Automated Test Data Generation For Programs Having Array Of Variable Length And Loops With Variable Number Of Iteration

H. Tahbildar and B. Kalita *

Abstract—**In this paper, we propose a heuristic called longest path criterion for test data generation of programs having arrays of variable length, loops with variable number of iteration. Our heuristic is computed by a mathematical relation of array size(arr) and number of iteration(k). The relation we found is based on our experiment result at a saturation point after which no more longer path is executed. Our relation computed k can be found directly in stead of trial and error procedure for finding k. Test data with our k achieves better code coverage. We achieved 97% accuracy for predicting the value of k generating the longest path. The time required to generate test data for longest path criterion shows the effectiveness of our heuristic.**

Keywords: ***longest path, saturation point, lmax,*** k_L, k_S

1 Introduction

Software testing is a process, which is used to identify the correctness, completeness and quality of a software. Software testing is very expensive. Statistics says that 50% of the total cost of software development is required for testing phase. There are various methods of test data generation [3]: Random [5, 10, 12], Path oriented [14, 4, 9, 1], and Goal oriented [17]. Our ultimate aim is to get 100% automation in test data generation process [7]. Test data can be generated using static approach [15] based on symbolic execution, dynamic approach [14] based on actual value. It has been found that neither static nor dynamic method of test data generation is efficient. A combined approach which takes the merits of both is more efficient. Test data generation for programs having arrays of variable length, loops with variable number of iteration is a challenging problem. Although test data for such program is generated using path prefix method but it causes a combinatorial explosion in the number of execution paths. Therefore it imposed coverage of only those paths containing number of iterations within user defined limit k. The method fails to cover upper bound testing. In [11], Williams combine the merits of both static and dynamic techniques. The method avoids the traditional path finding step and cover only feasible path but problem is with k path criterion. It is seen that k value exponentially increase the number of infeasible paths. Without testing with upper bound of k, which is done in domain testing, we can not have reliable test data. In this paper we modified the $k = 2$ user defined limit path criterion to longest path criterion that can be computed from our mathematical relation and it will generate test data for all paths containing the longest path. We found a mathematical relation for finding k by inputting array length which covers longest path criterion. The relation we found is based on our experiment's result at a saturation point after which no more longest path is created. We have eliminated the problem of user defined k-path criterion by introducing longest path criterion. In [11], user determines k value using trial and error method and k is limited to 2. Therefore the time required for finding k will be more than our longest path criterion and code coverage of longest path criterion will be more than $k = 2$ path criterion.

The rest of the paper is organized as follows: The section 3 presents a survey of related works of path oriented test data generation. The section 4 describes our method of test data generation. The section 5 shows our experimental results. Section 6 explain our mathematical relation for longest path criterion k with justification. Finally in section 7 we conclude with some observations and future research direction for automatic test data generation.

2 Related Work

There are various methods of implementation for test data generation: Symbolic value [15, 16], actual value [14, 9], path prefix [13], combined symbolic and actual value [11], prioritized constraints and data sampling

*Hitesh Tahbildar, Department of Computer Engineering, Assam Engineering Institute, Guwahati-781003, INDIA Tel/Fax:91-9864018339 Email: tahbil@rediffmail.com, B. Kalita, Department of Computer Application, Assam Engineering College, Guwahati-781013, INDIA Tel/Fax:91-9954936116 Email: bichitra1_kalita@rediffmail.com, Manuscript submission: September'2009

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

scores [2, 8, 6]. Nicky Williams in [11], presented the PathCrawler prototype tool for automatic test data generation that satisfies all-paths criterion, with user defined limit on the number of iterations. It is based on path prefix method [13] and it does code instrumentation and constraint solving. But PathCrawler does not use any heuristic to determine the value of k(number of iterations). It compromises from rigorous all-paths criterion to $k = 2$ path criterion. It does not take any effort to achieve the saturation point for variable length array. It selects a single test case for each feasible path and avoids all other test-cases that follows an already traversed path. Information collected during execution of the program under test influences the test selection.

3 Our Approach

3.1 k Vs execution time

We have observed the execution time behavior of the program with respect to different values of k. Graphs appear to be linear for lower values of k and becomes exponential at higher end as shown in figure 1.

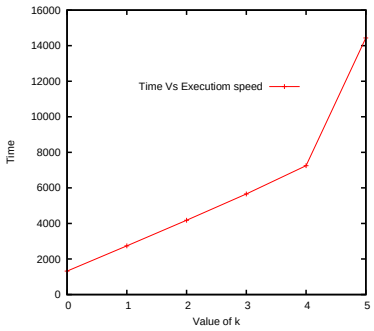

Figure 1: k Vs execution speed of program

3.2 k Vs new and unique feasible paths

The lower value of k does not satisfy upper bound testing for large array size. If k is small compared to the array size then many feasible paths are left out. After a certain value of k for a particular array size, the number of feasible paths become constant and do not increase any further even if k is increased. This is the point of saturation. The k value at this point is denoted by k_S. We have successfully achieved the saturation point for arrays of size 5 at $k = 940$ generating 926 unique feasible paths as shown in figure 2.

3.3 k Vs longest feasible path

We studied the behaviour of feasible paths produced with different values of k. No more new feasible paths will be created after saturation point. We have observed that the length of the paths generated, increases with increase of k as shown in figure 3.

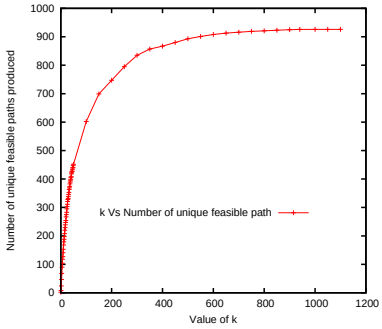

Figure 2: k Vs number of unique feasible paths

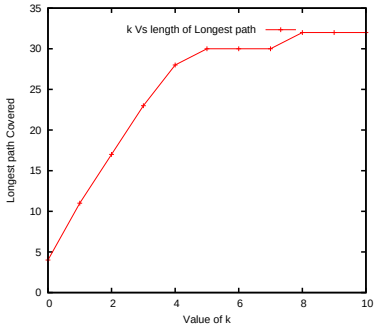

Figure 3: k Vs length of longest feasible path

Although k is increased further, at a particular value, the length of the paths become constant and does not increase further even if the value of k is increased. This is the longest path criterion. We are denoting the value of k at this point as k_L. In the figure 3, the length of the paths become constant at 32.

3.4 Our Heuristic

It is fact that k_S is optimum value of k. We can get best coverage and best test data if we generate our test data with k_S. But running a program with k_S increases the number of execution path and time exponentially for large array size. Our heuristic finds k_L and we can generate test data of all paths containing longest path. Although the k value that is determined by our heuristic is not optimum, yet it gives better coverage than $k = 2$ path criterion [11]. Instead of only a single test case, several test cases are selected for each path which improves the chances of detecting coincidental correctness. In our implementation we are considering the 1000 as domain range for the sample program given in Annexure I.

4 Mathematical Relation

After the study of different behaviors in which the feasible paths respond to different values of k, we have derived a mathematical relation between k_L (minimum number of

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

iteration where longest path is covered), arr (array size) and k_L, $lmax$ (longest path length).

We have evaluated the $lmax$ and k_L for different values of arr. Then we found out their relation in the form of $lmax/k_L$ and k_L/arr. The experimental data are shown in Table 1

Table 1: Computation of α and β

arr size	k_L	L_{path}	$\beta = lmax/k_L$	$\alpha = k_L/arr$
4	4	24	6	1
5	8	32	4	1.6
6	5	35	7	0.83
7	7	44	6.3	1
8	8	49	6.13	1
9	9	53	5.9	1
10	11	62	5.64	1.1
11	11	68	6.2	1

Relation between k_L and arr

$$k_L/arr = \alpha$$
$$k_L = alpha \times arr,$$
where $alpha = 1.0663$

Therefore,
$$k_L = alpha \times arr$$

The constant $alpha$ is average value of different array size

Relation between k_L and $lmax$ is:

$$lmax/k_L = \beta$$
$$lmax = \beta \times k_L,$$
where $\beta = 5.896$

The constant $beta$ is average value of different array size

Therefore,
$$lmax = beta \times k_L$$

From above two equations, we get
$$lmax = 6.2869 \times arr$$

4.1 Justification

To check the value of k_L and $lmax$ we are taking different array size. For example $arr = 12$ using the above two derived equations we check for their accuracy with the true value.

Expected value of k_L for array of length 12:
$$k_L = 1.0663 \times arr$$
$$= 1.066312$$
$$= 12.79$$
$$= 13\text{(approximately)}$$

Expected value of $lmax$ for array of length 12:
$$lmax = 5.896 \times k$$
$$= 5.896 \times 13\text{(derived from above)}$$
$$= 76\text{(approximately)}$$

True Value

For $arr = 12$, using our tool we get the result as shown in Table 2
$k_L = 12$ and $lmax = 74$

Table 2: Value of k and longest path

k	lpath	value of k	lpath
0	4	7	47
1	11	8	53
2	17	9	59
3	23	10	64
4	29	11	70
5	35	12	74
6	42	13	74

4.2 Error Analysis for Array Size

For given array size

Absolute Error =|True value−Expected Value|
$$=| 12 - 13 |$$
$$= 1$$
Relative Error =Absolute Error/True Value
$$= 1/12 = 0.083$$
Percentage Error =Absolute Error/True value×100
$$= 8.3\%$$
Percentage of Accuracy $= (100 - 8.3)\%$
$$= 91.7\%$$

For longest path length

Absolute Error =|True Value−Expected Value|
$$=| 74 - 76 |$$
$$= 2$$
Relative Error = Absolute Error/True Value
$$= 2/74$$
$$= 0.027$$

Percentage Error=Absolute Error/True value×100
$$= 2.7\%$$
Percentage of Accuracy $= 100 - 2.7\%$
$$= 97.3\%$$

4.3 Comparison between theoretical and practical values

Our experiment results show that the value computed from our heuristic (Theoretical) and the value found from our experiments as shown in Table 3 differ a little amount that has negligible effect on test data generation process. The

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

graph of theoretical value and experimental value of k_L and $lmax$ is shown in figure 4 and 5 respectively.

Table 3: Theoretical and practical values of k_L, lmax

Array size	Theoretical		Practical	
arr	k_L	lmax	k_L	lmax
2	2.136	12.59	3	14
4	4.27	25.14	4	24
6	6.4	37.72	6	35
8	8.53	50.29	8	49
10	10.66	62.87	11	62
20	21.33	125.74	21	121
25	26.66	157.17	26	151
30	31.99	188.6	31	182
35	37.34	220.04	37	212

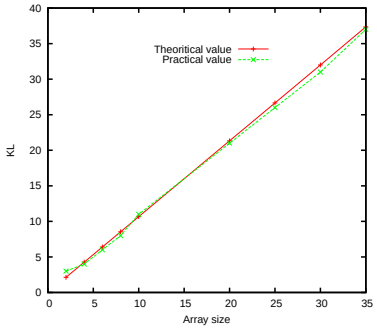

Figure 4: Comparison between Theoretical and practical value of k_L

4.4 Accuracy

The accuracy of the k_L is inversely proportional to the array size. The average accuracy of k_L computed from our heuristic is 97.3%

5 Conclusion and Future Work

The automatic test data generation for programs with variable length of array with variable number of iterations using traditional path oriented method is very costly. Because the number of infeasible paths increase exponentially. Although the path prefix method solves this problem to some extent but it fails in terms of code coverage. we have got a mathematical relation between number of iterations and array size that can easily compute best k for maximum coverage.

Of course, our k can not cover all-paths criterion but it covers longest path criterion. The percentage of code coverage is improved. No backtracking is required and thus, time is saved as there is no need to check for maximum code coverage with k value less than that computed from the tool. Our method

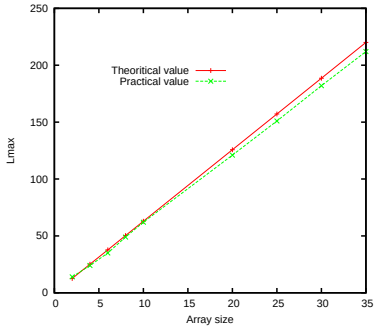

Figure 5: Comparison between Theoretical and practical value of lmax

does not guarantee testing of boundary values of the domain where bugs are often found. Our approach can be modified to statically inject the values at the domain boundaries into the array extreme positions. The accuracy and scalability of our relation can be improved by making more observations using different array size, loop construct, and varying number of iterations. Finally the execution time depends to a great extent, on the machine configuration. The graph will vary from machine to machine. Future research can be done on the heuristic constant α and β to improve our accuracy.

References

[1] C. Cadar and D. Engler, *Execution Generated Test Cases: How to Make systems Code Crash Itself*, Technical Report, Computer Systems Laboratory Standford University, Standford CA-94305, U.S.A. 2005.

[2] M. Gittens, K. Romanufa, D. Godwin, J. Racicot, *All Code Coverage is not created equal: A case study in prioritized code coverage*, Technical Report, IBM Toronto Laboratory, 2006.

[3] J. Edvardsson,, "A Survey on Automatic Test Data Generation," *In Proceedings of the Second Conference on Computer Science and Systems Engineering(CCSSE'99)*, Linkoping, pp. 21-28 10/1999.

[4] Chen Xu, J. Zhang, Xiaoliang Wang, "Path Oriented Test Data Generation Using Symbolic execution and Constraint solving Techniques," *In Proceedings of the Second International IEEE Conference on Software Engineering and Formal Methods(SEFM'04)*,2004.

[5] A. Gotlieb, M. Petit, "Path-Oriented Random Testing," *Proceedings of the First International Workshop on Random testing(RT'06)*, Portland, ME, USA, 07/2006

[6] Xiao Ma, J. Jenny Li, and David M. Weiss, "Prioritized Constraints with Data Sampling Scores for Automated Test Data Generation," *Eighth ACIS International Conference on Software Engineering, Aritificial Intelligence, Networking, and Parallel/Distributed Computing*, 2007.

[7] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," *In Future of Software Engineering(FOSE'07)*,2007.

[8] J. Jenny Li, "Prioritize Code for Testing to Improve Code Coverage of Complex Software," *In Proceedings of*

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the 16th IEEE International Symposium on Software Relaibility Engineering(ISSRE'05), 2005.

[9] Jun-Yi Li, Jia-Guang Sun, Ying-Ping Lu, "Automated Test Data Generation Based on Program Execution," *In Proceedings of the Fourth IEEE International Conference on Software Engineering Research, Management and Applications(SERA'06)*, 2006.

[10] N. Klarlund, P. Godefroid, and K. Sen, "Directed Automated Random Testing," *In Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation(PLIID'05)*, 2005

[11] N. Williams, B. Marre, P. Mouy, and M. Roger, "PathCrawler: Automatic Generation of Path Tests by Combining Static and Dynamic Analysis," *Springer-Verlag Berlin Heidelberg*, LNCS 3463, pp. 281-292, 2005.

[12] Koushik Sen, Darko Marinov, Gul Agha, "CUTE: A Concolic Unit Testing Engine for C ," *ACM*, pp. 5-9, 09/2005

[13] R. E. Prather, J. P. Myers, "The Path Prefix Software Engineering," *IEEE Trans on Software Engineering*, SE-13(7), pp. 761-766, 07/1987

[14] B. Korel, "Automated Software Test Data Generation," *IEEE Trans on Software Engineering*, Vol. 16, No.8, pp. 870-879, 08/1990

[15] L. A. Clarke , "A System to Generate Test Data and Symbolically Execute Programs," *IEEE Trans on Software Engineering*, Vol. SE-2, No.3, pp. 215-222, 09/1976

[16] J. Zhang, Xiaoxu Wang., "A Constraint Solver and its Application to Path Feasibility Analysis," *International Journal of Software Engineering and Knowledge Engineering*, 11(2): pp. 139-156, 2001.

[17] R. FerGuson, B. Korel, "The Chaining approach for Software Test Data Generation ," *ACM Transactions on Software Engineering and Methodology*, 5(1): pp. 63-86, 01/1996.

Annexure I

```
void Merge(int t1[], int t2[], int l1, int l2, int q) {
int i=0, j=0, k=0, t3[20];
while(i < l1 && j < l2)
{
if(t1[i] < t2[j])
{
t3[k]=t1[i];
i++;
}
else
{
t3[k]=t2[j];
j++;
}
k++;
}
while(i < l1)
{
t3[k]=t1[i];
i++;
k++;
}
while(j < l2)
{
t3[k]=t2[j];
j++;
k++;
}
}
```

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)