchmark, calculated by dividing the average runtime of FLASH over that of other methods. Higher ratio is better.

Answer 3: The design choices made for VEER are capable to resolve the model disagreement problem. Moreover, it does not compromise the performance of the original optimization model (FLASH, in this case).

7.4 RQ4

Can VEER reduce the execution time?

While executing VEER during the experimentation, one of the most obvious bonuses is that VEER runs much faster than prior methods when applied to the holdout set. As shown in Fig 6, the average execution time of VEER when applied to the holdout set is much faster than that of other benchmark methods. The fitted VEER model can be 1000+ times faster than other benchmark methods when applied on the largest dataset. Note that we also observe a larger variance in runtime when the size of datasets increases. This is because the runtime is also hugely influenced by the size of the returned solutions set: when there are more non-dominated solutions in the holdout set, the time till the termination of the non-domination sort will increase proportionally. VEER does not...
suffer from such complexity given its design of “compressing” multi-objective space into a single dimension.
In summary, we answer RQ4 as follows:

**Answer 4**: As shown in the experiment result, VEER takes a much shorter time to generate the solution set out of the holdout set than other methods. Moreover, as the size of the holdout set increases, the execution time of VEER grows much slower than that of other methods, which indicates better scalability of VEER.

8 DISCUSSION
In this section we discuss what makes VEER novel and more useful than prior methods in optimizing multi-objective configuration.

**Final interpretations**: In the domain of multi-objective optimization, final solutions are yielded by processing non-dominated sorting first. However, this part is non-parametric and cannot be used to extract interpretation. Therefore, traditional SMBO optimizers are only capable of providing preliminary interpretations from single-objective models. VEER, on contrary, provides a final interpretation about how to optimize a configuration in general.

**Uncompromising**: Our approach cannot be generalizable to all configurable software systems: when two objectives are competing (with strong negative correlation), VEER is doomed to fail since a configuration that optimizes one objective will inevitably compromise the other one by the same extent. However, we show that this is not the case in datasets explored so far. In fact, VEER can always generate disagreement-free interpretations without compromising the performance.

**Fast**: Non-dominated sorting is a computation-intensive step with an optimizer. VEER replaces it with the hyper-space model that mimics the behavior of the non-dominating sorting procedure. By directly learning the relationship between the non-dominated ranks and the independent variables (configuration options), VEER can achieve significantly (up to $10^3$ in the largest system) faster execution time.

**Adjustable**: In addition to the merits above, VEER can also incorporate with the preference-based approach by simply adjusting the distance calculation function insides ZIGZAG. As shown in Fig 7, by customizing the preference weights (e.g., $[2 : 1]$ means optimizing one objective is twice as important as optimizing the other one), VEER can generate solutions that reflect such user preferences.

9 THREATS TO VALIDITY
Given the complexity of our experiments in 11 real-world configurable software systems, many factors can threaten the validity of our results.

**Internal Validity**: First of all, the multi-objective optimizer learns from benchmark measurements that we collected from various configurable systems. While we have rejected measurements with relatively high variance, it remains possible that some measurements are incorrect, which can bias the learning procedure of the optimizer and may result in worse solution sets.

Secondly, we measure the level of model disagreement using a rank correlation test, namely Kendall’s \( r \) test. One shortcoming of this test is that it has to be performed on ranks assigned to the same set of variables, which forces us to compute the correlation on the whole population (since optimal solutions defined by different objectives are hardly the same). As is well-known, it is actually the optimal (non-dominated) solution set that an optimizer cares about, and the very majority of the whole population are only sub-optimal solutions. Therefore, it could be a possible case that there is actually little disagreement within the optimal solutions but much disagreement among the sub-optimal space.

**External Validity**: While we believe our findings is generalizable as supported by the experimentation result, this does not guarantee the model to be automatically scalable to systems with larger spaces and dimensionalities. However, to increase the external validity of our study, we did intentionally choose datasets from various sizes, dimensions, and application domains.

Secondly, we select CART as the embedded surrogate model because prior research has shown that CART is capable to achieve good performance and CART has great interpretability with feature interactions being presented as decision conditions within the tree [27]. Another consideration is that since we use FLASH, one of the most recent state-of-the-art optimization models, as the benchmark method, we would prefer to control variables within the experiment so that our comparison is sufficiently “fair”. However, it is possible that other white-box machine learners (e.g., logistic regression, Naive Bayes) can achieve better and more generalizable performance than CART while we have not explored yet.

10 CONCLUSION
We have proposed a confusion-free multi-objective configuration optimizer, VEER, which is built on top of a state-of-the-art sequential model-based optimizer FLASH. We have shown that VEER has not only inherited many merits of FLASH (good performance and
VEER: A Fast and Disagreement-Free Configuration Optimizer

ACKNOWLEDGEMENTS

This work was partially funded by a research grant from the Laboratory for Analytical Sciences, North Carolina State University.

REFERENCES

[1] Amritanshu Agrawal, Tim Menzies, Leandro L. Minku, Markus Wagner, and Zhe Yu. 2020. Better software analytics via “TXU”: Data mining algorithms using-used-by optimizers. Empirical Software Engineering 25, 3, (2020), 2099–2136.
[2] Michal Artukiewicz, Kasper Bäk, Alexandru Murașkin, Rafael Olabeaga, Jia Hui Liang, and Krzysztof Czarnecki. 2013. Clider tools for product line engineering. In Proceedings of the 17th international software product line conference co-located workshops. 130–135.
[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyperparameter optimization. Advances in neural information processing systems 24 (2011).
[4] James Bergstra, Dan Yamins, David D Cox, et al. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in science conference. Vol. 13. Citeseer, 20.
[5] Di Chen, Wei Fu, Rahul Krishna, and Tim Menzies. 2018. Applications of psychological science in software engineering. In Proceedings of the 2018 17th Joint Meeting on Foundations of Software Engineering. 456–467.
[6] Di Chen, Wei Fu, Rahul Krishna, and Tim Menzies. 2018. Sampling as a Baseline Optimizer for Search-based Software Engineering. IEEE Transactions on Software Engineering (pre-print) (2018), 1–1.
[7] J. Chen, V. Nair, R. Krishna, and T. Menzies. 2018. “Sampling” as a Baseline Optimizer for Search-based Software Engineering. IEEE Transactions on Software Engineering (pre-print) (2018), 1–1.
[8] Carlos A. Coello Coello and Margarita Reyes Sierra. 2004. A study of the parallelization of a coevolutionary multi objective evolutionary algorithm. In Mexican international conference on artificial intelligence. Springer, 688–697.
[9] Kalyanmoy Deb, Amit Pratap, Sameer Agarwal, and TAMSIT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation 6, 2 (2002), 182–197.
[10] Premkumar Devanbu, Thomas Zimmermann, and Christian Bird. 2016. Belief & evidence in empirical software engineering. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE, 108–119.
[11] Gerd Gigerenzer. 2009. Why heuristics work. Perspectives on psychological science 3, 1 (2008), 20–29.
[12] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google zipper: A service for black-box optimization. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 1487–1495.
[13] Juamei Guo, Dingyu Yang, Norbert Siegmond, Sven Apel, Aftish Sarkar, Pavel Valov, Krzysztof Czarnecki, Andzej Wawrzoniak, and Haoping Yu. 2018. Data-Efficient Performance Learning for Configurable Systems. Empirical Software Engineering (EMSE) 23, 1 (2018), 1826–1867.
[14] Hidetomo Herodotou, Harold Lim, Gang Lu, Nedyalko Borisov, Liang Dong, Fatma Bägen Cetin, and Shivnath Babu. 2011. Starfish: A Self-tuning System for Big Data Analytics. In Conference on Innovative Data Systems Research.
[15] Melissa R Hess and Jeffrey D Kromrey. 2004. Robust confidence intervals for effect sizes: A comparative study of Cohen’s d and Cliff’s delta under normality and heterogeneous variances. In Annual meeting of the American Educational Research Association. 1–30.
[16] John H Holland. 1992. Genetic algorithms. Scientific american 267, 1 (1992), 66–73.
[17] Simon Hrabuš, Mark Hingtgen, Lyndon Whale, and Laszlo Barone. 2003. An evolution strategy with probabilistic mutation for multi-objective optimisation. In The 20th Congress on Evolutionary Computation, 2003. CEC’03., Vol. 4. IEEE, 2284–2291.
[18] F. Hutter, H. H. Hoos, and K. Leyton-Brown. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In IJCB JCN.
[19] Jarayus Jarapakde, Chadkrit Tanthamuthavorn, and John Grandy. 2021. Practitioners’ Perceptions of the Goals and Visual Explanations of Defect Prediction Models. arXiv preprint arXiv:2102.12007 (2021).
[20] Christian Kaltenecker, Alexander Gre bowel, Norbert Siegmond, and Sven Apel. 2020. The interplay of sampling and machine learning for software performance prediction. IEEE Software 4 (2020), 58–66.
[21] Maurice George Kendall. 1948. Rank correlation methods. (1948).
[22] Sergiy Kolesnikov, Norbert Siegmond, Christian Kästner, Alexander Grebnow, and Sven Apel. 2019. Tradeoffs in modeling performance of highly configurable software systems. Software & Systems Modeling 18, 3 (2019), 2265–2283.
[23] M. Lasannan, L. Thiele, K. Deb, and E. Zitzler. 2002. Combining convergence and diversity in evolutionary multiobjective optimization. Evolutionary computation (2002).
[24] Guillermo Macheth, Eugenia Ruzanmyskyj, and Rubén Daniel Ledesma. 2011. Cliff’s Delta Calculator: A non-parametric effect size program for two groups of observation. Universität Psychologica 10, 2 (2011), 545–555.
[25] Nikolas Mitas and Letheris Angelis. 2012. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Transactions on software engineering 39, 4 (2012), 537–551.
[26] Vivek Nair, Tim Menzies, Norbert Siegmond, and Sven Apel. 2017. Using bad learners to find good configurations. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 257–267.
[27] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmond, and Sven Apel. 2018. Finding faster configurations using Flash. IEEE Transactions on Software Engineering 44, 7 (2018), 914–911.
[28] Atri Sarkar, Juamei Guo, Norbert Siegmond, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-efficient sampling for performance prediction of configurable systems (t). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 342–352.
[29] Robert Sawyer. 2011. BI’s Impact on Analyses and Decision Making Depends on the Development of Less Complex Applications. IJBIR 2 (07), 32–63. https://doi.org/10.46038/IJBIR.201107094
[30] NC Shrikanth and Tim Menzies. 2020. Assessing practitioner beliefs about software defect prediction. In 2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEP). IEEE, 182–190.
[31] Norbert Siegmond, Alexander Grebnow, Sven Apel, and Christian Kästner. 2015. Performance-Influencer Models for Highly Configurable Systems. In Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE). ACM, 284–294.
[32] J. Snook, H. Larochelle, and R. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In NIPS - Volume 2.
[33] Shiang-Tien Tan and Taianan Chan. 2016. Defining and conceptualizing actionable insight: a conceptual framework for decision-centric analytics. arXiv preprint arXiv:1606.03510 (2016).
[34] Vivek Nair, George Papadimitriou, Marian Kran, Cong Wang, Anubh Madan, Ewa Doremi, and Tim Menzies. 2021. Mining Workflows for Anomalous Data Transfers. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) 1–12. https://doi.org/10.1109/MSR52588.2021.00013
[35] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic Data-base Management System Tuning Through Large-scale Machine Learning. In International Conference on Management of Data. ACM.
[36] David A Van Veldhuizen. 1999. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical Report. AR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING.
[37] Tianpei Xia, Rahul Krishna, Jasupeng Chen, George Mathew, Xiping Shen, and Tim Menzies. 2018. Hyperparameter optimization for effort estimation. arXiv preprint arXiv:1805.00336 (2018).
[38] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanzuan Zhou, Shankar Paukula, and Runpai Tawalderi 2015. Hey, you have given me too many knobs! understanding and dealing with over-designed configuration in system software. In Foundations of Software Engineering.
[39] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on evolutionary computation 11, 6 (2007), 712–731.
[40] Han Zhu, Junqi Jin, Chang Tan, Fei Fan, Yifan Zeng, Han Li, and Kun Gai. 2017. Optimized Cost per Click in Taobao Display Advertising. In arXiv preprint (2017).
[41] Eckart Zitzler,Marco Loymanns, andLothar Thiele. 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. TKK-report 103 (2001).
[42] Marcela Zuluaga, Andreas Krause, and Markus Pischel. 2016. ε-pal: an active learning approach to the multi-objective optimization problem. The Journal of Machine Learning Research 17, 1 (2016), 3649–3660.