Improving Evolutionary Test Data Generation with the Aid of Symbolic Execution

M. Papadakis[1] and N. Malevris[1]

Abstract Recently, search based techniques have received great attention as a means of automating the test data generation activity. On the contrary, more traditional methods that automate the test data generation usually employ symbolic execution by incorporating a path generation phase and constraint solvers to produce the sought test data. In this paper, the benefits of both schools of thought are bridged in an attempt to investigate whether a mixed strategy approach could be employed when evaluating a coverage criterion. To this effect, a strategy that uses symbolic execution and dynamic domain reduction in order to enhance the initial population and approximately prune the search space considered by evolutionary based methods is proposed. This suggestion is also put under a number of tests which clearly show a dramatic improvement of its effectiveness. This suggests that the combination of evolutionary based and symbolic execution approaches can be beneficial toward, automating the generation of test data.

1. Introduction

It is a well known fact that the cost of software testing can reach 50% or even 60% of the total software development cost. In order to reduce the cost overhead, a lot of effort has been put by the software engineering community, in an attempt to automate the testing activity and thus reduce the overall software development cost. The usual way to evaluate the test thoroughness of a piece of software is to establish a collection of testing requirements that must be fulfilled when the software is executed against test cases, through a number of test coverage criteria that guide and evaluate the effectiveness of the test data generated.

Test data generation techniques based on Genetic Algorithms (GAs) have been adopted in the literature [7, 16] and a number of researchers have proposed vari-

[1] Athens University of Economics and Business {mpapad, ngm}@aueb.gr

ous algorithms based on both local [2, 16] and global searches [7, 8, 16] with an appropriate adaptation of their fitness function according to the coverage criterion considered. Evolutionary Algorithms (EA) for test data generation approaches have been widely studied in the literature [2, 7, 8, 13, 16] although a number of unresolved research issues still remain [7, 13], mainly on the efficiency and effectiveness of these approaches.

A hybrid approach to test data generation that integrates GAs and symbolic execution in a complementary way is proposed. Existing attempts [18, 19] use structural methods and EAs in an effort to facilitate the testing exercise. The purpose of this study is to guide the input domain selection in such a way so as to improve and make more efficient the EA. Symbolic execution is used not as a complimentary tool but as a yardstick, towards this purpose. It is used to guide the test data selection rather than evaluating a suggested path. In this respect, the proposed method have no common philosophy with the refs above [13, 18, 19]. Our goal is to demonstrate that information from path testing and symbolic execution can be used effectively in a combined way, in order to improve the efficiency and effectiveness of the EA. The proposed work stems from suggestions in the literature about the reduction of the search space [9, 12] and the observation that candidate solutions near the target ones can be efficiently refined through local search algorithms. We argue that the fulfillment of these two directions can be achieved through path testing in a systematic and automated way, relying on the ESPM method (hereafter called "Y&M") suggested by Yates and Malevris [4]. Through this approach a set of linearly independent candidate paths is produced, while all candidate paths have a high probability of being feasible. Here, these two attributes of the Y&M method are exploited in order to refine the initial population set of the EA. In order to reduce the search space when targeting to a particular branch [4], the assumption of the Y&M method is utilized which suggests that a feasible path has higher probability to be contained in a set of the first *k*-shortest paths. Based on this assumption, which was substantiated in [4], some irrelevant variables are pruned away and domain reduction is performed by using the DDR procedure proposed in [5], over the common constraints that constitute a subset of the constraints derived from the selected path set. The reduced domain space then forms the search space of the EA.

Our approach has been empirically investigated with reference: a) the impact of the initial population enhancement; b) the impact of the input domain reduction procedure; and c) the overall improvement of the proposed approach, clearly showing a dramatic improvement of the evolutionary approaches when guided by path testing.

2 Incorporated methods

Symbolic Execution: The symbolic evaluation process [1] of a program consists of assigning symbolic values to variables in order to deduce an abstract alge-

braic representation of the program's computations and representation. This technique is based on the selection of paths from its control flow graph and the computation of symbolic states. The symbolic state of a path forms a mapping from input variables to symbolic values and a set of constraints called *path conditions* over those symbolic values. Path conditions represent a set of constraints called *symbolic expressions* that form the computations performed over the selected path. Solving the path conditions results in test data which if input to the selected path, this will be executed. If the path condition has no solution the path is infeasible.

In this paper, the Y&M method [4] was chosen because of its flexibility and its ability to generate likely to be feasible paths. The method can be detailed as: ***Step1:*** Generate a set of program paths, whose constituent paths each involve a minimum number of predicates, and cover the target elements of code unit under test. ***Step2:*** Symbolically execute the current path set and determine the achieved coverage. ***Step3:*** Select uncovered elements and generate alternative path sets giving priority to those containing a lower number of predicates. Steps 3 and 2 are repeated *k* times or until the target coverage is achieved.

Evolutionary Algorithms: A GA test data generator employs a genetic algorithm as its primary search engine in seeking suitable test data according to a target test adequacy criterion. The basic steps of a GA are the following [7, 13]: ***Step1***. Create an initial population of candidate solutions. ***Step2***. Compute the fitness values of each of these candidates. ***Step3***. Select all the candidates that have the fitness values on or above a threshold. ***Step4***. Perturbate each of these selected candidates using genetic operators.

These steps, except the initialization step, are repeated until the coverage goal is met or until the time limit has been reached. Before the use of a GA we need to define the following: Some domain-dependent attributes, a representation of the problem solution in terms of genetic inputs (chromosomes), the fitness function, the candidate selection methods (chromosomes reproduction) and the genetic operators.

```
Input: Program P, Target Coverage, Time_Limit
Output: A set of test Inputs
Graph ← GenerateGraph(P);
Tests ← RandomInputs();
K = 1;
repeat
1    Paths ← GenerateY&M_Paths(Graph, k);
2    Tests ← Symbolic Execution ( Paths );
3    Targs ← GenTargets(Graph, Tests, Targs);
4    Tests ← EvolutionaryTesting(Targs);
5    for all ( Targs not covered )
         DomainReduction (Targs);
     K = K + 1;
until (Coverage==Target OR Time<Time_Limit)
return;.
```

Fig. 1. Proposed Algorithm

3 Approach Description

The motivation behind our work is the combination of static and dynamic approaches namely symbolic execution and evolutionary testing, in order to improve their performance. Our framework tries to guide the search along two directions. First by enhancing the initial population with results from established testing techniques such as [4] and second, by using the same method to dynamically and ap-

proximately reduce the input search space. Our approach uses *k* paths from the path set of the Y&M method [4] in order to prune away all the irrelevant input variables and uses the common parts of those paths to define a common set of constraints. The algorithm in Figure 1 outlines the proposed test data generation scheme. Given a program P, the algorithm iteratively computes sets of inputs based on both symbolic execution and evolutionary testing in order to achieve the targeting coverage. The algorithm loop as in figure 1 contains three basic steps (lines 1-5 of fig. 1). (a) Starts by generating and executing symbolically the current set of paths according to the Y&M method. The resulting test data enhance the randomly generated initial population of the evolutionary algorithm. (b) The test generation process continues with the evolutionary algorithm phase, where newly test inputs are constructed. (c) The final step of the loop is the domain reduction process according to uncovered branches. The algorithm terminates when the time limit is exceeded or the coverage goal is met.

Symbolic execution: The Symbolic execution process phase is carried out based on the Y&M method for a given parameter value *k* see [4] for details. Here we use some approximations in order to overcome, abstract away, unhandled expressions and generate test data. Note that solutions will be refined from the evolutionary algorithm so we only need to generate solutions close to the target ones. Unhandled expressions such as non linear constraints are ignored or replaced with new symbolic values. Constructs such as pointer values will be approximated through simple expressions containing only (in)equality constraints. The resulting path conditions are then passed to the constraint solver for feasibility check. If the considered path condition is found infeasible, the path is marked as infeasible. Otherwise, test data will be produced which enhance the current population of the evolutionary algorithms.

Evolutionary testing: The evolutionary algorithm phase takes the instrumented version of the program under test, the program's CFG and evolutionary parameters, and produces tests according to the evolutionary method used. In the present study we use the SGA&D[2] as used in [7, 16]. In our study we evaluate, based on symbolic execution, the improvement made over these two algorithms by considering two approaches (approach 1, 2) and a third which combines previous two (approach 3).

Standard GA (approach 1) setup: ***Representation:*** A chromosome as a bit string representing test cases. ***Evaluation function:*** Each chromosome is evaluated simply by using the program's CFG and accounting the number of decisions covered. ***Selection:*** A new population is formed by selecting the best chromosomes in terms of coverage and ***Recombination:*** Performing mutation and crossover operations.

Differential GA (approach 2) setup: ***Representation:*** A chromosome as a bit string representing test cases. ***Evaluation function:*** Each chromosome is evaluated according to its percentage coverage contribution. ***Selection:*** New population is formed by selecting the best chromosomes in terms of coverage and ***Re-***

[2] Standard Genetic Algorithm and Differential

combination: For each test case A={a_1, a_2, … a_n} the algorithm selects randomly two different mates B={b_1, b_2, … b_n} and C={c_1, c_2, … c_n} from the population. Then according to probability (Change Factor) it selects input values (a_i) for alteration. For all selected values (a_i) the algorithm calculates new ones based on the formula: $a_i' = a_i + \alpha (b_i - c_i)$ according to α (Factor). If the resulting A′={ a_1', a_2', … a_n'} solution performs better based on the objective function, the solution A is replaced by A′.

Hybrid approach of Standard and Differential GA (approach 3) setup*:* This algorithm forms the combination of the two preceding ones by applying the standard one first and then the differential one by using the final population of the standard algorithm as the initial population to the differential one.

Domain reduction: The goal of the proposed approach is to guide the search by reducing the search space and thus improve its application as stated in [9]. The main idea behind the reduction algorithm is to use the set of paths of Y&M method in order to dynamically define a domain approximation of the targeting branch. Through our research in path testing we have observed that many paths that covering particular branch share many parts. This observation helps as to reduce the search space by using domain reduction approaches as in [5] only for the common parts of the Y&M path set. We are interested about the common parts of the paths and so we eliminate all non common conditions. Additionally we detect irrelevant variables as those not used in any path condition of our set. The common set of constraints forms the targeting reduced domain approximated by using a similar to the Dynamic Domain Reduction method [5], alternative to the constraint propagation as used in [14]. The main assumption behind the approach is the same with the Y&M method: at least one feasible path is contained in a set of selected k-shortest paths and so reducing the search space based on them directs the search to the contained feasible paths.

Our approach reduces the search space using binary search on input domain space based on the concerned constraints. For each constraint in the set, the algorithm reduces appropriately the domain space based on the variables, their comparison use in the constraint set and their assigned domain space. The domains of input variables are split in half *(binary search)* based on the constraint comparison.

4 Empirical Setup and Results

In order to evaluate the proposed approach we have implemented a semi automated prototype tool. Our tool consists of the *symbolic execution module*, the *domain reduction module* and the *evolutionary testing module*.

Test Subjects. We used a set of thirteen java and C programs for our experiment. This set of test objectives contains some common programs used in test data generation studies [5, 7, 16] such as the Triangle classification, the Quadratic formula, the Euclidian algorithm, Binary Search, Bubble sort and the Calendar. The

other programs were taken from study [10]. Table 1 shows details of the pro grams.

Evaluation Procedure. The equipment and standards used in our experiments include:

1. Adopt the GA parameters for: **Approach 1:** (Standard GA) infinite MaxGens, MaxTime 30 Sec and MaxPopulation = 1000. **Approach 2:** (Differential GA) Infinite MaxGens, MaxTime 30 Sec and MaxPopulation = 1000, factor="0.5" changefactor="0.5". **Approach 3:** (Both Standard and Differential GA) same setting with MaxTime 20 Sec for each of the two approaches.
2. Run the prototype tool on a Core 2 Duo 1.66GHz with 2 GB Ram computer running the Windows Vista operating system.

Table 1. Subject programs

Test Object	Lines of Code	No. of Branches
Triangle	40	46
Quadratic	12	6
Triangle2	38	35
Euclidian	9	10
Binary search	20	17
Bubble sort	14	11
Calendar	22	12
Insert	26	20
Dbll	86	78
C prog One	45	23
C prog Two	32	17
C prog Three	64	31
C prog Four	28	30

3. For every approach and in order to avoid any form of bias from random effects we introduced 3 experimental cases under which we ran every experiment 10 times and compared their average values. ***For case 1:*** First run the EA in isolation and then the algorithm for parameter k = 1 (i.e. include steps 1, 2, 3, 4 of the proposed algorithm, fig. 1).***For case 2:*** First run the EA in isolation and then the domain reduction procedure followed by the evolutionary algorithm (i.e. include steps 4, 5 of the proposed algorithm fig. 1). ***For case 3:*** Run the proposed algorithm for k = 60 and for MaxTime = 30s. (i.e. include all algorithm steps from 1-5, fig. 1)

Case 1. In the current case we tried to examine the impact of population initialization through path testing on evolutionary based test data generation. In order to simulate and observe the behavior of GAs through the guidance from the symbolic execution process we initialized the population using random testing and symbolic execution based only on the basic path set produced from the use of the Y&M method (parameter value *k=1*).

Table 2 records the average coverage achieved within the same time limit for both the original evolutionary algorithm (Before) and the proposed approach (After) which uses randomly generated initial population (Before) and the proposed approach which enhance the initial population (After). The results of this experiment show that, for the programs used, the initial population enhancement improved the evolutionary algorithm effectiveness by on average 9.82% for Standard GA, 8.49% for Differential and 7.83% for their combination.

Table 2: Branch Coverage achievement before and after initial population enhancement.

Test Object	Cov. Before/After (Approach1)	Cov. Before/After (Approach2)	Cov. Before/After (Approach3)
Triangle	61.06% / 82.33%	57.87% / 86.59%	62.55%/ 87.74%
Quadratic	100% /100%	100% /100%	100% /100%
Triangle2	63.06% / 97.22%	60.83% / 98.89%	65.56 / 97.78%
Euclidian	*100% /100%*	*100% /100%*	*100% /100%*
Binary search	*71.655% /100%*	*98.647%/100%*	*98.647%/100%*
Bubble sort	*100% /100%*	*100% /100%*	*100% /100%*
Calendar	*100% /100%*	*100% /100%*	*100% /100%*
Insert	*100% /100%*	*100% /100%*	*100% /100%*
Dbll	*79.23% / 81.54%*	*84.74 % / 88.52%*	*88.52% / 96%*
C prog One	*72.17% / 73.91%*	*70.87% / 73.91%*	*73.91%/ 73.91%*
C prog Two	*52.94% / 88.24%*	*52.94% / 86.47%*	*52.94%/ 85.29%*
C prog Three	*84.52% / 89.03%*	*90.32% / 92.26%*	*89.03%/ 92.26%*
C prog Four	*76.67% / 76.67%*	*76.67% / 76.67%*	*76.67%/ 76.67%*
Average	**81.64%/91.46% (+9.82%)**	**84.07%/92.56% (+8.49%)**	**85.22%/93.05% (+7.83%)**

Case 2. In the current case we tried to examine the impact of domain reduction on the performance of GA by considering the domain reduction process described in section 3. We used k-value: 60 (number of selected paths). In order to simulate and observe the behavior of GAs through the guidance of the reduced domain we concentrated only on those branches that were left uncovered from the evolutionary algorithm on the initial search space. Table 3 records the average coverage improvement in the same time limit over the initial approach.

Table 3: Branch Coverage improvement achieved through domain reduction.

Test Object	Cov. Improvement (Approach1)	Cov. Improvement (Approach2)	Cov. Improvement (Approach3)
Triangle	*4.3%*	*11.2%*	*12.1%*
Triangle2	*1.2%*	*5.2%*	*5.6%*
Binary search	*0%*	*0%*	*0%*
Dbll	*0%*	*0%*	*0%*
C prog One	*1.8%*	*2.9%*	*0%*
C prog Two	*1.4%*	*2.3%*	*2.3%*
C prog Three	*3.2%*	*-1.2%*	*1.7%*
C prog Four	*5.2%*	*6.8%*	*5.8%*
Average	***1.55%***	***2.47%***	***2.50%***

The results of this experiment show that, the domain reduction procedure improves the evolutionary algorithm effectiveness by on average 1.55% for Standard GA, 2.47% for Differential and 2.50% for their combination.

Case 3. In the current case we tried to examine the overall improvement in terms of coverage of the proposed approach (proposed algorithm figure 1) over the evolutionary testing. The experiments where conducted within the same time limit (30s) for all approaches. Table 4 records the average coverage achieved in the same time limit for the evolutionary methods (Approaches 1, 2 and 3). Based on the positive findings of cases 1 and 2, case 3 incorporates these results as they are reflected by the algorithm in Figure 1.

Table 4: Branch Coverage achievement within the same time limit.

Test Object	Cov. GA/proposed (Approach1)	Cov. GA/ proposed (Approach2)	Cov. GA/ proposed (Approach3)
Triangle	*61.06% / 100%*	*57.87% / 100%*	*62.55% / 100%*
Quadratic	*100% /100%*	*100% /100%*	*100% /100%*
Triangle2	*63.06% / 100%*	*60.83% / 100%*	*65.56 / 100%*
Euclidian	*100% /100%*	*100% /100%*	*100% /100%*
Binary search	*71.655% /100%*	*98.647%/100%*	*98.647%/100%*
Bubble sort	*100% /100%*	*100% /100%*	*100% /100%*
Calendar	*100% /100%*	*100% /100%*	*100% /100%*
Insert	*100% /100%*	*100% /100%*	*100% /100%*
Dbll	*79.23% / 100%*	*84.74 % / 100%*	*88.52% / 100%*
C prog One	*72.17%/76.27%*	*70.87%/76.27%*	*73.91%/76.27%*
C prog Two	*52.94% / 100%*	*52.94% / 100%*	*52.94% / 100%*
C prog Three	*84.52% / 100%*	*90.32% / 100%*	*89.03% / 100%*
C prog Four	*76.67% / 97%*	*76.67% / 97%*	*76.67% / 97%*
Average	***81.64%/98% (+16.36%)***	***84.07%/98% (+13.93%)***	***85.22%/98% (+12.78%)***

The results of this experiment show that, for the test programs considered, the proposed algorithm in figure 1 outperforms the EA by on average 16.36% for Standard GA, 13.93% for Differential and 12.78% for their combination.

5 Related Work

Generally there has been a considerable amount of work in the area of automatic test generation based on both symbolic execution [1, 3, 5] and evolutionary techniques [2, 7, 8, 11, 13, 15, 16, 17]. Closest to our research is the work by Xie [6] where two automated tools, one for evolutionary testing and one for symbolic execution have been integrated in order to improve the structural testing of object-

oriented programs. They propose a framework that attempts to improve the coverage by targeting to sequences of method calls through evolutionary testing and to their coverage improvement by symbolic execution. The integration is based only on static input initialization from one tool to the other and not by simultaneously and dynamically completing one another as in our approach. In [12], another similar to [6] approach that combines static analysis and search based methods is proposed. The main objective of this work is a theoretical and empirical evaluation of the effects of domain reduction through irrelevant variable removal. In their empirical study the authors used static analysis based on program slicing in order to identify and discard the irrelevant variables. In [2] a search reduction was made considering the path coverage criterion. Each input variable was ranked according to influence graph constructed using dynamic data flow information. The variable value would remain unchanged if it was likely to impact segments that were currently being traversed correctly or if it did not affect the path. In [16], a framework that utilizes two optimization algorithms, the Batch-Optimistic and the Close-Up is proposed. This approach uses a domain control mechanism witch starts with small domain spaces and modifies its boundaries to larger ones at subsequent phases. Nevertheless, they did not guide the domain reduction through symbolic execution. In [11] Andreou et al, propose a specially designed genetic algorithm for data flow criteria which relies on the data flow graph.

6 Conclusions and Future work

This paper presented a test data generation technique that integrates symbolic execution and GAs in an effective and complementary way. The motivation behind our proposal is to combine both static and dynamic analysis techniques in order to complement each other and result in an improved generation process. This technique tries to guide the search of EA through symbolic execution and domain reduction, via two main directions. First by using symbolic execution on a limited set of paths and second by identifying common constraints that form a reduced domain. We also described the results obtained for evaluating the effectiveness of our approach, by using both guidance directions in isolation and in combination.

Here, the attempt has been to investigate whether symbolic evaluation can assist an existing genetic algorithmic approach rather than to be compared against it. Under such circumstances the feeling of the authors is that the amelioration in the results presented here will hold with different GAs. The choice of the most effective genetic algorithm to be used as a basis in the enhancement process, is however a matter of future research.

In all three experiments with a set of programs, three different GAs were used (basically two as the third is a combination of the other two). In all experiments, the support offered by symbolic execution via path generation, improved the coverage ability on two counts. First by increasing the achieved coverage by 8.7% on average as in case 1, while for cases 2 and 3 the improvement was on average

again 2.17 and 14.4% respectively and second by improving on the run time as it provided a reduction in the domain from which sample values ought to be generated. These two activities do highlight the strength of the proposed method i.e. of combining symbolic execution with the employment of a search based technique in an attempt to evaluate a coverage criterion when structurally testing a piece of software. Future work is directed towards conducting more experiments in order to statistically validate the claims of the present findings. Series of experiments are also planned to determine the optimal use of symbolic execution and the domain reduction process.

References

1. King, J. C. (1976). Symbolic execution and program testing. *Commun ACM* **19**(7), 385-394.
2. Korel, B. (1990). Automated Software Test Data Generation. *IEEE Trans. Softw. Eng*. **16**(8), 870-879.
3. Koutsikas, C., Malevris, N. (2001). A Unified Symbolic Execution System. *AICCSA* 466-469.
4. Yates, D. F., Malevris, N. (1989). Reducing the Effects of Infeasible Paths in Branch Testing, *in proc of Symposium on Testing, Analysis, and Verification,* 48-54.
5. Offutt, A. J., Jin, Z., Pan, J. (1999). The Dynamic Domain Reduction Procedure for Test Data Generation. *Softw., Pract. Exper.* **29**(2), 167-193.
6. Inkumsah, K., Xie, T. (2008). Improving structural testing of object-oriented programs via integrating evolutionary testing and symbolic execution. *In Proc. ASE*, 297–306.
7. McGraw, G., Michael, C., Schatz, M. (2001). Generating software test data by evolution. *IEEE Trans. Softw. Eng*, **27**(12), 1085-1110.
8. Pargas R., Harrold M., Peck R. (1999). Test-data generation using genetic algorithms. *Software Testing, Verification and Reliability*, **9**(4), 263-282.
9. Chen, S., Smith, S. (1999). Improving genetic algorithms by search space reductions. *GECCO*, 135-140.
10. Malevris, N., Yates, D. F. (2006). The collateral coverage of data flow criteria when branch testing. *Information & Software Technology* **48**(8), 676-686.
11. Andreou, A. S., Economides, K. A., Sofokleous, A. A. (2007). An Automatic Software Test-Data Generation Scheme Based on Data Flow Criteria and Genetic Algorithms. *CIT* 867-872
12. Harman, M., Hassoun, Y., Lakhotia, K., McMinn, P., Wegener, J. (2007). The impact of input domain reduction on searchbased test data generation. *ACM FSE*, 155-164.
13. McMinn, P. (2004). Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, **14**(2), 105-156.
14. Hentenryck, P., Saraswat, V., Deville, Y., (1998). Design, implementation and evaluation of the constraint language cc(fd) . *Journal of Logic Programming*, **37**, 139-164.
15. Xanthakis, S., Ellis, C., Skourlas, C., Gall, A. L., Katsikas, S., Karapoulios, K. (1992). Application of genetic algorithms to software testing *ICSEA*, 625-636.
16. Sofokleous, A. A., Andreou, A. S., (2008). Automatic, evolutionary test data generation for dynamic software testing. *Journal of Systems and Software* **81**(11), 1883-1898.
17. Ferguson, R., Korel, B. (1996). The chaining approach for software test data generation. *IEEE Trans. Softw. Eng,* **5**(1), 63-86.
18. Girgis, M. R., (2005). Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm, *Jucs*, **11**(6), 898-915.
19. Ahmed, M. A., Hermadi, I. (2008). GA-based multiple paths test data generator, *Computers and Operations Research* **35**(10), 3107-3124.