Increasing the Efficiency of Search-Based Unit Test Generation using Parameter Control

Thomas White
The University of Sheffield, Department of Computer Science
Regent Court, 211 Portobello
Sheffield, S1 4DP, United Kingdom
tdwhite1@sheffield.ac.uk

ABSTRACT

Automatically generating test suites with high coverage is of great importance to software engineers, but this process is hindered by the vast amount of parameters the tools use to generate tests. Developers usually lack knowledge about the workings of the tools that generate test suites to set the parameters to optimal values, and the optimal values usually change during runtime. *Parameter Control* automatically adapts parameters during test generation, and has shown to help solve this problem in other areas. To investigate any improvements parameter control could have in search-based generation of test suites, we adapted multiple methods of controlling mutation and crossover rate in EvoSuite, a tool that automatically generates unit test suites. Upon evaluation, clear benefits to controlling parameters were found, but surprisingly, controlling some parameters can sometimes be more harmful to the search than beneficial through increased computation costs.

Categories and Subject Descriptors. D.2.5 [Software Engineering]: Testing and Debugging – *Testing Tools*;

Keywords. Parameter control, genetic algorithm, testing, test suite generation, search based software engineering

1. INTRODUCTION

Object-oriented classes can be automatically tested using *Search-Based Software Testing* (SBST) [6]. One such way to achieve this is by using *Genetic Algorithms* (GA) [9]. A genetic algorithm follows Darwin's theory of Natural Selection [3] to solve a complex problem; searching through an infinite search space with only a function that evaluates an individuals fitness as guidance on which way the search should go.

Algorithms like GAs use numerous parameters which effect runtime. Arcuri and Fraser found that setting these parameters to poor values leads to inadequate results [2], and this was reinforced by Sayyad et al. [10]. Tuning these parameters to an optimal setting before the algorithm starts is difficult, and most software developers usually lack the knowledge to set these parameters to a suitable value. Furthermore, the optimal values are also likely to fluctuate during runtime, not remain constant [2, 7].

Solving the parameter problem would lead to an increase in efficiency of search-based test generation. This would improve the quality of output from the search-based test generation tools. An example of an improvement would be a higher code coverage on unit tests produced. Research on *Parameter Control* (PC) aims to automatically adapt parameters throughout the search [4, 7]. As parameter control was successfully applied in other domains, this raises the question whether it could also be applied to search-based test generation [1].

During this project, we adapted and applied different parameter control methods to unit test generation, specifically the EvoSuite [5] search-based unit test generator, and evaluated any observations found during runtime.

When controlling the parameters in EvoSuite, some methods of parameter control outperformed others, leading to an increase in the final fitness of the output (unit test suite) produced. This was observed to increase code coverage on unit tests produced by as high as 13%. In these cases, we can imply that the genetic algorithm is more efficient with parameter control. Surprisingly, we discovered that controlling the different parameters in EvoSuite can also lead to complications, such as increased fitness evaluation costs which countered the positive effects of having improved parameters leading to a decrease in code coverage. Further research is needed to take into account the extra computation time required when controlling parameters. The full findings are published in [9].

2. PARAMETER CONTROL

There are two major forms for setting parameters, parameter control (PC) and parameter tuning [4]. Parameter control modifies parameters during runtime of a program whereas parameter tuning finds some static optimal value before the run starts and never changes them. More on parameter tuning can be found in [2] but it can be deduced that changing parameters dynamically during runtime finds more suitable values than tuning or manual assignment [4, 11].

There are three main types of parameter control [4]. Firstly, deterministic, which uses no feedback from the algorithms current state and commonly only uses current resource vs maximum resource (e.g. current iteration vs maximum iteration) to control parameters. Deterministic excels where there is a pattern for a parameter to follow to increase the effectiveness the parameter has. An example of this would be

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08
http://dx.doi.org/10.1145/2786805.2807556

mutation rate, which benefits from starting high to diversify individuals (*exploration*), and finishing low to construct the best individual from the current population (*exploitation*).

The second type is adaptive which works by tweaking parameter values every iteration depending on the fitness impact of the parameter. If one parameter is outperforming another, then it will be rewarded with greater impact in the genetic algorithm, and if it is under-performing, punished with a negative change. An example of an adaptive method is by Ming et al [8], and computes the crossover rate based on two individuals and their similarity. Ming et al aimed to limit crossover from contrasting individuals as when this happened, less fit offspring were produced [8].

The third type is self adaptive, where parameters are assigned to each individual. Self adaptive works due to individuals with more suitable parameters evolving superior and surviving longer in the genetic algorithm, allowing their parameters to spread to other individuals through crossover. Self adaptive is useful in that it can control any number of parameters easily. This was taken advantage of when comparing control of mutation to crossover rate. We ran 3 self adaptive experiments, one controlling only mutation rate, one only crossover rate and one which controlled both simultaneously.

3. PARAMETER CONTROL IN SBST

We implemented multiple different methods of parameter control into EvoSuite, a *Search-Based Software Testing* (SBST) tool [5], to observe the effect on runtime that the different methods had. The implementation also required several adaptations specifically for EvoSuite. One example is that the deterministic method relied on current iterations vs maximum iteration and needed a constant individual size. We adapted this to rely on time alone and as EvoSuite uses variable length individuals, this constant individual size was substituted for other values [9]. Secondly, EvoSuite did not use a mutation rate, so multiple experiments were done to find the best way to implement mutation rate into EvoSuite. We found that using a mutation strength, which controls the quantity of mutations, would be best for this project [9]. Experiments undertaken were aimed at solving the following research questions:

1. Does PC improve the fitness of the result over baseline given a time limitation?
2. Which factors of PC in test generation have the most impact?
3. Which is the best suited method of PC in SBST?

Five experiments were undertaken using a modified version of EvoSuite, which included a deterministic, adaptive and the three self adaptive experiments mentioned in Section 2. The experiments ran 50 times on 10 classes randomly selected from the Apache Commons library [12]. Mutation and crossover rates were controlled during EvoSuite's runtime, and the result used to determine if parameter control could increase efficiency when testing object-oriented classes. The success rate was measured by the fitness of the best individual during runtime, which had a value of 1 for an unfit individual, scaling to 0 for an optimal individual.

4. RESULTS

The experiments show that controlling parameters does influence the effectiveness of search-based software testing,

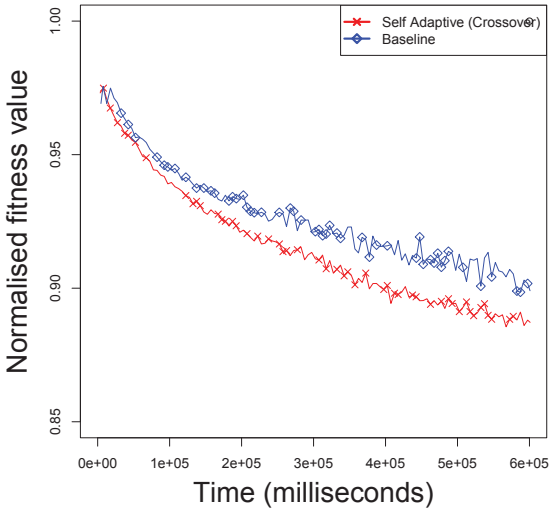

Figure 1: Average fitness of the best individual compared to baseline using Self Adaptive controlling only crossover across 50 runs on 10 classes. [9]

but with the complex individuals in the genetic algorithm and the increased cost of fitness evaluations per mutation, it was found that when a parameter control method increased mutation rate, the computation time per iteration also increased [9]. This caused less evolution to happen when running for a constant time, resulting in less fit individuals produced and negating the goal of parameter control. Despite this, the deterministic control method increased the efficiency of EvoSuite in seven out of 10 classes. The self adaptive method was most effective when controlling only crossover rather than mutation or both, as crossover required less fitness evaluations than mutation. Figure 1 shows the average fitness of the results across all 10 classes from the Apache Commons library. The adaptive method performed poorly, which could be due to the adaptive method consistently increasing mutation rate to a high value. This lead to eight out of 10 classes achieving lower overall fitness [9]. Overall, the results varied per class and PC worked better on some classes than others.

5. CONCLUSION

In conclusion to the project, the three types of parameter control were adapted and implemented into EvoSuite. We found some methods did give an increase in efficiency, such as deterministic. We also found that, as mentioned in Section 4, controlling different parameters gives varied results, as some increase computation time per iteration which could negate the effect of parameter control [9].

As future research, we plan to research into ways to control how individuals are mutated, as opposed to just changing mutation strength [9]. Furthermore, the current methods of parameter control will need to be adapted, taking into account increased computation cost of different parameter values, as this lead to drawbacks through less iterations in the algorithm at runtime.

6. ACKNOWLEDGEMENTS A special thanks to other project members, D. Paterson, J. Turner and G. Fraser.

7. REFERENCES

[1] A. Aleti and L. Grunske. Test data generation with a kalman filter-based adaptive genetic algorithm. *The Journal of Systems & Software*, 103(Complete):343–352, 2015.

[2] A. Arcuri and G. Fraser. Parameter tuning or default values? an empirical investigation in search-based software engineering. *Empirical Software Engineering*, 18(3):594–623, 2013.

[3] C. Darwin. *The origin of species*. Wordsworth Editions Ltd, Hertfordshire, 1998.

[4] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.

[5] G. Fraser and A. Arcuri. EvoSuite: Automatic Test Suite Generation for Object-Oriented Software. In *ACM Symposium on the Foundations of Software Engineering (FSE)*, pages 416–419, 2011.

[6] G. Fraser and A. Arcuri. Whole Test Suite Generation. *IEEE Transactions on Software Engineering (TSE)*, 39(2):276–291, 2013.

[7] G. Karafotias, M. Hoogendoorn, and A. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, April 2015.

[8] L. Ming, Y. ming Cheung, and Y.-P. Wang. A dynamically switched crossover for genetic algorithms. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004.*, volume 5, pages 3254–3257 vol.5, Aug 2004.

[9] D. Paterson, J. Turner, T. White, and G. Fraser. Parameter Control in Search-based Generation of Unit Test Suites. In *Proceedings of Symposium on Search-Based Software Engineering, 2015.*, 2015. To appear.

[10] A. S. Sayyad, K. Goseva-Popstojanova, T. Menzies, and H. Ammar. On parameter tuning in search based software engineering: A replicated empirical study. In *3rd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2013.*, pages 84–90. IEEE, 2013.

[11] J. Smith and T. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation, 1996.*, pages 318–323, May 1996.

[12] The Apache Software Foundation. Apache commons released components. `http://commons.apache.org/components.html`, 2015 (last accessed March 2015).