A Review On Parameter Tuning In Search Based Software Engineering

Amritanshu Agrawal, North Carolina State University, aagrawa8@ncsu.edu

Abstract—**When applying search-based software engineering (SBSE) techniques one is confronted with a multitude of different parameters that need to be chosen: Which population size for a genetic algorithm? Which selection mechanism to use? What settings to use for dozens of other parameters? This problem not only troubles users who want to apply SBSE tools in practice, but also researchers performing experimentation. This review provides a look on these parameters and how can we play with the parameters. Various studies have been done on this regard and based on some of the highly cited papers, I am giving an overview on how to handle parameter tuning.**

***Keywords*—Search based, Software Engineering, parameter tuning, control, testing.**

I. Introduction

Search-based techniques have been shown to be a promising approach to tackle many kinds of software engineering tasks, particularly software testing. Although automated generation of test cases for structural coverage has received particular attention. This is because many of the different parameters that influence search-based software testing (SBST) are not well understood. Investigating these techniques is therefore of practical value.

Some People prefer Statistical testing which is a highly effective technique for generating test inputs for software verification. Test data is sampled from a probability distribution over the input domainin a similar manner to random testingwhere the probability distribution is carefully chosen so that every part of the software is covered by the test set.

The issue of setting the values of various parameters of an evolutionary algorithm is crucial for good performance. Researchers have paid special attention to setting parameters on-the-fly. This has the potential of adjusting the algorithm to the problem while solving the problem. I chose to interleave a number of examples throughout the text. Thus I hope to both clarify the points I wish to raise as I present them, and also to give the reader a feel for some of the many possibilities available for controlling different parameters. Finding the appropriate setup for an evolutionary algorithm is a long standing challenge of the field. The main problem is that the description of a specific EA contains its components, such as the choice of representation, selection, recombination, and mutation operators, thereby setting a framework while still leaving quite a few items undefined. For instance, a simple GA might be given by stating it will use binary representation, uniform crossover, bit-flip mutation, tournament selection, and generational replacement. For a full specification, however, further details have to be given, for instance, the population size, the probability of mutation and crossover, and the tournament size. These data called the algorithm parameters or strategy parameters.

Globally, there are two major forms of setting parameter values: parameter tuning and parameter control. By parameter tuning, it is meant the commonly practised approach that amounts to finding good values for the parameters before the run of the algorithm and then running the algorithm using these values, which remain fixed during the run. Parameter control forms an alternative, as it amounts to starting a run with initial parameter values that are changed during the run.

Parameter tuning is a typical approach to algorithm design. Such tuning is done by experimenting with different values and selecting the ones that give the best results on the test problems at hand. However, the number of possible parameters and their different values means that this is a very time-consuming activity. Considering four parameters and five values for each of them, one has to test 625 different setups. Performing 100 independent runs with each setup, this implies 62,500 runs just to establish a good algorithm design.

Rest of the paper is classified as section 2 talks about the background. Section 3 talks about the review on various papers work in the fields of parameter tuning. Section 4 gives the final conclusion and discussion.

II. Background

Eiben et al. [1] presented a survey on how to control and set parameter values of evolutionary algorithms. A particular value that is good at the beginning of the search might become sub-optimal in the later stages. For example, in a genetic algorithm one might want to have a high mutation rate (or large population size) at the beginning of the search, and then decrease it in the course of the evolution; this would be conceptually similar to temperature cooling in simulated annealing. In this paper we only deal with parameter tuning. Parameter control is a promising area of research, but mainly unexplored in SBSE.

Smit and Eiben [2] carried out a series of experiments on parameter tuning. They consider the tuning of six parameters of a genetic algorithm applied to five numerical functions, comparing three settings: a default setting based on common wisdom, the best tuning averaged on the five functions (which they call generalist), and the best tuning for each function independently (specialist). Only one fixed search budget (i.e., maximum number of fitness evaluations as stopping criterion) was considered. Evaluating all possible parameter combinations is infeasible in practice. Techniques to select only a subset of configurations to test that have high probability of being optimal exist, for example regression trees (e.g., used in [3]) and response surface methodology (e.g., used in [4]). The goal of this paper is to study the effects of parameter tuning, which includes also the cases of sub-optimal choices. Such type of analysis requires an exhaustive evaluation. This is done only for the sake of answering research questions (as for example to study the effects of a sub-optimal tuning). In general, a practitioner would be interested only in the best configuration.

Drawback of the parameter tuning approach recall how we defined it: finding good values for the parameters before the run of the algorithm and then running the algorithm using these values, which remain fixed during the run. However, a run of an EA is an intrinsically dynamic, adaptive process. The use of rigid parameters that do not change their values is thus in contrast to this spirit. Additionally, it is intuitively obvious, and has been empirically and theoretically demonstrated, that different values of parameters might be optimal at different stages of the evolutionary process. The use of static parameters itself can lead to inferior algorithm performance. A straightforward way to overcome the limitations of static parameters is by replacing a parameter p by a function p(t), where t is the generation counter. Designing optimal dynamic parameters (that is, functions for p(t)) may be even more difficult. A well-known instance of this problem occurs in simulated annealing [5] where a so-called cooling schedule has to be set before the execution of the algorithm.

It is thus a natural idea to use an EA for tuning an EA to a particular problem. This could be done using two EAs: one for problem solving and another one the so-called meta-EA to tune the first one. It could also be done by using only one EA that tunes itself to a given problem, while solving that problem. Self adaptation, as introduced in Evolution Strategies for varying the mutation parameters, falls within this category. In the next section we discuss various options for changing parameters, illustrated by an example.

III. Review

A. On parameter tuning in search based software engineering [6]

(i) KEYWORDS:

(i1) Search Based Software Engineering: SBSE converts a software engineering problem into a computational search problem that can be tackled with a metaheuristic. This involves defining a search space, or the set of possible solutions. This space is typically too large to be explored exhaustively, suggesting a metaheuristic approach.

(i2) Parameter Tuning: Parameter-tuning meant to optimize parameters of a classifier in order to maximize or minimize its objective; typically the maximization of efficiency or error minimization.

(i3) Crossover Rate: Whenever two individuals are selected from the parent generation, this parameter specifies the probability with which they are crossed over. If they are not crossed over, then the parents are passed on to the next stage (mutation), else the offspring resulting from the crossover are used at the mutation stage.

(i4) Elitism Rate: Elitism describes the process that the best individuals of a population (its elite) automatically survive evolution.

(ii) KEYPOINTS:

(ii1) Motivation: Current researches don't talk about what population size to use for a genetic algorithm, which selection mechanism will be best? What other parameters to use to get the best result?

(ii2) Data: They have generated the data using object-oriented software using EVOSUITE tool. This data consists of 20 software classes as case study. They considered this much to finsih the experiments in feasible time but still this led to more than one million experiments.

(ii3) New Results:

* Different Parameter settings cause very large variance in the performance.

* Default parameter settings perform relatively well, but are far from optimal on individual problem instances.

* 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.

* Available search budget has strong impact on the parameter settings that should be used.

(iii) IMPROVISATIONS:

* A problem related to tuning is the search budget too. With a genetic algorithm population size will be more and this will take more computational time. Search budget work could have done separately to not conflict the results.

* Some problems cannot be deterministically found in reasonable time. This area could be a potential to work on.

B. Comparing mining algorithms for predicting the severity of a reported bug [7]

(i) KEYWORDS:

(i1) Naive Bayes Classifier: Naive Bayes is a technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values. It is a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. Naive Bayes model works without accepting Bayesian probability or using any Bayesian methods.

(i2) Severity: The impact of a bug on the successful execution of the software system. It is an absolute classification.

(i3) Stemming and stopwords: Stemming is the process for reducing inflected (or sometimes derived) words to their word stem, base or root formgenerally a written word form. Stop words are words which are filtered out before or after processing of natural language data. Though stop words usually refer to the most common words in a language.

(i4) Tf-idf: Term frequencyinverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The number of times a term occurs in a document is called its term frequency. An inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

(ii) KEYPOINTS:

(ii1) Motivation: Severity of a bug is very important for a smooth working of a system. For predicting bug report severity, we should have a good classification algorithm. How much data/bug reports are necessary to train a data to get a reliable prediction? Are there couple of classifies which can predict severity very well?

(ii2) Hypotheses:

* Assumptions: Report contains summary of observed malfunction and a longer description. The reports are posted by users who have technical knowledge so they will write a detailed technical report.

* Experiment: Extraction and organization of bug reports. Reports are prepocessed using the common steps of stopwords removal, stemming. Reports are categorized into severe and non-severe and classifier is trained. A number of classifiers are trained like Naive bayes, Naive Bayes Multinomial, 1-Nearest neighbor, SVM. For training the Tf-idf scores are maintained for feature extraction from data. They have used TPR (The rate of true positives) and FPR (the rate of false positives) for concluding the results.

(ii3) New Results:

* The Naive Bayes Multinomial classifier has the best accuracy and is also the fastest when classifying the severity of reported bugs.

* The Naive Bayes and Naive Bayes Multinomial classifiers are able to achieve stable accuracy the fastest. Also, we need about 250 bug reports of each severity when we train our classifier.

* Each component tends to have its own particular way of describing severe and non-severe bugs. Thus, terms which are good indicators of the severity are usually component-specific.

(iii) IMPROVISATIONS:

* The classifier is trained on component basis, but a single report can belong to different components. Instead of single lables they should consider multi labels.

* They are dependent too much on the users giving detailed reports of bugs. There can be users who don't have that much technical knowledge.

* They have considered only 2 softwares bugs Eclipse and Genome. Validity needs to be checked on other softwares too.

* Reports aren't reliable as what they enter in description is according to their senses. But they might fall into different category.

* Duplicate bug reports will shift the training curve to fall each report into 1 category.

C. Parameter control in evolutionary algorithms [8]

(i) KEYWORDS:

(i1) Single-point crossover: A single crossover point on both parents' organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms. The resulting organisms are the children.

(i2) Bit-flip mutation: Bit-flip mutation is a common mutation operator for evolutionary algorithms applied to optimize functions over binary strings. Computing the probability distribution of fitness values of a binary string undergoing uniform bit-flip mutation. This probability distribution can be expressed as a polynomial in p, the probability of flipping each bit.

(i3) Multivariate normal distribution: The multivariate normal distribution or multivariate Gaussian distribution, is a generalization of the one-dimensional (univariate) normal distribution to higher dimensions. One possible definition is that a random vector is said to be k-variate normally distributed if every linear combination of its k components has a univariate normal distribution.

(i4) Covariance matrix: A covariance matrix (also known as dispersion matrix or variancecovariance matrix) is a matrix whose element in the i, j position is the covariance between the i-th and j-th elements of a random vector. Each element of the vector is a scalar random variable, either with a finite number of observed empirical values or with a finite or infinite number of potential values specified by a theoretical joint probability distribution of all the random variables.

(ii) KEYPOINTS:

(ii1) Motivation: Setting the values of various parameters of an evolutionary algorithm is crucial for good performance. Special attention to setting parameters on-the-fly. This has the potential of adjusting the algorithm to the problem while solving the problem.

(ii2) Hypotheses: Parameters are not independent, but trying all different combinations systematically is practically impossible. The process of parameter tuning is time consuming, even if parameters are optimised one by one, regardless of their interactions. For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.

(ii3) Patterns: Evolution strategies:

* Gaussian Mutations: The main operator of ES is the Gaussian mutation, that adds centered normally distributed noise to the variables of the individuals. The most general Gaussian distribution is the multivariate normal distribution.

* Adapting the Step-Size: The step-size of the Gaussian mutation gives the scale of the search.

* Self-Adaptive ES: the parameters of the mutation (both the step-size and the covariance matrix) are attached to each individual, and are subject to mutation, too. Those personal mutation parameters range from a single step-size, leading to isotropic mutation, where all coordinates are mutated independently with the same variance, to the non-isotropic mutation

* CMA-ES: a Clever Adaptation: The basic idea in CMA-ES is to use the path followed by the algorithm to deterministically update the different mutation parameters.

(ii4) Sampling Procedures:

In classifying parameter control techniques of an evolutionary algorithm, many aspects can be taken into account. For example:

1. What is changed? (e.g., representation, evaluation function, operators, selection process, mutation rate, population size, and so on)

2. How the change is made (i.e., deterministic heuristic, feedback-based heuristic, or self-adaptive)

3. The evidence upon which the change is carried out (e.g., monitoring performance of operators, diversity of the population, and so on)

4. The level of change (e.g., population-level, individual-level, and so forth)

(iii) IMPROVISATIONS:

* Varying Several Parameters Simultaneously.

* Population, Selection, Crossover, Mutation. These all operators need to be played with.

* Evaluation function needs to be verified with other functions too.

D. A principled evaluation of the effect of directed mutation on search-based statistical testing [9]

(i) KEYWORDS:

(i1) Statistical testing: Statistical testing generates test inputs by sampling from a probability distribution that is carefully chosen so that the inputs exercise all parts of the software being tested.

(i2) Directed mutation: When evaluating potential solutions, feedback is normally returned to the search algorithm in the form a fitness metric, and this information is used to guide the algorithms selection operator.

(i3) Adaptive mutation: The overall mutation rate is modifiedover time, based on the recent changes in fitness, or by self-adaption as part of the representationbut the mutation operator itself is unchanged.

(i4) Response surface methodology: RSM explores the relationships between several explanatory variables and one or more response variables.

(ii) KEYPOINTS:

(ii1) Motivation: Sets of inputs are needed to detect more faults than test sets generated using traditional random and structural testing techniques. Significant improvement in algorithm performance, and so increases both the cost-effectiveness and scalability of search-based statistical testing.

(ii2) Hypotheses: The search algorithm proposed in this paper has the following key features :-

a) Representation: They treated each argument independently since they will interact with each other in the code. Therefore a joint (multivariate) distribution is necessary, and a Bayesian network is used to represent such a distribution. A Bayesian network is a directed acyclic graph.

b) Fitness metric - The fitness metric used here is an estimate of the probability lower bound achieved by the candidate probability distribution.

c) Search Method: The search method used is stochastic hill climbing.

(ii3) Patterns: To implement directed mutation, they created three mutation groups formed using the mutation operators:

* edge consists of the operators that modify edges in the Bayesian network: add and rem.

* bins consists of the remaining operators that modify bins directly: len, spl, joi, and prb, and applies to all bins.

* drct consists of the same bin-modifying operators as bins, but applies them only to bins that contributed input vectors that executed the least-exercised structural element.

(ii4) Sampling Procedures:

The choice of starting point, i.e., the initial algorithm parameter settings. This may have little effect if there isas RSM assumesa single optimum reachable by hill-climbing, but if there are multiple local optima, then the starting point may have a significant effect on the outcome. One possibility is to pick a random starting point and/or use a number of different starting points. The identification of the minimum along the path of steepest decision.

(iii) IMPROVISATIONS:

* There is a need to keep playing with the representation, fitness metric and search method itself with the objective of further performance improvements.

* Search based statistical testing can be enhanced to generate test sequences, and to accommodate more complex test input data types.

E. The seed is strong: Seeding strategies in search-based software testing [10]

(i) KEYWORDS:

(i1) Seeding: Seeding is referred to any technique that exploits previous related knowledge to help solve the testing problem at hand.

(i2) Control dependence graph : A control dependence graph is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

(i3) Tournament: For each of the N positions it is needed to fill in the population, we generate 10 random test suites, and add only the one that has highest fitness.

(i4) AllMethods: During the initialization, each time a new method call is inserted, it is not chosen randomly; instead, it is chosen based on a ring buffer of all methods.

(ii) KEYPOINTS:

(ii1) Motivation: The objective of such a search is to automatically generate test suites that maximize branch coverage. The problem should be solvable even without using previous knowledge.

(ii2) Patterns: SEEDING STRATEGIES FOR CLASS TESTING :

* Evolutionary Testing of Classes

* Seeding Constants

* Optimizing the Initial Population

* Incorporating Previous Solutions

(ii3) Sampling Procedures:

- What is the impact of using constants from the bytecode for seeding?

- Which are the best pre-processing techniques to seed an improved population before starting a SBST search?

- Given a solution to improve from, which are the best seeding strategies to initialize a new population for SBST?

(iii) IMPROVISATIONS:

* Threats to construct validity are on how the performance of a testing technique is defined.

* Threats to internal validity might come from how the empirical study was carried out.

* Studying the interactions/relations of the different parameter configurations of EVOSUITE (e.g., population size and crossover probability) with the seeding strategies and the chosen search budget.

IV. Conclusion

Summarising this paper a number of things can be noted. First, parameter control in an EA can have two purposes. It can be done to avoid suboptimal algorithm performance resulting from suboptimal parameter values set by the user. The basic assumption here is that the applied control mechanisms are intelligent enough to do this job better than the user could, or that they can do it approximately as good, but they liberate the user from doing it. Either way, they are beneficial. The other motivation for controlling parameters on-thefly is the assumption that the given parameter can have a different optimal value in different phases of the search. If this holds, then there is simply no optimal static parameter value; for good EA performance one must vary this parameter.

Acknowledgment

I would like to thank Dr Tim Menzies for giving me opportunity to explore the content in the Automated Software Engineering. I have gained sufficient knowledge in this area and more importantly the procedure to go about the literature review.

References

[1] Eiben, A., Michalewicz, Z., Schoenauer, M., Smith, J.: Parameter control in evolutionary algorithms. Parameter Setting in Evolutionary Algorithms, 1946 (2007)

[2] Smit, S., Eiben, A.: Parameter tuning of evolutionary algorithms: Generalist vs. specialist. Applications of Evolutionary Computation, 542551 (2010)

[3] Bartz-Beielstein, T., Markon, S.: Tuning search algorithms for real-world applications: A regression tree based approach. In: IEEE Congress on Evolutionary Computation (CEC), pp. 11111118 (2004)

[4] Poulding, S., Clark, J., Waeselynck, H.: A principled evaluation of the effect of directed mutation on search-based statistical testing. In: International Workshop on Search-Based Software Testing, SBST (2011)

[5] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated anealing. Science, 220:671680, 1983.

[6] Arcuri, Andrea, and Gordon Fraser. ”On parameter tuning in search based software engineering.” In Search Based Software Engineering, pp. 33-47. Springer Berlin Heidelberg, 2011.

[7] Lamkanfi, Ahmed, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. ”Comparing mining algorithms for predicting the severity of a reported bug.” In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pp. 249-258. IEEE, 2011

[8] Eiben, Agoston E., Zbigniew Michalewicz, Marc Schoenauer, and James E. Smith. ”Parameter control in evolutionary algorithms.” In Parameter setting in evolutionary algorithms, pp. 19-46. Springer Berlin Heidelberg, 2007.

[9] Poulding, Simon, John A. Clark, and Hlne Waeselynck. ”A principled evaluation of the effect of directed mutation on search-based statistical testing.” In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, pp. 184-193. IEEE, 2011.

[10] Fraser, Gordon, and Andrea Arcuri. ”The seed is strong: Seeding strategies in search-based software testing.” In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pp. 121-130. IEEE, 2012