ISSN:2320-0790

COMPUSOFT
An International Journal of Advanced Computer Technology

Optimizing Programs Using Call Graphs

Pravin Kandala[1], Jasmeen Kaur[2], Prof. T.Venkat Narayana Rao[3]

[1,2]Student, Department of CSE, Guru Nanak Institutions Technical Campus, Hyderabad,
[3]Professor, Department of CSE, Guru Nanak Institutions Technical Campus, Hyderabad, India

Abstract: When working with complex software, visualization improves understanding considerably. Thus enhancing the ability of programmers to picture the relationships between components in a complex program not only saves time but becomes progressively mission- critical with increasing software complexity. Call Graph Generation Tool is a visualization tool which provides programmers different metrics to assess the software code. The different metrics include total number of lines in the function, total number of executable lines, number of unreachable lines, and cyclomatic complexity of the program. It provides a graphical representation of the function calls in a tree like structure. The tool accepts a 'C' program and generates a functions call graph along with the functional metrics. The call graph generation tool provides both static and dynamic view. The whole programming is done using java technology. Thus, this tool helps the developer to know the program flow and thereby decide the optimality of the program. In situations where in, a single program is to be selected from available programs, this tool helps to figure out it. This paper depicts usage of call graph Generator to assess the reachability and exactness of the programs.

Keywords: Call graph, cyclomatic complexity, C program, optimality

I. INTRODUCTION

Call graph is a directed graph that represents calling relationships between subroutines in a computer program. Thus, a cycle in the graph indicates recursive procedure calls. Call graphs are a basic program analysis result that can be used for human understanding of programs, on basis for further analyses, such as an analysis that tracks the flow of values between procedures. Application of call graphs is finding procedures that are never called. Call graphs can be dynamic or static. A Call graph generation tool provides with a wide range of tools to which when used over a specific piece of code enables a programmer to get a vivid understanding of the code. It also enhances the performance of the program in a direct manner by improving its overall functioning and understanding.

A. *Purpose*

The central focus of this system is to help the developer analyze the program that he wrote or that he wants to analyze. It is difficult for the developer to know the complete flow of the program by just viewing it. So here comes the need for a specialized and sophisticated tool which provides the developer with the graphical representation of the program. Using which the user can improve the performance of the code by just calculating the cyclomatic complexity of each function.

B. *Objectives of the system*

Intention behind the proposal of this system is to generate a call graph to a 'c' program. Here we are taking a C program as an input to the system. The call graph is drawn in a hierarchical way starting from the main() function of the and moving down until all the functions used in the program have been exhausted. The nodes in the graph at each level represent the function called by the functions in the previous level. The relationship is represented by the directed edges from one node to other. This facilitates the user to understand the program clearly.

C. *Cyclomatic Complexity*

The cyclomatic complexity metric is based on the number of decisions in a program [1]. This is important to testers because it will provide an indication of the amount of testing (including reviews) necessary to practically avoid defects, areas of code identified as more complex are candidates for reviews and additional dynamic tests. A more formal definition regarding the calculation rules is provided in the glossary as shown in figure 1. While there are many ways to calculate cyclomatic complexity, sum the number of binary decision statements (e.g. if, while, for, etc.) and add 1 to it [2]. The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code [3]. For example, if the source code contained no decision points such as IF statements or FOR loops, the complexity will be 1, since there is only a single path through the code. A single IF statement of the code containing a single condition there will be two paths through the code: 1 path where the IF statement is evaluated as TRUE and one path where the IF statement is evaluated as FALSE [4][5] .

Mathematically, the cyclomatic complexity of a structured program_[note 1] is defined with reference to the control flow graph of the program, a directed graph contains the basic blocks of the program with an edge between two basic blocks if control may pass from the first to the next. The complexity M is then defined as :
M = E − N + 2P, Where,
E = the number of edges of the graph,
N = the number of nodes of the graph,
P = the number of connected components

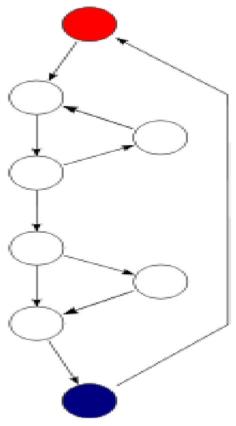


Fig1: Illustrates Cyclomatic Complexity

D. Using of JgraphT API

JGraphT is a free Java graph library that provides mathematical graph-theory objects and algorithms [8]. It supports various types of graphs including:

- Directed and undirected graphs.
- Graphs with weighted/un-weighted/labeled or any user-defined edges.
- Various edge multiplicity options are simple-graphs,
- Multi graphs, pseudo graphs.
- Unmodifiable graphs - allow modules to provide "read-only" access to internal graphs.
- Listenable graphs - allow external listeners to track modification events.
- Sub graphs that are auto-updating sub graph views on other graphs.
- All compositions of above graphs.

Java Generics: JGraphT is designed to be simple and type-safe -- JAVA GENERICS

***Jgraph and JgraphT*:** These are two different libraries which are intended for different purposes.

JGraphT: It is focused on data structures and algorithms. Although powerful, it is designed to be type-safe and simple. Ex: graph vertices can be of any objects. Creating graphs based on XML document, Strings, URLs etc; Even create graphs of graphs is possible.

Other features offered by JGraphT*:* Graph visualization using the JGraph library

- complete source code included, under the terms of the GNU Lesser General Public License and the EPL as well via dual licensing).
- Comprehensive Javadocs.
- Easy extensibility.

JGraph: Is focused on GUI-based editing and rendering. These libraries are complementary and can be used together via the JGraphModelAdapter provided by JGraphT; this adapter allows a JGraphT graph data structure to be used as the model being viewed and controlled via JGraph.

E. *Dependencies*

- JGraphT requires JDK 1.6 or later to build.
- JUnit is a unit testing framework. JUnit is needed only to run the unit tests. XMLUnit extends JUnit with XML capabilities. XMLUnit will be needed only to run the unit tests.
- JGraph is a graph visualization and editing component. JGraph will be needed only when creating graph visualizations using the JGraphT-to-JGraph adapter.
- JGraphX is the successor to JGraph. This will be needed only when using the JGraphXAdapter to visualize the JGraphT graph interactively via JGraphX.
- Touchgraph is a graph visualization and layout component. Touchgraph is needed when creating graph visualizations using the JGraphT-to-Touchgraph converter.

F. *Advantages*

Faster way of analyzing a program. Allows the user to analyze multiple ‘c’ files related to the same software. Helps to find out the unusual functions called on fronts on fronts in the program. Enables easy review and modification of the program. Platform independent as the programming language is Java.

***G.* Specification**

The specification here include an systematic way of providing information about the program which is successfully done by the call graph [6][7]. Thus making the programmers find a better way of analyzing a program. The human understanding of the programs becomes difficult as they become complex and no statistical information of them is provided. This requires a systematic analysis of the program for the programmers to clearly understand what is actually written in the source code.

II. METHODOLOGY

Call Graphs provides the systematic information about the programs by indicating the calling relationships between all the subroutines in the 'C' program. The Call graph generation for mission critical software is to enable the people better understand the program so that they can make an easy review of it, modify the code or use the defined functions in the program in other programs. The company administrative people will maintain a database of all the 'c' programs which they want to analyze and store the information in another data base that is useful for them in above mentioned ways. The proposed system is a software tool that facilitates the user needs. So, it does not have any administrator to maintain. When the user clicks the icon of it, it prompts for the input file which after given, a call graph is shown with all the subroutines in the program and also can provide the basic metrics about the functions like cyclomatic complexity, number of lines in the program. It also makes the user to know about the total number of executable lines in the given program and list of unused functions in a given 'c' program.

This system consists of three modules:

- Accepting 'c' files from the user
- Identifying the functions in the program, list out calling functions.
- Generating the call graph and calculation of function metrics.

The user will be prompted to select a c file which he likes to analyze. Then the given c file is splitted into tokens and the tokens are passed into another file. Then the obtained tokens are checked for function prototype. The functions obtained are shown to the user for his confirmation. Then the function calls for each function will be obtained and stored for each function object. The function objects have different set of parameters that are used to find the metrics for each function. The metrics of every function is presented to the user in the graphical manner using tables .The user has the option to print the data presented in the table for future verification.

The call graph is generated using JgraphT library package, where the nodes of the graph indicates the functions and the links indicate the function calls. Custom Look And Feel package is used for better presentation of the user interface.

III. IMPLEMENTATION

***A.* Identification of the functions**

This module involves thorough scanning of the input file to identify all the subroutines used. The logic to identify the name of the function is to read the input file into a buffer and using string tokenizer split it into tokens. Now, each tokenizer is analyzed for the function names, the token before the opening parenthesis is the function name. Different styles of writing the function prototype are considered here. Function names are distinguished from the keywords like for, if, switch, while, etc. which are followed by opening parenthesis. If a function name is repeated, the duplication is avoided by checking in the function names array. Thus, every function is identified and stored in the function array for analysis in the further stages.

***B.* Identification of the function metrics**

This module deals with the scanning of each of the function's definition listed in the function name array. For each function the number of lines in the function definition are counted. The logic for this is to increment the bracket counter when an opening flower bracket is encountered and decrementing the bracket counter when a closing bracket is encountered in the function definition and counting the no. of lines in between these flower brackets. When the bracket counter becomes zero, the function definition is said to be completed. After that, the function calls are analyzed. For this the logic is, function name followed by parenthesis and a semicolon. In this way, the function calls and number of lines in the function are outputted in a table for all the functions in the function name array. During scanning of the functions or the program, we should take care of the comment lines as they are not executed and are to be skipped off here.

***C.* Call Graph Generation**

This module deals with the generation of the call graph from the information obtained from the modules 1 and 2. The call graph is shown in a frame which is generated as a normal application but not as an applet window. Each function name is presented as a vertex of the graph in a rectangular box. Care should be taken in the alignment of the vertices using the x and y co-ordinates of the position, as the graph should not get cluttered after generation. The graph is drawn in a hierarchical way starting from the "main" function. The function calls are represented by the edges drawn from the calling function to the called function with the calling sequence mentioned above the edge. The generated graph can also be manually aligned after auto alignment so as to get the analysis of the function calls in the user desired way. The orphan nodes in the graph, i.e nodes with no other node connected to it can be omitted from the program as it not being called by any other function in the program.

IV. RESULTS AND DISCUSSION

A. *JgraphT implemetation*

The package org.jgrapht.demo includes small demo applications to help you get started. If you spawn your own demo app and think others can use it, please send it to us and we will add it to that package. To run the graph visualization demo, try executing this command in the lib directory: *java –jar jgrapht-demo-x.y.z.jar*

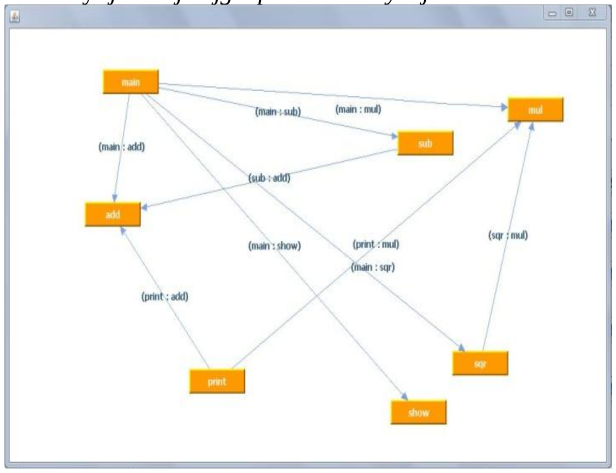
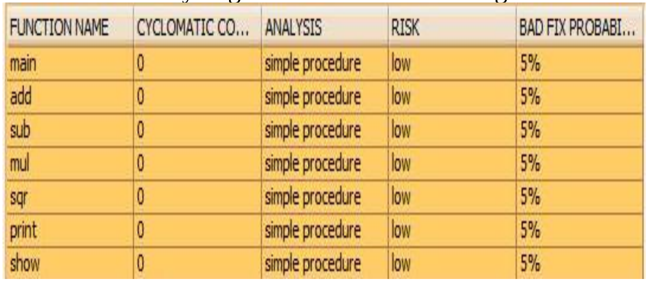


Fig 2: Output of Call graphs

After calculating the cyclomatic complexity, user will see this table for analyzing the code as shown in figure 2 and 3.



FUNCTION NAME	CYCLOMATIC CO...	ANALYSIS	RISK	BAD FIX PROBABI...
main	0	simple procedure	low	5%
add	0	simple procedure	low	5%
sub	0	simple procedure	low	5%
mul	0	simple procedure	low	5%
sqr	0	simple procedure	low	5%
print	0	simple procedure	low	5%
show	0	simple procedure	low	5%

Fig 3: Analysis by cyclomatic complexity

B. *Problem Specification*

The human understanding of the programs becomes difficult as they become complex and no statistical information of them is provided. This requires a systematic analysis of the program for the programmers to clearly understand what is actually written in the source code.

C. *Evaluation and Synthesis*

The system has to be designed only after complete evaluation of the problem, upon which we can see that a lot depends on the analysis of the program. In the proposed system the information about the program is very effective and convenient. The whole information is shown in an organized way which helps the programmers with graphical information as well as the statistical information about the program.

D. *Limitation*

Call graph only provides the basic metrics for the program, since it causes to be a static and dynamic linking program. If there is no main function it will show just an error message and terminates. The links will be increased when function calls are high. So there is a possibility for overlap.

E. *Scope and Future*

The scope of the proposed call graphs in underlying theories and practical problems is undeniable because of its approach towards solving and analyzing programs Its platform independence as a programming tool gives a wide scope for its usage and future. On broader aspects the following are some of the main points that can be listed out as the scope for call graphs in mere future.

- Provision for identifying the unused variables in the program.
- A scrollable presentation of the Call Graph
- Self-organized graph can be developed.
- Dynamic call graph is more useful than static one.
- More metrics can be shown to the user.

V. CONCLUSION

This paper implements a system that can be considered as a software analytic tool which is supportive in understanding and analyzing the C program. An uncomplicated tool that would make the programmer supplement with the functionalities of the program through call graphs. The automatically generated call graphs and assist the programmers to modify, test, document, explain and maintain the code or read other's code. Unlike the graphs that are made by the programmer with paper and pen. The following are some of the major applications that are taken into consideration.

- Orphans or unused functions can be found through call graphs.
- Review of a software program.
- Modifying of the program can be done.
- Maintenance of the code.
- The utility of the functions used in a program in another program can be checked in the future through call graphs.
-

VI. REFERENCES:

[1] A J Sojev. "Basis Path Testing".

[2] McCabe (December 1976). "A Complexity Measure". IEEE Transactions on Software Engineering: 308–320. Template:Working link

[3] Belzer, Kent, Holzman and Williams (1992). Encyclopedia of Computer Science and Technology. CRC Press. pp. 367–368.

[4] Harrison (October 1984). "Applying Mccabe's complexity measure to multiple-exit programs". Software: Practice and Experience (J Wiley & Sons).

[5] Diestel, Reinhard (2000). Graph theory. Graduate texts in mathematics 173 (2 ed.). New York: Springer. ISBN 0-387-98976-5.

[6] McCabe (December 1976). "A Complexity Measure". IEEE Transactions on Software Engineering: 315.

[7] McCabe, Watson (1996). "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric".

[8] John Sichi, JGraphT - Project Administrator.

AUTHORS BIOGRAPHIES

#1 Pravin Kandala is currently in his 4th year at the Guru Nanak Institutions Technical Campus pursuing Bachelors in Computer Science and Engineering. And he is a Microsoft Student Partner. Member at National App Review Board Microsoft. Holds Microsoft Associate, Microsoft Certified Professional and Microsoft Certified Solution Developer on Web platform and Windows store. Has developed apps on Windows 8, windows phone, blackberry platforms. Over the past four years he has been actively participating in projects, technical seminars and workshops. He avers Computer research oriented study and has been working on Cloud computing, Mobile applications, algorithms, Augmented Reality, Artificial Intelligence, Image processing. He can be reached at pravinkandala07@gmail.com

#2 Jasmeen Kaur is currently in her 4th year at the Guru Nanak Institutions Technical campus pursuing Bachelors in Computer Science and Engineering. Over the past four years she has been actively participating in several technical presentations, seminars & workshops. Member and organizer at Hyderabad Youth Assembly. She holds certifications for Microsoft technology associate and Microsoft certified solution developer. Has developed Windows 8 apps. Her major interest of research lies in the areas of Computer and Developing, Network Security, Computer vision, Theoretical Computer Science, Data Structures, Machine learning. She can be reached at jasmeenkaur807@gmail.com

Professor T.Venkat Narayana Rao, received B.E in Computer Technology and Engineering from Nagpur University, Nagpur, India, M.B.A (Systems), holds a M.Tech in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, A.P., India and a Research Scholar in JNTU. He has 21 years of vast experience in Computer Science and Engineering areas pertaining to academics and industry related I.T issues. He is presently working as Professor, Department of Computer Science and Engineering, Guru Nanak Institutions Technical Campus, Ibrahimpatnam, R.R.Dist., A.P, INDIA. He is nominated as an Editor and Reviewer to 38 International journals relating to Computer Science and Information Technology and has published 52 papers in international journals. He is currently working on research areas, which include Digital Image Processing, Digital Watermarking, Data Mining, Network Security and other emerging areas of Information Technology. He can be reached at tvnrbobby@yahoo.com