

ZEPHYRUS: Determinant Approach to Path Generation

Praveen Ranjan Srivastava¹, Anuj Prateek²

^{1,2}Computer Science & Information System Group

Birla Institute of Technology and Science (BITS), Pilani, Rajasthan India

Email: praveenranjansrivastava, anuj.prateek@gmail.com

Abstract- Software Testing is one of the most an indispensable part of software development lifecycle and glass box testing is one of the most widely used testing paradigms for testing various software. Glass box testing relies on code path identification, which in turn leads to identification of effective paths. The current paper aims at presenting a simple and novel algorithm for the effective path identification by using the basic property of the matrix, namely the determinant. The matrix used in the method finds its origin in the control flow graph and finds its representation in adjacency matrix. This novel approach uses certain set of rules to find out all the effective paths. The method concentrates on generation of paths, equal to the cyclomatic complexity. Towards the end, an end-to-end path generation algorithm is also proposed.

Keywords - Software testing, Path testing.

I. INTRODUCTION

Software Testing [1, 4] forms one of the indispensable parts of the software development life cycle [7] and heavily depends upon the design of the software. It also depends upon the software tester, guidelines of the organization, which developed the software and various other factors. The diverse nature of the software Testing leads to standardization of the testing process. Some of the major methods of Software Testing include black box testing [4] and glass box testing [4]. Glass box testing, also known as white box or structural testing depends upon generation of various control flow paths. Generation of control flow paths [4] [8] [10] is a challenging problem in the domain of software engineering [7]. Various methods like brute force, constraint based heuristics, symmetric matrix algorithm etc [8] [6] suggested for generation of control flow paths, offer solution to the problem, but none of these offer a simple and optimized solution. The current paper proposes a novel algorithm for the path generation in a simpler and effective manner. The algorithm also aims at reducing the number of control flow paths to the cyclomatic complexity [4] [6] of the software. The method proposed, depends upon the mathematical property of the matrix namely determinant [7] [9]. The proposed algorithm uses a tweak of determinant calculation by set of proposed rules for the path generation. The paper also proposes an end-to-end path generation algorithm based on the effective paths generated, again utilizing the property of the matrix, namely matrix traversal. The proposed algorithm named as Zephyrus and divided into two parts, namely Zephyrus path generation

algorithm and Zephyrus end-to-end path generation algorithm. This paper describes Zephyrus algorithm in detail.

II. ZEPHYRUS PATH GENERATION ALGORITHM

The control flow graph [2, 7] is represented by the help of the adjacency matrix [3]. The processing of the adjacency matrix involves determinant calculation. The determinant calculation, governed under certain set of Zephyrus algorithm specific rules, leads to generation of independent and effective paths during every step of the calculation. Zephyrus performs a tweak of determinant calculation, which renders the mathematical definition of determinant calculation untrue in this case, and in the process, it leaves us with an algorithm of path generation. The determinant calculation involves reduction of the matrix by removal of the first row and the first column at every step of the determinant calculation. At any step of determinant calculation, for a_{11} , all the elements a_{i1} where $1 < i < n$ (n = number of rows), are reduced to zero by row reduction operation. After the reduction has been carried out, all the independent i.e. singleton entities in the first row are added to the set of paths, if they do not appear in the sub matrix formed after removal of first row and first column. Non-independent entities i.e. the expression appearing in the first row are reduced to independent entities by applying normal mathematical operations in light of the under given rules: Addition is equivalent to subtraction, Any quadratic power is equivalent to one, Multiplication is equivalent to division, Non-expression operands of addition represents addition paths, Non-expressions operands of multiplication represents sequential paths, Expression operands of multiplication, reduced to the form representing non-expression operands of expression, then the expression, removed from the multiplication operands results into a reduced multiplication expression, Multiplication with zero yields no effect on the other operand. Reduction and removal of the rows, carried out until the matrix size, reduced to 2×2 or less in special cases, leading to completion of the Zephyrus path generation algorithm. The algorithm exhibits the following properties and uses the following notions defined over the adjacency matrix,

1. Nodes at same level in the Control Flow Graph form row of the matrix.
2. Nodes at subsequent levels in the Control Flow Graph form the column of the matrix.
3. In determinant calculation, nodes at same level are never multiplied.
4. In determinant calculation, nodes at different levels are never

added. **5.** All operations at the time of calculation are stable. **6.** All mathematical operations are performed in light of the mathematical rules described earlier. **7.** Sub-matrix is the matrix that we get after removing the first row and first column. **8.** Determinant calculation propagates from upper diagonal element to the lower diagonal element. At every step of calculation we aim at making all elements other than diagonal elements as zero for the first row and column. **9.** All subsequent determinant calculation steps, effective paths are generated

The pseudo-code [6] of the Zephyrus path generation algorithm, given below describes the process in a more complete manner, followed by the description of the pseudo-code.

Algorithm EffectivePathCalculator starts:

Program's global Variable: EffectivePathSet GEPS

Input: Labelled Graph G (E, V)

```

Matrix M
M = generateAdjacencyMatrix(G(E,V))
DeterminantExpression DE
DE = getDeterminantExpression(M)
/* In the calculation of DeterminantExpression, EPS is
being generated along with and appended to GEPS.*/
EffectivePathSet EPS
EPS = getEffectivePathSet(DE)
GEPS = GEPS + EPS
GEPS = reducePathSet(GEPS, M)

```

Algorithm EffectivePathCalculator ends

Algorithm getEffectivePathSet starts:

```

Input: DeterminantExpression DE
Output: EffectivePathSet EPS
DeterminantExpression DTEMP
DTEMP = \ getConstantDeterminantExpression(DE)
EPS = \ parseDeterminantExpressionToEffectivePathSet(D\
TEMP)
Return EPS

```

Algorithm getEffectivePathSet ends

Algorithm getConstantDeterminantExpression starts:

```

Input: DeterminantExpression DE
Output: DeterminantExpression DC
DC = DE
While DC not constant
DC = ReducePath(DC)
DC = OrderCorrector(DC)
End While, Return DC

```

Algorithm getConstantDeterminantExpression ends

Algorithm reducePathSet starts:

```

Input: GlobalEffectivePathSet GEPS,
AdjacencyMatrix M
Output: GlobalEffectivePathSet GEPS
GlobalEffectivePathSet tempGEPS
tempGEPS = GEPS
GEPS = NULL, Index1 = 0, Index2 = 0
Path P
While tempGEPS.pathCount != 0
P = tempGEPS.getPathAt(Index1)
Pick all diagonal elements to P in M

```

```

Append all elements picked to P
Delete all elements picked, from tempGEPS
P = OrderCorrector(P)
GEPS.addPathAt(Index2, P)
Index1 = Index1 + 1
Index2 = Index2 + 1
End While, Algorithm reducePathSet ends

```

III. ZEPHYRUS END-TO-END PATH GENERATION

Zephyrus path generation algorithm generates a set of independent paths from the adjacency matrix representation of the control flow graph. These paths may find their origin at the starting point of the program and even in between the computational steps. Glass box testing requires end-to-end paths i.e. paths originating at entry point of the program and ending at the exit point [4] [7]. These end-to-end paths are essential for exhaustive testing of the software and generation of the test cases for the testing. Zephyrus end-to-end path generation algorithm works on the effective path set generated and converts them into end-to-end paths by utilizing various traversal rules of the adjacency matrix, proposed by the current paper. These traversal rules are as follows: **1.** Movement along y-axis corresponds to travelling up in the adjacency matrix (move up). **2.** Movement along negative y-axis means travelling down the adjacency matrix (move down). **3.** Movement along x-axis means travelling right in the adjacency matrix (move right). **4.** Movement along negative x-axis means travelling left the adjacency matrix (move left). **5.** A single element of the adjacency matrix corresponds to a node. **6.** If the current node is on any of the boundaries of the adjacency matrix, then the moves will be restricted depending on the boundary with the exception that the starting and ending element nodes are not found, in which case the matrix is considered wrap around. **7.** In the notion of the wrap around moves in the matrix, we do not consider an element twice in our moves. **8.** The notion of wrap around matrix, utilizes the start and end nodes, and the algorithm considers multiplicity in the same.

Zephyrus end-to-end path generation algorithm picks one independent path from the effective path set and converts it into an end-to-end path by help of the traversal defined. The pseudo-code of end-to-end path generation algorithm, given next, describes the process in detail.

Algorithm generateEndToEndPath starts:

Input: Global EffectivePathSet GEPS

Output: EndToEndPathSet EEPS

```

Path abovePath, bottomPath, tempPath
For every path in GEPS
tempPath = GEPS[i]
abovePath = getAbovePath(tempPath)
belowPath = getBelowPath(tempPath)
EEPS = EEPS + combinePath(abovePath, \ belowPath,
tempPath), End for, Return EEPS
Algorithm generateEndToEndPath ends.

```

The algorithm described above uses two important functions, namely getAbovePath and getBelowPath. These

functions work on the adjacency matrix and treat it in as wrap around mesh [1]. Moves around the matrix are not possible, once one gets to the start or the ends of the Control Flow Graph, represented in the adjacency matrix, as described by the traversal rules proposed earlier. An important consideration here is, that the adjacency matrix must be in the right form i.e. in the first column and the last row, only obtuse-diagonal (diagonal connecting elements at $0, 0$ and N, N , where N is the size of the matrix) elements, must be one, and rest should be zero. This definition, along with posing the restriction, gives the flexibility to have multiple starts and ends.

Algorithm getAbovePath starts:

Input: Path tempPath, Output: Path abovePath

Node node = tempPath.node[0]

While (flag = TRUE), Move one step up

If (any non zero or non one node exists in left)

If connected(node, abovePath.getLastNode)

abovePath = abovePath + node

node = new node found in left

End if, End if

if (no move possible), flag = FALSE

End if, End while

Algorithm getAbovePath ends.

Algorithm getBelowPath starts:

Input: Path tempPath

Output: Path abovePath

Node node = tempPath.node[0]

While (flag = TRUE), Move one step right

If (any non zero or non one node exists below)

If connected (belowPath.getLastNode , node)

belowPath = belowPath + node

node = new node found below

End if, End if

if(no move possible), flag = FALSE

End if, End while

Algorithm getBelowPath ends.

After the determination of the abovePath and the belowPath, the paths, combined with the node to generate the complete path. Combining of the paths, done in light of the following rules: If the node is single, then for node N , abovePath P and belowPath Q , the resultant end to end path is $P-N-Q$. If the node is a chain and forms a loop, then for chain N_1-N_N , abovePath P and belowPath Q , the resultant paths are $P-N_1-Q$ and $P-N_1-N_N-N_1-Q$, when the path generation was carried on N_1 . If the node is a chain and do not form a loop, then for chain N_1-N_N , abovePath P and belowPath Q , the resultant paths is $P-N_1-N_N-Q$. GenerateEndToEndPath () is dependent upon two functions namely, abovePath() and belowPath() to generate a single end-to-end path from one effective path. This function works straight forward for a single node. In case of node chains, which form a loop, only the first node of the chain, passed to the function, suffices. In case of node chains, not forming a loop, all the nodes form input to the end-to-end path generation function. A node chain requires satisfying the following condition to form a loop. In circular fashion, all the adjacent nodes, namely N_1 and N_2 , must have,

one stored in the adjacency matrix, at the intersection of the horizontal projection of N_1 and vertical projection of N_2 . The algorithm uses a test for connectivity of two nodes. Two nodes, one at (row₁, col₁) and other at (row₂, col₂) are said to be connected if and only if the element at index (col₁, row₂) is one. After the generation of the end-to-end paths, the paths generated, corresponding to node chains, not forming loop undergoes cleaning process. Cleaning here refers to removal of extra or irrelevant nodes. A node chain N_1-N_N , followed by a sequence of nodes from N_1-N_N after N_N node ending at one, reflects the possibility of irrelevant or extra nodes. The end-to-end path generation algorithm finds its limitation in finding the end-to-end paths for an independent path that represents the single entry-exit point loops. This scenario, handled by help of combining the independent path, with the path generated by the node, present obtuse-diagonally, to the starting point of the independent path loop utilizes a different combine algorithm, based on insertion.

IV. CONCLUSION

Zephyrus proves to be a convenient and simple algorithm for reduction of potentially infinite set of paths to finite domain. Zephyrus algorithm is highly selective in nature and identifies the effective paths in a way that the reduction further goes down to the cyclomatic complexity of the software. The algorithm also generates end-to-end paths, which can facilitate generation of test cases using backtracking appropriately. The algorithm also specialises at identification of the loop constructs in the control flow graph, though is not limited to it. The paper hence presents a novel approach to the problem and demonstrates how the tweaked mathematical rules can serve the need of specialized problems. Our ongoing work focuses at the application of Zephyrus in real life situation. Future extension of the algorithm aims at proposing the backtracking approach for test case generation utilizing various mathematical properties.

REFERENCES

- [1] A.P. Mathur, software testing, Pearson education, 2007
- [2] G. Chartrand, Introductory Graph Theory, Dover Publications, New York,
- [3] Gilbert Strang, Introduction to Linear Algebra, Wellesley-Cambridge Press, Cambridge, June 1998
- [4] Ian Sommerville, Software Engineering, 6/e, Addison Wesley, Boston, August 11, 2000
- [5] Jon Mathews, Robert L. Walker, Methods of Mathematical Physics, 2/e, Addison Wesley, 1971
- [6] Robert Sedgewick, Algorithms in Java, Third Edition, Part 5: Graph Algorithms, Addison Wesley, Boston, July 15, 2003
- [7] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, Dec 1976, pp. 308-320
- [8] V. Bhattacharjee, Suri, P. Mahanti, "Application of Regular Matrix Theory to Software Testing", European Journal of Scientific Research, Vol.12 No.1 (2005), pp. 60-70
- [9] V. N. Faddeeva, Computational Methods of Linear Algebra, Dover Publications, New York, 1959
- [10] W. E. Howden, Functional program Testing and Analysis. McGraw-Hill, 1987.