2010 Second WRI World Congress on Software Engineering

Genetic Algorithm Based Path Testing: Challenges and Key Parameters

Irman Hermadi
UNSW@ADFA, Australia
i.hermadi@student.adfa.edu.au

Chris Lokan
UNSW@ADFA, Australia
c.lokan@adfa.edu.au

Ruhul Sarker
UNSW@ADFA, Australia
r.sarker@adfa.edu.au

***Abstract*— Although many studies have used Genetic Algorithms (GA) to generate test cases for white box software testing, very little attention has been paid to path testing. The paper aims to expose some of challenges posed by path testing, and to analyze what control parameters most affect GA's performance with respect to path testing. Each step in path testing is analyzed based on its complexity and automation. Experiments consist of running GA-based path testing on 12 test problems taken from the literature, using different combinations of values for important control parameters (population size, number of generations, allele range, and mutation rate). The results show that population size matters most in terms of path coverage and number of fitness evaluations, followed by allele range. Changing number of generations or mutation rate has less impact. We also make some observations about what sorts of paths are most difficult to cover. The understanding gained from these results will help to guide future research into GA-based path testing.**

Keywords: genetic algorithm based path testing.

I. Introduction

Hundreds of papers have appeared in the literature on testing and debugging in Search Based Software Engineering, from as early as 1976 [1]. Most have concentrated on branch coverage or statement coverage; very few have considered path testing. All that did consider path testing made use of genetic algorithms (GAs) to evolve test cases.

Path testing allows deeper logical error(s) to be found that may not be found if branch or statement coverage is used, because errors associated with different numbers of iterations through loops can be exposed. Each different number of iterations (once, twice, or more) is considered a different path.

Path testing requires more work, e.g. target path generation and infeasible path confirmation. In addition, complete path coverage is unlikely or impossible because a loop can go infinitely. One way to overcome this is by limiting the number of executions for a loop to a reasonable number; even though this is not complete path coverage, the hope is that it is enough to expose the most likely errors.

This work aims to analyze some challenges of path testing; identify which control parameters have the most effect on the performance of GA-based path testing; and identify good values for those control parameters. We conduct experiments on 12 test problems from the literature, varying each parameter through a set of feasible values. In a two-step process we determine the best parameter settings and also which parameters have the most effect.

The rest of the paper is organized as follows: Section II describes theoretical background for GA based path testing. The experimental approach and test problems are explained in Section III, which is followed by Section IV which describes the results. Analysis of the results is presented in Section V. Threats to validity are described in Section VI. Section VII concludes the paper.

II. Background

A. Path Testing

The aim of path testing is that every possible logical execution path in a program must be exercised at least once.

A given test case causes a program to take exactly one logical path. However, one single logical path can be triggered by multiple test cases. If many test cases can cause the execution of one path, but few can cause the execution of another path, searching for test cases that trigger the rare path is like looking for a needle in a haystack.

A domain of possible data of a program makes a search space for path testing. Generating data that traverse required paths for path testing adequacy is a search. So, path testing is a search for test data that meets path coverage adequacy.

B. Path Testing Challenges

Path testing is the strongest coverage criterion in white box testing. It poses some challenges: justifying its use in the first place, adequate target paths generation, guiding heuristic, computational complexity, and recognizing infeasible paths.

Currently, the standard for coverage testing is branch coverage. Justifying a stronger coverage requirement needs benefits to be shown to counter the higher computational effort.

Generating an adequate set of target paths is not a trivial task. The number of target paths in a no-loop program is just equal to its cyclomatic complexity (CC). The presence of loops, especially nested loops, can increase the number of target paths enormously. A challenge is to find a subset of target paths that adequately represents the complete set.

Searching for test data that causes a certain path in a program to be followed requires information to guide the search. Blind or simple search techniques are not effective

978-0-7695-4303-1/10 $26.00 © 2010 IEEE
DOI 10.1109/WCSE.2010.82

and efficient enough, because of the search space characteristics, i.e. vast, multi modal, discontinuous, noisy, and no (direct) gradient information. Metaheuristic search techniques are mostly used to overcome these characteristics, e.g. evolutionary algorithms (including genetic algorithms, evolutionary strategies, genetic programming), simulated annealing, and particle swarm optimization. All of these techniques make use of guiding heuristics and a fitness function to direct the search.

A disadvantage of path testing is that it requires more work than other coverage criteria (eg for target path generation, and path traversal to evaluate executed input data), therefore it requires more computational resources. In addition, target paths that remain uncovered cannot be easily classified as infeasible paths: an uncovered path may be due to either the search being incomplete, which means it has not been covered yet, or it really is an infeasible path. Further analysis is needed to confirm whether to continue the search or to decide that the path is infeasible. If it is infeasible, part of the test problem that is containing this path must be corrected.

C. Genetic Algorithm

GA is an optimization algorithm that is inspired by evolutionary processes in nature. The following are its main steps:

1) Initial population generation
2) Fitness evaluation
3) While terminating condition has not been met Do: Selection, Mating, Perturbation, Fitness Evaluation

GA involves a set of solution candidates (population) for a particular problem. A solution candidate represents an individual or chromosome in an evolutionary process. Each individual is evaluated according to a function that reflects its fitness as a solution to the problem: the fitter an individual, the better the solution.

III. Approach and Test Problems

The experiments described in this paper aim to shed some light on the challenges and limitations of path testing, described in Section II-B. These experiments build on the previous work of Ahmed and Hermadi in 2008 [3]. The previous experiments concentrated on the effect of using different fitness functions. These experiments seek to reveal behaviors of path testing as parameter settings are changed, e.g. path coverage achievement as population size and other parameters vary. Test problems and their target paths are selected from the literature. More target paths are added for some test problems.

To assist in making our experimental work reproducible, the following subsections describe the details of our experiments.

A. Test Problems

The 12 test problems used in the experiments are taken from the literature. They are called Triangle ([4], [5], [6], [7], [3], [8]), Minimaxi ([9], [3]), Insertion ([7], [3]), Remainder ([4], [8], [10]), Bisection ([5], [10]), Binary ([4], [3]), Bubble ([5], [3]), GCD [7], MmTriangle [3], TriangleMansour2004 (TM2004) ([11], [12]), ExpintRapps1985 (ER1985) ([13], [12]), and QuotientGallagher1997 (QG) ([14], [12]). They were selected because of representativeness of their logical characteristics.

B. Target Path Representation

A target path is represented as a sequence of pairs of predicate (or branch) number (e.g. B1 for first branch), and its decision (1 or 0 for TRUE or FALSE respectively). Repetitions of sub-path(s) in a path indicates the presence of loop(s).

C. Test Data Representation

For each test problem, a test case is one set of input values. Each input (chromosome) has its own domain, which is further decomposed into sub-domains, i.e. allele range. For Triangle, the input is three integers, e.g. (3, 4, 5), and each integer has its own domain, e.g. 0 to 10000.

D. Search Setup

GA is used as the search technique in these experiments, in line with other recent research in this area [1].

The following settings are used: random seeded initial population generation, Roulette Wheel selection (RWS), single point crossover, and generation gap. Generation gap is set to 90%, i.e. 90% of the population experiences GA's operators. 30 runs were made with each combination of parameter settings, always using the same 30 random number seeds.

The fitness function used in the experiments combines approximation level and branch distance [3]. Approximation level (AL) measures similarity/dissimilarity between the path taken by an input data and a target path; it is counted as the number of matched/unmatched branches. The count continues as far as the first unmatched branch encountered. Branch distance (BD) is calculated as Korel's distance function [2], if the path taken differs from the target path of interest. AL and BD can be combined in different ways, each combination representing a different fitness function. 32 different fitness functions were studied in [15]; the one found to be best in that work is used here.

E. Control Parameters

Four GA parameters are varied in these experiments: population size, number of generations, allele range, and mutation rate. Others are fixed, as mentioned in section III-D. Decisions on which ones to vary, and sensible ranges of values for those parameters, were based on previous experience [15], [3].

As the experiments progressed, the ranges of parameter values was narrowed as it became clear that some wider values were beyond the range of being useful. This approach to setting GA parameters is similar to the racing algorithm [16].

To begin with, 9 different values (10, 30, 50, 70, 100, 150, 200, 250, 300) were investigated for population size, 5 values (50, 100, 250, 500, 1000) for the number of generations, 4 ranges ([0 10000], [0 1000], [0 100], [0 10]) for allele range, and 2 values (0.15, 0.3) for mutation rate.

The Triangle problem was investigated first. 360 different combinations of parameter values, with 30 runs per combination, required 10800 runs. Results suggested that the number of generations and the allele range could be narrowed. For the

next 5 problems, 216 combinations of parameter values were used (dropping 1000 for the number of generations, and the widest of the four allele ranges). 30 runs per combination were still done, requiring 6480 runs.

Based on the analysis of results from the first 6 test problems, the parameter ranges for the next 3 problems were further reduced to (30, 100, 250) for population size, (50, 500) for number of generations, and 2 different allele ranges. Each problem required 360 runs (12 combinations $\times$ 30 runs).

The final 3 test problems were run using the same parameter settings as had been used for the previous test problem(s) with the most similar program characteristics (i.e. tackling the same or similar problems, input type, and input length). For example, ER1985 and QG1997 have similar characteristics to some other test problems in terms of input type and length e.g. Binary, Bubble, GCD, Insertion, Minimaxi, and Remainder. For both of these, a low population size (50) was best.

In case of test problems that have variable-sized chromosomes, the range of chromosome lengths is determined by two extreme values related to the number of loops in the test problems, i.e. exercising no loop and many loops. For example, a test problem may require at least two positive integers to either avoid a loop or enter a loop once, but it needs three positive integers to execute a loop twice, thus its input length must be varied between two and three positive integers.

IV. Experimental Results

For each test problem, a two-phase analysis was done: best parameter settings were identified by cumulative plotting of the results over all runs as parameters vary; deciding which parameter matters most was then done by seeing the effect of varying each parameter by itself while each other parameter is fixed at its best value.

Table I shows the best parameter settings.

TABLE I
Best Parameter Settings

No	Problem	Pop.	Gens.	Allele	Mutation
1	Triangle	150	25	0 10	0.30
2	Minimaxi	200	100	-10 10	0.15
3	Insertion	70	50	-10 10	0.15
4	Bisection	100	100	1.72 1.75	0.30
5	Binary	30	50	10 10	0.15
6	Bubble	50	50	-10 10	0.30
7	GCD	250	50	0 20	0.10
8	Remainder	250	500	0 20	0.10
9	MMTriangle	250	50	0 20	0.10
10	TM2004	150	25	0 10	0.30
11	ER1985	50	50	-10 10	0.30
12	QG1997	50	50	1 10	0.30

Table II presents path coverage for all test problems. Ten out of 12 test problems have 100% coverage of feasible paths. Bisection has 3 paths remaining not found, this is due to the requirement that the root of the equation being solved need to be exactly 0 (zero), so any approximations are not accepted. In other words, these 3 paths are feasible but they are less likely to find unless the root is exactly 0.0. The least coverage is seen with Bubble, which has double nested loops.

TABLE II
Path Coverage (Cov. is coverage in %)

No	Problem	CC	Loops	Feasible	Found	Cov.
1	Triangle	[4:12]	0	4	4	100.0
2	Minimaxi	[4: 4]	1	13	13	100.0
3	Insertion	[3: 4]	2	5	5	100.0
4	Bisection	[5: 7]	1	9	6	66.7
5	Binary	[3: 3]	1	7	7	100.0
6	Bubble	[4: 5]	2	13	4	26.7
7	GCD	[4: 4]	1	5	5	100.0
8	Remainder	[4: 4]	1	4	4	100.0
9	MMTriangle	[7:16]	1	20	20	100.0
10	TM2004	[5: 5]	0	7	7	100.0
11	ER1985	[4: 4]	1	5	5	100.0
12	QG1997	[4: 4]	2	4	4	100.0

In general, more target paths are found, regardless of the test problem, with larger population sizes, more generations, and narrower allele ranges. Mutation rate displays no trend. If these four parameters are ordered in decreasing order of impact, they are: population size, allele range, number of generations, and mutation rate. However, the magnitude of increment varies from one test problem to the next.

Further observations are: (1) the hardest target paths for GA to cover involve generating multiple input numbers with exactly the same values, regardless of the test problem, (2) the longer path the longer time is consumed, (3) the more complex a branch the longer time is required, (4) number of paths found increases (almost) exponentially as number of generations increases, (5) the presence of loop(s) greatly increases the computation complexity, (6) higher CC number means more difficult, and (7) variable length input means more time complexity due to larger search space.

V. Analysis

Analysis of the results presented here is based on best parameter settings, test problem characteristics, path testing adequacy, and search technique used.

Based on Table I, population size can be set to 100 for low or medium complexity problems, and 250 for medium or high complexity problems. The number of generations can be set to 50 for low or medium complexity problems, or 500 for medium or high complexity problems. Having allele range as narrow as possible is always better, as long as region of feasible solutions is not removed. Changing mutation rate does not really help, it just can be set to one reasonable value.

Test problem characteristics can be used to describe computational complexity of generating test data for required paths. Looking at Table II, CC alone could not be used to estimate test problem complexity with respect to path testing. However, CC, number of loops, and loop configuration together are good candidate to measure the complexity.

Increasing the population size increases effectiveness more than increasing the number of generations. According to the schemata theorem of genetic algorithms, the number of good/promising chromosomes is proportional to population size. A formula for deriving suitable population size was

presented by Goldberg et. al. [17]. The population sizes we studied align with that formula.

Narrowing the allele range significantly improves the effectiveness of the test data generator. Narrowing the allele range means reducing the search space itself. However, it is more risky to do this, because optimal solution(s) can be missed unless the domain of the search space, and possibly optimal solution location(s), are well known.

Not surprisingly, hard-to-cover paths are those whose input includes numbers with exactly the same values. These were always the last paths to be covered.

Also hard to cover are paths that depend on producing an exact result from an approximation, or that require the input of an exact real number. For example, in Bisection one of the predicate expressions needs an exact variable value of zero to be computed from the previous statements.

GA has following characteristics: incompleteness, probabilistic, and randomness. This means that (1) whenever a path is not found it does not mean that the path is infeasible, it just may not have been found yet; and (2) running GA with the same parameters but different seeds can give different outputs.

VI. Threats to Validity

There are at least two threats to the validity of the approach: problem scaling, and determining the range of values for variable-length chromosomes.

As the test problem gets larger, sizing population, determination of allele range, and generating target paths may be slightly different. There should be a more rigorous or formulated approach for deciding what must be the range of chromosome length for test problems that require variable-length chromosomes.

The approach used in the experiments to decide the range is done manually based on the number of loops required by target paths. This approach is not practical for larger problems.

So, both population sizing and formulated variable length chromosome must be related to problem size or at least to its input size.

VII. Conclusions

The experimental results have shown that varying the population size and the allele range generally has significant impact on path testing performance, while varying the number of generations and mutation rate has less impact in term of number of paths found.

Population size and number of generations can be set to 100 or 250 and 50 or 500, respectively, for low-medium or medium-high complexity problems. The narrowest possible allele range really helps the search, as long as feasible solution regions are not omitted as a result. Mutation rate can be set to a single reasonable value.

CC alone is not enough to estimate complexity of test problem with respect to path testing. It has to be combined with some other measures that give more information on path coverage.

These conclusions will assist us in future work on GA-based path testing. Further investigation is needed to understand test problem complexity, such as combination of CC, number of loops, and loops configuration. Further work can also involve how to decide whether a path is infeasible whenever no test data has been found for it; and whether other types of search techniques, for example combining GA with local search, are suitable for path testing.

Acknowledgment

The authors would like to thank the Indonesia General Directorate of Higher Education (IGDHE), Ministry of National Education, Republic of Indonesia for providing PhD research grant for this project.

References

[1] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Apr. 2009, technical Report TR-09-03.

[2] B. Korel, "Automated software test data generation," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 870–879, Aug 1990.

[3] M. A. Ahmed and I. Hermadi, "GA-based Multiple Paths Test Data Generator," *Computers & Operations Research*, vol. 35, pp. 3107–3124, Oct. 2008.

[4] B. Jones, H.-H. Sthamer, and D. Eyres, "Automatic structural testing using genetic algorithms," *Software Engineering Journal*, vol. 11, no. 5, pp. 299 –306, sep 1996.

[5] R. P. Pargas, M. J. Harrold, and R. R. Peck, "Test-data generation using genetic algorithms," *Software Testing, Verification And Reliability*, vol. 9, pp. 263–282, 1999.

[6] J.-C. Lin and P.-L. Yeh, "Using genetic algorithms for test case generation in path testing," in *Test Symposium, 2000. (ATS 2000). Proceedings of the Ninth Asian*, 2000, pp. 241–246.

[7] E. Alba and F. Chicano, "Observations in using parallel and sequential evolutionary algorithms for automatic software testing," *Computers & Operations Research*, vol. 35, pp. 3161–3183, Feb. 2007.

[8] R. Sagarna and X. Yao, "Handling constraints for search based software test data generation," in *ICSTW '08: Proceedings of the IEEE International Conference on Software Testing Verification and Validation 2008*, Apr. 2008, pp. 232–240.

[9] M. Pei, E. D. Goodman, Z. Gao, and K. Zhong, "Automated software test data generation using a genetic algorithm," Michigan State University, Tech. Rep., Jun. 1994.

[10] R. Blanco, J. Tuya, and B. Adenso-Diaz, "Automated test data generation using a scatter search approach," *Information and Software Technology*, vol. 51, no. 4, pp. 708–720, Apr. 2009.

[11] N. Mansour and M. Salame, "Data generation for path testing," *Software Quality Control*, vol. 12, no. 2, pp. 121–136, 2004.

[12] P. Bueno and M. Jino, "Automatic test data generation for program paths using genetic algorithms," *International Journal of Software Engineering & Knowledge Engineering (IJSEKE)*, vol. 12, no. 6, pp. 691–709, 2002.

[13] S. Rapps and E. Weyuker, "Selecting software test data using data flow information," *IEEE Transactions on Software Engineering*, vol. 11, pp. 367–375, 1985.

[14] M. J. Gallagher and V. Narasimhan, "Adtest: A test data generation suite for ada software systems," *IEEE Transactions on Software Engineering*, vol. 23, pp. 473–484, 1997.

[15] I. Hermadi, "Genetic Algorithm based Test Data Generator," Master's thesis, King Fahd University of Petroleum & Minerals (KFUPM), Jun. 2004.

[16] M. Birattari, T. Stutzle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2002, pp. 11–18.

[17] D. E. Goldberg, K. Deb, and J. H. Clark, "Genetic algorithms, noise, and the sizing of populations," *COMPLEX SYSTEMS*, vol. 6, pp. 333–362, 1991.