A Novel Approach to Bug Localization using Call Graph Reduction

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Prabhdeep Singh
801031022

Under the supervision of:
Dr. Shalini Batra
Designation of Supervisors

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"An Novel Approach to Bug Localization using Call Graph Reduction"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Shalini Batra* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Prabhdeep Singh

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Dr.Shalini Batra
Assistant Professor,
CSED,
Thapar University,
Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgment

I express my sincere and deep gratitude to my guide Dr. ShaliniBatra, Assistant Professor in Computer Science & Engineering Department, Thapar University Patiala, for the invaluable guidance, support and encouragement. He provided me all resource and guidance throughout thesis work.

I am heartfelt thankful to Dr. Maninder Singh, Head of Computer Science & Engineering Department, Thapar University Patiala, for providing us adequate environment, facility for carrying out thesis work.

I would like to thank to all staff members who were always there at the need of hour and provided with all the help and facilities, which I required for the completion of my thesis. At last but not the least I would like to thank God and my parents for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Prabhdeep Singh
801031022

Abstract

Testing the software is the process of validating and verifying a software program. Every software is expected to meet certain needs so, when software is developed major requirement is to check whether it fulfills those needs or not. Bug localization is an important part of software development and elimination of bugs from the software depends upon how much efficiently testing is done on the software.Automated solutions for bug localizations aim to reduce human effort and software maintenance cost. One of the techniques for automating bug localization is usage of call graph and the one of the waysto utilize call graphs of program executions in graph mining is by applying various algorithms to fix such problem. Since size of the call graph generated is quite large, researchers have proposed various techniques and methods for call graph reduction.

An algorithm is proposed for the call graph reduction which uses call graph frequency to store the information of the each weight of the node and matrix to store the information about the node. It has been experimentally depicted that the applied algorithm reduces the size of the call graph without changing the basic structure without any loss of information.

Once the graph is generated from the source code,it is stored in the matrix and reduced appropriately using call graph frequency. After call graph reduction various graph mining techniques can be applied to localize the bug.The proposed algorithm is compared to another call graph reduction techniques and it has been experimentally evaluated that the proposed algorithm significantly reduces the graph and make it efficient for bug localization. The output generated using the proposed methodology shows promising results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's era, software industries are competing with quality of the software which depends upon the sound software testing phase. Most of software contains some bugs after being released so it is most challenging to localize bug automatically and fix them before release. A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program.

1.1 Software testing

1.1.1 Introduction to Software Testing

Testing the software is the process of validating and verifying a software program. Every software is expected to meet certain needs so, when software is developed major requirement is to check whether it fulfills those needs or not. For example, banking software is entirely different from software required in a shop. The needs and requirements of both those software are different. Hence it is important to check its potential. The main goal of software testing is to know the errors of the software before the user finds them[1].

1.1.2 Software testing in software industry

Sometime it is experienced that software does not work as expected and it can have a large impact on an organization. It can lead to many problems including:

- Loss of money – this can include losing customers right through to financial penalties for non-compliance to legal requirements
- Loss of time – this can be caused by transactions taking a long time to process that can include staff not being able to work due to a fault or failure
- Damage to business reputation – if an organization is unable to provide service to their customers due to software problems the customers will lose confidence or faith in this organization (and probably take their business elsewhere)

- Injury or death – It might sound dramatic but some safety-critical systems could result in injuries or deaths if they don't work properly (e.g. flight traffic control software)

1.1.3 Testing in software development

Earlier software was handed over to the client after a quick check carried out on it. Testing the application was considered as waste of time and resources. But now a days software testing is as much important as developing software. Some major issues are necessary in Software development phase which include:

i. **Localization of bugs and errors:** The bugs and errors in the application can be determined by testing. The bugs can be at unit level or system level.
ii. **Information of software:** By testing, the state of Software and service quality is known. The stakeholders get information through testing phase about service quality also.
iii. **Improvement in Software quality:** Testing starts with known conditions, uses predefined procedures, and has predictable outcomes and helps to improve product quality.
iv. **Technical importance:** As testing has to come out with best technically sound application, it is important for technical aspect in any Software Development lifecycle.
v. **Bug Free Software:** If bug free software is delivered to the client, software testing has to be carried out in detail. Even a small mistake in the software can affect the client.
vi. **Safe and secure application:** With getting away in all phases of testing, more sound, safe and secure software application is developed.
vii. **Economic importance:** A well-tested software application will have good economic impact. This is because any one would love to go with reliable and trusted application in market.

There are many kinds of Software Development Life Cycle models and each of these models makes use of testing phase. Hence, this makes testing a very important part in any software development life cycle. Various types of testing like unit testing, integration

tests, system testing, regression tests and user acceptance testing helps any developer to come up with a reliable and trusted web application that can be useful. Testing also follows its own lifecycle like test analysis, test plan, test design and test execution.

1.2 Bug localization

1.2.1 Debugging

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or art of determining the location a bug in a program. It starts from possibly unknown initial conditions and the end cannot be predicted, except statistically and the duration of debugging, cannot be constrained. It demands intuitive leaps, conjectures, experimentation, intelligence and freedom which is impossible without detailed design knowledge. Usually, the most difficult part of debugging is finding the bug in the source code. Programs known as debuggers exist to help programmers locate bugs by executing code line by line, watching variable values, and other features to observe program behavior. Without a debugger, code can be added so that messages or values can be written to a console or to a window or log file to trace program execution or show values.

1.2.2 Bug Localization

Bug localization is an important part of software development. It is a process of identifying the set of statements in a program that cause the program to fail. It is a routine task in maintenance of software. Bug can be defined as the abnormal behavior of the software. The elimination of bugs from the software depends upon how much efficiently testing is done on the software. Bug localization becomes harder due to increasing size and complexity of current software. Extensive research has been made in field of software reliability and various algorithms have been developed for bug localization.

Technique for bug localization is either static or dynamic. Static technique deals with source code and program executions are analyzed in dynamic technique. Automated solutions for bug localizations aim to reduce human effort and software maintenance cost. Katz and Anderson [2] describe the debugging process of a program as chain of three steps:

i. Finding the potential location of bug.
ii. Fixing it, and
iii. Testing the program.

1.2.3 Approaches for locating bugs in software

There are many ways to find bugs in software. Current solutions for software bug localization can be classified into two approaches static and dynamic

1.2.3.1 Static Approach

Static approach finds bugs by analyzing the source code and does not require running the program. Major advantages of static techniques are:

Early Detection of Software Errors: If the error is detected in the first phase of software development it is cheaper to fix it. The earlier software bugs are located, the cheaper it is to fix them. Static analysis works on program source. It can be carried out immediately after the code is written.

Coverage: Static analysis can explore all possible execution scenarios thoroughly and systematically. It can offer better coverage than dynamic tools. Static analysis is particularly effective at finding bugs on obscure code paths; it does not require test cases.

Cost and applicability: Static analysis can be used to analyze low-level system software that has stringent timing and resource constraints.

1.2.3.2 Dynamic approach

A dynamic approach is used which instrument the program and locate bugs by analyzing information collected from one or more executions. The advantage is that dynamic techniques consider feasible paths in a program. Another advantage of this is precision. With a few exceptions, they generate accurate bug reports as the information upon which bug reports are based is collected from actual program executions. However, there are some limitations on dynamic approach:

i. Performance: Due to instrumentation dynamic approach typically impose significant time and space overheads. For example, programs instrumented by Purifyrun approximately two to five times slower and consume over 50% more

memory than their unmodified counterparts. As such, dynamic approach are mostly used during testing and debugging and rarely deployed in production environments [3].

ii. Coverage: In dynamic techniques information is collect from concrete program executions and is therefore limited by the quality of the test suite. While they are effective at exploring bugs along commonly executed code paths, bugs hidden in obscure corner cases remain undetected.

iii. Limited applicability: Instrumentation introduces incompatibilities and therefore the applicability of dynamic approach is limited. Programs with stringent timing requirements and resource constraints (e.g., operating system kernels) are not usually checked by using dynamic approach.

1.3 Graph Mining

Graph mining is relatively young disciplines in data mining. There are many different techniques as well as numerous applications for graph mining. The most prominent application is the analysis of chemical molecules. As the NP-complete problem of sub graph isomorphism is an inherent part of frequent sub graph mining algorithms[4]. As a result, common graph mining algorithms do not scale for these graphs. Graph mining is the key to deploy a suitable call-graph-reduction technique to make use of call graphs which reflect the invocation structure of specific program executions. Such techniques help to alleviate the scalability problems to some extent and allow making use of graph-mining algorithms in a number of cases

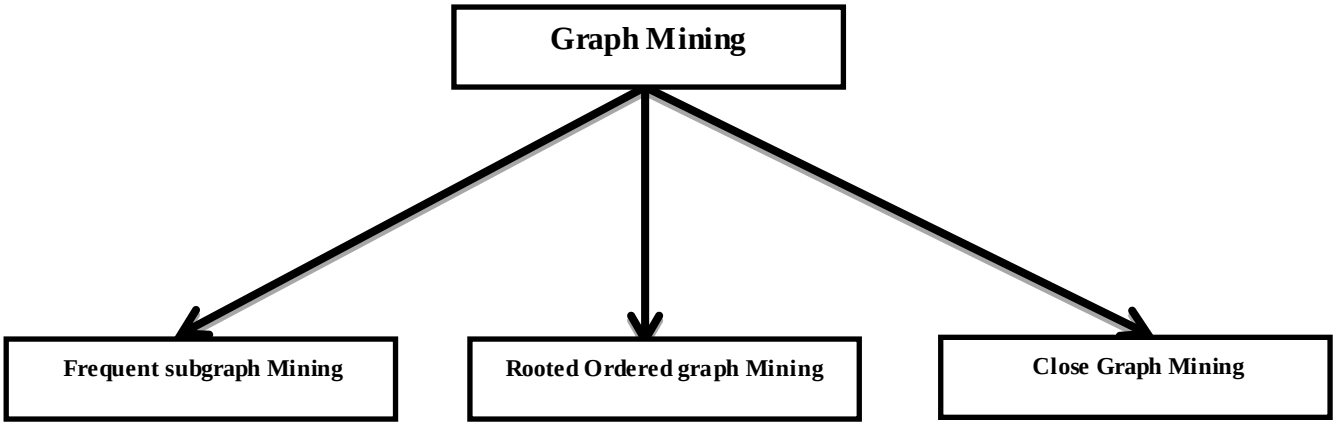


Figure 1.1: Types of Graph Mining Techniques

- **Frequent sub graph mining**: Frequent subgraph mining searches for the complete set of subgraphs which are frequent within a database o graphs, with respect to a user defined minimum support. Respective algorithms can mine connected graphs containing labeled nodes and edges. Most implementations also handle directed graphs and pseudo graphs which might contain self-loops and multiple edges. In general the graphs analyzed can contain cycles. A prominent mining algorithm is gSpan [5].
- **Rooted ordered graph mining**: Graph mining algorithms work on databases of graph and exploit their characteristics. Rooted ordered graph mining algorithms work on rooted ordered graphs. Rooted graph has the main method which is considered as root node in call graphs. Ordered graphs preserve the order of outgoing edges of a node, which is not encoded in arbitrary graphs. Thus, call graphs can keep the information that a certain node is called before another one from the same parent. Rooted ordered graph mining algorithms produce result sets of rooted ordered graphs. They can be embedded in the graphs from the original graph database, preserving the order. Such algorithms have the advantage that they benefit from the order, which speeds up mining significantly. Techniques in the context of bug localization sometimes use the FREQT rooted ordered graph mining algorithm [6]. But this can only be done when call graphs are not reduced to graphs containing cycles.
- **Closed frequent subgraph mining**: Closed mining algorithms differ from regular frequent subgraph mining in the sense that only closed sub-graphs are contained in the result set. A subgraph 'sg' is called closed if no other graph is contained in the result set which is a supergraph of 'sg' and has exactly the same support. Closed mining algorithms therefore produce more concise result sets and benefit from pruning opportunities which may speed up the algorithms[7].

1.3.1 Bug localization using graph mining

Steps followed in bug localization using graph mining include:

Step 1: Making call graph from program executions

Finding bug from the program execution become easier when converted into call graph. Therefore conversion of program execution into call graph is the foremost requirement. All the methods from software are found and appropriate call graph is crested.

Step 2: Reduction of call graph

Call graphs are very large in size therefore the complexity in finding bug from program execution increases. Hence reduction of call graph is required to overcome the huge size of call graph. The most important focus while reducing a call graph is that no information of program is lost or changed. Many techniques are available for reducing call graph as total reduction, total reduction with edge weights,*etc*.

Step 3: Mining of reduced call graph

For bug localization from a reduced call graph, the first step is to mine the call graph. Then the results are analyzed and bug is localized. Algorithms are available for mining reduced call graph as closegraph,frequent sub graph mining(FREQT),rooted ordered graph mining,gSpan.

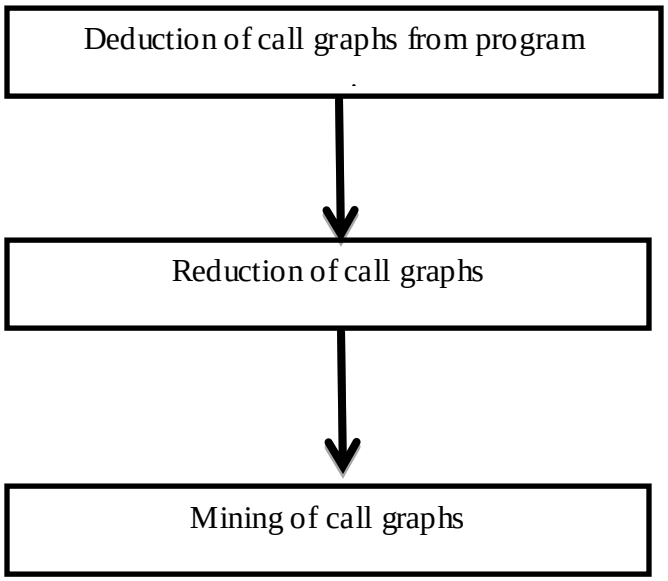


Figure 1.2: Steps of Bug Localization using Call Graphs

1.3.2 Role of call graph in graph mining

Graph is one of the most widely used structures for abstraction. In computer science graphs are used for different applications as program dependency graph, control flow graph and Call graph. Program dependency graph capture the data/control dependency among program statements. Control flow graph capture information about how the control is transferred by a program. By PDG and CFG it is very difficult to retrieve frequent sub structure because they contain minute details of program. They also contain start node and end node but they don't have the proper information about functions. So call graph is proposed. Call graph is well defined, well studied structure in call graph.

When call graph of a program is made frequent sub structures are found by different mining algorithms. After analyzing its results it is used in different applications as bug localization.

1.3.3 Need of call graph reduction

In the application of graph mining in software bug localization, reduction plays an important role. No doubt several techniques as total reduction, total reduction with edge weight, etc. are available to reduce the call graph but there are drawbacks in some cases while reducing call graph by these techniques, sometime information of program is lost or changed, loosing or changing of information affects the accuracy of program that is unacceptable[8]. A new technique is proposed to overcome the draw backs.

1.4 Structure of the Thesis

The rest of the thesis is organized in the following order:

Chapter 2 – Literature Review: This chapter reviews software testing techniques, call graph for storage and reduction for localization of bugs in software.

Chapter 3 - Problem Statement: It states the problem and provides the methodology used to solve it.

Chapter 4 - Call Graph Representation: This chapter gives a detailed introduction about call graph, algorithm is proposed for the storage and reduction without changing its basic structure and information.

Chapter 5 - Implementation Details: It includes the implementation of algorithm using a real time example and the results evaluated.

Chapter 6 - Conclusion and Future Scope: It concludes the thesis and provides suggestions for future work. Thesis concludes with references and list of publications.

Chapter 2

Literature Review

Debugging process in software industry has been ongoing for decades but in the past few years it has tremendously contributed in the development of reliable and robust software. It includes everything from dealing with test cases, observing program executions, localizing defects and ultimately fixing them. Most important task in software testing is bug localization, in which a developer uses information from a bug report to locate the source code elements in the subject software system that must be modified to correct the bug [9, 10]. It has also been described as the process of relating a failure to an infection to defect. Bug localization is an emerging area of research for computer scientists using graph theory to identify bug in a wide range of software product [11, 12]. Similarly graph mining has been expanding in all directions at an astonishing rate during the last few decades.

Software industry is reliable only when its software has reliability and it depends on its being bug free. Two approaches for finding bugs in software are used: static analysis and dynamic analysis [13]. Silly mistakes of programmer like spelling mistakes, typing mistake are found by static analysis. Static represent of source code are called Control-flow graphs (CFGs) which are frequently used in compiler technology [14]. In a CFG, this is divided into several so-called basic blocks. On the other hand when mistake found in modules in software as calling of functions are not proper, control flow graph and other graphs do not find these things properly. So call graph is used to verify the calling of functions, this approach is a dynamic approach [15].

A graphical model is devised that supports the process of debugging software by guiding developers to code that is likely to contain defects. The model is trained using execution traces of passing test runs; it reflects the distribution over transitional patterns of code positions. The model is designed such that Bayesian inference has a closed-form solution[16].Depending on the nature of the problem, different techniques can be used to formulate and solve a typical bug localization problem. The most valuable data structure

such as graph deals with localization problems, in which the objective and constraints can be formulated using graph reduction with respect to identify bugs. The term 'Bug' has been part of engineering jargon for many decades. It's been originally in hardware engineering to describe mechanical malfunctions or problems. During World War II Problems with radar electronics are referred as bugs or glitches. There is evidence that usage of this term date back to much earlier. This term is mentioned letter from Edison to an associate in 1878 [17].

In the beginning of computer era, debugging was something of a hit-or-miss procedure for a quite a few years. Early debugging efforts were mostly centered on either data dumps of the system or used output devices, such as printers and display lights to indicate when an error occurred. Programmer wrote the code line by line until they could determine the location of the problem [18].

The next evolution of debugging came with the advent of command-line debuggers. These simple programs were amazing step forward for the programmer. This allowed a programmer working in an assembler to look directly at the registers and the memory blocks that contained the address of the local variables of the program. Reproducing something is the next step toward making it a scientific or engineering process. As the software projects started getting bigger and the same technique that worked well for small projects no longer worked when program reached certain size. Scalability was an issue even at the beginning of software complexity curve. Compiler vendor discovered that the information they had while they parsing high level languages such a C, FORTRAN, COBOL, *etc.* could be kept in a separate file called symbol map, which could map the variables names in the programs to the actual memory address into which they could load at runtime. The ability to look at variables names and map them to memory address allowed programmer to dump memory by name.

Next big thing in debugging was the ability to set breakpoints, the term breakpoint means the ability to "break" into the code and stop program execution at a given point. The program still loaded in memory but it just would not be running at the breakpoint when certain critical areas of code were reached and allowed programmer to dump the contents of variables (symbols) before the program crashed or before continuing execution [17].

Before breakpoint improvement, programmer had only two states: Initial state of application before it ran and the final state of the application. Due to breakpoint programmers can check state when they want it and still guessing where to set a breakpoint was still difficult. Next big thing in modern debuggers is Turbo Pascal with IDE (Integrated development environment) and this was introduced by Borland where you can edit, compile, link and debug code in the same system. No need to run programs and no need to load special symbol table or use special compiler options. Turbo Pascal signified the dawn of the modern age of debugging.

Next comes true multi-threaded, multi-tasking UNIX operating system, where using debuggers to debug multi-threading was very hard. Previous application debuggers were text based. But, to debug GUI based applications needed additional thought for debugging. Existing debugger were not supporting this, one need to flip back between application and debugger screen, but this was of little help when it was an application that "painted" screen. If the application was not running, the screen would be blank. IDE and debuggers are able to debug large projects but, still there are lot of innovation should happen in debugging areas. Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated 59.5 billion annually, or about 0.6 percent of the gross domestic product [19].

For the purpose of bug localization, concept of retrieval from large software libraries is also used which compare generic text models and certain composite variations. The task is to locate the files that are relevant to a bug reported in the form of a textual description by a software developer. The retrieval effectiveness for the various models was measured using the some metrics [20].

Graphs, an important role in computer engineering, can represent the complex problems of computer science pictorially and the solution of problem becomes easy [21]. Many approaches for converting the problem into graph are available which are represented as graph theory which shows relationship between objects, dependency among objects, flow of data, etc. showing of relationship is flexible as graphs can be either directed or undirected. The study of graphs theory was first systematically investigated by D. Konig in the 1930s. Similarly graphs in software testing can also be used to represent many

different concepts; for example if the graph represents a source code of a program, the node could represent the objects and flow of program is represented by edges. Graph mining has an important application in a wide variety of practical fields, including computational biology, sociology, software bug localization, keyword search, and computer networking. It also helps to transform the graphical data into graphical information to investigate recurring patterns in real-world graphs, to gain a deeper understanding of their structure [22].

Call-graph-based defect localization naturally relies on call graphs. Such graphs are representations of program executions [23]. Graph reduction technique is used to reduce the complexity and represent in the form of frequent sub graph to analyze it further to determine bug area. Raw call graphs typically become much too large for graph-mining algorithms, as program might be executed for a long period and frequently call other parts of the program, which adds information to the graph. Therefore, it is essential to compress the graphs; this process is also called reduction which is usually done by a lossy compression technique. This involves the trade-off between keeping as much information as possible and a strong compression.

Unreduced call graphs can be obtained by tracing a program execution. They are rooted ordered trees [24]. Nodes stand for methods and one edge stands for each method invocation. The order of the nodes is the temporal order in which the methods were executed.

The total reduction technique is probably the easiest technique, and it yields good compression. In totally reduced graphs at the function level, every distinct function is represented by exactly one node. When one function has called another function at least once in an execution, a directed edge connects the corresponding nodes. Another technique for call graph reduction is based on the compression of iteratively executed structures using frequent usage of iterations in today's software.

After going through various research proposals in the area of localizing bug, it was realized that algorithm should be designed to reduce of call graph without changing its basic structure and information as it is based on the idea that nodes of a call graph can be

stored with its parent information in a matrix under some meaningful definition of equivalence. So the whole thesis work can be grouped into following basic steps:

i. Consider a call graph and identify the functions.
ii. Generate a new reduced call graph and store it using proposed algorithm.
iii. Once the reduced call graph is obtained, make it ready for mining to localize bug using graph mining techniques.

Chapter 3

Problem statement

In present time the quality of software is a major factor in software industry and it can be maintained if software is bug free. Once the bugs are identified in the software all the effort is put to remove it using various static and dynamics techniques. Graph mining is effective dynamic technique for bug localization as it uses call graph to represent the program. Call graph is mined efficiently if it is reduced in a proper way. Researchers have proposed a number of techniques to reduce the graph but most of the techniques suffer from one or more shortcomings. Majority of techniques are not able to store the graph with all information in computer memory. So, some new techniques or algorithms are needed to store the information of graph in computer memory and reduce the graph in such a manner that its information is not lost and graph is easily mined.

Major objectives laid down in this thesis are:

- To find an efficient method to store the graph in computer memory with information about all the nodes and its parents.
- Once the storage is done efficiently, reduction of graph is required. Graph must be reduced in such a manner that its information is not lost and it should have minimum edges and nodes.
- Structure of graph should not be changed after reduction.

Methodology

- The step-by-step methodology for in storing, reducing and analyzing the call graph is as follows:
- Matrix has been used to store in computer memory the information of every node with its parent.

- Reduction has been done without changing the connectivity of graph level as well as the structure.
- To point out call frequency, every edge is annotated with a numerical weight. It represents the total number of calls of the callee function from the caller function.

Chapter 4

Call Graph

Graph representations of data are increasing in a variety of applications, including computational biology, social network analysis, web applications, and many others. There has been much work on reducing the complexity of graph data; in particular, graph reducing algorithms have been developed for both classification and regression on graphs. Graphs are mathematical concepts that have found many application of computer science, have many different flavors of representation: CFG, PDG and Call Graph.

4.1 Graph

A.Aho and J.Ulman acknowledge that "Fundamentally, computer science is a science of abstraction." Computer scientists must create abstractions of real-world problems that can be represented and manipulated in a computer [25]. Most of the time the process of abstraction is simple, for example, scheduling final exams. For successful scheduling we have to take into account associations between courses, students and rooms. Such set of connections between items is modeled by graphs. The basic idea of graphs was introduced in 18th century by the great Swiss mathematician Leonhard Euler. He used graphs to solve the famous Konigsberg bridge problem. German city of Konigsberg (now it is Russian Kaliningrad) was situated on the river Pregel. It had a park situated on the banks of the river and two islands. Mainland and islands were joined by seven bridges. A problem was whether it was possible to take a walk through the town in such a way as to cross over every bridge once, and only once

4.1.1 Graph representation in memory

Graphs can be implemented by two standard ways. One, called adjacency lists, is used to implementation of binary relations in general. The second, called adjacency matrices, is a new way to represent binary relations, and is more suitable for relations where the number of pairs is a sizable fraction of the total number of pairs that could possibly exist over a given domain.

Adjacency Matrix

Each cell a_{ij} of an adjacency matrix contains 0, if there is an edge between i-th and j-th vertices, and 1 otherwise [26].

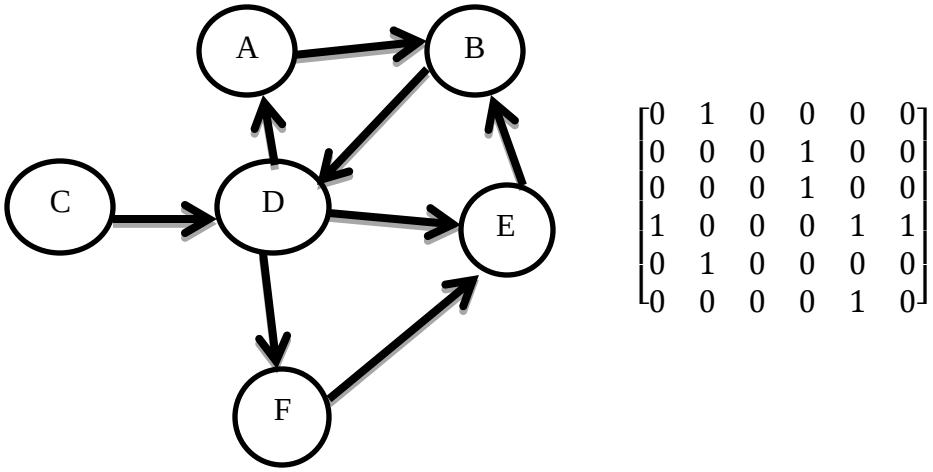
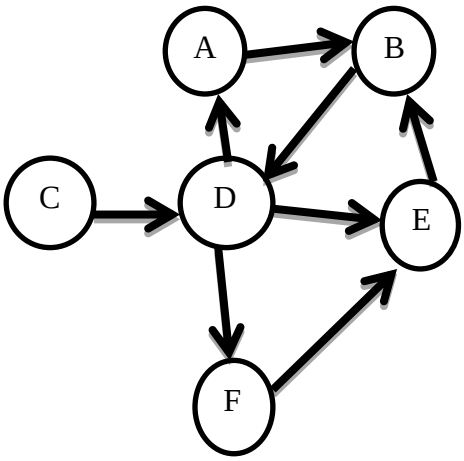


Figure 4.1 Graph Representation using adjacency Matrix

The symmetry of the matrix means that only elements on or below the main diagonal need to be stored. Therefore, all the information contained in the graph is contained in the lower triangular section of the matrix, and the graph could be reconstructed from this array. The adjacency matrix for a directed graph is not symmetric.

Adjacency list

This kind of the graph representation is one of the alternatives to adjacency matrix. It requires less amount of memory and, in particular situations even can outperform adjacency matrix. For every vertex adjacency list stores a list of vertices, which are adjacent to current one.



A	B
B	D
C	D
D	A E F
E	B
F	E

Figure 4.2 Graph Representation with adjacency List

4.1.2 Graphs in Software Engineering

The aim of software engineering is to develop large software systems that meet quality and cost requirements. The development process which moves from the initial problem to the software solution is based on models that describe and support the development during the different phases. Models are key elements of many software engineering methodologies to capture structural, functional and non-functional aspects. Popular methodologies prescribe various models with different degrees of formality, flexibility, and analyzability to solve the different problems. For example, UML proposes some semi-formal diagrammatic languages: class, object, component, and deployment diagrams for modeling structural aspects, and use case, sequence, activity, collaboration, and state chart diagrams for modeling behavioral aspects [27].

Graphs have been used for a long time in different sub disciplines of software engineering. The most important distinction in the graphs is static or dynamic, i.e., whether they represent aspects from the source code or from program executions, respectively. Graphs are very useful means to model concepts and ideas and to describe complex structures and systems in a direct and intuitive way. In particular, they provide a simple and powerful approach to a variety of problems that are typical to software engineering [28]. For example, bubbles and arrows are often the first means to reason on

a new project, but also the structure of an object-oriented system or the execution flow of a program can be seen as a graph.

4.1.2.1 Program Dependence Graph

A Program Dependence Graph (PDG) consists of nodes and directed edges [29]. Nodes represent program statements and directed edges represent control or data dependences between the nodes. They carry all information about the dependencies within programs and so they can be used to determine the influence of a change or extension on proven properties. They present the opportunity to interpret different parts of the same program in different ways. It depends on the part being executed or the input being used or both. It was conceived specifically for program optimization, accommodating standard optimizations as well as including control- and data-dependence information. Some of the advantages of PDG over other graph based representations are:

They are easier to understand, besides being a vehicle for performing optimizations.

They enable "cross-compilation" of programs written in a language suited to one execution paradigm onto a different one.

They support data- and demand-driven semantics within one representation, facilitating hybrid evaluation schemes like operator nets where some parts of the program graph may be evaluated eagerly and other parts lazily depending on the "need" of the values [30].

4.1.2.2 Control Flow Graphs

Control Flow Graphs (CFGs) are static representations of source code, frequently used in compiler technology [31]. A control-flow graph is a directed graph where nodes in the graph represent actions and the edges indicate the flow of control from one action to another. A control flow graph has two special nodes: the start node and the stop node. The stop node has no outgoing edge and every node in a flow graph lies on some path from the start node to the stop node A node with one outgoing edge is called an action node while a node with two or more outgoing edges is called a branch node. Flow graphs that can be decomposed with sequencing and nesting into elementary flow graphs are called structured flow graphs [32].

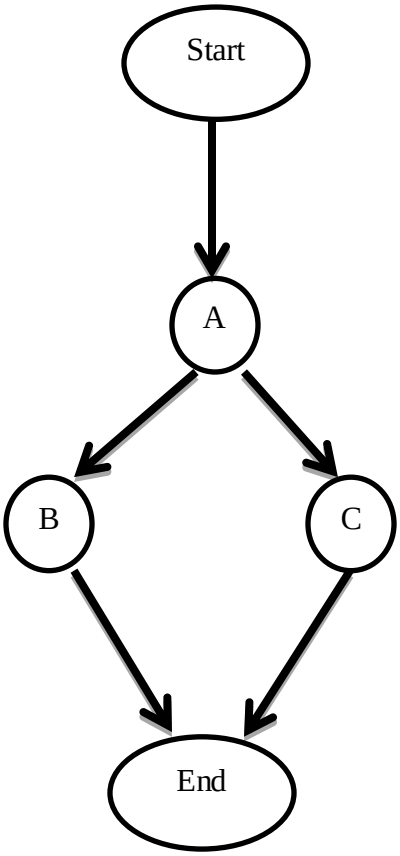


Figure 4.3 If Loop

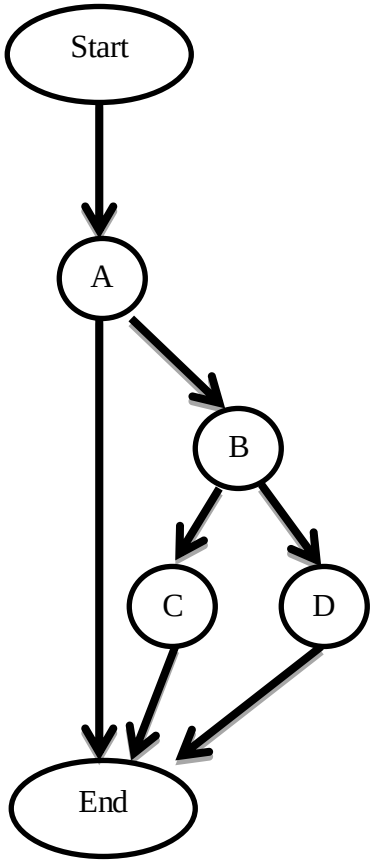


Figure 4.4 Nested If

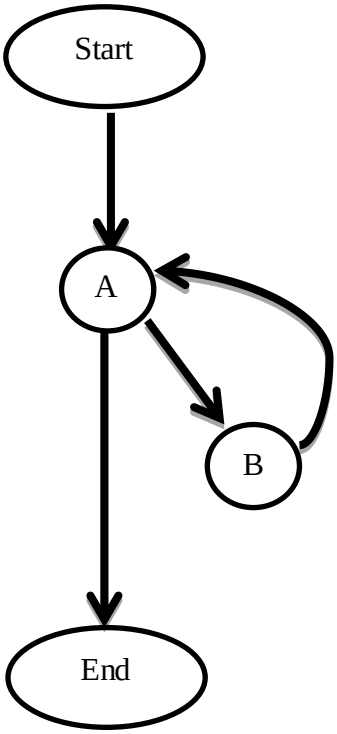


Figure 4.5 While loop

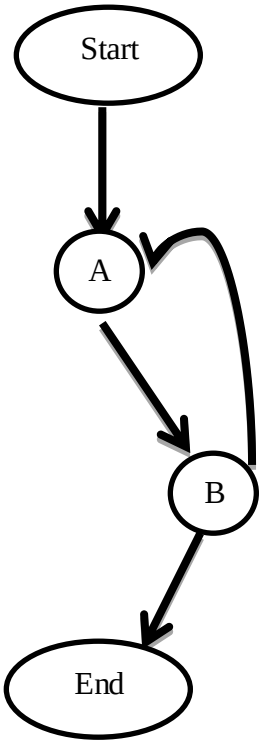


Figure 4.6 Do while

Graphs have been used for a long time in different sub disciplines of software engineering. The most important distinction is if the graphs are static or dynamic, i.e., if they represent aspects from the source code or from program executions, respectively.

4.1.2.3 Call graph

Call graph is one of the most widely used structures for abstraction. It is very useful as a data structure to support the automation of propagating information across program elements. It is useful when rendered visually to help programmers understand the code. Call graph is different from the regular procedure graphs. Each procedure is represented with one node and is identified with a name. Dependencies are represented with directed edges. If, for example, procedure A calls procedure B there will be a directed edge from procedure A to procedure B.

4.2 Call Graph Representation

4.2.1 Call graph definition

A call graph is a directed graph whose nodes represent the functions of program and directed edges symbolize function calls [33]. Nodes can represent either one of the following two types of functions:

i. Local functions, implemented by the program designer.
ii. External functions: system and library calls.

Local functions are the most frequently occurring functions in any program. They are written by the programmer of the binary executable. External functions, as system and library calls, are stored in a library as part of an operating system. Contrary to local functions, external functions never invoke local functions. Call graphs are formally defined as follows:

Definition (Call Graph): A call graph is a directed graph G with vertex set V=V(G), representing the functions, and edge set E=E(G), where $E(G) \subseteq V(G) \times V(G)$, in correspondence with the function calls[34].

4.2.2 Types of Call Graph

Static Call Graph

A static call graph can be obtained from the source code. It represents all methods of a program as nodes and all possible method invocations as edges. Discovering the static call graph from the source text requires two steps: finding the source text for the program, and scanning and parsing that text.

Dynamic Call Graph

A dynamic call graph is the invocation relation that represents a specific set of runtime executions of a program. Dynamic call graph extraction is a typical application of dynamic analysis to aid compiler optimization, program understanding, performance analysis etc.

4.3 Reduction

4.3.1 Reduction definition

Reduction means to simplify a source code, graph *etc*. by removing all unnecessary things. The aim of reduction is to reduce the complexity without losing important information. Reduction is applied in different areas of computer science as network routing consensus and cooperation in multiagent systems, image processing, statistical learning, neuroscience studies of functional relationships in the brain, and in distributed control of networked dynamical systems etc.

4.3.2 Graph Reduction

Even smallest software is made after thousands lines of code which have many functions, methods, etc. It is required that all these things must be stored in computer memory without losing information. When a program is converted into graph its size is extremely big and there is complexity in it. A large memory space in computer is needed which affects the cost of a program. That is why graph is reduced. The unnecessary nodes or edges are deleted in such a way that important information is not lost. When the size becomes smaller the program is interpretable, its complexity is reduced. It will take smallest space in computer memory and the cost of program becomes lower. Graph reduction can be divided into two classes: pure graph reduction and programmed graph

reduction. The pure graph reduction applies fixed reduction rules to a graph according to an appropriate reduction order. The programmed graph reduction executes programs of the graph nodes selected by the demand-driven execution. In the programmed graph reduction, a program can be user-defined function.

4.3.3 Call Graph Reduction

Call graph are representations of program executions. Raw call graphs typically become much too large. The program might be executed for a long period. Therefore, it is essential to compress the graphs by a process called reduction. It is usually done by a lossy compression technique. This involves the trade-off between keeping as much information as possible and a strong compression. A number of different call-graph representations are proposed, standing for different degrees of reduction and different types and amounts of information encoded in the graphs. Call graph doesn't control the flow of program, it represent the relationship among functions/methods. When call graph is being reduced it is essential that no function call is missed. The specifications of all the functions/methods must be clear. It is also noticeable that no information is lost when label is changed. Call frequency must be clearly specified.

4.3.4 Approaches of call graph reduction

There are two approaches of reducing software call graphs

i. Total Reduction (Liu *et al.*)
ii. Zero-one-many reduction (DiFatta *et al.*)

Total reduction is proposed by Liu *et al.* [35].In totally reduced graphs, every function is represented by a node. A direct edge is connected with the corresponding nodes when one function has called another function. Total reduction technique shortens the size of source call graph. This technique has been introduced by Liu *et al.* In this technique, every method occurs just once within the graph. The major shortcoming of this technique is that it changes the structure of the graph. On the other side, much information about the program execution is lost, e.g., frequencies of the execution of methods and information on different structural patterns within the graphs. So it is very difficult to retrieve required information from this reduced graph.

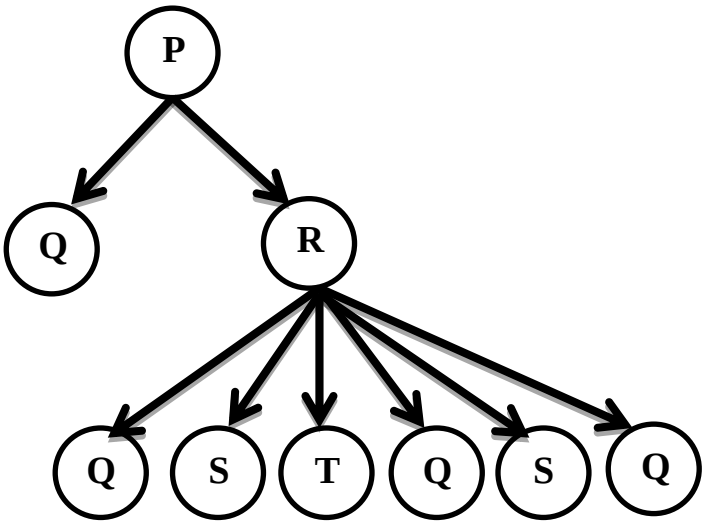


Figure 4.7 Source Call Graph

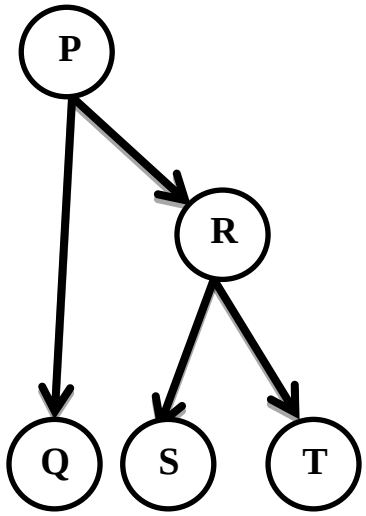


Figure 4.8 Reduced Call Graph

The above graph is derived from the source code, called source call graph. As seen in figure4.7 the graph has 3 levels where P is root node and Q and R are its children. In next level R is considered as Q,S and T's parent. Q is called 3 times so 3 direct edges are connected from R to Q and S is called 2 times so 2 edges are connected with R to S. Function T is called single time so edge is connected from R to T singly. After applying Liu *et al.* approach reduced graph is shown in Figure 4.8. Using this technique the 2nd level children are reduced from two to one. In source call graph P is connected with Q at 2nd level but after reducing it is directly connected with Q in 3rd level of graph. This reduced call graph doesn't show the call frequency of nodes. Its structure has also been changed.

The other approach given by DiFatta *et al.* covers the drawback of Liu *et al.* approach as it doesn't change the structure but the reduction is not properly done [36]. The improper reduction increases its complexity and it is difficult to find frequent sub structure from graph. Reduced graph can provide near information about call frequency but exact information is not known. Call frequencies are important for detecting certain groups of bugs.

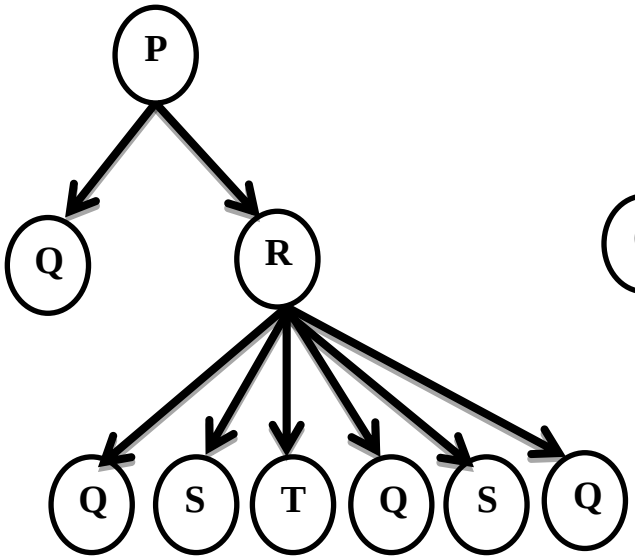


Figure 4.9 Source Call Graph

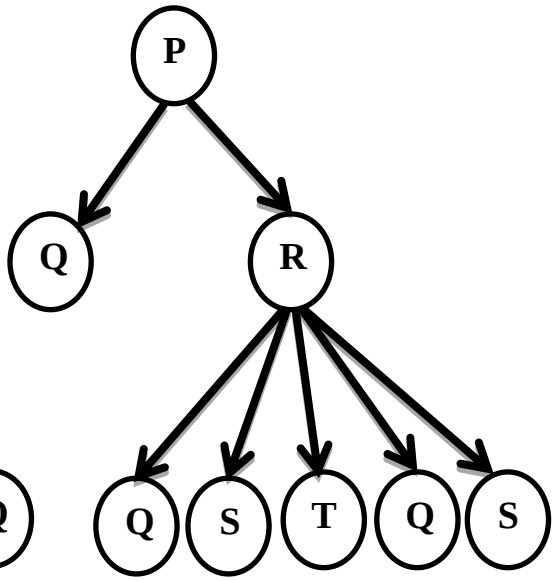


Figure 4.10 Reduced Call Graph

In the figure 4.10 reduced callgraph by approach of DiFatta *et al.* has been shown. Frequency of nodes has been changed but the structure of call graph remains same. In this approach of reduced call graph node is shown two times if it is shown more than two times in source code. The exact frequency of nodes is not known by using this approach.

4.4 Proposed Approach

To overcome the drawbacks of both techniques a new approach is proposed. In this approach the reduced call graph shows the call frequency of each node without changing the structure of source call graph. First of all, all functions of source code are labeled so that it can easily be interpreted. Then a call graph is made using these labeled functions. The main task is to save the node into computer memory with its parent's information which is not possible with adjacency list or adjacency matrix. Therefore the parent of each child is stored in the matrix. Rows represent the levels of call graph as 1st row represent 1st level's nodes, 2nd row represent 2nd level's nodes and so on. Every node also contains the information about its parent. The proposed algorithm to save the node with its parent's information in the computer memory and efficiently reduce the graph is as follows:

Algorithm: Reducing call graph

1. **Input:** Matrix of structure containing children, label, parent
2. **Output:** Reduced call graph
3. **Set** j=Getstr[100][]
4. **foreach** aa←1 to 10 **do**
5. **Set** j[aa - 1] = **Getstr**[200]
6. **end for**
7. **Set** count= **GetArray**(level)
8. **foreach** i←0 to levels-1 **do**
9. print "Enter no. of children at level 0"
10. **Input**(children)
11. count[i]=children
12. **foreach** x←0 to children -1 **do**
13. print "Enter the label of **(x)** children"
14. j[i][x].label = **Input**(label)
15. print "Enter the parent of **(x)** children"
16. j[i][x].parent = **Input**(parent)
17. j[i][x].count = 1
18. **end for**
19. **end for**
20. **foreach**k←levels down to 1 **do**
21. **foreach** l←0 to count[levels-1] -1 **do**
22. **foreach** m←l+1 to count[levels -1] -1 **do**
23. **if** j[k-1][l].label=j[k-1][m].label **AND** j[k-1][l].parent=j[k-1][m].parent **AND** j[k-1][l].parent ! = -1 **then**
    i. j[k-1][m].parent = -1
    ii. j[k-1][l].count++
24. **end if**
25. **end for**
26. **end for**
27. **end for**

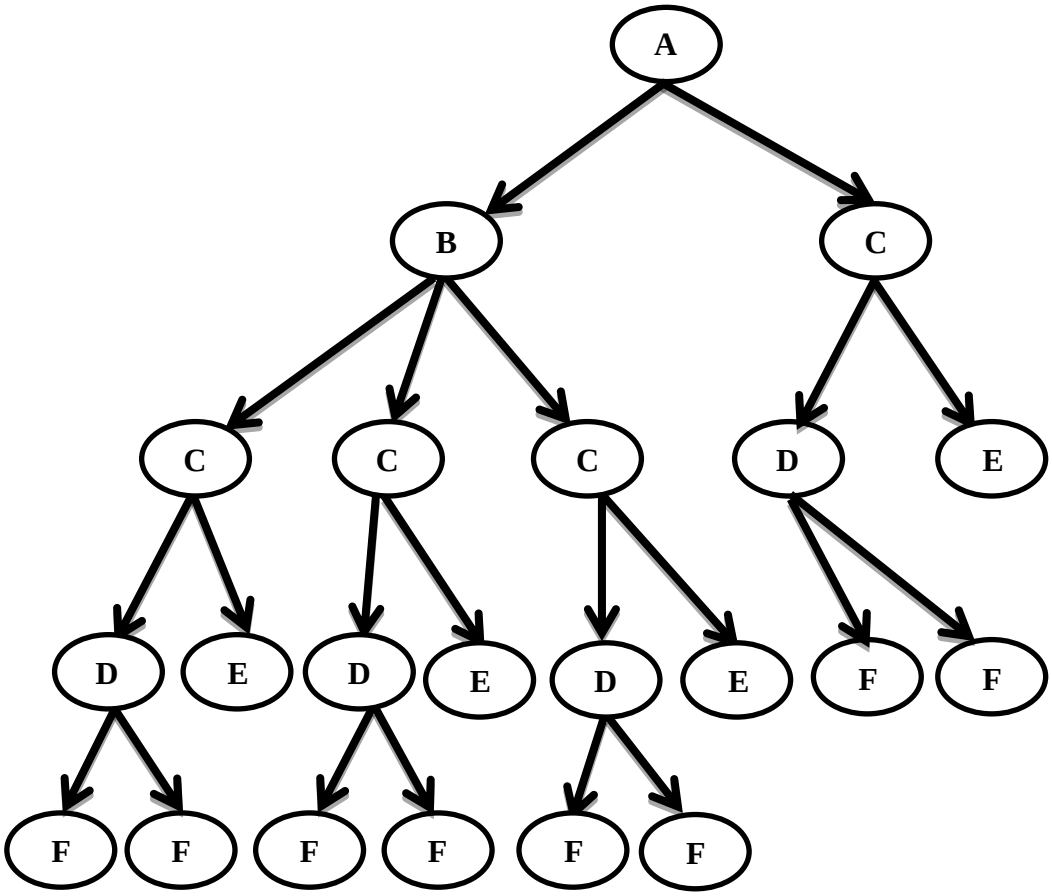
Algorithm1: Reducing Call Graph

One example is given which will be stored in computer memory and reduced step by step. This is the source code to find clones among two or more source code files. Cloning means detecting the same lines of code.

```
publicstaticlist<string> lines1 = newlist<string>();
publicstaticlist<string> lines2 = newlist<string>();

// this is equiavalent to function A()
void main()
{
string file1 = file.readalltext("d://file1.txt");
string file2 = file.readalltext("d://file2.txt");
string file3 = file.readalltext("d://file3.txt");
console.writeline("enter 1 for type 1 and 2 for multiple files\n");
string val = console.readline();
if (int.parse(val) == 1)
            type1fnscall (file1, file2);
else
multipletype1fnscall (file1, file2,file3);
 }

// this is equivalent to function B()
public static void multipletype1fnscall(string file1, string file2, string file3)
{
type1fnscall(file1, file2);
 type1fnscall(file2,file3);
 type1fnscall(file1,file3);
}

// this is equivalent to function C()
publicstaticvoid type1fnscall(string file1, string file2)
      {
                  compare(file1,file2);
                  display();
      }


// this is equivalent to function D()
publicstaticvoid compare(string file1, string file2)
      {
          file1=removespaces(file1);
  file=removespaces(file2);
string[] lines1 = file1.split('\n');
string[] lines2 = file2.split('\n');
for (int i = 0; i < lines1.length; i++)
```

```
            lines1.add(lines1[i]);
for (int j = 0; j < lines2.length; j++)
            lines2.add(lines2[j]);
while (lines1.remove("")) ;
while (lines2.remove("")) ;
for (int i = 0; i < lines1.count; i++)
        {
for (int j = 0; j < lines2.count; j++)
            {
if (lines1[i].equals(lines2[j]))
                adjmat[i, j] = 1;
else
                adjmat[i, j] = 0;
            }
        }
    }

// this is equivalent to function F()
publicstaticstring removespaces(string file)
    {
int index;
for (int i = 0; i < file.length; i++)
        {
            index = file.indexof(" ");
if (index < 0)
break;
            file = file.remove(index, 1);
            i = index - 1;

        }
return file;
    }

// this is equivalent to function E()
publicstaticvoid display()
    {
string mat = "";
for (int i = 0; i < lines1.count; i++)
        {
for (int j = 0; j < lines2.count; j++)
              mat = mat + adjmat[i, j].tostring() + "\t";
            mat = mat + "\n";
        }
console.write(mat);

}
```

Firstly the code will be analyzed and all the functions will be determined to make the call graph. In this code there are total 6 functions named main, multipletype1Fnscall, type1Fnscall, Compare, and RemoveSpaces, Display. All the functions will be labeled with different names to make it easy to understand. Nodes will be made in call graph for all the 6 functions. Obviously the first function is the main function. This can be called root of call graph which is labeled as A. the 2nd function multipletype1Fnscall is labeled as B. and the 3rd one type1Fnscall is labeled as C.Compare, and RemoveSpaces, Display are labeled as D,E and F respectively. The main function calls multipletype1Fnscall and type1Fnscall functions. so direct edges are connected from A to B and A to C. in next levelmultipletype1Fnscall calls type1Fnscall 3 times so 3 nodes labeled C are connected with B. In the same way edges are connected with nodes according to their calling and new call graph is generated as shown in figure 4.11.



4.11 Call Graph Generated from Source Code

Chapter 5

Implementation Details

The call graph will be saved in the computer memory by using matrix. There are 5 levels in the call graph and so matrix will have 5 rows. 1st row represent 1st level second row represent 2nd level and so on. Corresponding nodes with parent's information will be saved. 1st row stores the information about A node which is in 1st level and hasn't any parent as it is root node so store the information as -1.at 2nd level B and C stored in 2nd row with its parent information which is A and so on.

```
struct field

{

char label;

int parent;
} ;

struct field b[n][n];////where n is the number of nodes
```

$$\begin{bmatrix} A_{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B_0 & C_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_0 & C_0 & C_0 & D_1 & E_1 & 0 & 0 & 0 \\ D_0 & E_0 & D_1 & E_1 & D_2 & E_2 & F_3 & F_3 \\ F_0 & F_0 & F_2 & F_2 & F_4 & F_4 & 0 & 0 \end{bmatrix}$$

Matrix representation of call graph

According to algorithm the next step is to reduce the call graph. This approach is bottom up approach so 5th level is considered first and then move to 4th, 3rd and so on.5thlevel has 6 nodes labeled as F. According to the proposed algorithm the same level nodes same parent remerged resulting in a single node with same label. Information of call frequency is also saved. 2 node labeled F having same parent labled D are merged and indicated as F with frequency 2.

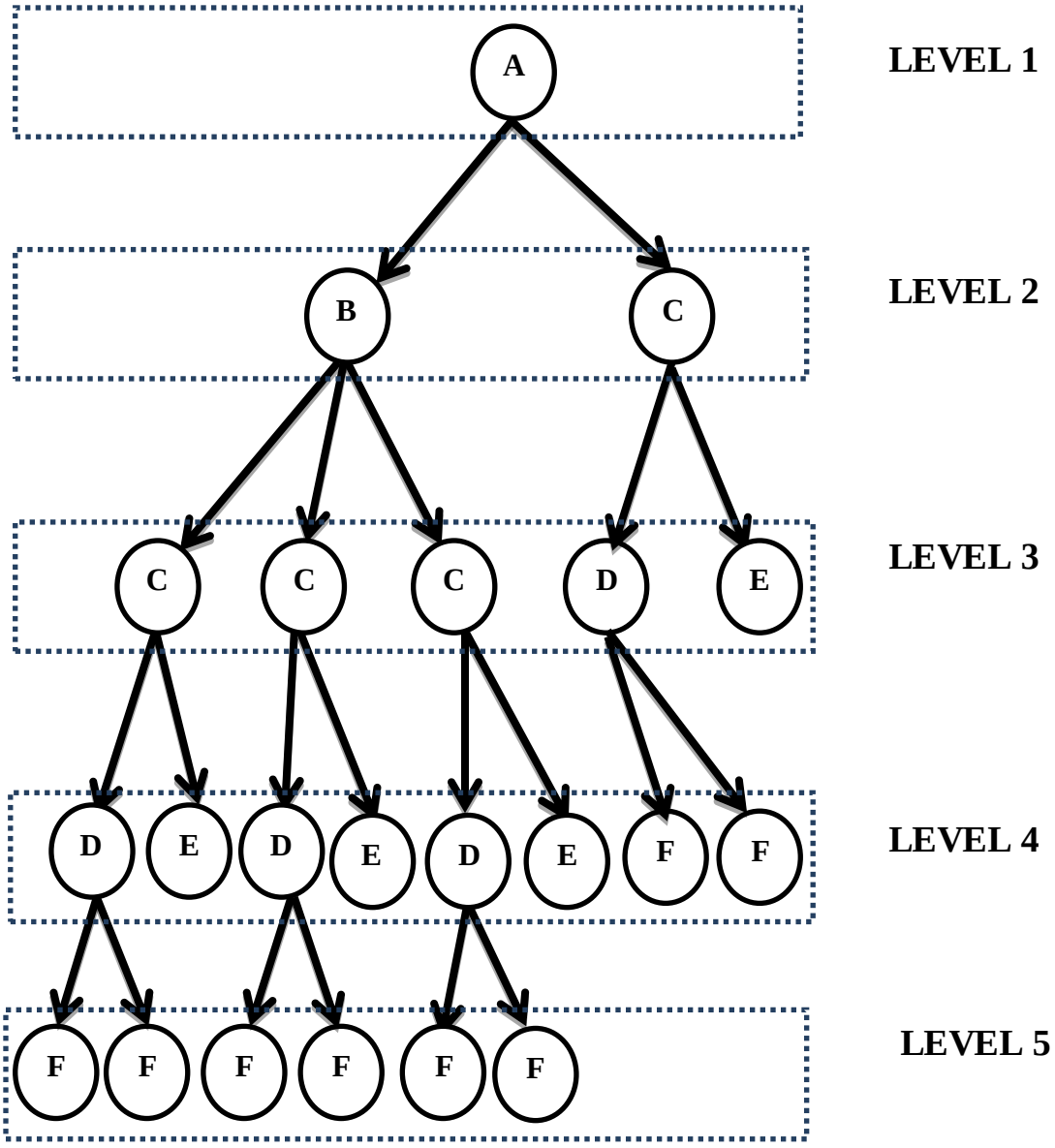


Figure 5.1: Source Call Graph

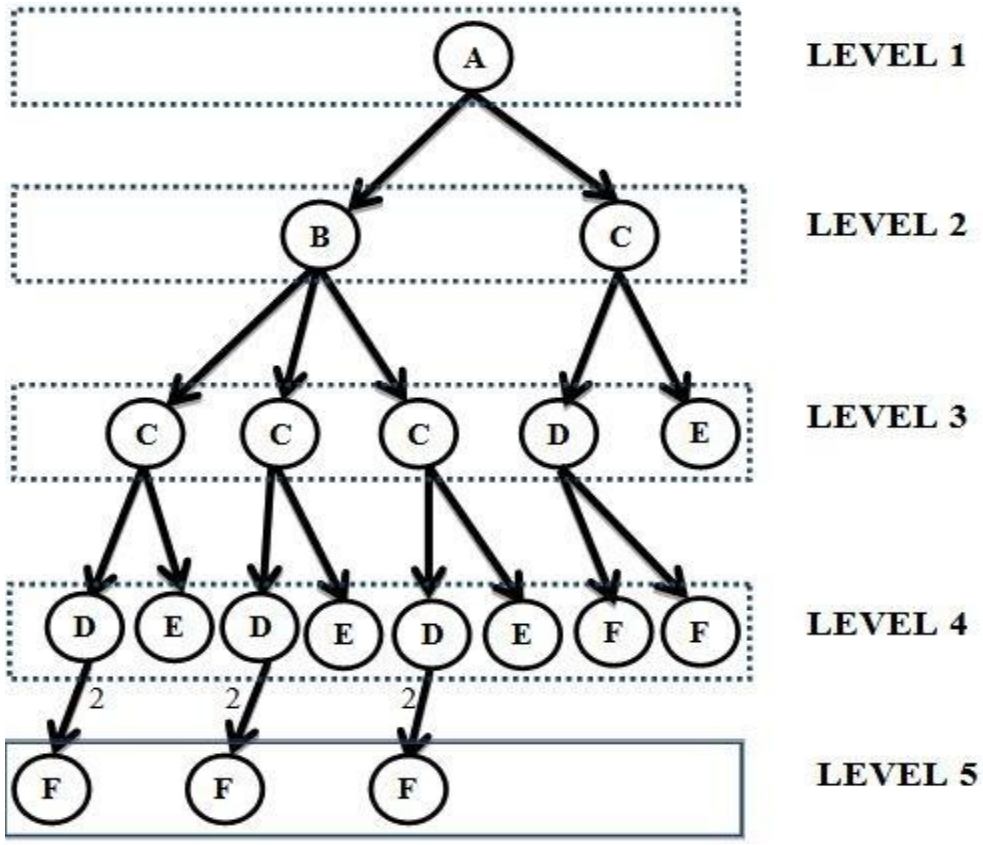


Figure: 5.2 Reduction at 5th level

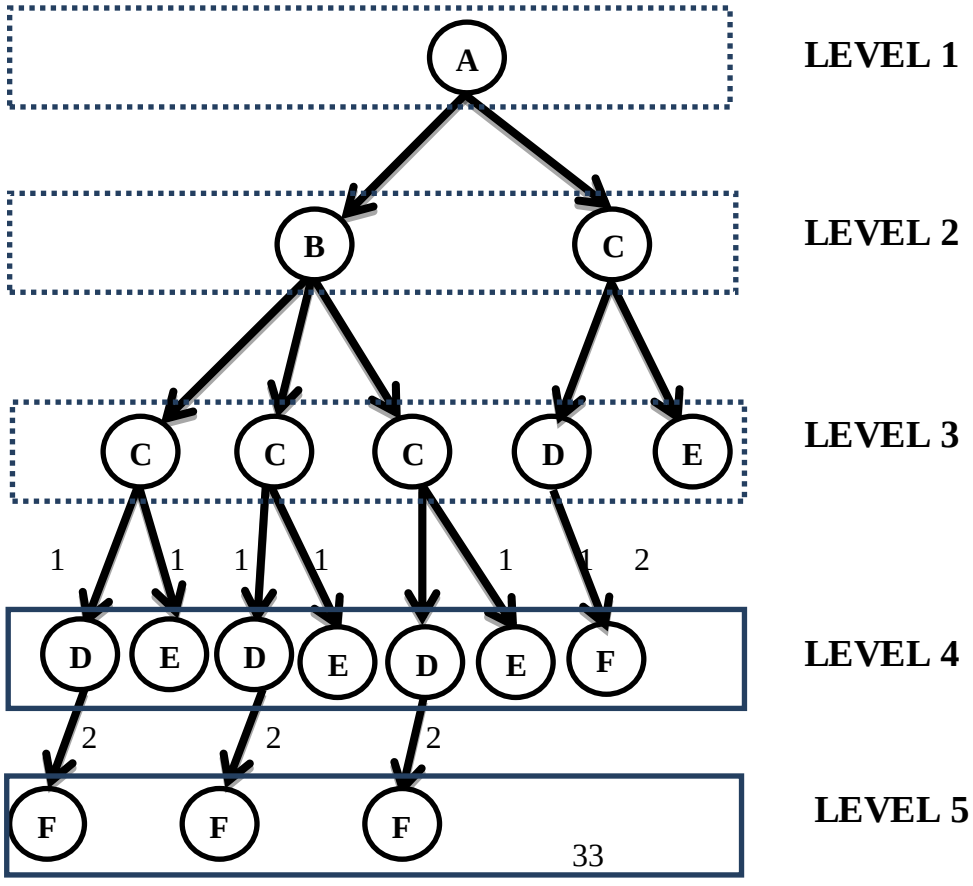


Figure 5.3:Reduction on 4th level

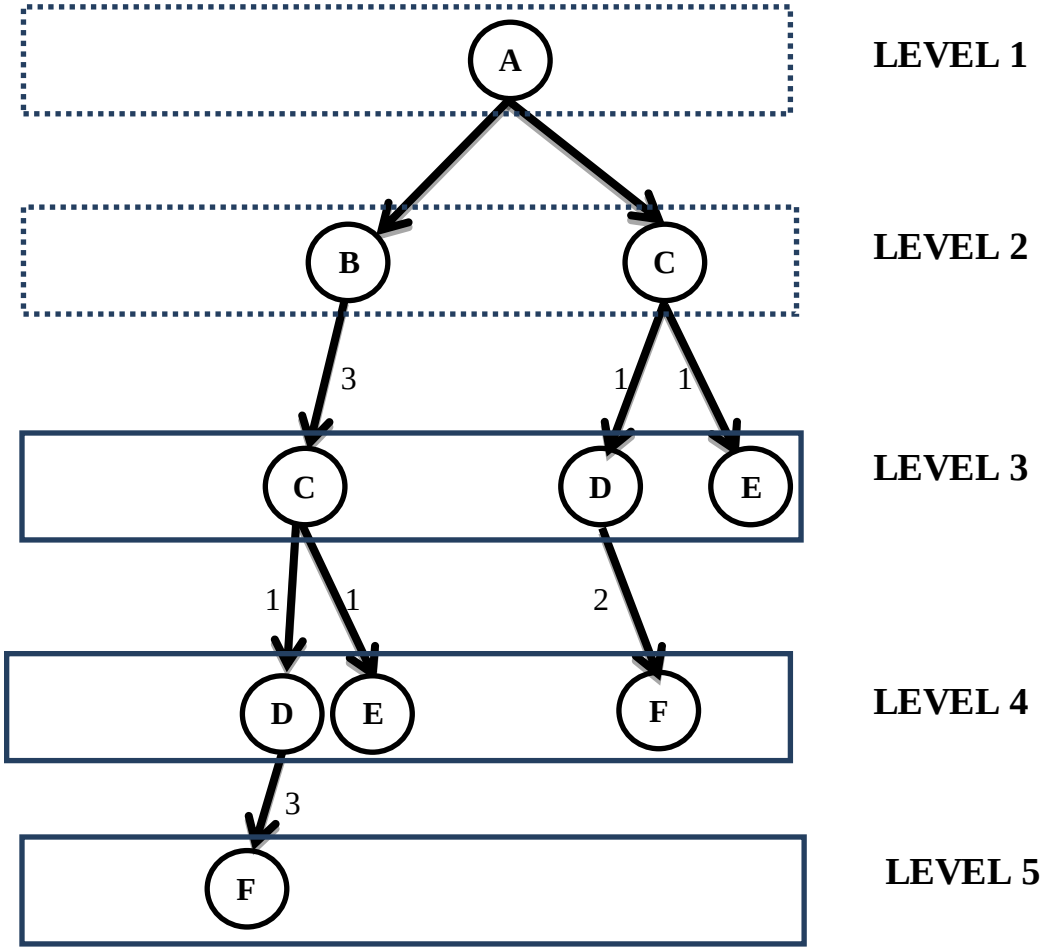


Figure 5.4: Reduction on 3rd level

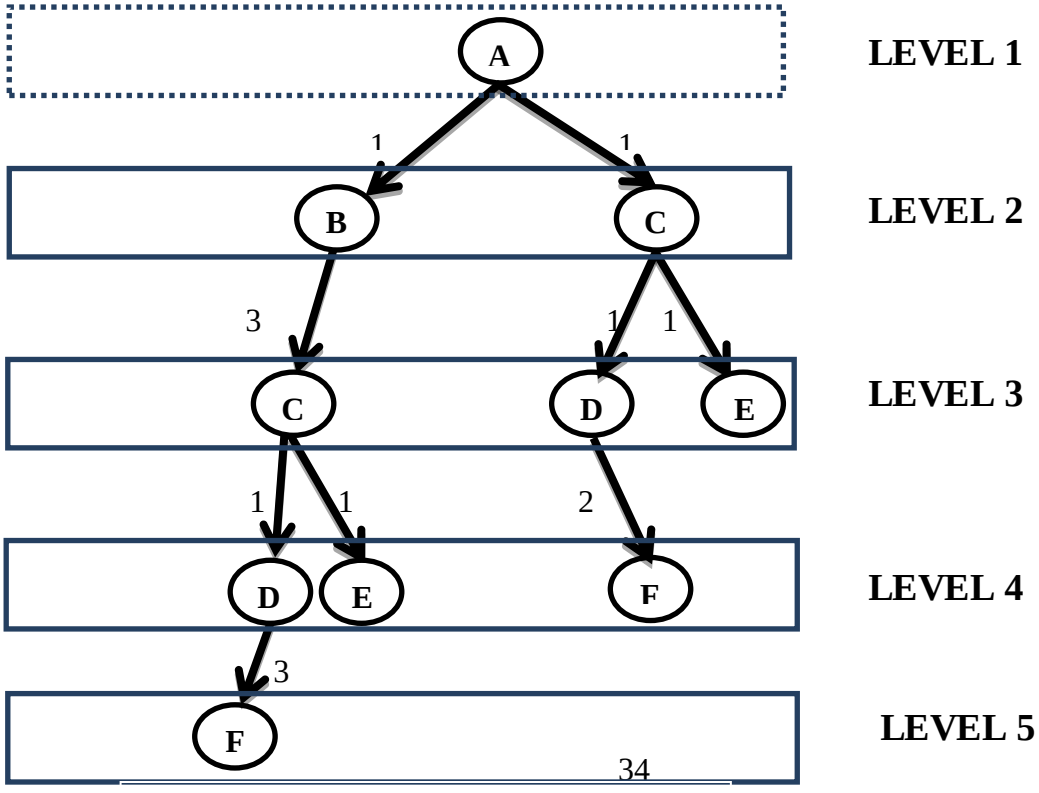


Figure 5.5: Reduction on 2nd level

Same process is applied in level 4 resulting in the figure 5.4.

The 3rd level will be worked atC appears 3 times D and E appear singly. So C is concentrated on. 3 Cs will be merged resulting in single C with frequency 3. This will also affect its children. So they will also be reduced according to the same procedure as shown in figure 5.5.

Finally reduced call graph is shown above is created. In this graph every node has the information about call frequency.

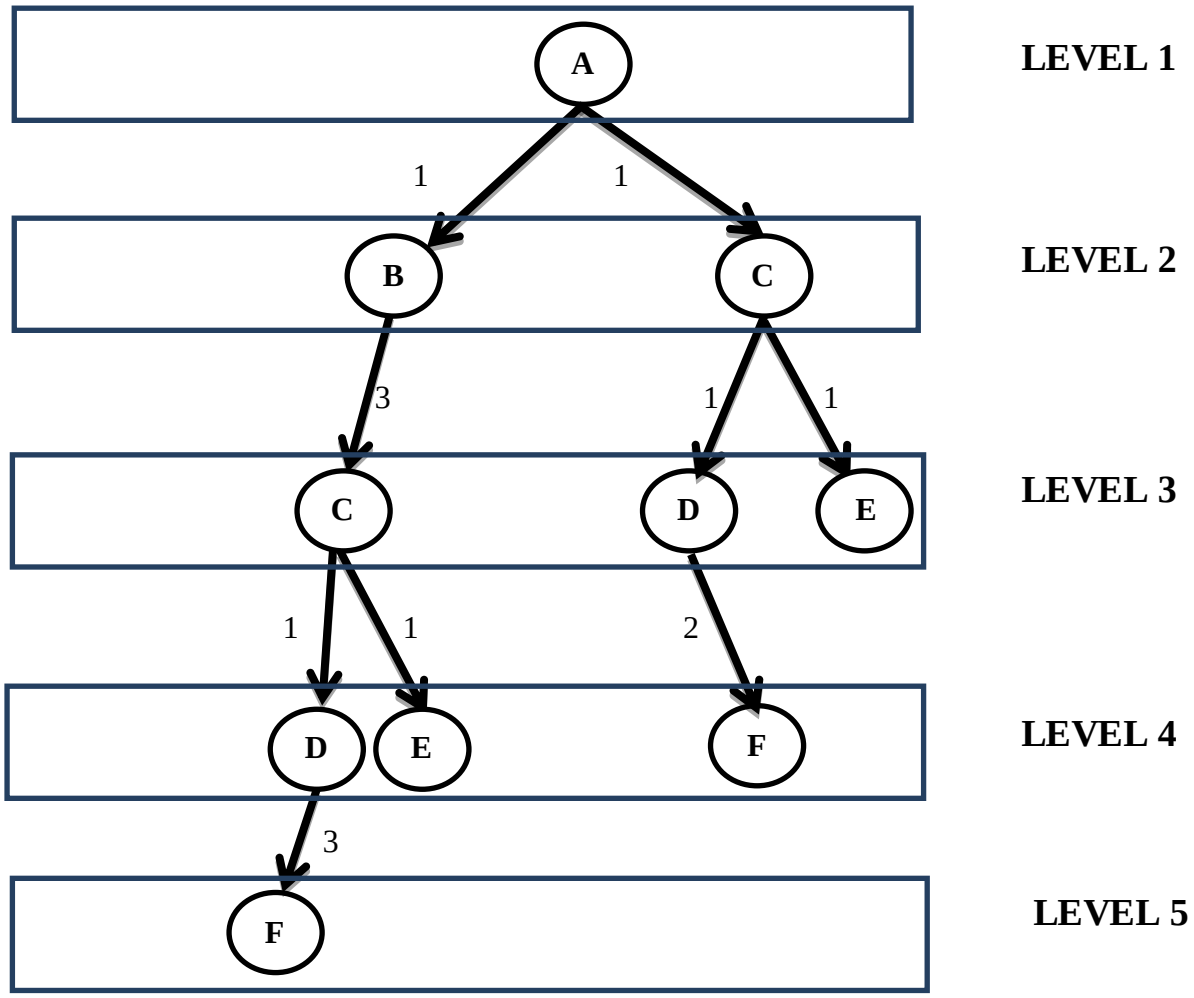


Figure 5.6: Completely Reduced Graph

Comparison

The graph obtained from the same source code is also reduced with the techniques given by DiFatta *et al* and Liu *et al* .As shown in the figure the graph generated from the Liu et al technique has reduced the graph with lesser edges and nodes but it could not able to retain the basic structure of the call graph where in the other hand as shown in figure 5.9Zero-one-many reduction could not reduce the same and lost the information of nodes.

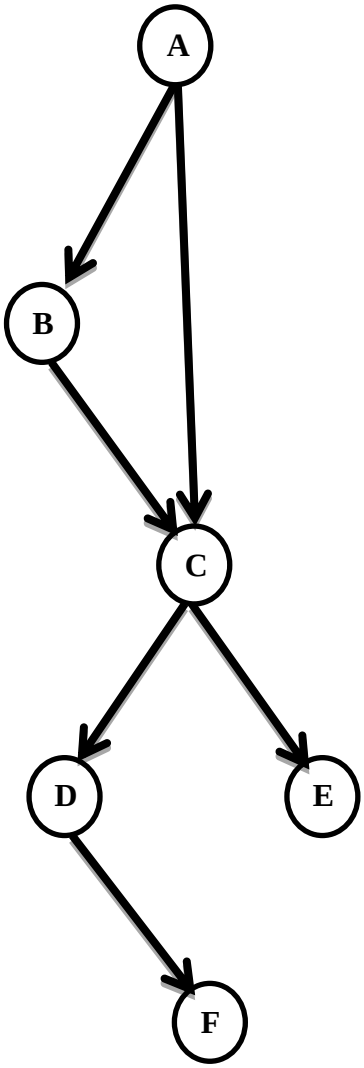


Figure 5.7 Reduced with total reduction technique (Liu *et al.*)

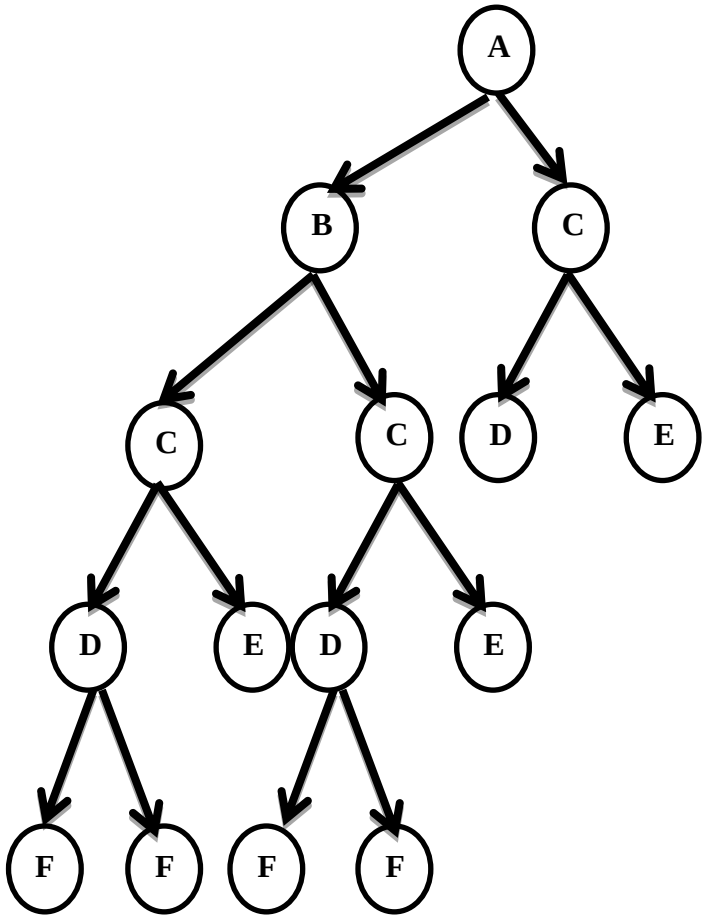


Figure 5.8 Reduced with Zero-one-many reduction (DiFatta*et al.*)

In contrast of both of them the call graph generated from the proposed algorithm is able to reduce the graph and retain the information of nodes as well. The comparison of results obtained from each technique of call graph reduction is shown in table no 5.1.

Table 5.1:Comparison among various call graph reduction techniques

Reduction	**No Of Nodes**	**No of edges**	**Effects on Structure**
Source code	*22*	*21*	
Total Reduction (Liu *et al.*)	*6*	*6*	*Lost information and Changed structure*
Zero-one-many reduction (DiFatta *et al.*)	*15*	*14*	*Lost information but remain same structure*
Reduction with proposed Algorithm	*10*	*9*	*No loss in information and Remain Same structure*

Both techniques Total Reduction and Zero-one-many reduction lost the information of the nodes and reduce the graph from 22 to 6 and 15 nodes along with edges from 21 to 6 and 14 respectively. The result obtained from proposed algorithm have positive results with reducing graph without losing information and basic structure i.e. from 22 nodes to 10 nodes and 21 edges to 9 edges.

Chapter 6

Conclusion & Future Scope

6.1 Conclusion

New static and dynamic approaches for bug localization have been developed, the diffusion into other disciplines has proceeded at a rapid pace, and knowledge of all aspects of the field has grown even more profound. At the same time, one of the most striking trends in graph theory is constantly increasing emphasis on the interdisciplinary nature of the field. Graph mining today is basic research tool in all areas of engineering, medicine, and the sciences. The bug localization techniques based on graph mining are successfully applied in a wide range of practical problems arising in software industry.

In this thesis a novel algorithm for call graph reduction has been proposed In order to use the respective call graphs for bug localization, the developed technique stores the parent information in the matrix and reduced at each level drastically. Information about each node is retained by using the call frequency by annotating each edge with a numerical weight. Similarly the algorithm used to reduced call graph has various advantages over traditional techniques. It takes various parameters for consideration such as information of nodes, basic structure of graphs and call frequency. Here the detailed study of call graph reduction in graph mining made the study of various other techniques in bug localization very easy

6.2 Future Scope

The proposed algorithm works only when there are same types of nodes at a particular level in a call graph.

In future this work can be extended to multiple levels of call graph will make the graph mining algorithm efficiently.

Secondly the storage of graph can be upgraded with any new storage technique where it would require lesser storage space as well as lesser access time leading to further optimize reduction of call graph.

References

[1] W. Sadiq,M.E. Orlowska,"Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models Distributed Systems Technology", Proceeding of the 11th International Conference on Advanced Information Systems Engineering, 1999,pp.195–209.

[2] I.R.Katz, J.R.Anderson,"Debugging: an analysis of bug-location strategies", Human Computer Interaction,1987, pp.351–399.

[3]R.Hastings,B.Joyce, "Purify: Fast detection of memory leaks and access errors",Proceedings of the Winter USENIX Conference,1992,pp.125.

[4] O.Lhotak,"Comparing Call Graphs",Proceedings of the 7th ACM Sigplan-sigsoft Workshop on Program Analysis for Software Tools and Engineering, PASTE 07, San Diego, California, USA, 2007, pp. 37-42.

[5] J.L.Gersting,"Mathematical Structures for Computer Science", Freeman, 2007.

[6] W.Ren,R.Beard,E.Atkins, "Information consensus in multivehicle cooperative control" Control Systems, IEEE, Vol.27, No. 2,2007, pp.71 -82.

[7]X.Yan, J. Han,"CloseGraph: Mining Closed Frequent Graph Patterns",Proceeding of Int. Conf. Knowledge Discovery and Data Mining,2003.

[8] G.Shi,Weiwei"A Graph Reduction Approach to Symbolic Circuit Analysis"Proceeding of Design Automation Conference, 2007.

[9] A.Zeller,"Why Programs Fail: A Guide to Systematic Debugging",2nd Edition, Morgan Kaufmann, 2009.

[10]S.K.Lukins,N.A.Kraft,L.H.Etzkorn,"Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation,"Proceedings of the 15th Working Conference on Reverse Engineering, 2008, pp.155-164.

[11] R.Tzoref, S.Ur,E.Yom,Tov,"Instrumenting Where it Hurts – An Automatic Concurrent Debugging Technique", Proceedings of the 16th International Symposium on Software Testing and Analysis, 2007.

[12] W.Eric,Wong,V.Debroy, "A Survey of Software Fault Localization",2009.

[13] D.Binkley,"Source Code Analysis: A Road Map"Proceedings of the 29th International Conference on Software Engineering,2007.

[14] Boris Beizer, "Software Testing Techniques", Van Nostrand Reinhold Co., 2nd Ed., 1990.

[15] D.Hovemeyer,W.Pugh,"Finding Bugs is Easy",SIGPLAN 2004, pp.92-106.

[16] L.Dietz,V.Dallmeier,A.Zeller,T.Scheffer"Localizing Bugs in Program Executions with Graphical Models", Advances in Neural Information Processing Systems 22,Proceedings of the Conference, Vancouver, Canada, 2009, pp. 468-477.

[17]J.Dibling, "testing and debugging"Dr. Dobb's JournalVolume 30, Issue 6, No. 373, 2005.

[18]Edison Papers, Edison National Laboratory, U.S. National Park Service, West Orange, NJ, Cited in Thomas P. Hughes, American Genesis: A History of the American Genius for Invention, Penguin Books, 1989, ISBN 0-14-009741-4 , on pp. 75,13 November 1878

[19] M. Kaufmann, J. S. Moore,"Proof Search Debugging Tools in ACL2". Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics, 2008, pp.17-21.

[20] S.Rao,A.Kak,"Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models" Proceedings of the 8th Working Conference on Mining Software Repositories,2011,pp. 43-52.

[21] J.C.M.Baeten,K.M.Van,Hee,"Role of graph in computer science", 2007.

[22] D.Kavitha,M.Rao,V.KishoreBabu "A Survey on Assorted Approaches to Graph Data Mining", International Journal of Computer Applications IJCA Journal, 2011.

[23] N.Rutar,C.B.Almazan,J.S. Foster, "A Comparison of Bug Finding Tools for Java", Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004,pp. 245 - 256.

[24] G.Di Fatta,S.Leue,E.Stegantova, "Discriminative Pattern Mining in Software Fault Detection". Proceedings of the 3rd International Workshop on Software Quality Assurance,2006, pp. 62.

[25] N.L.Biggs,R.J. Lloyd,R.J.Wilson,"Graph Theory 1736 – 1936", 1976.

[26] T.Zaslavsky, "Matrices in the Theory of Signed Simple Graphs" 2010.

[27] OMG. Unified Modeling Language (UML), version 1.5.OMG Standard, 2003.

[28]C. Ghezzi, M. Jazayeri, and D. Mandrioli."Fundamentals of Software Engineering".Prentice Hall Int 1991.

[29] J. Ferrante, K. J. Ottenstein,J. D. Warren,"The Program Dependence Graph and its Use in Optimization",ACM Trans. on Programming Languages and Systems Volume 9,1987.

[30] N.E.Fenton,R.W.Whitty,"Axiomatic approach to software metrication through program decomposition", Computer Journal, vol. 29, no. 4, 1986,pp.329-339.

[31] E. A. Frances."Control Flow Analysis".ACM SIGPLAN1970.

[32] N.E.Fenton, S.L.Pfleeger, "A Rigorous & Practical Approach",1996

[33] B. Ryder, "Constructing the call graph of a program", Software Engineering, IEEE Transactions on,vol. SE-5, no. 3, 1979, pp.216 – 226.

[34] X. Hu, T. Chiueh, K. G.Shin,"Large-scale malware indexing using function-call graphs", ACM Conference on Computer and Communications Security,E.AlShaer,S. Jha, A. D.Keromytis,Eds,2009, pp. 611–620.

[35] C.Liu, X Yan, H Yu,J Han,., P.S. Yu, "Mining Behavior Graphs for \Backtrace" of Noncrashing Bugs". In: Proc. of the 5th Int.Conf. on Data Mining, 2005

[36] Di Fatta, G. Leue, S.Stegantova,"Discriminative Pattern Mining in Software Fault Detection". In: Proc. of the 3rd Int. Workshop on Software Quality Assurance,2006.

List of Publications

Accepted

[1] Prabhdeep Singh, Shalini Batra. “A novel technique for call graph reduction for bug localization”;International Journal of Computer Applications, Vol. 47, 2012 (Impact Factor: 0.7).