An Improved Technique for Storage and Reduction of Call Graph in Computer Memory

Prabhdeep Singh
Punjab Technical University
42 PUBLICATIONS **542** CITATIONS
SEE PROFILE

Kiran Deep Singh
Khalsa College of Engineering & Technology, Amritsar
42 PUBLICATIONS **637** CITATIONS
SEE PROFILE

An Improved Technique for Storage and Reduction of Call Graph in Computer Memory

Prabhdeep Singh[1], Vivek Gupta[2] and Kiran Deep Singh[3]
[1]*Student (ME), Department of CSE, Thapar University Patiala, India*
[2]*Lecturer, Department of CSE, SGI, Ganganagar, India*
[3]*Assistant Professor, Department of CSE, KCET, Amritsar, India*
e-mail: [1]ssingh.prabhdeep@gmail.com, [2]vivek@gmail.com, [3]kdkirandeep@gmail.com

***Abstract*—In this paper a new algorithm is proposed 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. The proposed algorithm is also 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.**

I. INTRODUCTION

In many applications of graph mining, 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.

A. *Call Graph Definition*

A call graph is a directed graph whose nodes represent the functions of program and directed edges symbolize function calls [1]. Nodes can represent either one of the following two types of functions:

1. Local functions, implemented by the program designer.
2. External functions: system and library calls.

Call graphs are formally defined as follows:

1) *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 [2].

B. *Types of Call Graph*

1) *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.

2) *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.

II. CALL GRAPH REDUCTION

Call graph are representations of program executions. Raw call graphs typically become much too large [5, 6]. 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 [7, 8]. 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.

A. *Approaches of Call Graph Reduction*

There are two approaches of reducing software call graphs

1. Total Reduction (Liu *et al.*)
2. Zero-one-many reduction (DiFatta *et al.*)

Total reduction is proposed by Liu *et al.* [3]. 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.

The other approach given by DiFatta *et al.* covers the drawback of Liu *et al.* approach as it doesnot change the structure but the reduction is not properly done [4]. 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.

III. Problem Statement

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 of this paper 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.

IV. Proposed Approach

Researchers proposed many approaches for reduction of call graph but they could not proposed any approach to save the call graph in computer memory [9, 10,11]. 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. Basic structure for save the node with its parent information is shown in figure1.

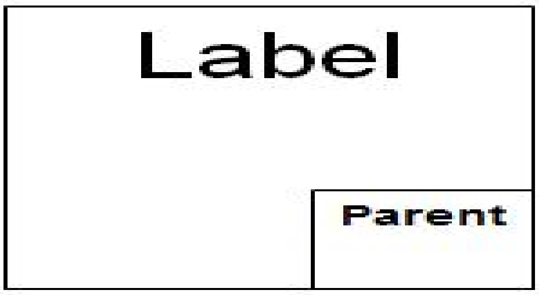

Fig. 1: Structure for Node Storage

```
structfeild
{
char label;
int parent;
} ;
structfeild b[n][n]; ////where n is the number of nodes
```

Call graph has n number of nodes so to store all the nodes in matrix it is shown like figure 2.

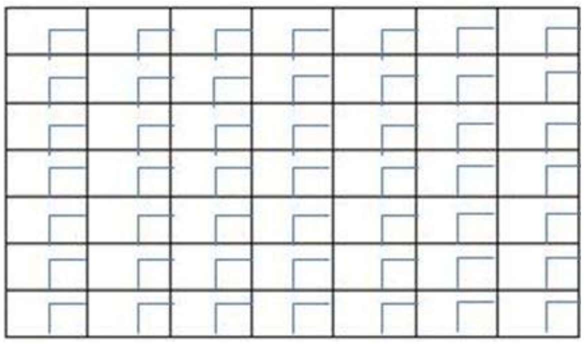

Fig. 2: Structure for Call Graph Storage

Once the call graph is saved in memory it is reduced using proposed algorithm. 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 labelled so that it can easily be interpreted. Then a call graph is made using these labelled functions. To understand the algorithm call graphs is constructed from any code and apply the algorithm step by step. Figure 3 shows the call graph which is generated by a code.

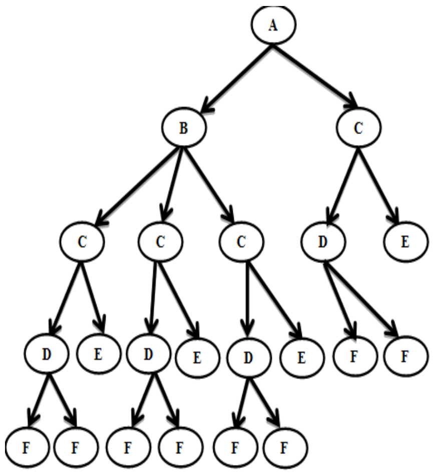

Fig. 3: Source Call Graph

```
Algorithm: Reducing Call Graph
Input: children, label, parent
Output: Reduced Call Graph
1.  Set j=Getstr[100][]
2.  foreach aa←1 to 10 do
3.  Set j[aa-1] = Getstr[200]
4.  end for
5.  Set count= GetArray(level)
6.  foreach i←0 to levels-1 do
7.  print "Enter no. of children at level 0"
8.  Input(children)
9.  count[i]=children
10. foreach x←0 to children -1 do
11. print "Enter the label of (x) children"
12. j[i][x].label = Input(label)
13. print "Enter the parent of (x) children"
14. j[i][x].parent = Input(parent)
15. end for
16. end for
17. foreach k←levels down to 0 do
18. foreach l←count[levels-1] down to 0 do
19. foreach m←l-1 down to 0 do
20. if j[k-1] [m-1]. label=j [l-1] [m-1]. label
    AND j[l-1] [m-1].parent=j[k-1] [m-1].
    parent AND j [k-1] [m-1].label ≠ '\0' then
21. j[k][l].parent = -1
22. end if
23. end for
24. end for
25. end for
```

V. IMPLEMENTATION AND RESULTS

First part of algorithm is used for call graph storage. 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.

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

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 with same parent are merged resulting in a single node with same label. In this graph same level is level 5, same label is F and same parent is D. so after applying the algorithm 3 F's are merged to single F with call frequency.

The result is as shown in figure 4.

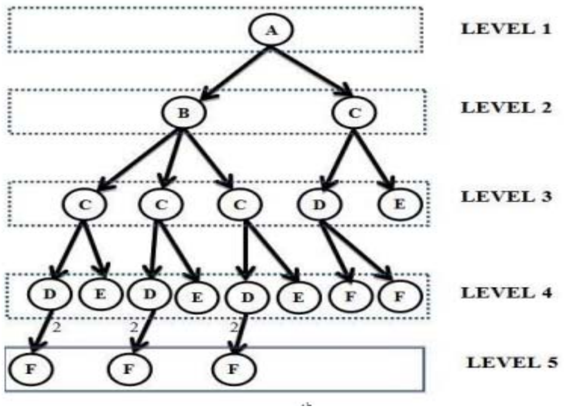

Fig. 4: Reduction at 5th Level

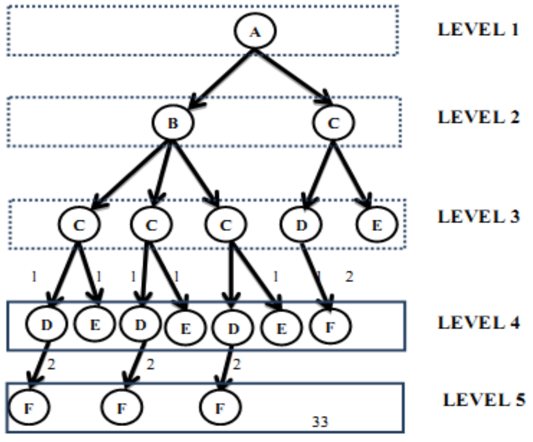

Fig. 5: Reduction at 4th Level

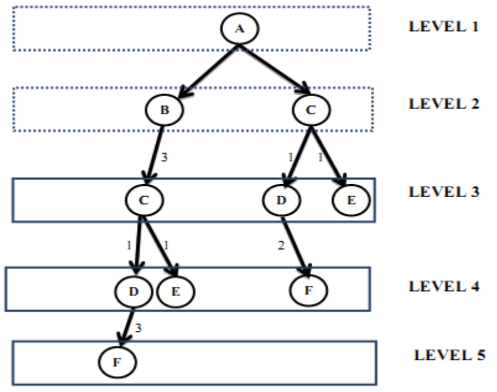

Fig. 6: Reduction at 3rd Level

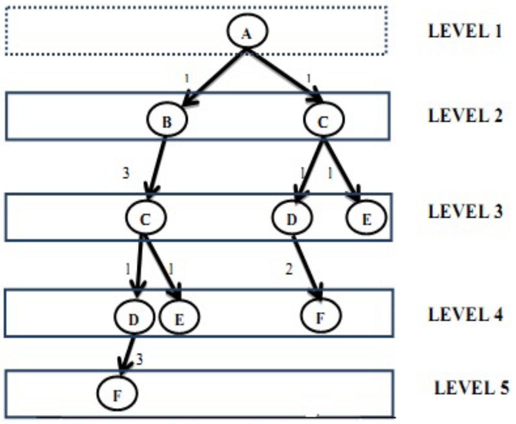

Fig. 7: Reduction at 2nd Level

Same process is applied in level 4 resulting in the figure 5.The 3rd level will be worked at C 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.Finally reduced call graph is shown above is created. In this graph every node has the information about call frequency.

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 Zero-one-many reduction could not reduce the same and lost the information of nodes.

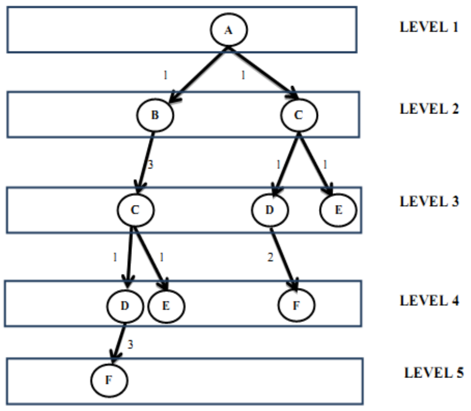

Fig. 8: Completely Reduced Graph

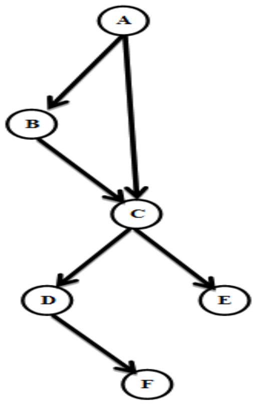

Fig. 9: Reduced with Total Reduction Technique (Liu *et al.*)

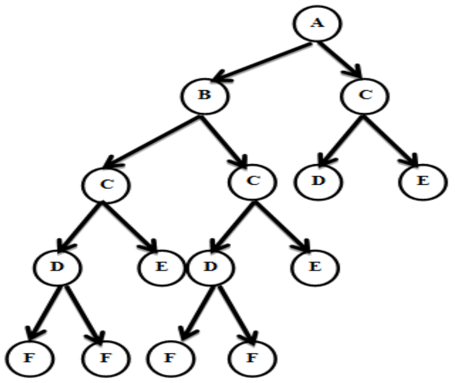

Fig. 10: Reduced with Zero-one-many Reduction (DiFatta *et al.*)

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.

TABLE 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.

VI. CONCLUSION & FUTURE SCOPE

In this paper a novel algorithm for call graph reduction has been proposed. 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.

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

[2] 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.

[3] 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

[4] 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.

[5] 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.

[6] 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.

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

[10] N. L. Biggs, R. J. Lloyd, R. J. Wilson, “Graph Theory 1736–1936”, 1976.

[11] T. Zaslavsky, “Matrices in the Theory of Signed Simple Graphs” 2010.