Global Journal of Computer Science and Technology: B
Cloud and Distributed
Volume 22 Issue 1 Version 1.0 Year 2022
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 | Print ISSN: 0975-4350 |

A Call Graph Reduction based Novel Storage Allocation Scheme for Smart City Applications

By Prabhdeep Singh, Rajbir Kaur & Diljot Singh
ECE Punjabi University

Abstract- Today's world is going to be smart even smarter day by day. Smart cities play an important role to make the world smart. Thousands of smart city applications are developing in every day. Every second very huge amount of data is generated. The data need to be managed and stored properly so that information can be extracted using various emerging technologies. The main aim of this paper is to propose a storage scheme for data generated by smart city applications. A matrix is used which store the information of each adjacency node of each level as well as the weight and frequency of call graph. 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 the proposed algorithm. 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 store the smart city application data efficiently.

GJCST- B Classification: *D.4.2*

Strictly as per the compliance and regulations of:

A Call Graph Reduction based Novel Storage Allocation Scheme for Smart City Applications

Prabhdeep Singh [α], Rajbir Kaur [σ] & Diljot Singh [ρ]

Abstract- Today's world is going to be smart even smarter day by day. Smart cities play an important role to make the world smart. Thousands of smart city applications are developing in every day. Every second very huge amount of data is generated. The data need to be managed and stored properly so that information can be extracted using various emerging technologies. The main aim of this paper is to propose a storage scheme for data generated by smart city applications. A matrix is used which store the information of each adjacency node of each level as well as the weight and frequency of call graph. 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 the proposed algorithm. 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 store the smart city application data efficiently.

I. Introduction

Internet of Things (IoT) is a new technology, which is rapidly gaining momentum in the smart city applications. This concept enables the ubiquitous presence around us of a variety of objects or things such as RFID tags, sensors, actuators, and cell phones. The rise of IoT has affected many areas of smart city applications, such as e-learning, e-health, transportation, waste management, etc. By 2020, more than 5 billion devices will be connected worldwide will become the pioneer to provide information accessible across the globe in milliseconds. Almost all smart city applications are running using the internet and they utilized the services to the cloud[1]. The IoT technology is upgrading day by day with the rapid implementation of a smart city[7]. Every smart city application requires the processing speed to process data and storage space to store the processed data for further use. The main challenge of the smart city [8,9] applications are to store the data and analysis it. Data is stored in a memory in the form of the graph. 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, sometimes information of the program is lost or changed, loosing or changing of information affects the accuracy of program that is unacceptable.

Author α: Department of CSE Punjabi University, Patiala, Punjab, India.
e-mail: ssingh.prabhdeep@gmail.com
Author σ: Department of ECE Punjabi University, Patiala, Punjab, India.
e-mail: rajbir277@yahoo.co.in
Author ρ: Department of CSE KCET, Amritsar.
e-mail: diljot.asr@gmail.com

II. Call Graph

A call graph is a directed graph whose nodes represent the functions of the program and directed edges symbolize function calls [2,14]. 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:

a) Definition

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) ⊆ V(G)×V(G), in correspondence with the function calls.

b) Types of Call Graph

The call graph can be classified as static and dynamic call graph.

i. 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[15,19].

ii. Dynamic Call Graph

A dynamic call graph is the invocation relation that represents a specific set of run-time executions of a program. Dynamic call graph extraction is a typical application of dynamic analysis to aid compiler optimization, program understanding, performance analysis etc[3,4].

c) Call Graph Reduction

The call graph is representations of program executions [11]. 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[5]. This involves the trade-off between keeping as much information as possible and a strong compression. When call graph is being reduced it is essential that no function call is missed[16]. The specifications of all the functions/methods must be clear. It is also noticeable that no information is lost when the label is changed[6]. Call frequency must be clearly specified. Two approaches are center of the focus to reduce the call graph

1. Total Reduction
2. Zero-one-many reduction

In total reduction, a node represents every function. A direct edge is connected with the corresponding nodes when one function has called another function. Total reduction technique shortens the size of the source call graph. 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 the required information from this reduced graph[17,18].

The other approach is Zero-one-many reduction which covers the drawback of Total Reduction as it does not change the structure but the reduction is not properly done. The improper reduction increases its complexity and it is difficult to find frequent substructure from the graph. The reduced graph can provide near information about call frequency but exact information is not known.

III. Problem Statement

Smart city applications generated the very high amount of data every millisecond, which required a very high amount of storage space[10]. Various existing techniques reduce the size of data but the originality and quality of data may also suffer. Surely, the size of data is reduced but the complete information of that data is also changed[13]. 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[12]. So, some new techniques or algorithms are needed to store the information of the graph in computer memory and reduce the graph in such a manner that its information is not lost and the 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, the reduction of the graph is required. The graph must be reduced in such a manner that its information is not lost and it should have minimum edges and nodes.
- Structure of the graph should not be changed after reduction.

IV. Proposed Approach

Researchers proposed many approaches for reduction of call graph but they could not propose any approach to saving the call graph in computer memory. 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.

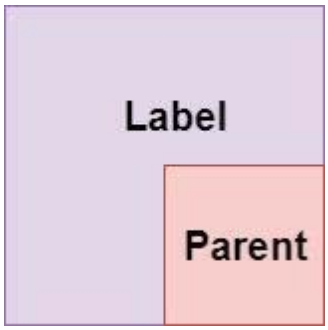

Figure 1: Structure for Node Storage

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 basic structure for saving the node with its parent information is shown in fig.

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

Call graph has n number of nodes to store all the nodes in the matrix. It is shown as figure 2.

$$\begin{bmatrix} A_{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B_0 & c_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_0 & c_0 & c_0 & D_1 & E_1 & 0 & 0 & 0 \\ D_0 & E_0 & D_1 & E_1 & D_2 & E_2 & F_3 & F_3 \\ F_0 & F_0 & F_2 & F_2 & F_4 & F_4 & 0 & 0 \end{bmatrix}$$

Once the call graph is saved in memory, it is reduced using the proposed algorithm. In this approach, the reduced call graph shows the call frequency of each node without changing the structure of the source call graph. First, all functions of source code are labeled so that it can easily be interpreted. Then a call graph is made using these labeled 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

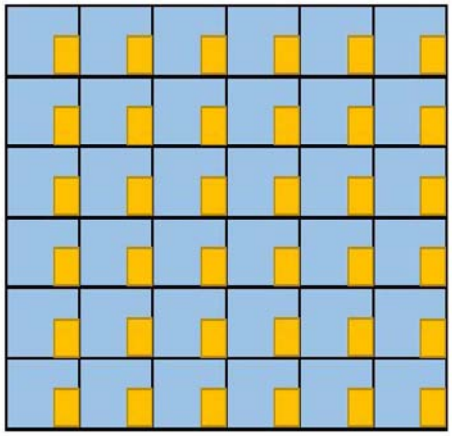

Figure 2: Structure for Call Graph Storage

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

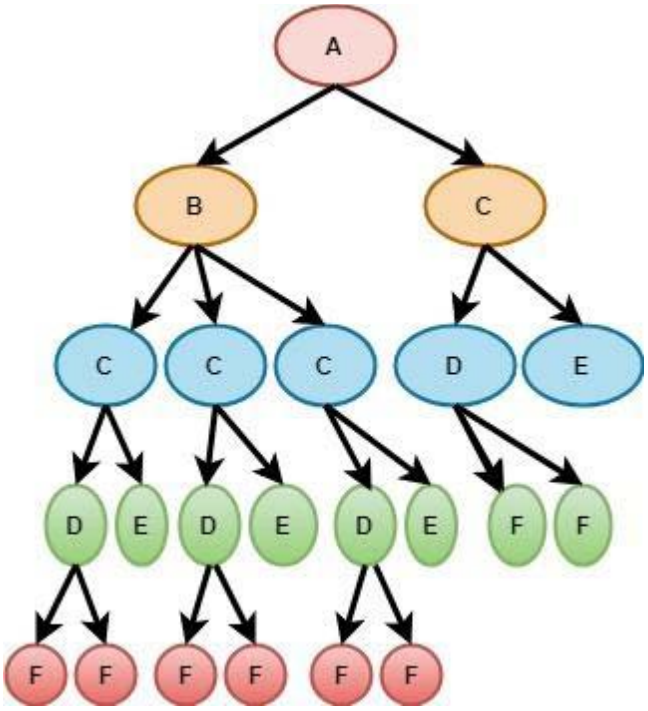

Figure 3: Source Call Graph

V. Implementation and Results

The first part of the 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 the proposed algorithm the same level nodes with the same parent are merged resulting in a single node with the same label.

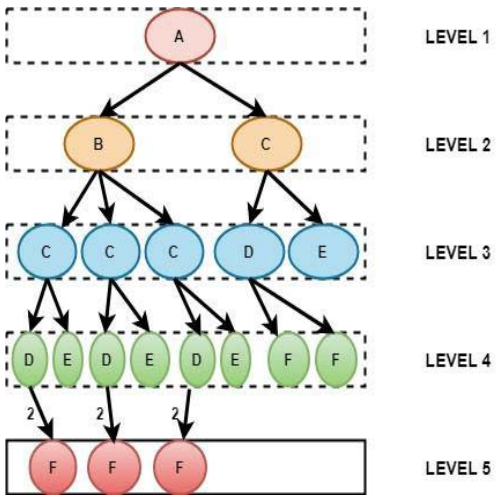

Figure 4: Reduction at 5th Level

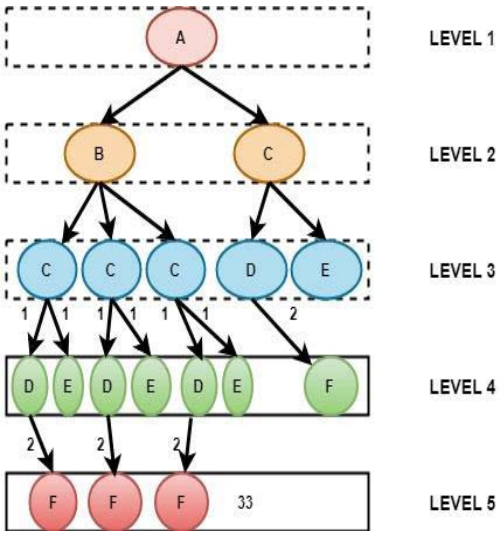

Figure 5: Reduction at 4th Level

In the graph same level is level 5, the same label is F and the 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 4.1st level the 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.

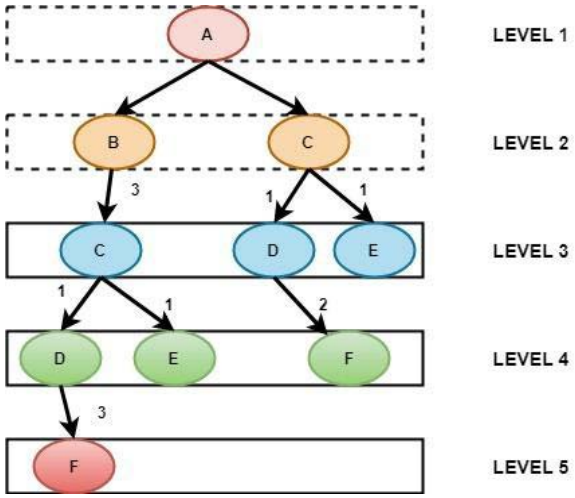

Figure: 6: Reduction at 3rd Level

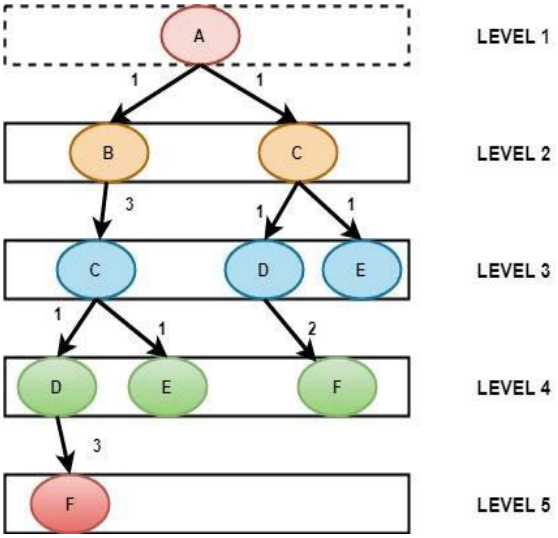

Figure 7: Reduction at 2nd Level

According to the algorithm, the next step is to reduce the call graph. This approach is bottom-up approach so the 5th level is considered first and then move to 4th, 3rd, and 4th Fig. 3: Source Call Graph so on.5 level has 6 nodes labeled as f. according to the

The same process is applied in level 4 resulting in figure 5.

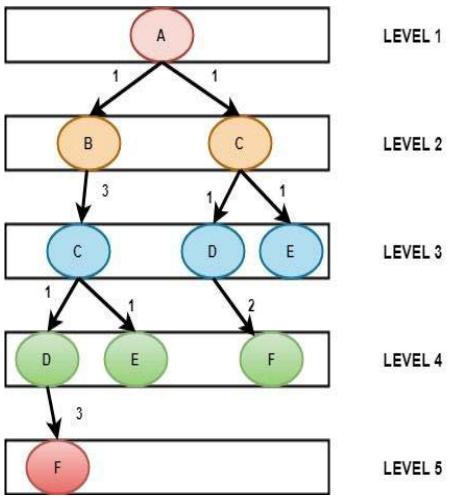

Figure 8: Completely Reduced Graph

The 3rd level will be worked at C appears 3 times D and E appear singly. Hence, C is concentrated on. 3 Cs will be merged resulting in single C with frequency 3. This will also affect its children. Therefore, 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.

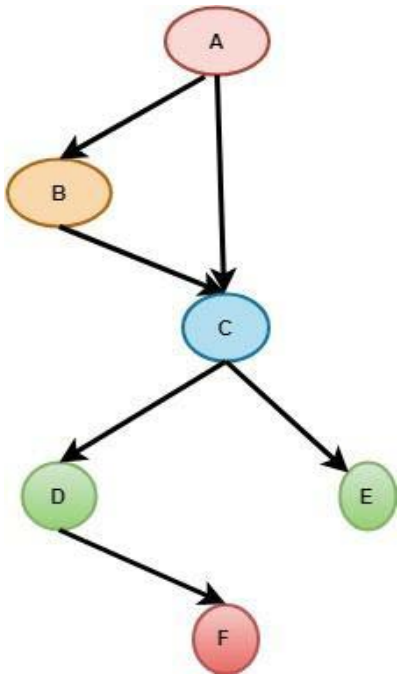

Figure 9: Reduced with Total Reduction Technique

The graph obtained from the same source code is also reduced with both techniques.

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 wherein the other hand as shown in figure Zero-one-many reduction could not reduce the same and lost the information of nodes. In contrast to 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.1

Table 1: Comparison Among Various Call Graph Reduction Techniques

Reduction algorithm	No Of Nodes	No of Edges	Effects on Structure
Source code	22	21	
Total Reduction	6	6	Lost information and Changed structure
Zero-one-many reduction	15	14	Lost information but remain same structure
Proposed Algorithm	10	9	No loss in information and Remain Same structure

Both techniques Total Reduction and Zero-one-many reductions 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 the 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

Every year the government to their tradition city to update it to the smart city spends a very huge amount. Smart city applications generate the high amount of data at every second. In this paper, the data storage scheme is proposed. The main benefit of this algorithm is to develop a technique to 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 Références Referencias

1. Zhou, L., Wu, D., Chen, J., & Dong, Z. (2018). Greening the smart cities: Energy-efficient massive content delivery via D2D communications. *IEEE Transactions on Industrial Informatics*, *14*(4), 1626-1634.
2. Lin, C. C., Deng, D. J., Kuo, C. C., & Liang, Y. L. (2018). Optimal charging control of energy storage and electric vehicle of an individual in the internet of energy with energy trading. *IEEE Transactions on Industrial Informatics*, *14*(6), 2570-2578.
3. Darivianakis, G., Eichler, A., Smith, R. S., & Lygeros, J. (2017). A Data-Driven Stochastic Optimization Approach to the Seasonal Storage Energy Management. *IEEE control systems letters*, *1*(2), 394-399.
4. Lee, C. C., Chen, C. T., Wu, P. H., & Chen, T. Y. (2013). Three-factor control protocol based on elliptic curve cryptosystem for universal serial bus mass storage devices. *IET Computers & Digital Techniques*, *7*(1), 48-56.
5. Tang, Z., Liu, A., & Huang, C. (2016). Social-Aware Data Collection Scheme Through Opportunistic Communication in Vehicular Mobile Networks. *IEEE Access*, *4*, 6480-6502.
6. Colmenar-Santos, A., Luis-Molina, E., Rosales-Asensio, E., & Lopez-Rey, Á. (2018). Technical approach for the inclusion of superconducting magnetic energy storage in a smart city. *Energy*.
7. Cebe, M., & Akkaya, K. (2018). Efficient Certificate Revocation Management Schemes for IoT-based Advanced Metering Infrastructures in Smart Cities. *Ad Hoc Networks*.
8. Han, J., Li, Y., & Chen, W. (2018). A Lightweight And privacy-preserving public cloud auditing scheme without bilinear pairings in smart cities. *Computer Standards & Interfaces*.
9. Kumar, H., Singh, M. K., Gupta, M. P., & Madaan, J. (2018). Moving towards smart cities: Solutions that lead to the Smart City Transformation Framework. *Technological Forecasting and Social Change*.
10. Osman, A. M. S. (2019). A novel big data analytics framework for smart cities. *Future Generation Computer Systems*, *91*, 620-633.
11. Wang, Y. F., & Ding, D. W. (2015, April). Topology characters of the linux call graph. In *2015 2nd International Conference on Information Science and Control Engineering (ICISCE)* (pp. 517-518). IEEE.
12. Blokhin, K., Saxe, J., & Mentis, D. (2013, July). Malware similarity identification using call graph based system call subsequence features. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops* (pp. 6-10). IEEE.
13. Kato, T., Hayashi, S., & Saeki, M. (2012, October). Cutting a method call graph for supporting feature location. In *2012 Fourth International Workshop on Empirical Software Engineering in Practice* (pp. 55-57). IEEE.
14. Agrawal, G. (1999). Simultaneous demand-driven data-flow and call graph analysis. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*(pp. 453-462). IEEE.
15. Yousefi, A., & Wassyng, A. (2013, March). A call graph mining and matching based defect localization technique. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* (pp. 86-95). IEEE.
16. Xue, J., Hu, C., Wang, K., Ma, R., & Leng, B. (2009, December). Constructing a Knowledge Base for Software Security Detection Based on Similar Call Graph. In *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on* (Vol. 1, pp. 593-597). IEEE.
17. Chroni, M., & Nikolopoulos, S. D. (2012, July). An embedding graph-based model for software

watermarking. In *Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2012 Eighth International Conference on* (pp. 261-264). IEEE.

18. Wang, L., Gong, J., & Shi, L. (2015, November). OLAP Visual Analytics on Large Software Call Graphs with Hierarchical ChordMap. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on* (pp. 675-679). IEEE.
19. Gurukar, S., & Ravindran, B. (2014, January). Temporal analysis of telecom call graphs. In *Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on* (pp. 1-6). IEEE.