On Parameter Tuning in Search Based Software Engineering: A Replicated Empirical Study

Abdel Salam Sayyad Katerina Goseva-Popstojanova Tim Menzies Hany Ammar
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV, USA
asayyad@mix.wvu.edu katerin.goseva@mail.wvu.edu tim@menzies.us hany.ammar@mail.wvu.edu

***Abstract*—Multiobjective Evolutionary Algorithms are increasingly used to solve optimization problems in software engineering. The choice of parameters for those algorithms usually follows the "default" settings, often accepted as "rule of thumb" or common wisdom. The fact is that each algorithms needs to be tuned for the problem at hand. Previous work [Arcuri and Fraser, 2011] has shown that variations in parameter values had large effects on the performance of the algorithms. This project seeks to partially replicate the statistical analysis performed by Arcuri and Fraser. We seek to investigate the effects of parameter tuning on the performance of the two algorithms: Indicator-Based Evolutionary Algorithm (IBEA), and Nondominated Sorting Genetic Algorithm (NSGA-II) when applied to the problem of configuring Software Product Lines (SPLs) in the presence of stakeholder preferences such as cost and reliability. The results of this study confirm and strengthen the findings in the original study by Arcuri and Fraser.**

Keywords—Parameter Tuning; Search Based Software Engineering; Software Product Lines.

I. INTRODUCTION

Evolutionary algorithms (EAs) are popular for solving problems that are otherwise intractable; i.e. problems that become impossible to solve as their dimensions increase. EAs are able to perform this task because they navigate the solution space guided by heuristics, without having to check every possible solution. When arguing about the performance of EAs, it is essential to account for the randomness in their behavior, hence the need for repeated experiments and statistical analysis to assess the strength of conclusions.

This paper is concerned with the performance assessment of EAs when different values are assigned for the different parameters involved. A study was performed by Arcuri and Fraser [1], [2], that performed the same experiment to the problem of generating test vectors for object-oriented software, and comparing the performance according to test coverage. This experiment seeks to emulate the work of Arcuri and Fraser in different domain. We investigate the effects of parameter tuning on the performance of two multiobjective evolutionary optimization algorithms (MEOAs), namely Indicator-Based Evolutionary Algorithm (IBEA), and Nondominated Sorting Genetic Algorithm (NSGA-II) when applied to the problem of configuring Software Product Lines (SPLs). Previous work by Sayyad et al [10] has shown the superiority of IBEA when optimizing up to 5 objectives related to cost and reliability. That work was performed with a fixed set of "default" parameters, which cast a little doubt as to the ability of NSGA-II to perform better under a different set of parameter values. This work confirms statistically that IBEA remains the algorithm of choice for this problem, and also confirms the findings by Arcuri and Fraser that parameter choice has a large effect on the performance of EAs.

The rest of this paper is organized as follows: Section II introduces background material on feature models and MEOAs. Section III presents the research questions. Section IV explains the experimental setup. Section V mentions the statistical analysis method, and section VI presents the results. In section VII we discuss potential threats to validity, and then in section VIII we offer our conclusions and directions for future work.

II. BACKGROUND

This section is adapted from [10] to provide background about Software Product Lines (SPLs), feature models, and multiobjective evolutionary optimization algorithms (MEOAs).

A. Software Product Line Engineering (SPLE)

Software product line engineering is a paradigm to develop software applications using platforms and mass customization. Benefits of SPLE include reduction of development costs, enhancement of quality, reduction of time to market, reduction of maintenance effort, coping with evolution and complexity [9], and automating the creation of family members [4].

B. Feature Models

A feature is an end-user-visible behavior of a software product that is of interest to some stakeholder. A feature model represents the information of all possible products of a software product line in terms of features and relationships among them. Feature models are a special type of information model widely used in software product line engineering. A feature model is represented as a hierarchically arranged set of features composed by:

- Relationships between a parent feature and its child features (or subfeatures).
- Cross-tree constraints (CTCs), which describe relationships between features from different branches in the tree. CTCs are typically inclusion or exclusion statements in the form: if feature F is included, then features A and B must also be included (or excluded).

This research work was funded by the Qatar National Research Fund (QNRF) under the National Priorities Research Program (NPRP) Grant No.: 09-1205-2-470.

978-0-7695-5121-0/13 $26.00 © 2013 IEEE
DOI 10.1109/RESER.2013.6

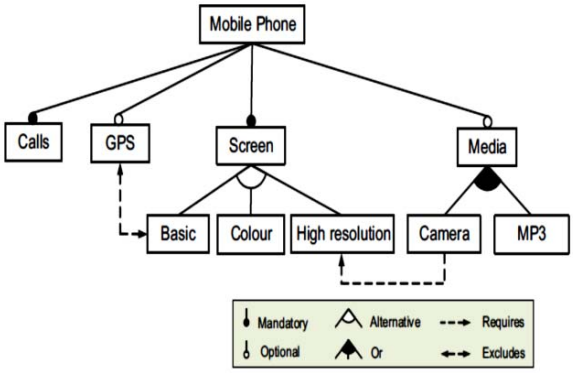

Fig. 1. Feature model for mobile phone product line, adopted from [3]

Fig. 1, adapted from [3], depicts a simplified feature model inspired by the mobile phone industry.

The full set of rules in a feature model includes:

- The root feature is mandatory.
- Every child requires its own parent.
- If the child is mandatory then the parent requires it.
- Every group adds a rule about how many members can be chosen.
- Every cross-tree constraint (CTC) is a rule.

The total number of rules will be used as the "full correctness" score in this experiment, thus making "correctness" one of the optimization objectives.

C. Genetic Algorithms

Both IBEA and NSGA-II algorithms have Genetic Algorithm at their core. The original Genetic Algorithm (GA) was developed with a single fitness value in mind, i.e. a single optimization objective. Fig. 2 lists an outline of the Genetic Algorithm, adapted from [7].

```
Initialization
        popsize ← desired population size
        P ← {}
        for popsize times do
                P ← P ∪ {new random individual}
        Best ← null
Repeat until Termination
    for each individual Pi ∈ P do
        AssignFitness(Pi)
        if Best = null or Fitness(Pi) > Fitness(Best) then
            Best ← Pi
    Q←{}
    for popsize/2 times do
        Parent Pa ← MatingSelection(P)
        Parent Pb ← MatingSelection (P)
        Children Ca,Cb ← Crossover(Copy(Pa),Copy(Pb))
        Q ← Q ∪{Mutate(Ca),Mutate(Cb)}
    P ← Q
Termination Best is ideal solution or we've run out of time
return Best
```

Fig. 2. Genetic Algorithm, adapted from **[7]**

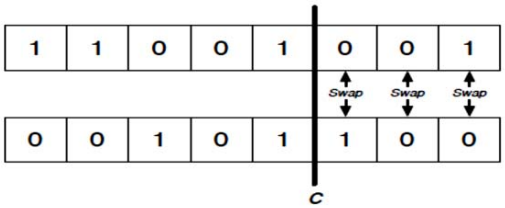

Fig. 3. Example of a Single-Point Crossover, from **[7]**

The SelectWithReplacement() function selects individuals from the population with a probability proportionate with their fitness. An individual may, by chance, be chosen for breeding multiple times.

The individual solutions are of the Boolean string type. The Crossover() function performs a single-point crossover, where a point is chosen randomly, and the bits are swapped between the two parents as shown in Fig. 3 [7]. Crossover between two parents is performed with a predetermined probability.

The Mutate() function performs bit-flip mutation, where each bit in the string may be flipped at a predetermined probability.

D. Multiobjective Optimization

Many real-world problems involve simultaneous optimization of several incommensurable and often competing objectives. Often, there is no single optimal solution, but rather a set of alternative solutions. These solutions are optimal in the wider sense that no other solutions in the search space are superior to them when all objectives are considered [15].

Formally, a *vector* $u = \{u_1, \ldots, u_k\}$ is said to be dominated by a vector $v = \{v_1, \ldots, v_k\}$ if and only if u is partially less than v, i.e.

$$\forall i \in \{1, \ldots, k\}, u_i \leq v_i \; and \; \exists \, i \in \{1, \ldots, k\} : u_i < v_i \quad (1)$$

The set of all points in the objective space that are not dominated by any other points is called the *Pareto Front*.

E. Multiobjective Evolutionary Optimization Algorithms (MEOAs)

Many real-world problems involve simultaneous optimization of several incommensurable and often competing objectives. Often, there is no single optimal solution, but rather a set of alternative solutions. These solutions are optimal in the wider sense that no other solutions in the search space are superior to them when all objectives are considered [15].

Many algorithms have been suggested over the past two decades for multiobjective optimization based on evolutionary algorithms that were designed primarily for single-objective optimization, such as Genetic Algorithms, Evolutionary Strategies, Particle Swarm Optimization, and Differential Evolution.

The algorithms we used in this study were already implemented in the jMetal framework [6]. They are:

1. IBEA: Indicator-Based Evolutionary Algorithm [14].
2. NSGA-II: Nondominated Sorting Genetic Algorithm, version 2 [5].

F. Quality Indicators

To assess the performance of the two algorithms, we use two quality indicators:

1. The primary indicator is %Correct: i.e. the percentage of fully-correct solutions, which is an indicator particular to this problem. Since correctness is an optimization objective that evolves over time, there maybe points in the final Pareto front that have rule violations. Such points are not likely to be useful to the user. We are interested in percentage of points within the Pareto front that have zero violations, and thus a full-correctness score.
2. The secondary indicator is Hypervolume (HV), which is an indicator of how close to optimality the solutions is. HV is implemented in jMetal [6].

In the results section, we list the values for %Correct. We use HV to rank solutions that have the same %Correct, but we don't show HV values for brevity.

III. Research Questions

In this paper, we consider 4 research questions. RQ1, RQ2, and RQ4 are considered by Arcuri and Fraser [1]. We added RQ3 since we're interested in comparing IBEA to NSGA-II.

RQ1: How Large is the Potential Impact of a Wrong Choice of Parameter Settings?

RQ2: How Does a "Default" Setting Compare to the Best and Worst Achievable Performance?

RQ3: How does the performance of IBEA's best tuning compare to NSGA-II's best tuning?

RQ4: If we Tune a Search Algorithm Based on a Set of Feature Models, How Will Its Performance Be On Other New Feature Models?

IV. Experimental Setup

In this section, we explain the experimental setup and the value choices for the different parameters. Where possible, we used the same parameter levels that were used by Arcuri and Fraser. In some cases, we used fixed values because the value levels were not available in jMetal. Table 1 shows the similarities and differences between Arcuri and Fraser and this study.

Each feature model (FM) was operated on for 10 seconds. The overall time to execute our experiment was = 250 configs x 15 runs x 20 FMs x 10 sec = 12500 minutes = 208.3 hours = ~8.7 days.

A. Default Values

The default parameter values according to jMetal are: Population = 100, Crossover rate = 0.8, and Mutation rate = 1/FEATURES.

B. Feature Models Used in this Study

20 non-trivial feature models were chosen from the SPLOT online repository [8]. Table 2 shows the 20 feature models that were used in this study.

The feature models were augmented with 3 attributes for each feature: *COST*, *USED_BEFORE*, and *DEFECTS*. The values were selected stochastically according to distributions that emulate software projects. COST takes real values distributed normally between 5.0 and 15.0, USED_BEFORE takes Boolean values distributed uniformly, and DEFECTS takes integer values distributed normally between 0 and 10.

C. Problem Representation

The feature models were represented as binary strings, where the number of bits is equal to the number of features. If the bit value is TRUE then the feature is selected, otherwise the feature is removed (i.e. deselected).

Table 1: Experimental setup

Parameter	***Arcuri and Fraser [1]***	***This paper***
Algorithm	Genetic Algorithm	NSGA-II, IBEA
Mutation rate	0.1	{0, 0.5, 1, 1.5, 2}/Features
Crossover rate	{0 , .2 , .5 , .8 , 1}	{0 , .2 , .5 , .8 , 1}
Population size	{4 , 10, 50 , 100 , 200}	{10, 50 , 100 , 150, 200}
Elitism rate	{0 , 1, 10% , 50%} or steady state	Elitism rate does not apply to either NSGA-II or IBEA. Steady state is implemented in jMetal for NSGA-II but not IBEA. Thus no variation will be considered here.
Selection	roulette wheel, tournament with size either 2 or 7, and rank selection with bias either 1.2 or 1.7	Binary tournament only.
Parent replacement check	activated or not	No.
Repeats	15	15
Test bed	20 classes	20 feature models
Number of configurations	5 x 5 x 5 x 5 x 2 = 1250	2 x 5 x 5 x 5 = 250

TABLE 2: FEATURE MODELS USED IN THIS STUDY

ID	Feature model	Features	CTCs	Total rules	ID	Feature model	Features	CTCs	Total rules
FM-43	Web portal	43	6	63	FM-66	bCMS system	66	2	109
FM-43B	Mobile Media 2	43	3	65	FM-67	HIS	67	4	121
FM-44	Documentation Generation	44	8	68	FM-70	DATABASE TOOLS	70	2	82
FM-45	Android SPL	45	5	74	FM-72	Car Selection	72	18	123
FM-46	DELL Notebook Computers	46	110	171	FM-72B	Reuso - UFRJ - Eclipse1	72	1	97
FM-52	Linea de Experimentos	52	4	87	FM-88	Billing	88	59	160
FM-53	Video Player	53	2	78	FM-94	Coche ecologico	94	2	151
FM-60	Smart Home v2.2	60	2	82	FM-97	UP structural	97	1	138
FM-61	Arcade Game PL	61	34	122	FM-137	xtext	137	1	179
FM-63	OW2-FraSCAti-1.4	63	46	129	FM-290	E-Shop	290	21	426

D. Defining the Optimization Objectives

In this work we optimize the following objectives:

1- Correctness; i.e. compliance to the relationships and constraints defined in the feature model. Since jMetal treats all optimization objectives as minimization objectives, we seek to minimize rule violations.

2- Richness of features; we seek to minimize the number of deselected features.

3- Features that were used before; we seek to minimize the number of features that weren't used before.

4- Known defects; which we seek to minimize.

5- Cost; which we seek to minimize.

E. Run Time as Stopping Criterion

In this experiment, we specify run time as the stopping criterion, rather than the commonly used approach of stopping after a given number of fitness evaluations. The number of evaluations is proportional to the total run time and the required CPU power. Yet, the total run time is affected by many other algorithm-dependent operations, including the fitness ranking of individuals in each generation. This leads to varying runtimes with the same number of evaluations. For instance, we noticed that IBEA took five times longer than NSGA-II to perform the same number of evaluations, which meant that IBEA spent far more time in fitness ranking than NSGA-II. We are of the opinion that each algorithm should be given a fixed amount of time to calculate its best approximation of the Pareto front. A better algorithm should score better on the quality indicators (HV, %correct) within that duration of time. Going back to the comparison between IBEA and NSGA-II, if both are given the same duration of time, then NSGA-II would perform far more evaluations than IBEA, and thus would be given a better chance to improve its results. As we will see in the coming section, providing NSGA-II with the chance to evolve more generations did not help it to overcome IBEA at producing more correct solutions or better HV.

V. STATISTICAL ANALYSIS METHOD

We performed the same analysis done by Arcuri and Fraser [1], in which they performed two-way comparisons composed of two parts: effect size and statistical significance.

For effect size, the Vargha-Delaney A measure [12] was used, as implemented in R by Thomas et al [11]. It tells us how often, on average, one technique outperforms the other. When applied to two populations such as the results of two techniques, the A measure is a value between 0 and 1: when the A measure is exactly 0.5, then the two techniques achieve equal performance; when A is less than 0.5, the first technique is worse; and when A is more than 0.5, the second technique is worse. The closer to 0.5, the smaller the difference between the techniques; the farther from 0.5, the larger the difference. [11]

For statistical significance, the Mann-Whitney U-test was used as implemented in R.

VI. RESULTS

This section has two parts, the main results, and the parameter training results.

A. Main Results

Table 3 shows a summary of the main results. For each feature model (FM), we show the average over 15 runs of the percentage of correct configurations (%correct) found by best parameter setting and the default parameter setting for both IBEA and NSGA-II. Then we calculate three effect size values:

1. $\hat{A}_{BDI}$: Vargha-Delaney effect size of best compared to default for IBEA.
2. $\hat{A}_{BDN}$: Vargha-Delaney effect size of best compared to default for NSGA-II.
3. $\hat{A}_{BIN}$: Vargha-Delaney effect size of best IBEA compared to best NSGA-II.

We also calculate the Mann-Whitney statistical significance measure, and we highlight the A-measure in bold when the Mann-Whitney p-value is less than 5%.

It is worth mentioning here that we don't list the worst performance values for IBEA and NSGA-II (as Arcuri and Fraser [1] do in their study) since those values were always 0% for all experimental runs.

The results in table 3 show the following:

1. For both IBEA and NSGA-II, the best parameter tuning beats the default parameter values by a large and significant amount in 15 out of 20 models. For the other 5 models there is improvement, but it is neither large nor statistically significant.
2. The best IBEA results beat the best NSGA-II results in 19 out of 20 models. The improvement was large and significant in 17 out of 19 models. In the one case where NSGA-II results beat IBEA results, the improvement was neither large nor significant.

Thus we are able to answer the first 3 research questions:

RQ1: How Large is the Potential Impact of a Wrong Choice of Parameter Settings?

We confirm Arcuri and Fraser's [1] conclusion: *Different parameter settings cause very large variance in the performance*.

RQ2: How Does a "Default" Setting Compare to the Best and Worst Achievable Performance?

Arcuri and Fraser [1] concluded that: Default parameter settings perform relatively well, but are far from optimal on individual problem instances.

With our results from table 3, we are able to make a stronger conclusion: *Default parameter settings perform generally poorly, but might perform relatively well on individual problem instances*.

RQ3: How does the performance of IBEA's best tuning compare to NSGA-II's best tuning?

Our results show that *IBEA's best tuning performs generally much better than NSGA-II's best tuning*. This RQ is of no concern to Arcuri and Fraser [1], but it confirms previous findings by the authors [10]. Specifically, we verify here that no parameter settings enable NSGA-II to achieve acceptable levels of correctness and optimality for the leaned configurations. This results was expected since we have identified the core fitness assignment method as the reason for IBEA's advantage over NSGA-II and other Pareto-based algorithms. IBEA is able to exploit the user preferences in ranking the solutions for selection to generate new solutions and to survive through the evolutionary process; whereas NSGA-II depends on absolute dominance as the deciding criterion, coupled with crowd pruning as a mechanism to force solution diversity, which ignores rich details from the user preferences. [10]

TABLE 3: SUMMARY OF MAIN RESULTS

FM	*Best IBEA*	*Default IBEA*	*Best NSGA-II*	*Default NSGA-II*	$\hat{A}_{BDI}$	$\hat{A}_{BDN}$	$\hat{A}_{BIN}$
FM-43	98%	91%	20%	7%	0.63	**1**	**1**
FM-43B	48%	43%	14%	1.3%	0.58	**1**	**1**
FM-44	60%	51%	13%	1.4%	**0.76**	**1**	**1**
FM-45	27%	11%	9%	0.8%	**0.88**	**1**	**1**
FM-46	5%	0%	0.3%	0%	**0.8**	**0.80**	0.68
FM-52	22%	2%	23%	7%	**0.95**	**1**	0.55
FM-53	79%	71%	27%	16%	0.53	**1**	**1**
FM-60	85%	62%	14%	11%	0.61	**0.86**	**1**
FM-61	39%	9%	21%	5%	**0.90**	**1**	**0.86**
FM-63	54%	19%	11%	1.0%	**0.98**	**1**	**1**
FM-66	14%	0%	5%	0.3%	**0.9**	0.66	**0.85**
FM-67	14%	0%	5%	0.6%	**0.93**	0.59	**0.86**
FM-70	100%	100%	10%	2%	0.5	**1**	**1**
FM-72	27%	13%	0.7%	0.2%	**0.82**	0.47	**1**
FM-72B	97%	66%	10%	1.0%	0.64	**1**	**1**
FM-88	27%	0%	12%	0.3%	**0.90**	**1**	**0.80**
FM-94	14%	0%	0.7%	0%	**0.97**	0.53	**0.95**
FM-97	67%	22%	3%	0.1%	**0.82**	0.61	**0.96**
FM-137	96%	23%	23%	0.3%	**0.86**	**1**	**1**
FM-290	37%	0%	22%	0%	**0.73**	**1**	0.45

B. Parameter Training Results

Next, we examine the possibility of finding the best parameter values for a feature model by training on the 19 other feature models. To achieve this, we computed the overall average for the %correct indicator achieved by each configuration across all feature models.

We found that the best configuration overall, and also the best whenever an individual FM is removed, was {IBEA, Population = 50, Crossover rate = 0, Mutation rate = 0.5/FEATURES}.

The performance under this set of values is compared to the best performance achieved for each FM and also to the performance under default values, and the results are shown in table 4. We calculate $\hat{A}_{TD}$, the Vargha-Delaney effect size for the "trained" tuning compared to the default tuning; and $\hat{A}_{TB}$, the effect size for the "trained" tuning compared to the best tuning. We highlight the result in bold if the Mann-Whitney test shows statistical significance.

The results in table 4 show the following:

1. In general, the trained parameter settings achieve better results than the default settings. This is true for 16 out of 20 feature models, and significant in 10 out of 16.
2. For 12 FMs, the best tuning was better than the trained tuning by a large and significant margin. The remaining 8 FMs had small and insignificant differences.

Thus we provide the following answer to RQ4.

RQ4: If we Tune a Search Algorithm Based on a Set of Feature Models, How Will Its Performance Be On Other Feature Models?

We answer by: *Tuning on a sample of problem instances does not, in general, result in the best parameter values for a new problem instance*, *but the obtained setting are generally better than the defaults settings*.

On this RQ, Arcuri and Fraser [1] find that: Tuning should be done on a very large sample of problem instances. Otherwise, the obtained parameter settings are likely to be worse than arbitrary default values. This is because they obtained mixed, and generally poor, parameter settings with the training technique. In our experiment, the result of training was uniform, i.e. the same settings were obtained for all feature models, and the performance compared well with the best and default settings. This again points to the difference in the problem domains and model structure, which affects the way the algorithms need to be tuned in order to best explore the search space at hand.

TABLE 4: TRAINED TUNING COMPARED TO BEST AND DEFAULT TUNING

FM	*Best*	*Trained*	*Default (IBEA)*	$\hat{A}_{TD}$	$\hat{A}_{TB}$
FM-43	98%	88%	91%	**0.26**	**0**
FM-43B	48%	46%	43%	0.58	0.52
FM-44	60%	57%	51%	0.65	**0.28**
FM-45	27%	26%	11%	**0.84**	0.53
FM-46	5%	2%	0%	**0.67**	0.33
FM-52	23%	21%	2%	**0.85**	0.59
FM-53	79%	67%	71%	**0.28**	**0.21**
FM-60	85%	77%	62%	0.49	**0.24**
FM-61	39%	43%	9%	**0.96**	0.56
FM-63	54%	51%	19%	**0.95**	0.43
FM-66	14%	6%	0%	**0.67**	**0.30**
FM-67	14%	3%	0%	0.60	**0.16**
FM-70	100%	81%	100%	**0.07**	**0.07**
FM-72	27%	17%	13%	0.56	**0.12**
FM-72B	97%	95%	66%	0.56	**0.29**
FM-88	27%	25%	0%	**0.9**	0.39
FM-94	14%	7%	0%	**0.7**	**0.29**
FM-97	67%	52%	22%	**0.79**	**0.26**
FM-137	96%	90%	23%	**0.87**	0.5
FM-290	37%	3%	0%	0.53	**0.29**

VII. Threats to Validity

A. Threats to Construct Validity

Threats to construct validity are the confounding factors that are not considered in the study. One such factor is the search budget, i.e. the amount of time that is allocated for optimizing each configuration. As the search budget increases, the differences in performance among different parameter settings start to disappear. Still, restricting the run time to 10 seconds puts the algorithms under stress to achieve interactive-grade response, i.e. response within time to keep the attention of a live user working to interactively configure the software package. Arcuri and Fraser address the search budget as a factor in their study [1], but we leave it to future work.

Another possible factor is the use of synthetic data as attributes of features, i.e. COST, DEFECTS, and USED_BEFORE. The use of synthetic data is common in software engineering literature. The difficulty of obtaining real data comes from the fact that such numbers are usually associated with software components, not features. When available, such data is often proprietary and not published. Nevertheless, the results we obtained have such a large margin of superiority achieved by IBEA over other algorithms which couldn't possibly be biased by the synthetic data. Future work should attempt to collect real data for use with other MEOAs to optimize product configuration.

B. Threats to Internal Validity

A possible threat to internal validity is the fact that the MEAOs are implemented in Java with its garbage collection utility, which lies outside of any programmer's control. Nevertheless, the use of statistical testing should alleviate this threat and assure the strength of conclusions.

C. Threats to Conclusion Validity

In comparing the performance of MEOAs, we performed statistical significance testing (Mann-Whitney U test) on 15 different runs of each MEOA. The goal was to remove the randomized nature of the algorithms as a confounding factor. It may be argued that 15 runs are not enough for a strong conclusion, but we achieved high significance (low p-values) for most of the comparisons, which supports our conclusions.

D. Threats to External Validity

A threat to external validity is that we are unable to generalize our findings to other evolutionary algorithms, or other software engineering problems. Nevertheless, when combining the results of this work with the original study, which used Genetic Algorithms applied to test case generation for OO software, we can make a strong case for the conclusions and encourage further empirical investigations.

VIII. Conclusions and Future Work

The results of this replication confirm the findings in in the original study by Arcuri and Fraser [1], and even make a stronger case for parameter tuning as opposed to default values, as in our answer to RQ2. This confirms what is generally known as the "no-free-lunch" theorems for optimization [13], which deny the existence of "one-size-fits-all" algorithm or configuration of an algorithm. We also confirm the findings by Sayyad et al [10] on the superiority of IBEA over NSGA-II, by proving that the best parameter tuning for NSGA-II still can not outperform the best parameter tuning for IBEA.

One research question considered by Arcuri and Fraser [1] was not considered here, which relates to the dependency of our result on the search budget, i.e. the time allocated for finding an optimum solution. We have used a fixed search duration (10 seconds), and we do expect that parameter tuning would lose significance when the algorithm is given an extended time to run, as asserted by Arcuri and Fraser [1]. Such experiment, however, would consume a much longer time than the experiment performed here, and thus was not done in this paper. We may also consider other methods for parameter tuning such as those in the extended work by Arcuri and Fraser [2].

References

[1] A. Arcuri and G. Fraser, "On Parameter Tuning in Search Based Software Engineering," in Proc. SSBSE, 2011, pp. 33-47.

[2] A. Arcuri and G. Fraser, "Parameter Tuning or Default Values? An Empirical Investigation in Search-Based Software Engineering," Empir Software Eng, Feb 2013.

[3] D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," Information Systems, vol. 35, no. 6, pp. 615-636, 2010.

[4] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and Variability in Software Engineering," IEEE Software, vol. 15, no. 6, pp. 37-45, 1998.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, 2002.

[6] J.J. Durillo and A.J. Nebro, "jMetal: A Java Framework for Multi-Objective Optimization," Advances in Engineering Software, vol. 42, pp. 760-771, 2011.

[7] S. Luke, Essentials of Metaheuristics. 2009: Lulu. [Online]. http://cs.gmu.edu/~sean/book/metaheuristics/

[8] M. Mendonca, M. Branco, and D. Cowan, "S.P.L.O.T. - Software Product Lines Online Tools," in Proc. OOPSLA, Orlando, USA, 2009.

[9] K. Pohl, G. Böckle, and F.J. van der Linden, Software Product Line Engineering. New York: Springer-Verlag, 2005.

[10] A.S. Sayyad, T. Menzies, and H. Ammar, "On the Value of User Preferences in Search-Based Software Engineering: A Case Study in Software Product Lines," in Proc. ICSE, San Francisco, USA, 2013, pp. 492-501.

[11] S.W. Thomas, H. Hemmati, A.E. Hassan, and D. Blostein, "Static test case prioritization using topic models," Empirical Software Engineering, July 2012.

[12] A. Vargha and H.D. Delaney, "A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong," Journal of Educational and Behavioral Statistics, vol. 25, no. 2, pp. 101-132, 2000.

[13] D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67-82, 1997.

[14] E. Zitzler and S. Kunzli, "Indicator-Based Selection in Multiobjective Search," in Parallel Problem Solving from Nature. Berlin, Germany: Springer-Verlag, 2004, pp. 832–842.

[15] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," IEEE Transactions on Evolutionary Computation, vol. 3, no. 4, pp. 257–271, 1999.