International Journal of Information Technology and Knowledge Management
July-December 2008, Volume 1, No. 2, pp. 401-405

GRAPH THEORY APPLICATIONS TO SOFTWARE AND ASSEMBLY LANGUAGE PROGRAMMING

Manoj Duhan & Parvinder Singh

Abstract

We are in a process to study different techniques to compute the cyclomatic complexity of a software or assembly language program. We are having more than two components in any signal flow graph. We will be able to convert the software or assembly language program with the help of flow graph components into the form of algorithm to compute the cyclomatic complexity of a program, which is the slight modification of McCabe'y algorithm. After implementing it to many programs it is possible to find out the upper limit of the cyclomatic complexity.

Introduction

A linear graph (or simply a graph) $G = (X, Y)$ consists of set of objects $X = (X_1, X_2,\ldots)$ and another set $Y = (Y_1, Y_2,\ldots)$, whose elements are called edges, such that each edge y_k is identified wiyh an ordered pair (X_i, Y_j) of vertices. The vertices X_i, Y_j associated with each edge y_k. The most common representation of a graph is by the means of a diagram, in which the vertices are represented as points and each edge as line segment joining its end vertices. Often this diagram itself is referred to as graph [1].

McCabe's Cyclomatic Number

McCabe's cyclomatic number introduced in 1976 is, after lines of code, one of most commonly used metrics in software development. Also called "McCabe's cyclomatic complexity measure" from the title of the original journal article, it is based on the graph theory's cyclomatic number. McCabe tries to measure the complexity of a programe. The premise is that complexity is related to the control flow of the program [2–4].

Cylomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. The value computed for cyclomatic complexity defines the number of independent path in the basis set of program and provides us with an upper bound for the number of the test that must be conducted to ensure that all the statements have been executed at least once.

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of flow

graph, an independent path must flow move along at least one edge that has not been traversed before the path is defined. Cylomatic complexity is a useful metric for predicting those modules that are likely to be erroe prone [4-5]

How do we know that for how many paths, we have to look for? The computation of cyclomatic complexity provides us with the appropriate answer.

Cyclomatic complexity has a foundation in Graph Theory and provides us with extremely useful software metric. Complexity is computed by following algorithm.

Algorithm 1: A planar graph is a graph that can be drawn without line crossing. The Swiss mathematicians Leonard Euler proved for planar graphs that $2 = n - y + r$, where

r = number of regions, y = number of edges, and n =number of nodes. A region is that area enclosed by arcs. Using algebra, this can be converted to $r = y - n + 2$. Therefore the number of regions on the planar graph equals the cyclomatic complexity $Y(G)$.

$$Y(g) = r = y - n + 2 \tag{1}$$

Algorithm 2: Calculating the Cyclomatic complexity from control flow graph is time consuming. Constructing a control flow graph from a large program would be prohibitively more time consuming. McCabe found a more direct method for calculating his measure. He found that the number of regions is usually equal to one more than the number of decisions in the program, $V(r) = D + 1$, where D is the number of decisions. In source code, an If statement, a WHILE loop or a FOR loop is considered as one decision.

$$Y(G) = r = D + 1 \tag{2}$$

Algorithm 3: For a graph $G = (X, Y)$ with n nodes, e edges and C connected components, the cyclomatic complexity $Y(G)$ is defined as

$$Y(G) = y - n + C \tag{3}$$

Modified Algorithm: To compute the complexity of a program by using modified algorithms you will have to use the equation (4) which is the slight modification of McCabe's algorithm.

$$Y(g) = \text{total number of edges } (y) - \text{number of nodes } (n) + \text{number of connected component (C)} \tag{4}$$

Where

$$C = \begin{cases} 1 & \text{if there is an arc from exit node to start node,} \\ 2 & \text{otherwise.} \end{cases}$$

In this equation "2" is nothing but a number of connected components. If we have more than 2 connected components, then number of independent path will be different. Here

we are giving a simple representation of determining the cyclomatic complexity from the control flow graph through a simple example. Here we have converted a simple program to its flow graph.

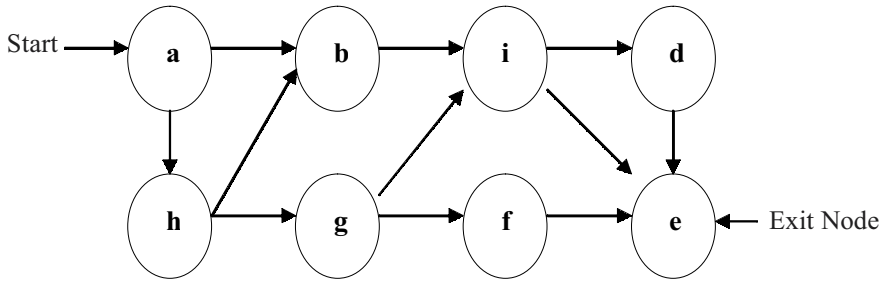

Figure 1: Control Flow Graphe

Algorithm 1

We can assume any assembly language program for finding its complexity. Here is such an example in which we are having 8 nodes. There are 11 arcs so $e = 1$. The cyclomatic complexity is $Y(G) = 11 – 8 + 2 = 5$. we have to now label the regions in the control flowgrpah as shown in Fig. 1

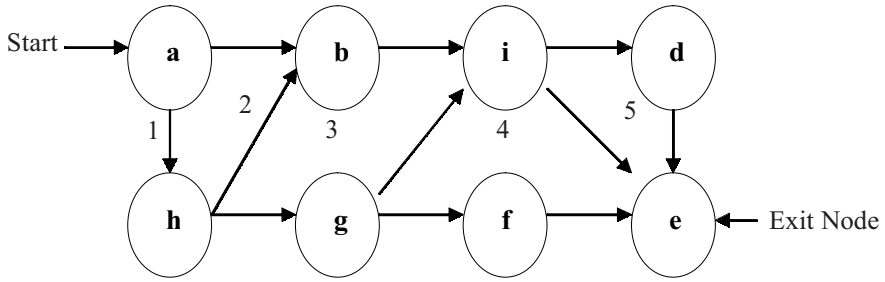

Figure 2: Control Flow Graph With Regions Marked, $Y(g) = 5$

Algorithm 2

Label the decisions in the control flow graph with capital alphabets. As shown in Fig. 3 from node *a,* there are three arcs, so there must be two decisions, labeled *L* and *M*. From nodes *h*, *g* and *i*, there are two arcs and so one decision each. The other nodes have at most one exit and so no decisions. There are four decisions from where we will be able to calculate $Y(G) = 4 + 1 = 5$.

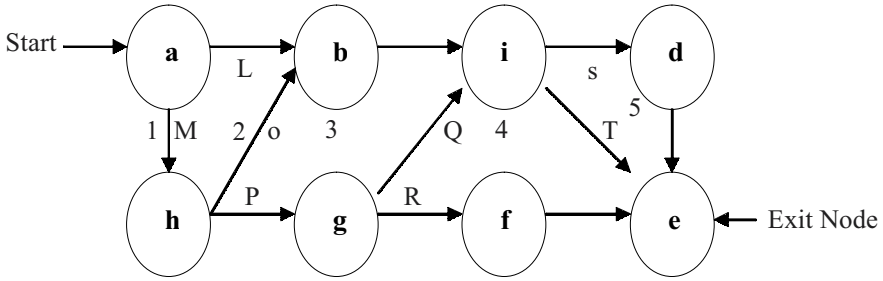

Figure 3: Control Flow Graph With Capital alphabets

Algorithm 3

There are 08 nodes in the control flow graph with an extra exit node to start node so $n = 8$, there are 12 arcs therefore $c = 12$. Number of connected component is one, we can calculate (modified algorithm) $Y(G) = 12 - 8 + 1 = 5$.

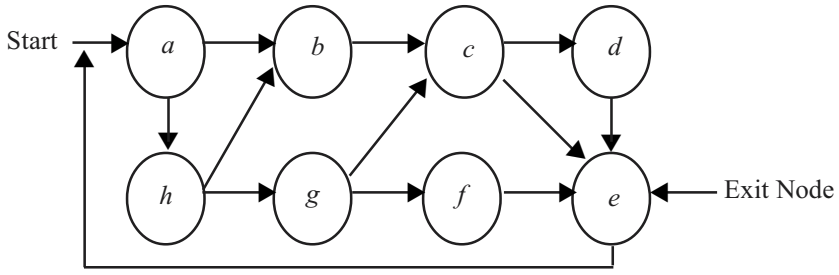

Figure 4

CONCLUSION

After implementing it on large number of programs in the same way, we come to the conclusion that if there are more than two connected components in a flow graph, then cyclomatic complexity of a given program will increase and the number of independent paths will also increase. McCabe proposed that the cyclomatic complexity will be kept below 10. Experiments indicate that the cyclomatic complexity is highly correlated to the size of the program module in Line of Code. Further we conclude that it can be effectively implemented to calculate the complexity of a software or assembly language program.

REFERENCES

[1] Deo, Narsingh, "Graph Theory with Applications to Engineering and Computer Science", Prentice-Hall of India Pvt. Ltd. New Delhi, (1999).

[2] Jalote, Pankaj, "An Integrated Approach to Software Engineering", 2nd Edition. Narosa Publishing House, New Delhi, (2001).

[3] Pressman. Roger, "Software Engineering: A Practitioner's Approach", 3rd Edition, McGraw Hill.

[4] McCabe, T., "A Software Complexity Measures", *IEEE Trans. Software Engineering*, Se-**2**, (1976), 308–320.

[5] Gross. J. L., Yellen. J, "*Graph Theory and its Applications*", CRC Press LLC, (1998).

[6] West. D. B., "Introduction to Graph Theory", Prentice Hall, (1996).

[7] Tibor Jordán, "A Characterization of Weakly Four-connected Graphs", *Journal of Graph Theory*, **52**, (3), (July 2006), 217–229.

Manoj Duhan[1] & Parvinder Singh[2]
Dept. of Electronics Engineering[1], Dept. of Computer Science & Engineering[2]
Deenbandhu Chhotu Ram University of Science & Technology
Murthal, India