IJRSET September 2015 Volume 2, Issue 7 Pages: 42-47

ISSN 2394-739X

International Journal for Research in Science Engineering and Technology

# CLUSTERING OF SOFTWARE MODULES WITH MULTIPLE VIEWS FOR EFFECTIVE MAINTENANCE COST

[1] R. Nandhini
[1] Assistant Professor
[1] Department of Information Technology
[1] Sree Saraswathi Thyagaraja College,
[1] Pollachi.

**ABSTRACT:** Software clustering is the process of combining multiple systems or applications into a cluster that act as a single system. The subsystems are reverse engineered to extract design data. When large software systems are reverse engineered there is possibility of the system decomposition hierarchy. This hierarchy shows the system's subsystems, the contents of the subsystems like modules or other subsystems. The Bunch's software clustering tool shows how meta-heuristic search algorithms can be applied to the software clustering problem, successfully. But some uncertainty prevails whether Bunch provides optimal solution. The optimal solution for trivial systems may be achieved using an exhaustive search. Hence a clustering of software modules method with single graph is used in the existing method to obtain optimal solution by low coupling and high cohesion criterion. The advantage of using spectral methods is that the results this technique produced are within a known factor of the optimal solution. But the spectral single graph shows all the clusters in a single MDG which is complex and costly to maintain. The single graph view shows every system as a node which enables further complex cluster views. This study proposes a method called clustering of software modules with multiple graph views which improves the optimal efficiency and reduces the complexity. In this study the usage of multiple graphs for separate systems also reduces the maintenance cost to a greater extent.

**Keywords: [software clustering, maintenance, Module Dependency Graph].**

## 1. INTRODUCTION

Software clustering tools create abstract structural views of the entities and relations present in the source code. These views, which can be considered a ''road map'' of a system's structure, can help software engineers cope with the complexity of software development and maintenance. The typical design extraction process is to determine the entities and relations in the source code and store the resultant data in either a database or a set of files. This data can then be queried by the user to obtain information about the code structure. In this work readily available source code analysis tools are used for this purpose. After the entities and relations have been stored in a database, the database can be queried to derive a Module Dependency Graph (MDG). The MDG is considered to be a directed graph that represents the software modules (e.g., classes,

files, packages) as nodes, and the relations (e.g., function invocation, variable usage, class inheritance) between modules as directed edges. Once the MDG is created, clustering algorithms can be used to partition the MDG. The clusters in the partitioned MDG represent subsystems that contain one or more modules, relations, and possibly other subsystems. But the single view graph MDG increases the maintenance problems. Hence clustering with multiple views can be introduced to reduce cost.

### 1.1 Problem statement

Software clustering is an efficient method in computing but even the efficient method might have problems. The software systems are needed to be reverse engineered in order to extract data or to troubleshoot a system connected to the cluster. During this process the software cluster enables system decomposition hierarchy which creates breakdown of network into individual systems. The Bunch software clustering tool applies meta-heuristic search algorithms such as hill-climbing, simulated annealing and genetic. However there are limitations in meta-heuristic search algorithms such as their inability to guarantee the proximity of their solutions to the optimal solution and poor results because they converge to local optimum solutions that are far from the optimal one. The spectral single view graph provides global optimal solution but due to a single MDG the cost of maintenance increases. Hence an efficient clustering algorithm with multiple views is proposed to overcome the problem.

## 2. PROPOSED SYSTEM

In the proposed system, in order to reduce the maintenance cost of the clustering tools and to reduce the complexity in the single graph view, clustering of software modules of the data with multiple graphs is used. Spectral Multi-view is proposed such that the complex graph is divided using suitable partitioning techniques to produce multiple graph views. The partitioning helps in the aligning multiple views individually to every system in the cluster. This method increases the efficiency of every cluster that it reduces the complexity in viewing the graph. The partitioning of the graph enables the software of the cluster to be fragmented to simpler individual systems with each one assigned a specific solution. These simpler multiple graphs are easier to analyze and can be reverse engineered whenever the need arises. Multi-view learning is utilized in many other situations. In scientific publication classification, a citation network over the articles, where each node indicates an article and each directed link a citation from one article to another can be constructed. A coauthor network over the articles, where there is a link between two articles if they have an author in common can also be constructed. In social network analysis, there are multiple types of relationships among individuals. For example, they can be email networks, organization hierarchy, and collaboration. As in web categorization, for clustering or classifying scientific publications or individuals, the consideration to utilize several networks together rather than a single network only can be utilized. The problem of software clustering can be avoided by this fragmentation while using these simpler graphs the maintenance cost is reduced significantly.

- Low maintenance cost
- Simple partitioning of graph can be easy to view

**3. Architecture diagram**

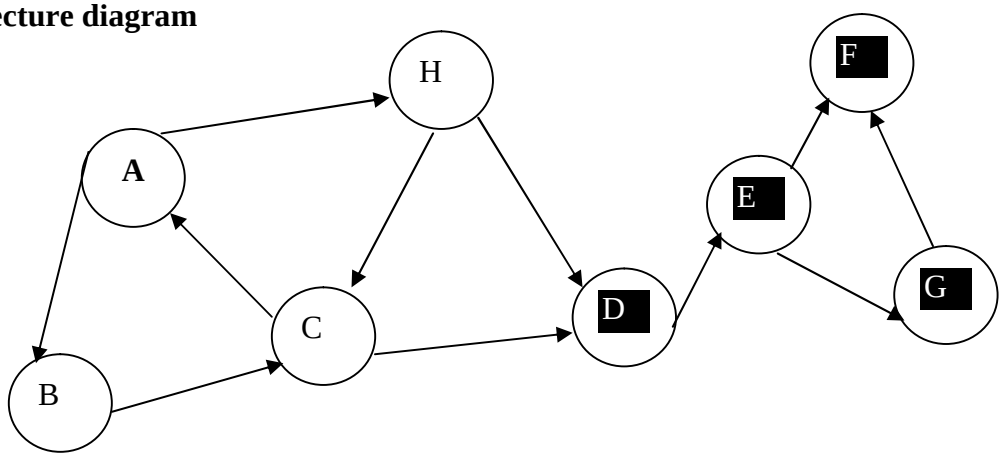
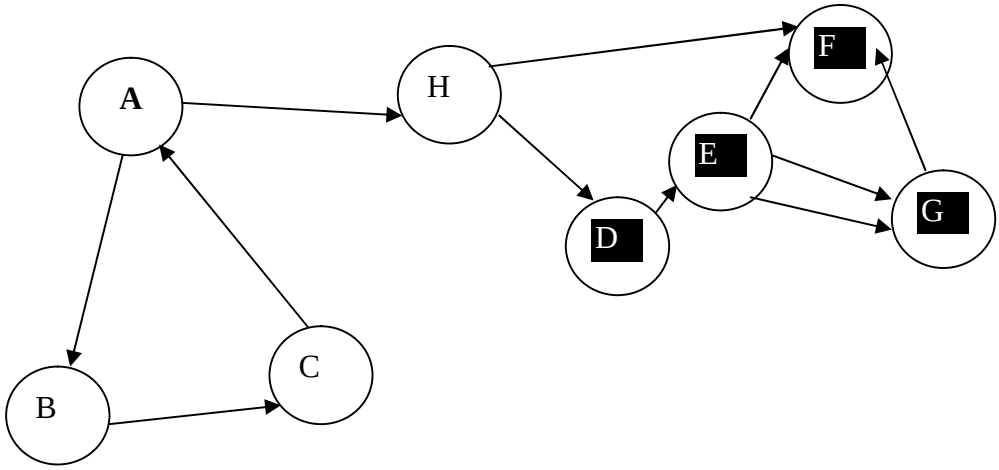



**Figure 2- Clustering of software modules of two different directed graphs with the shaded systems representing the different vertices**




**Algorithm**

1. Initialize the tool
2. Fix the initial adjustments
3. Load the eclipse-ant module
4. Implement the eclipse-ant module
5. Start the source file loading
6. Copy the jar files in Lib folder
7. Paste that jar file in C:\Program Files\Java\jdk1.7.0_60\jre\lib\ext
8. Check initial setup for identifying the packages and interfaces
9. Check initial setup for identifying the object class
10. Add the list of components required
11. Open cmd and goto project location
12. Initialize project files
13. Assign the location for saving the result
14. Run the project using 'java Main' command
15. Analyze the result

## 4. BUNCH SOFTWARE CLUSTERING TOOL ALGORITHM

The Bunch tool implements a variety of meta-heuristic search algorithms to cluster graphs. Bunch's hill-climbing clustering algorithm starts by generating a random partition of the MDG. Modules from this partition are then rearranged systematically in an attempt to find an ''improved'' partition. If a better partition is found, the process iterates, using the improved partition as the basis for

finding even better partitions. The hill-climbing search algorithm eventually converges when no improved partitions of the MDG can be found. The Bunch genetic algorithm (GA) uses operators such as selection, crossover, and mutation to determine a ‘‘good’’ partition of the MDG. This technique is especially good at finding solutions quickly, but we have found that the quality of the results produced by Bunch’s hill-climbing algorithms is typically better. Although each of Bunch’s search algorithms works differently, they all examine partitions from the very large search space of MDG partitions. Note that the number of MDG partitions, the Bell number, is O (N!), where N is the number of modules in the MDG. Thus, Bunch’s search algorithms require a way to determine if one MDG partition is ‘‘better’’ than another. To address this need we define an objective function, which is called as Modularization Quality (MQ), to evaluate the relative quality of MDG partitions. The MQ function works by calculating a value which is called as the Cluster Factor (CF) for each cluster. Given an MDG partitioned into k clusters, MQ is calculated by summing CF for each cluster of the partitioned MDG. CFi for cluster i (1 6 i 6 k) is defined as a normalized ratio between the total weight of the internal edges (edges within the cluster) and half of the total weight of external edges. The weight of the external edges is split in half in order to apply an equal penalty to both clusters that are connected by an external edge. We refer to the internal edges of a cluster as intra-edges, and the edges between two distinct clusters i and j as inter- edges. If edge weights are not provided by the MDG, we assume that each edge has a weight of 1. The MQ measurement design is based on the assumption that good software systems consist of a set of highly-cohesive subsystems clusters in the MDG that are loosely coupled together.

$$MQ = \sum_{i=1}^{k} CF_i$$

Where,

$$CF_i = \begin{cases} 0 & for\ \mu i = 0 \\ \dfrac{2\mu i}{2\mu i + \sum_{j=1}^{i} \varepsilon j, i + \varepsilon i, j} & otherwise \end{cases}$$

The clustering problem, as solved by Bunch, can be stated as a good partition of an MDG graph. We use the term partition in the traditional mathematical sense, that is, the decomposition of a set of elements such as all nodes of a graph into mutually disjoint clusters. By a good partition" we mean a partition where highly interdependent modules (nodes) are grouped in the same subsystems and, conversely, independent modules are assigned to separate subsystems. Finding a good graph partition involves systematically navigating through a very large search space of all possible partitions for that graph. Bunch treats graph partitioning or clustering as an optimization problem. The goal of the optimization is to maximize the value of an objective function, called Modularization Quality (MQ). MQ determines the quality of an MDG partition quantitatively as the trade between interconnectivity (i.e., dependencies between the modules of two distinct subsystems) and intra-connectivity (i.e., dependencies between the modules of the same subsystem). This trade is based on the assumption that well-designed software systems are organized into cohesive subsystems that are loosely interconnected. Hence, MQ is designed to reward the creation of highly cohesive clusters, and to penalize excessive coupling between clusters. All values of MQ are between -1 (no internal cohesion) and +1 (no external coupling). This algorithm is not practical for MDGs with a large number of modules, because the number of partitions of a graph grows exponentially with respect to its number of nodes. Thus, Bunch uses more efficient search algorithms to discover acceptable sub-optimal results. These algorithms are based on hill-climbing and genetic algorithms. Meta-heuristic is a higher-level procedure or heuristic designed to find, generate, or select a lower-level procedure or heuristic (partial search algorithm) that may provide a sufficiently

good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity. Meta-heuristics sample a set of solutions which is too large to be completely sampled. Meta-heuristics may make few assumptions about the optimization problem being solved, and so they may be usable for a variety of problems. Compared to optimization algorithms and iterative methods, meta-heuristics do not guarantee that a globally optimal solution can be found on some class of problems. Many meta-heuristics implement some form of stochastic optimization, so that the solution found is dependent on the set of random variables generated. By searching over a large set of feasible solutions, meta-heuristics can often find good solutions with less computational effort than algorithms, iterative methods, or simple heuristics. As such, they are useful approaches for optimization problems.

## 5. RESULTS AND DISCUSSION

In this section, the performance of the existing and the proposed system is compared. In the existing system, the meta-heuristic and clustering of software modules with single graph is used. In the proposed system the clustering of software modules with multiple graphs is used to reduce complexity. Evaluation of software clustering algorithms is typically done by comparing the clustering results to an authoritative decomposition prepared manually by a system expert. A well-known drawback of this approach is the fact that there are many, equally valid ways to decompose a software system, since different clustering objectives create different decompositions. Evaluating all clustering algorithms against a single authoritative decomposition can lead to biased results.

## CONCLUSION

The clustering algorithms are used to simplify the server applications into smaller clusters to enable higher efficiency denoted by the optimal global solution. The existing method of bunch clustering tool uses the meta-heuristic approach to analyze the functioning of the clusters which is called as spectral clustering. The spectral clustering uses a single graph view which is greatly efficient for small no of clusters. But when the number of clusters in the system increases the graph becomes complex thus affecting the optimal solution. This approach requires separate partitioning methods to simplify the view but still the system could not attain optimum solution. The lack of a global solution makes the clusters to be more time consuming and thus the performance degrades. As the result of this, the maintenance cost also increases. Hence the enhanced method should be implemented. Clustering of software modules method is used to overcome the short comings of the meta-heuristic search algorithm. The clustering of software modules initially uses the single graph view model which provides global solution compared to the Bunch software clustering. But due to the usage of single graph to analyze an entire cluster the system lacks simplicity and causes high maintenance cost. Hence the clustering of software modules with multiple graph view model is introduced which partitions the single complex graph into smaller multiple graphs. Thus the multiple view method reduces complexity as well as maintenance cost to a greater extent compared to the existing model. The system can be enhanced to fit into the individual users in a more affordable cost.The security issues are also a major concern for further research. The security threat is very high in cluster devices. But the cluster devices, at times make this as an advantage to analyze their security. The use of enhanced features additionally in the near future may help to improve the security of the clusters. So still there exists room for enhancement.

## REFERENCES

[1] N. Anquetil, A comparison of graphs of concepts for reverse engineering. In:

Proceedings of the International Workshop on Program Comprehension, 2000.
[2] N. Anquetil, T. Lethbridge, Recovering software architecture from the names of source files. In: Proceedings of Working Conference on Reverse Engineering, 1999.
[3] B. Mitchell, S. Mancoridis. Craft: a framework for evaluating software clustering results in the absence of benchmark decompositions. In: Proceedings of the Working Conference on Reverse Engineering, 2001.
[4] J. Clark, J.J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B.S. Mitchell, S. Mancoridis, K. Rees, M. Roper, M. Shepperd, Reformulating software engineering as a search problem Journal of IEE Proceedings Software 150 (3), 161–175, 2003.
[5] D. Doval, S. Mancoridis, B. Mitchell, Automatic clustering of software systems using a genetic algorithm. In: Proceedings of Software Technology and Engineering Practice, 1999.
[6] Q. Han, Y. Ye, H. Zhang, J. Zhang, on approximation of max-vertex-cover. In: 17th International Symposium on Mathematical Programming in Atlanta, Georgia, 2000.
[7] R. Kannan, S. Vempala, A. Vetta, on clusterings: good, bad and spectral. In: Proceedings of 41st Symposium on Foundations of Computer Science, FOCS 00, Redondo Beach, CA, 2000.
[8] J. Korn, Y. Chen, E. Koutsofios, and Chava: reverse engineering and tracking of Java Applets. In: Proceedings of the 6th Working Conference on Reverse Engineering, pp. 314–325, 1999.
[9] D. McWherter, M. Peabody, W.C. Regli, A. Shokoufandeh, Transformation invariant shape similarity comparison of models. In: Proceedings of ASME Design Engineering Technical Conferences, 2000.
[10] S. Russell, P. Norvig, Artificial intelligence: a modern approach Series in Artificial Intelligence, Prentice Hall, Englewood Cliffs, NJ, 2001.
[11] K. Sartipi, K. Kontogiannis, Component clustering based on maximal association. In: Proceedings of Working Conference on Reverse Engineering (WCRE 01), 2001.
[12] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (8), 888–905, 2000.
[13] V. Tzerpos, R.C. Holt, ACDC: an algorithm for comprehension driven clustering. In: Proceedings of the Working Conference in Reverse Engineering (WCRE 00), 2000.
[14] A. Ando, T. Zhang, Learning on graph with Laplacian regularization, Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2000.
[15] A. Argyriou, M. Herbster, M. Pontil, Combining graph Laplacians for semi-supervised learning Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2006.
[16] S. Rauping, T. Schefier, Learning with multiple views, Proc ICML Workshop on Learning with Multiple Views, 2005.
[17] D. Zhou, J. Huang, B. Schaolkopf, Learning from labeled and unlabeled data on a directed graph, Proc 22th International Conference on Machine Learning, 2005.
[18] C. Castillo, D. Donato, L. Becchetti, P. Boldi, M. Santini, S. Vigna, A reference collection for web spam, SIGIR Forum, 2006.
[19] M. Jordan, A. Ng, Y. Weiss, On clustering of software modules: analysis and an algorithm. In: Advances in Neural Information Processing Systems, number 14, 2001.
[20] X. Zhu, Z. Ghahramani, J. Lafferty, Semi-supervised learning using Gaussian fields and harmonic functions Proc 20th International Conference on Machine Learning, 2003.