Multi-Objective Configuration Sampling for Performance Ranking in Configurable Systems

Yongfeng Gu, Yuntianyi Chen, Xiangyang Jia, Jifeng Xuan*
School of Computer Science, Wuhan University, China
{yongfenggu, yuntianyichen, jxy, jxuan}@whu.edu.cn

***Abstract*—The problem of performance ranking in configurable systems is to find the optimal (near-optimal) configurations with the best performance. This problem is challenging due to the large search space of potential configurations and the cost of manually examining configurations. Existing methods, such as the rank-based method, use a progressive strategy to sample configurations to reduce the cost of examining configurations. This sampling strategy is guided by frequent and random trials and may fail in balancing the number of samples and the ranking difference (i.e., the minimum of actual ranks in the predicted ranking). In this paper, we proposed a sampling method, namely MoConfig, which uses multi-objective optimization to minimize the number of samples and the ranking difference. Each solution in MoConfig is a sampling set of configurations and can be directly used as the input of existing methods of performance ranking. We conducted experiments on 20 datasets from real-world configurable systems. Experimental results demonstrate that MoConfig can sample fewer configurations and rank better than the existing rank-based method. We also compared the results by four algorithms of multi-objective optimization and found that NSGA-II performs well. Our proposed method can be used to improve the ranking difference and reduce the number of samples in building predictive models of performance ranking.**

***Index Terms*—configuration sampling, multi-objective optimization, configurable systems, performance ranking**

I. Introduction

Highly configurable systems often provide tunable configuration options for users to achieve specific functional or non-functional properties. A *configuration option* is a customized parameter or tunable feature in a configurable system. Among non-functional properties, performance is viewed as one of the most crucial properties [1]–[4]. The definition of performance is diversified, e.g., the maximum response rate in a web server, like Apache HTTP Server or the encoding time in a video stream encoder, like x264 [1]. In a configurable system, each configuration option is assigned to a parameter value; then the value assignment of all configuration options is called a *configuration*. However, finding the optimal (or near-optimal) configuration is challenging. On the one hand, the increasing number of options in configurable systems results in a large number of configurations, which make picking out "good" configurations difficult [5], [6]. A default configuration of a configurable system may result in low performance; users have to try many times to find a better configuration than the default one [7]. On the other hand, deploying and measuring a configuration system is time-consuming, e.g., rebooting a server in a long time [8]; this adds a huge cost to the examination of configurations.

To cope with the unknown performance of configurable systems, researchers attempted to build learning models by predicting the performance of each configuration [2]–[4], [9]. However, building such learning models requires a large number of known configurations, i.e., a large training set. It is expensive in constructing this training set due to the cost of examining the performance of configurations.

Recently, to reduce the cost of known configurations, Nair et al. [10] proposed a rank-based method, which ranks "good" configurations to the top with a sampling strategy, called *progressive sampling*. Their experiments demonstrated that the rank-based method outperforms other predictive models in ranking configurations. In their work, a dataset is divided into a training pool, a validation pool, and a test pool. The rank-based method iteratively samples configurations from the training pool to learn a predictive model and evaluates the model with the validation pool. The validation pool is used to measure the sampling process, which iterates until the validation result reaches a pre-defined criterion. The test pool is used to evaluate the predictive model. Therefore, in this sampling method, sampled configurations from the training pool and all configurations from the validation pool have to be examined to obtain the performance. Compared with non-sampling methods, this sampling method saves the measurement cost of configurations. This sampling strategy is also applied in other methods, such as Flash [8] and AutoConfig [11].

We followed Nair et al. [10] and defined the task of finding "good" configurations in a configurable system as the *performance ranking* problem. We chose the work by Nair et al. [10] since it is recognized as the state-of-the-art method [4], [8], [11]. The goal of the performance ranking problem is to recommend the optimal (or near-optimal) configurations to users. We refer to the number of sampled configurations as the *measurement cost*. The rank-based method by Nair et al. [10] is limited by its randomness and uncertainty. In the rank-based method, the newly sampled configurations are guided by frequent and random trials; meanwhile, the sampling strategy in the rank-based method fails in balancing the measurement cost and the ranking difference (i.e., the minimum of actual ranks in the predicted list in this paper).

In this paper, we proposed a **M**ulti-**o**bjective **Config**uration

*Corresponding author

sampling method (**MoConfig**) to select configurations. MoConfig is designed to optimize the performance ranking problem for highly configurable systems. In our method, we aim at balancing two objectives: the measurement cost and the ranking difference. However, since the performance of configurations is unknown before sampling, it is infeasible to directly calculate the ranking difference of sampled configurations. Instead, we defined three alternative objectives (Detailed definitions are given in Section III-C): *entropy* – the entropy of configuration options in sampled configurations, *variance* – the variance of distances between sampled configurations and the average of configurations, and *density* – the density of configuration options in sampled configurations. Measuring these three alternative objectives is unsupervised; that is, given a set of configurations, we can directly calculate values of these objectives without knowing the performance of configurations.

In MoConfig, we consider the sampling of configurations as a multi-objective optimization problem with two objectives: the measurement cost and the entropy, variance, or density; then the sampled configurations are used to train a predictive model of performance ranking, such as a regression method. An algorithm of multi-objective optimization, NSGA-II [12], is used in MoConfig due to its fast sorting ability on non-dominated solutions. We conducted experiments on 20 datasets of configurations and investigate three research questions. In experiments, we compared MoConfig with the rank-based method [10] and evaluated results with the measurement cost and the ranking difference of performance ranking [8], [10], [11].

Experimental results demonstrate that MoConfig can achieve high ranking performance via sampling fewer configurations than the rank-based method. The defined variance of configurations is an effective objective in improving the ranking difference and reducing the number of samples. Meanwhile, the defined entropy of configurations is also an effective objective in several datasets. The entropy can reduce more samples than the variance, but may result in a worse ranking difference. We compared NSGA-II with three other algorithms of multi-objective optimization, including eMOEA, IBEA, and DBEA. We found that, all algorithms except DBEA, lead to similar ranking results. This indicates that different algorithms of multi-objective optimization can be used in MoConfig. We also measured the time of running MoConfig. Compared with the measurement cost of sampling configurations, the time of running MoConfig may be ignored. Experimental results show that MoConfig can be used to improve the ranking difference and reduce the number of samples in building predictive models of performance ranking.

The contributions of this paper are as follows,

- We proposed a multi-objective sampling method, MoConfig, for the performance ranking problem. MoConfig samples configurations via balancing the entropy, variance, or density of sampled configurations and the number of sampled configurations before the performance evaluation of any configuration.
- We conducted experiments on 20 datasets of configurations with four multi-objective optimization algorithms. We showed that MoConfig can achieve high ranking performance via sampling fewer configurations than the rank-based method.

The rest of this paper is organized as follows. Section II shows the basic background information. Section III presents the design of MoConfig. Experimental setup and results are presented in Sections IV and V. Section VI discusses threats to validity. Section VII shows the related work. Finally, Section VIII concludes the paper and lists the future work.

II. Background

A. Performance Ranking of Configurable Systems

The performance ranking problem is a task of finding the optimal (or near-optimal) configurations in a configurable system. For a configurable system, let X denote the set of all valid configurations and let $x_i \in X$ denote the i-th configuration in X. A configuration x_i can be presented as a vector of configuration option values; that is, $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,j}, ...)$, where $x_{i,j}$ denotes the value of the j-th configuration option in x_i. To solve the performance ranking problem, researchers aim to build a predictive model $f(x)$, mapping from each configuration x_i to its performance. In performance ranking, all valid configurations are ranked according to their predicted performance; that is, all $x_i \in X$ are sorted as a ranking list according to their predicted performance $f(x_i)$. In this ranking list, a sublist of top-k configurations is defined as $list(k)$.

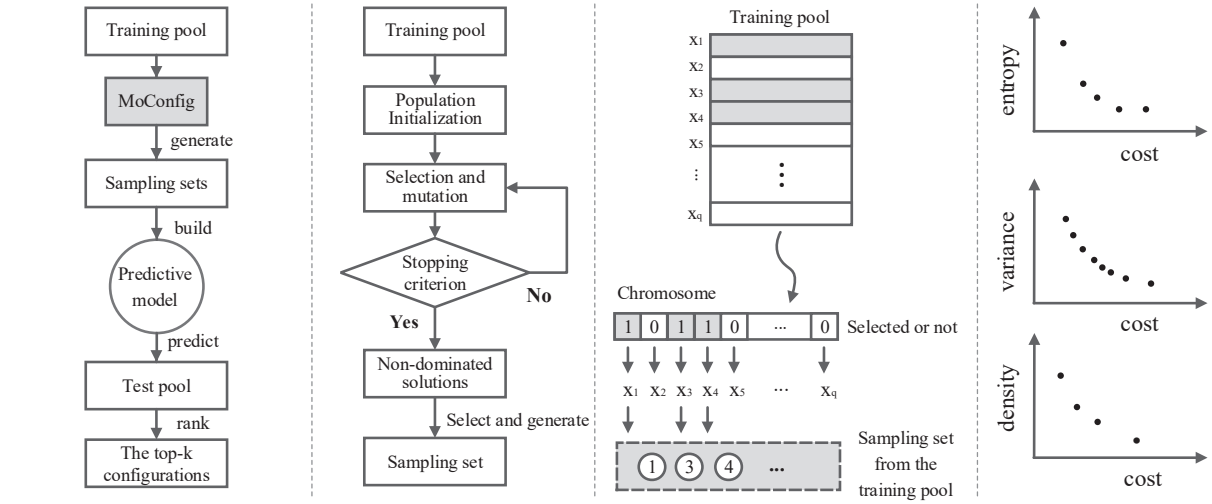
To evaluate the ranking performance, the minimum of actual ranks from top-k configurations is used. We followed Nair et al. [10] to define this minimum of actual ranks as follows,

$$RD(k) = min(rank_1, rank_2, ..., rank_k)$$

where $rank_l$ $(1 \leq l \leq k)$ is the actual rank of the l-th configuration in top-k predicted ranking list $list(k)$. That is, $RD(k)$ calculates the minimum rank of actual performance from the top-k configurations of the predicted ranking list.[1]

The rank-based method by Nair et al. [10] is considered as the state-of-the-art method in performance ranking [8], [11]. This rank-based method first divides the dataset into a training pool, a validation pool, and a test pool with a ratio of $2:1:2$. Then, the rank-based method uses a progressive sampling strategy [13] to iteratively sample configurations from the training pool and builds a predictive model (Classification And Regression Trees, CART) on sampled configurations. Nair et al. [10] found that the rank-based method can achieve better performance ranking for users than previous methods. Sampling configurations can reduce the measurement cost, but may hurt the ranking difference. This motivates our work of balancing the measurement cost and the ranking difference. We proposed the MoConfig approach, which uses multi-objective optimization to sample configurations.

[1]In this paper, we treat the performance rankings as a minimization problem; that is, a configuration with the lowest value of performance is the best configuration.



(a) Solving the performance ranking problem using MoConfig

(b) Details in MoConfig approach

(c) Chromosome representation

(d) Objective pairs in MoConfig

Fig. 1. The overview, the chromosome representation, and the multi-objective pairs in of MoConfig.

B. Multi-objective Optimization

Multi-objective optimization [12], [14], [15] considers two or more objectives together and finds out solutions that are not worse than the other solutions. In multi-objective optimization, the comparison that a solution z_a is "better" than another solution z_b is called that z_a dominates z_b. Note that in this paper, one solution z in MoConfig is a subset of configurations, i.e., $z \subset X$ (shown in Section III-B).

Taking two objectives as an example, let $g(z)$ and $h(z)$ be two objective functions of a solution z. We say z_a dominates z_b, denoted by $z_a \prec z_b$, if

$$g(z_a) \leq g(z_b) \text{ and } h(z_a) < h(z_b).$$

Hence, the output of multi-objective optimization is a set of non-dominated solutions, where no solution dominates the others. This set is also called the Pareto optimal set. All solutions in the set form the Pareto front [12].

In our work, we use an algorithm of multi-objective optimization, NSGA-II [12], to optimize two objectives. NSGA-II, inspired by Darwinian evolution, is a genetic algorithm targeting multiple objectives. The optimization in NSGA-II is an iterative process with multiple generations of solutions. In each generation, NSGA-II uses crossover and mutation operators to evolve new solutions and selects the solutions, which form a set of non-dominated solutions. The crossover operator generates offsprings by swapping some bits between two individuals while the mutation operator creates offsprings by changing some bits of individuals.

III. Multi-Objective Configuration Sampling

This section presents the overview, the chromosome representation, and the optimized objectives of MoConfig.

A. Overview

MoConfig is a pre-processing method to sample configurations for the construction of the training set in the performance ranking problem. Fig. 1 illustrates the usage of MoConfig in the performance ranking and the overview.

Fig. 1(a) shows the steps of solving the performance ranking problem with MoConfig: Since the actual performance of configurations in the training pool is unknown, MoConfig is used to generate a sampling set without the support of performance. The sampling set is then used to build a predictive model to predict the performance of configurations in the test pool. Configurations in the test pool are ranked according to their predicted performance and the top-k configurations are selected as the optimal ones.

Fig. 1(b) presents details of MoConfig. MoConfig takes the training pool as the input and outputs the sampling set from the training pool. The NSGA-II algorithm in Section II-B is used in MoConfig to balance the two objectives. First, MoConfig randomly initializes a population. Then it iteratively generates new populations with crossover operators and mutation operators. When the iteration terminates, the non-dominated solutions are generated. One solution, i.e., a sampling set of configurations from the training pool, is used to build a predictive model for performance ranking.

Chromosome representation in Fig. 1(c) and objective pairs in Fig. 1(d) will be shown in the following sections.

B. Chromosome Representation

In MoConfig, each chromosome corresponds to a sampling set of configurations. The chromosome determines which configurations to sample, that is, one chromosome corresponds to one sampling set. For a training pool with q configurations, a chromosome can be encoded as a vector of q bits. Each bit is a binary value that represents whether a configuration is sampled or not. Taking the chromosome in Fig. 1(c) as

an example, the first, second, and fourth configurations in the training pool are selected to form the sampling set.

C. Objective Pairs

The goal of MoConfig is to simultaneously optimize the measurement cost and the ranking difference in the sampling process. However, this sampling process is unsupervised: we cannot know the performance of the configurations before calculating the ranking difference. Hence, we define three alternative objectives, named the entropy, the variance, and the density, instead of the ranking difference. Let T be a sampling set of configurations and the size of T is $n = |T|$. Each configuration has m configuration options. Let $x_{i,j}$ be the j-th option value of the i-th configuration ($1 \leq i \leq n$ and $1 \leq j \leq m$). We defined three alternative objectives as follows.

The entropy of the sampling set T is defined via counting the sum of all entropy values for each configuration option. Let $p_{i,j}$ be the probability that the j-th configuration option equals to $x_{i,j}$; then the entropy of the j-th configuration option is defined as $entropy(j) = E(-\log p_{i,j}) = -\sum_{i=1}^{n} p_{i,j} \log p_{i,j}$, where $E(\bullet)$ is the expectation function. Then we defined the entropy of the sampling set T as follows,

$$entropy(T) = W - \sum_{j=1}^{m} entropy(j) = W + \sum_{j=1}^{m} \sum_{i=1}^{n} p_{i,j} \log p_{i,j}$$

where W is a pre-defined constant. W is used to convert the original problem of maximizing the sum of $entropy(j)$ into a minimization problem.[2]

The variance of T is defined via counting the variance of the Euclidean distance between each configuration and the average. Let $dist(x_i, \overline{x})$ be the Euclidean distance between a configuration x_i and the average $\overline{x}$ of all configurations in T. Then $dist(x_i, \overline{x}) = \sum_{j=1}^{m} (x_{i,j} - avg(j))^2$, where $avg(j)$ is the average value of the j-th configuration option and $avg(j) = \frac{1}{n} \sum_{i=1}^{n} x_{i,j}$. Then we defined the variance of the sampling set T as follows,

$$variance(T) = W - \sum_{i=1}^{n} dist(x_i, \overline{x}))$$

The density of T is defined via counting the ratio of unique values. Let $unique(j)$ is the number of unique values of the j-th configuration option. Then we defined the density of the sampling set T as follows,

$$density(T) = W - \frac{\sum_{j=1}^{m} unique(j)}{m \times n}$$

We used the above three objectives $entropy(T)$, $variance(T)$, and $density(T)$ to measure the entropy, the variance, and the density of a chromosome, which is a sampling set of configurations.

[2]For the sake of illustration, all objectives in our work are the minimization problems, which consider the lowest value as the optimal value. Any value can be used for W. We set $W = 10000$ in our work.

TABLE I
THE BASIC INFORMATION OF 20 DATASETS

Dataset name	Number of options	Number of configurations
rs-6d-c3-obj1	6	3840
rs-6d-c3-obj2	6	3840
sol-6d-c2-obj1	6	2803
sol-6d-c2-obj2	6	2803
wc+rs-3d-c4-obj1	3	196
wc+rs-3d-c4-obj2	3	196
wc+sol-3d-c4-obj1	3	196
wc+sol-3d-c4-obj2	3	196
wc+wc-3d-c4-obj1	3	196
wc+wc-3d-c4-obj2	3	196
wc-3d-c4-obj1	3	756
wc-3d-c4-obj2	3	756
wc-5d-c5-obj1	5	1080
wc-5d-c5-obj2	5	1080
wc-6d-c1-obj1	6	2880
wc-6d-c1-obj2	6	2880
wc-c1-3d-c1-obj1	3	1343
wc-c1-3d-c1-obj2	3	1343
wc-c3-3d-c1-obj1	3	1512
wc-c3-3d-c1-obj2	3	1512

In MoConfig, we simultaneously optimize two objectives. Based on the above definitions, we have three pairs of objectives, including the pairs of entropy-cost, variance-cost, and density-cost, where *cost* denotes the measurement cost, i.e., the number of sampled configurations.

IV. EXPERIMENTAL SETUP

A. Data Preparation

To evaluate our approach, we selected 20 datasets which are commonly used in previous studies [10], [16]. Existing datasets contain two kinds of configuration options: Boolean options and numeric options. In our study, only datasets that contain numeric options are used since the entropy, the variance, and the density in MoConfig may not be suitable for Boolean options. Table I lists the number of configurations options and the number of configurations of each dataset. All datasets are derived from different components of a widely-used distributed system, the Apache Storm. In Table I, *obj1* and *obj2* denote two types of performance: latency and throughput. For example, the dataset rs-6d-c3-obj1 records the latency of each configuration in a scenario *rs-6d-c3* while rs-6d-c3-obj2 records the throughput in the same scenario. The number of configurations in all datasets ranges from 196 to 3840. Experimental data are available on the website.[3]

B. Experimental Setting

In experiments, we compare our proposed method with the rank-based method. For each method in comparison, we set $k = 10$ in $RD(k)$; that is, we measure the minimum actual rank in the top-10 predicted ranking list as the ranking difference. For all algorithms in MoConfig, the initial population is generated randomly; the maximum times of generations is set to 10000.

[3]MoConfig, http://cstar.whu.edu.cn/p/moconfig/.

TABLE II
THE NUMBER OF SOLUTIONS, THE AVERAGE OF THE RANKING DIFFERENCE, AND THE AVERAGE OF THE MEASUREMENT COST OF METHODS.

Dataset name	Rank-based			entropy-cost			variance-cost			density-cost		
	Solutions	Cost	RD(10)	Solutions	Cost	RD(10)	Solutions	Cost	RD(10)	Solutions	Cost	RD(10)
rs-6d-c3-obj1	50	544.12	27.82	6	134.00	18.33	100	474.73	**12.04**	1	93.00	13.00
rs-6d-c3-obj2	50	551.52	36.38	3	141.67	8.00	100	470.42	**4.88**	1	100.00	93.00
sol-6d-c2-obj1	50	406.02	35.44	4	70.25	9.00	100	344.46	**3.32**	2	48.50	7.00
sol-6d-c2-obj2	50	410.38	42.46	3	68.33	6.67	100	341.61	**1.97**	1	45.00	12.00
wc+rs-3d-c4-obj1	50	52.04	3.20	8	4.50	6.38	34	25.03	**3.09**	1	1.00	18.00
wc+rs-3d-c4-obj2	50	50.64	**1.60**	8	4.50	8.00	37	25.89	3.14	1	1.00	10.00
wc+sol-3d-c4-obj1	50	50.42	3.64	8	4.50	18.38	35	23.54	10.57	1	1.00	**1.00**
wc+sol-3d-c4-obj2	50	51.68	**1.28**	9	5.00	8.22	25	20.56	2.12	1	1.00	24.00
wc+wc-3d-c4-obj1	50	52.00	1.32	8	4.50	8.75	35	24.80	**1.31**	1	1.00	41.00
wc+wc-3d-c4-obj2	50	52.28	**1.54**	8	4.50	9.38	30	20.73	3.70	1	1.00	2.00
wc-3d-c4-obj1	50	130.52	5.12	17	10.35	29.94	91	92.80	**1.59**	1	1.00	16.00
wc-3d-c4-obj2	50	130.06	4.80	16	14.88	12.50	89	83.19	**2.13**	1	1.00	25.00
wc-5d-c5-obj1	50	175.92	23.04	7	6.00	46.71	100	126.61	**3.64**	1	1.00	119.00
wc-5d-c5-obj2	50	178.82	13.94	7	6.43	83.86	100	117.64	**3.37**	1	1.00	2.00
wc-6d-c1-obj1	50	421.78	52.70	3	66.67	29.00	100	346.36	**11.28**	1	45.00	25.00
wc-6d-c1-obj2	50	419.00	33.62	1	68.00	96.00	100	343.76	**3.81**	1	42.00	39.00
wc-c1-3d-c1-obj1	50	214.38	14.92	13	29.15	20.00	68	131.74	**8.12**	1	2.00	194.00
wc-c1-3d-c1-obj2	50	214.68	**15.50**	13	36.38	38.00	79	152.08	17.80	1	2.00	33.00
wc-c3-3d-c1-obj1	50	239.96	22.00	11	42.91	16.91	75	161.73	**6.75**	2	4.50	29.50
wc-c3-3d-c1-obj2	50	233.46	21.46	11	33.91	17.73	82	159.70	**8.87**	1	2.00	8.00

Prototypes are implemented with Java JDK 8.0 and experiments are implemented in Python 3.6.2. All codes are run on a PC with Intel(R) Core(TM) i7-4790 CPU 3.6GHz and memory of 8GB RAM.

C. Research Questions

a) ***RQ1.*** *Can MoConfig sample fewer configurations than the rank-based approach in performance ranking?* We designed RQ1 to explore the capability of our approach in solving the performance ranking problem. We choose the rank-based method as the baseline method since it is a widely used sampling method to solve the performance ranking problem in previous studies [8], [10], [11].

b) ***RQ2.*** *Which multi-objective optimization algorithm can be used in MoConfig?* We designed RQ2 to study the best choice of multi-objective optimization algorithms in MoConfig. MoConfig uses the NSGA-II as the multi-objective optimization algorithm. We aim at analyzing which algorithm is more suitable for MoConfig than NSGA-II.

c) ***RQ3.*** *What is the time cost of MoConfig?* We designed RQ3 to measure how long does it take to run MoConfig. This can be used as a reference for applying MoConfig. Compared with the measurement cost of sampling configurations, the time of running MoConfig may be ignored.

V. EXPERIMENTAL RESULTS

A. RQ1. Can MoConfig sample fewer configurations than the rank-based approach in performance ranking?

Evaluation steps. We compared the prediction results (i.e., the ranking difference and the measurement cost) obtained by MoConfig and the rank-based method. The rank-based method divides the dataset into the training pool, the validation pool, and the test pool with a ratio of 2:1:2 [10]. For fair comparison, MoConfig uses the same test pool with the rank-based method and takes the rest of the data as its training pool. Then we count the number of sampled configurations for each method. That is, the ratio of the training pool and the test pool is $3:2$ in MoConfig. To avoid the bias of randomness, we run the rank-based method for 50 times and calculated the mean value of the measurement cost and the ranking difference.

Table II shows the solution numbers, the average measurement cost, and the average ranking difference of the rank-based method and MoConfig (in three objective pairs). The ranking difference is measured with the minimum of actual ranks $RD(10)$ and the measurement cost is the number of sampled configurations. For both of the ranking difference and the cost, the lower the better. We found that all three objective pairs in MoConfig have a lower cost than the rank-based method. The pair of variance-cost can obtain the best ranking differences in 15 datasets while the rank-based method only obtains the best ranking differences in 4 datasets. The pair of density-cost has the lowest measurement cost but achieves the worst ranking differences among all the methods.

We also recorded the distribution of non-dominated solutions and the prediction results of each dataset. Due to the space limitation, we only show an example of *rs-6d-c3-obj1* in Fig. 2. The non-dominated solutions generated by three objective pairs are shown in the first three sub-figures. The prediction results (i.e., the ranking difference and the cost) of MoConfig and the rank-based method are given in the last sub-figure. Experimental results of the remaining datasets are similar to the above figure and can be found on our website.

From the distribution of non-dominated solutions, we can find that the pair of variance-cost generates more non-dominated solutions and needs high measurement cost. The pair of density-cost produces fewer non-dominated solutions and requires low measurement cost. This difference can be explained by the choice of objective pairs because the definition of objectives determines the direction of optimization and further influence the convergence speed.

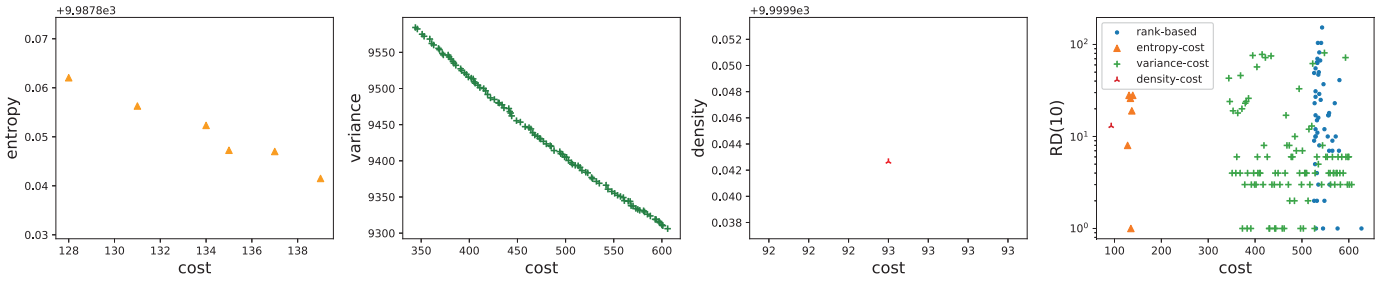
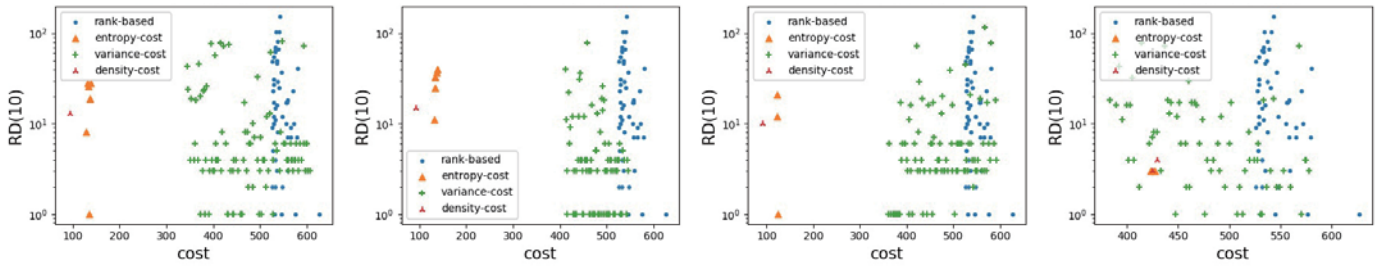
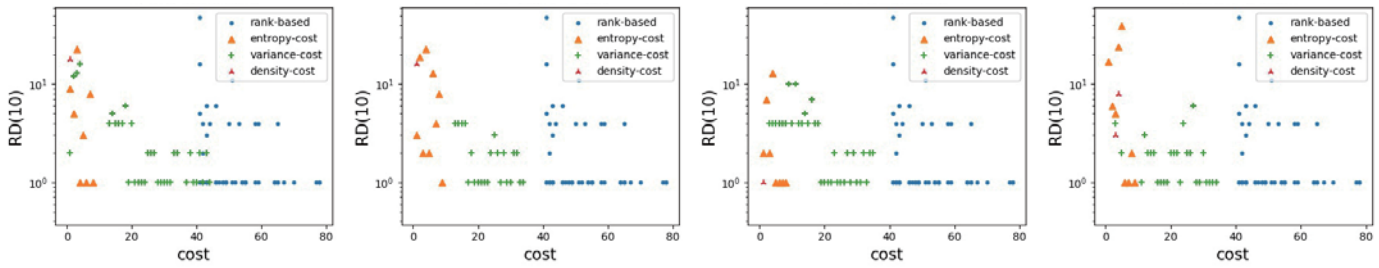


Fig. 2. The non-dominated solutions and the prediction results in rs-6d-c3-obj1



(a) MoConfig-NSGAII (b) MoConfig-eMOEA (c) MoConfig-IBEA (d) MoConfig-DBEA

Fig. 3. The prediction results by using different multi-objective optimization algorithms in rs-6d-c3-obj1



(a) MoConfig-NSGAII (b) MoConfig-eMOEA (c) MoConfig-IBEA (d) MoConfig-DBEA

Fig. 4. The prediction results by using different multi-objective optimization algorithms in wc+rs-3d-c4-obj1

From the prediction results, we found that both MoConfig and the rank-based method can obtain good ranking differences ($RD(10)$), but MoConfig requires fewer configurations than the rank-based method to reach the same ranking differences in most cases. For example in rs-6d-c3-obj1 in Fig. 2, MoConfig can obtain a ranking difference of 1 with at least 100 configurations while the rank-based method requires over 500 configurations. The pair of variance-cost always outperforms the rank-based method according to the results. This indicates that the objective of the variance outperforms the entropy and density in MoConfig.

We can conclude that MoConfig samples fewer configurations than the rank-based method in performance ranking. The objective of the variance is effective in MoConfig.

B. RQ2. Which multi-objective optimization algorithm can be used in MoConfig?

Evaluation steps. We try to compare the NSGA-II with other three multi-objective optimization algorithms, i.e., e-MOEA, IBEA, and DBEA. Thus, we have four variants of MoConfig, which are denoted as MoConfig-NSGAII, MoConfig-eMOEA, MoConfig-IBEA, and MoConfig-DBEA, respectively. We compare the measurement cost and the ranking differences obtained by each variant.

Fig. 3 and Fig. 4 present the detailed prediction results using four variants of MoConfig in datasets *rs-6d-c3-obj1* and *wc+rs-3d-c4-obj1*, respectively. We can find that all four variants can obtain a similar range of the ranking difference. However, the variant MoConfig-DBEA is unstable in some datasets. For example in Fig. 3, NSGA-II, eMOEA, and IBEA have a similar measurement cost while DBEA sometimes requires higher measurement cost. In Fig. 4, four variants are in a similar range of the measurement cost. These results demonstrate that algorithms NSGA-II, eMOEA, and IBEA are all suitable for MoConfig, while DBEA may be not a good choice of MoConfig. The remaining datasets follow the same experimental results, which can also be found on our website.

To summer up, MoConfig is effective with several multi-objective optimization algorithms except DBEA.

C. RQ3. What is the time cost of Moconfig?

Evaluation steps. We define the time of MoConfig as the sum of the evolution cost and building cost. The evolution cost is the time consumption in the generation of non-dominated solutions by NSGA-II. The building cost is the time consumption in building a CART model. However, the building cost is small and is determined by the performance of CART, which is beyond the scope of discussion in this paper. We approximate the evolution cost as the time cost of MoConfig. We run the MoConfig for 10 times and calculate the average time cost.

Table III presents the time cost (in milliseconds) of MoConfig in three objective pairs, where $Time_{entropy}$, $Time_{variance}$, and $Time_{density}$ denote the time of three objective pairs of entropy-cost, variance-cost, and density-cost, respectively.

We found that, in each dataset (each row in the table), the $Time_{density}$ is the smallest; the $Time_{entropy}$ is the second; and the $Time_{variance}$ is the largest (i.e., $Time_{density} < Time_{entropy} < Time_{variance}$). This difference is mainly caused by the computational complexity of different objectives. We can also estimate the computational complexity of these objectives from their definitions in Section III-C, where the variance needs more time to be calculated than the density and the entropy. Additionally, the average time is acceptable. Almost all the time is less than 2 seconds and the average time in each objective pairs is less than 1 second, indicating that our approach can successfully generate sampling sets in a short time.

We conclude that the average time of MoConfig is 413.4, 871.1, and 225.9 milliseconds in three objective pairs, respectively.

TABLE III
THE TIME OF MOCONFIG IN THREE OBJECTIVE PAIRS ON 20 DATASETS

Dataset name	$Time_{entropy}$	$Time_{variance}$	$Time_{density}$
rs-6d-c3-obj1	1143.2	2133.1	720.5
rs-6d-c3-obj2	1103.1	1647.1	701.7
sol-6d-c2-obj1	764.2	1536.8	477.6
sol-6d-c2-obj2	742.5	1311.7	461.1
wc+rs-3d-c4-obj1	90.6	271.7	44.1
wc+rs-3d-c4-obj2	105.3	231.9	46.3
wc+sol-3d-c4-obj1	89.2	240.2	43.0
wc+sol-3d-c4-obj2	93.8	197.1	43.0
wc+wc-3d-c4-obj1	116.2	204.8	48.2
wc+wc-3d-c4-obj2	96.9	178.3	41.7
wc-3d-c4-obj1	263.6	807.7	89.8
wc-3d-c4-obj2	242.4	743.5	83.3
wc-5d-c5-obj1	231.8	911.9	135.8
wc-5d-c5-obj2	241.1	1258.0	140.4
wc-6d-c1-obj1	758.1	1650.5	474.3
wc-6d-c1-obj2	766.2	1474.0	473.3
wc-c1-3d-c1-obj1	417.5	803.1	158.0
wc-c1-3d-c1-obj2	390.8	839.5	154.6
wc-c3-3d-c1-obj1	450.0	811.7	185.9
wc-c3-3d-c1-obj2	443.7	798.6	175.0
Average	**413.4**	**871.1**	**225.9**

VI. THREATS TO VALIDITY

Internal Validity. In MoConfig, we used three pairs of objectives to generate the final solutions for the performance ranking. The correlation between these objectives and the ranking difference could be investigated for further understanding. Meanwhile, we use CART as the predictive model in the performance ranking. Although CART is commonly used in the performance prediction [2], [8], [10], [17], other regression models should be evaluated. Different choices of regression models can bring a clear view of experimental results.

External Validity. In experiments, we selected 20 datasets, which contain numeric configuration options. The reason for such selection is due to the design of the entropy, the variance, and the density. This leads to a threat that the proposed method cannot solve the sampling problem of Boolean configurations. It is feasible to design other optimization objectives for Boolean configuration options.

VII. RELATED WORK

A. Performance Ranking Problem

Previous studies [1], [2], [9] solved the performance ranking problem by predicting the performance of configurations.

Siegmund et al. [18] proposed a polynomial-based model *SPL Conqueror* to predict the configuration performance by figuring out the inner feature interactions. However, this model needs to measure a large number of training samples to find out the whole feature interactions. Guo et al. [2] used the *Classification And Regression Trees* (CART [19]) to build the predictive model. Guo et al. [17] then proposed the *Data-Efficient CART* (DECART), which combines resampling with parameter tuning to ease the instability of CART.

Ha and Zhang [4] proposed the *DeepPerf* to construct a deep forward neural network to predict the performance. In this approach, they designed an hyper-parameter tuning strategy to narrow down the search space of parameters. Nair et al. [8] proposed the Flash, which uses the idea of single- and multi-objective optimization to find the optimal configurations with two definitions of performance. Singh et al. [20] used the NSGA-II algorithm to pick out the optimal configurations with three definitions of performance. Different from the above studies that consider multiple definitions of performance, our paper used multiple objectives to measure the sampling process.

B. Sampling Strategy

A sampling strategy plays a key role in both performance ranking and fault detection [9], [13], [21], [22].

Guo et al. [2] used a random sampling strategy in model training. They randomly select N, $2N$, $3N$ samples where N is the number of features in a specific system. They found that the built model is precise if the training set and the whole dataset have a similar performance distribution. Siegmund et al. [9] proposed three strategies (i.e., *option-wise, negative option-wise*, and *pair-wise*) for Boolean-option sampling and two strategies for numeric-option sampling. Chen et al. [23] proposed a comparative study on regression models of performance ranking of highly configurable Systems.

Sarkar et al. [13] analyzed the *progressive sampling* and *projective sampling* strategies, which are applied in some con-

figuration optimization studies [10]. Xuan et al. [22] proposed *genetic configuration sampling* (GCS) for the problem of fault localization of configurable systems. GCS is a single objective genetic algorithm that combines existing sampling strategies, such as *one-disabled* and *t-wise*, in a large-scale configurable system. Kaltenecker et al. [24] proposed the *distance-based sampling* strategy, which uses the discrete probability distribution and distance metrics to sample configurations.

Different from previous strategies, MoConfig samples configurations via simultaneously optimizing the measurement cost and the ranking difference. The ranking difference is indirectly obtained based on calculating the entropy, the variance, or the density of samples.

VIII. Conclusion and Future work

This paper proposed a multi-objective configuration sampling method, namely MoConfig, which aims to balances the trade-off between the measurement cost and the ranking difference in the performance ranking problem. MoConfig first uses a multi-objective optimization algorithm to generate non-dominated solutions, each of which is a sampling set of configurations. Sampled configurations can be used to build a predictive model to predict the performance of new configurations. Experimental results demonstrated that MoConfig can achieve high ranking performance via sampling fewer configurations than the rank-based method.

In future work, we plan to design to enhance the multi-objective optimization of sampling to further improve the performance. We also plan to evaluate on large real-world configurable systems to understand the issues of the performance ranking in practice.

Acknowledgment

The work is supported by the National Key R&D Program of China under Grant No. 2018YFB1003901 and the National Natural Science Foundation of China under Grant Nos. 61872273 and 61502345.

References

[1] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, pp. 167–177, 2012.

[2] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-aware performance prediction: A statistical learning approach," in 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013, pp. 301–311, 2013.

[3] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Faster discovery of faster system configurations with spectral learning," Autom. Softw. Eng., vol. 25, no. 2, pp. 247–277, 2018.

[4] H. Ha and H. Zhang, "Deepperf: performance prediction for configurable software with deep sparse neural network," in Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, pp. 1095–1106, 2019.

[5] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, and S. Apel, "A comparison of 10 sampling algorithms for configurable systems," in Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, pp. 643–654, 2016.

[6] I. Abal, C. Brabrand, and A. Wasowski, "42 variability bugs in the linux kernel: a qualitative analysis," in ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014, pp. 421–432, 2014.

[7] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, May 14-19, 2017, pp. 1009–1024, 2017.

[8] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," IEEE Transactions on Software Engineering, to appear.

[9] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, August 30 - September 4, 2015, pp. 284–294, 2015.

[10] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Using bad learners to find good configurations," in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017, pp. 257–267, 2017.

[11] L. Bao, X. Liu, Z. Xu, and B. Fang, "Autoconfig: Automatic configuration tuning for distributed message systems," in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, September 3-7, 2018, pp. 29–40, 2018.

[12] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Trans. Evolutionary Computation, vol. 6, no. 2, pp. 182–197, 2002.

[13] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki, "Cost-efficient sampling for performance prediction of configurable systems (T)," in 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, 2015, pp. 342–352, 2015.

[14] A. Santiago Pineda, H. J. Fraire Huacuja, B. Dorronsoro, J. E. Pecero, C. G. Santillán, J. J. G. Barbosa, and J. C. Soto Monterrubio, "A survey of decomposition methods for multi-objective optimization," in Recent Advances on Hybrid Approaches for Designing Intelligent Systems, pp. 453–465, 2014.

[15] R. Azzouz, S. Bechikh, and L. B. Said, "Dynamic multi-objective optimization using evolutionary algorithms: A survey," in Recent Advances in Evolutionary Multi-objective Optimization, pp. 31–70, 2017.

[16] P. Jamshidi and G. Casale, "An uncertainty-aware approach to optimal configuration of stream processing systems," in 24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2016, pp. 39–48, 2016.

[17] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, and H. Yu, "Data-efficient performance learning for configurable systems," Empirical Software Engineering, vol. 23, no. 3, pp. 1826–1867, 2018.

[18] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake, "SPL conqueror: Toward optimization of non-functional properties in software product lines," Software Quality Journal, vol. 20, no. 3-4, pp. 487–517, 2012.

[19] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees. Wadsworth, 1984.

[20] R. Singh, C. Bezemer, W. Shang, and A. E. Hassan, "Optimizing the performance-related configurations of object-relational mapping frameworks using a multi-objective genetic algorithm," in Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, March 12-16, 2016, pp. 309–320, 2016.

[21] M. Last, "Improving data mining utility with projective sampling," in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009, pp. 487–496, 2009.

[22] J. Xuan, Y. Gu, Z. Ren, X. Jia, and Q. Fan, "Genetic configuration sampling: learning a sampling strategy for fault detection of configurable systems," in Proceedings of the Genetic Improvement Workshop, GECCO 2018, Kyoto, Japan, July 15-19, 2018, pp. 1624–1631, 2018.

[23] Y. Chen, Y. Gu, L. He, and J. Xuan, "Regression models for performance ranking of configurable systems: A comparative study," in Proceedings of the Annual Conference on Software Analysis, Testing and Evolution, SATE 2019, Hangzhou, China, 2019, 2019.

[24] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel, "Distance-based sampling of software configuration spaces," in Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, pp. 1084–1094, 2019.