Modularizing Software Systems using PSO optimized Hierarchical Clustering

Monika Bishnoi
National Institute of Technology
Jalandhar, INDIA
m.bishnoi29@gmail.com

Paramvir Singh
National Institute of Technology
Jalandhar, INDIA
singhpv@nitj.ac.in

***Abstract*—Software modularization is an automated process for restructuring software entities into modules to refine the software's design. Software systems are required to evolve in order to accommodate the changes relating to their functionalities, performance, and the supporting platforms. As software undergoes the required changes over the time, its structure deteriorates. In recent times, various clustering techniques have been applied to improve the architecture of such systems. Weighted Combined Algorithm (WCA) is a hierarchical clustering-based technique for restructuring software systems, which provides a multi-level architectural view of the system. In this paper, we propose an approach for optimizing WCA using Particle Swarm Optimization (PSO) for software modularization. To analyze the performance of the proposed algorithm, five open source java software systems were considered under the experimental study. The results of this experimental study show that proposed approach outperforms both WCA and PSO clustering techniques when applied to software modularization.**

Keywords—Software Modularization; Hierarchical Clustering; Particle Swarm Optimization; Optimization Techniques.

I. INTRODUCTION

Software modularization is the design technique of organizing software system units into modules such that each module can perform its desired functions as independently as possible. Modules are formed in the form of clusters (groups of classes of a software system). Software systems undergo changes during their life cycle. It is critical for developers to change the functionality of the software without knowing its proper structure. However, the structural knowledge of a system is not so easily available, because most of the time design documentations are inconsistent and cannot be relied on. This problem necessitates a process that can provide high-level structural decompositions of the software systems. Software modularization is such a process and software clustering is the technique for producing such conceptual views. Clustering is the process of grouping entities together according to the similarity among them. Although many clustering techniques have been proposed and applied for modularizing a software system, there exist many techniques for software modularization that are not fully clustering based, even though they apply clustering one or the other way to modularize a software system. For example, clustering and association rule mining together are used to improve a software system's structure [1]. Hence, clustering is an essential technique with in many software modularization approaches.

Clustering techniques presented in [2], mainly categorize clustering algorithms into two types: - partition based and hierarchical. Partition based algorithms require knowledge of the number of clusters to be formed in advance and are computationally expensive. To minimize the computational expense of partition based algorithms, heuristic-search based approaches have been used by researchers to advance software modularization [3], [4], [5]. Also, partition based algorithms generate flat decompositions, but the real decompositions of software systems are usually nested decompositions [33], [34]. Hierarchical clustering algorithms produce multi-level decompositions. Furthermore, these algorithms do not need any prior information. Weighted Combined Algorithm (WCA) is a linkage hierarchical agglomerative clustering based algorithm widely used for modularizing software systems.

Our approach use meta-heuristic partition based technique called Particle Swarm Optimization (PSO) to enhance WCA. Main research question that we investigate in this work is: whether optimizing WCA using PSO overcomes the shortcomings in WCA and produces better clustering results, which has not been investigated yet in the past [6]. The remaining paper is compiled as follows: Section II presents the related work on software modularization. Section III introduces the proposed approach. Section IV briefs the experimental setup. Section V presents the experimental results and analysis. Section VI considers threats to validity, while Section VII presents conclusions with some future recommendations.

II. RELATED WORK

Anquetil et al. [7] evaluated four hierarchical clustering based algorithms including Complete linkage, Single linkage, Unweighted linkage and Weighted linkage and it was concluded experimentally that the Complete linkage generates more cohesive clusters in comparison to other linkage algorithms.

Saeed et al. [8] proposed Combined Algorithm (CA), a new linkage algorithm and proved it better than the Complete Linkage. Maqbool et al. [9] developed the WCA and proposed the Unbiased Ellenberg similarity measure that aims to reduce the formation of non-cohesive clusters. They compared it with Complete Linkage and CA and suggested that WCA

978-1-5090-0082-1/16/$31.00 ©2016 IEEE

outperforms both of them when used with Unbiased Ellenberg similarity measure.

Andritsos et al. [10] proposed an algorithm called LIMBO (scaLable InforMation BOttleneck algorithm). They compared the results with Complete Linkage, Single Linkage, Unweighted Linkage, Weighted Linkage, ACDC, NAHC-lib, SAHC and SAHClib algorithms, and concluded that LIMBO usually outperforms the rest of the algorithms.

Maqbool et al. [11] presented an extensive review of hierarchical clustering techniques for software module clustering and analyzed the working of distance and similarity measures for non-binary and binary features. Wang et al. [12] introduced LBFHC (LIMBO Based Fuzzy Hierarchical Clustering) algorithm, which gives better clustering results by producing more number of clusters with lesser number of arbitrary decisions.

Naseem et al. [13] developed Unbiased Ellenberg-NM (UENM), an enhanced similarity measure for non-binary features and concluded that although LIMBO performs better than WCA-UE, it is outperformed by WCA-UENM [14].

Muhammad et al. [14], [15] assessed a wide range of relationships among classes for software module clustering. They executed a bunch of hierarchical clustering algorithms, i.e. Complete, ACDC [16], Un-weighted, Weighted and Bunch, and concluded that indirect relationships provide superior clustering results as compared to direct relationships.

Hierarchical clustering based Cooperative approach [6], employed cooperation between more than one similarity measures. It is an optimistic approach to exploit the strengths of multiple similarity measures rather than using just a single similarity measure.

Researchers have applied a number of techniques for software modularization apart from the agglomerative hierarchical algorithms. Doval et al. [17] introduced software system as a directed graph termed as Module Dependency Graph (MDG), in which nodes represent entities, and edges represent relationships among entities. Hence software module clustering can be presented as a graph partitioning problem with the aim of maximizing cohesion (intra edges in a cluster) and minimizing coupling (inter edges among clusters) [18]. Graph partitioning problem produces immense solution space that requires exhaustive searching for finding an optimal solution, making it a NP hard problem.

To overcome the computational complexity, Search based approach was proposed by Mancoridis et al. [19] for software modularization. They introduced a tool called Bunch by implementing single objective hill-climbing and genetic algorithms to modularize software systems. The *single objective* used by them to modularize software systems is famously known as Modularization Quality (MQ). The Bunch tool was improved over the years to incorporate new traits [3], [20]. For example, multiple hill climbing was applied to software modularization by Mahdavi et al. [21].

Harman et al. [41] developed an automatic software modularization tool called BruBunch by re-implementing the hill climbing technique proposed by Mancoridis et al. [19]. Other search based techniques have also been used for software modularization such as multiple hill climbing [21], simulated annealing [32] etc. However, empirical studies show that hill-climbing algorithms provide superior and stable outcomes in acceptable time [22].

Praditwong et al. [22] modeled software modularization as a multi-objective search based problem in place of single-objective problem as illustrated in the previous research. They proposed two multi-objective approaches: Equal Cluster Approach (ECA) and Maximum Cluster Approach (MCA). They compared their two-archive algorithm [23] based multi-objective approaches with the hill climbing algorithm based single-objective approach of Bunch tool, and showed that the multi-objective approaches outperform the latter for weighted graphs. Barros [24] studied the effects of MQ and ECA on software modularization, and concluded that by reducing MQ, the multi-objective Genetic Algorithm (GA) produces superior results in lesser computational time.

Kumari et al. [25] proposed a Multi-objective Hyper heuristic Genetic Algorithm (MHypGA) for software module clustering. MHypGA produced superior results than the multi-objective two-archive algorithm [23] and in very less computational time as well.

Hussain et al. [26] presented a more effective optimization technique i.e. the real valued Particle Swarm Optimization (PSO) for software modularization and claimed that PSO outperforms GA. PSO has not been frequently used for modularizing software systems, but it has been applied to a number of other research fields and results have been found to be significant. Izakian et al. [27] used Discrete PSO for job scheduling problem and compared the results with Fuzzy PSO, Continuous PSO, Ant Colony Optimization (ACO) and GA. The results showed that discrete PSO outperforms rest of the algorithms. Cui et al. [28] applied a combination of PSO and K-means to the document clustering problem and concluded that hybrid PSO produces better clustering of documents in comparison with the clustering provided by K-Means and PSO alone. Hence, in this work we apply real valued PSO for optimizing the WCA for software modularization.

III. THEORATICAL FRAME WORK FOR THE PROPOSED APPROACH

WCA is a hierarchical clustering based algorithm given by Maqbool et al. [9]. It combines entities (classes of a software system) on the basis of similarities among them. When classes are joined to form clusters, information regarding number of classes accessing a feature (function calls, global variables etc.) is lost. WCA maintains this information by associating newly formed cluster with a new feature vector.

As mentioned in Section II, WCA-UENM outperforms other Hierarchical Agglomerative Clustering (HAC) techniques and produces better clusters with lesser arbitrary decisions [14]. WCA-UENM uses UENM similarity measure to calculate the similarities among entities.

$$Unbiased\ Ellenberg - NM = \frac{0.5*Ma}{0.5*Ma+b+c+n}$$

978-1-5090-0082-1/16/$31.00 ©2016 IEEE

$$= \frac{0.5*Ma}{0.5*Ma+b+c+(a+b+c+d)}$$

$$= \frac{0.5*Ma}{0.5*Ma+2(b+c)+a+d} \quad (1)$$

Where *Ma* depicts the summation of features existing in both the entities, *a* depicts the count of features that are "1" in both entities, *d* depicts the count of features that are "0" in both entities, whereas *b* and *c* depict the features that are "1" in one entity and "0" in the other.

A disadvantage in case of WCA is that it requires a cutoff point to be selected, which means that the step count after which to terminate the algorithm has to be decided in advance.

PSO is a meta-heuristic search based technique proposed by Kennedy et al. [29]. It is built on the basis of flocking practices of fishes and birds. Each individual entity is treated as a particle. Each particle's position in the swarm is affected both by its own most optimist position so far and the position of the most optimist particle in the swarm. PSO with inertia weights is an improved version of PSO [30]. The *velocity* and *position* of all the particles change according to (2) and (3).

$$v_{t+1} = w * v_t + c_1 r_1 (lbest - x_t) + c_2 r_2 (gbest - x_t) \quad (2)$$

$$x_{t+1} = x_t + v_{t+1} \quad (3)$$

where w,c_1 and c_2 are inertia factor, self-confidence and swarm confidence respectively. v_{t+1} and x_{t+1} are particle's velocity and position respectively after $t+1$ iterations. r_1 and r_2 are uniform random numbers in the range [0,1]. *gbest* is the best global value in the current swarm and *lbest* is the best position of each particle over the iterations. During each iteration, firstly the particle's velocity is updated and then according to the newly calculated velocity, the position of particle is changed. This process continues till the desired value of fitness function is achieved.

The fitness function used in this work is TurboMQ. The TurboMQ [26], [31] for an MDG partitioned into k clusters is calculated as:

$$TurboMQ = \sum_{i}^{k} CF$$

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{\mu_i}{\mu_i + \frac{1}{2}\sum_{\substack{j=1 \\ j \neq i}}^{j=k} (\epsilon_{i,j} + \epsilon_{j,i})} & \mu_i > 0 \end{cases} \quad (4)$$

Where CF is Cluster Factor, i.e. is based on coupling and cohesion for the cluster [26], [31]. CF is defined as a normalized ratio of cohesion (sum of internal edges) to coupling (half of the sum of external edges) of a cluster. μ_i is the cohesion or intra-connectivity of a cluster, and ϵ_{ij} is the inter-connectivity between cluster i and j.

Hence this study aims to optimize WCA-UENM hierarchical clustering algorithm using PSO in order to achieve better software modularization. We use real valued PSO with inertia weights [30].

The step by step procedure to compute the global optimal solution using WCA-PSO is as follows:

1. Associate each entity E_i with a feature vector f_i having components $f_{i1}, f_{i2} \ldots f_{ip}$.
2. Calculate similarity between each pair of entities using (1).
3. Repeat
   a. Cluster the two most similar entities.
   b. Associate the newly formed cluster with a new feature vector. Let E_i and E_j be the entities to be clustered, with feature vectors f_i and f_j, respectively. Let the number of entities in E_i be denoted by n_i and the number of entities in E_j by n_j.
      The feature vector f_{ij} is computed as:
      $f_{ij} = (f_i + f_j) / (n_i + n_j)$
      $= \{(f_{ik} + f_{jk}) / (n_i + n_j)\}$ where $k = 1, 2 \ldots p$.
   c. Treat the newly formed non-singleton cluster as a new entity and re-calculate similarity between this freshly composed cluster and rest of the entities.
   d. Calculate TurboMQ using (4).

   Until
   The needed count of clusters is achieved or TurboMQ stops increasing.
4. For each particle:
   a. Initialize particle *position* equal to clustering result obtained from Step 3.
   b. Initialize particle *velocity* to 0.
   c. Assign 0 to *lbest*.
   End.
5. Assign 0 to *gbest*.
6. Repeat
   a. Compute TurboMQ value using (4).
   b. If computed value is large than *lbest* value
      i. For each particle:
      Change *lbest* to current position of the particle.
      End.
   c. Change *gbest* to *lbest* of the cluster having maximum TurboMQ value.
   d. For all the particles:
      i. Compute *velocity* using (2)
      ii. Compute *position* using (3)
   End

   Until
   A maximum iterations criterion is not attained.

IV. Experimental Setup

We explain the test systems and experimental arrangements used for our research in this section. The algorithms have been implemented in Eclipse-Luna, on Intel® Core™ 2 Duo CPU T5670@1.80 GHz processor, 3 GB RAM and Ubuntu-14.04.2-desktop-i386 platform.

A. Test Systems

For conducting the empirical study, we selected five open source object oriented software systems written in java. Test systems cover small to moderate size software systems. A brief explanation of these systems is presented in Table I.

978-1-5090-0082-1/16/$31.00 ©2016 IEEE

B. Entities and Features

In our study, we have chosen *class* as an entity because all the test systems are object-oriented. From all the relationships that exist between entities, the static dependencies among classes were selected as features for finding the similarity among the entities. For extracting these dependencies, Jdeps tool is used. Jdeps [40] is a static class dependency analyzer tool for java and is useful for quickly identifying static dependencies of applications and libraries. Its output can be exported in 'dot' format. Jdeps comes as a built-in command line tool with JDK 8. The class dependencies involving java inbuilt library classes are not considered in this study as we target modularization of test system classes only.

C. Algorithmic Parameters

To cluster the most similar entities together, WCA, PSO and our proposed approach (WCA-PSO) were selected. Maximum number of iterations for WCA is n-1 (where n is the number of classes present in an entity) because after n-1 iterations WCA starts to merge all the clusters to form a single cluster. For this study, out of n-1 iterations for WCA, the iteration resulting in the highest TurboMQ is chosen.

Parameter values for PSO and WCA-PSO are shown in Table II. The output of WCA is used as an initial seed to the result optimizing part (PSO) of the proposed algorithm, which generates the final result.

D. Assessment Criteria

The decompositions obtained automatically using the clustering algorithms under study are assessed using criteria consisting of number of isolated clusters formed, cohesion, coupling and TurboMQ.

Number of clusters produced by an algorithm during the clustering process is an important factor for software modularization. A good clustering algorithm should not group all the entities into one cluster and not produce too many singleton (cluster having only one class) clusters as well. Cohesion is the number of intra-edges between classes in a single cluster, and coupling is the number of inter-edges among the clusters of software system. Cohesion should be high and coupling should be low in a well modularized software system. In this work, both cohesion and coupling are evaluated at the class level to guide the cluster formation (group of classes of a jar file) for a test system. Cohesion/coupling values of a modularized test system are calculated as aggregations of the cohesion/coupling values taken over all the individual clusters of the respective software system.

TABLE I. TEST SYSTEMS

Software Systems	Description	No. of Classes
Graphviz [35]	Graph Visualization Software	31
Launch4J [36]	Java Executable Wrapper	109
MidiQuickFix [37]	Editor and Player	160
JHawk [38]	Java Metrics Tools	314
Sellwin [39]	CRM Sales Application	322

TABLE II. PARAMETER VALUES FOR PSO AND WCA-PSO

Parameters	PSO	WCA-PSO
Population size	Equal to number of entities	Equal to number of entities
Iterations	100	WCA's iteration + 100
Initial position	Random	Results obtained from WCA
Initial velocity	0	0
Self-confidence (c1)	1 – 2	1 – 2
Swarm-confidence(c_2)	1.5 - 2.5	1.5 - 2.5
Inertia weight(w)	0.4-0.9	0.4-0.9

E. Methodology

Fig.1 provides the methodology adopted to analyze the performance of proposed approach. Firstly, java dependency analyzer tool called Jdeps was used to extract the dependencies among the classes of input test systems in 'dot' format. A dot parser was designed to convert the class dependencies stored in dot file into a numeric matrix called dependency matrix. The input class dependencies in the form of dependency matrix were provided to all 3 algorithms (WCA, PSO and WCA-PSO) separately. These algorithms group classes into different clusters. Once the clusters are formed, the next step is the evaluation of these modularization algorithms by using the selected assessment criteria. The comparative results for all these algorithms were then analyzed.

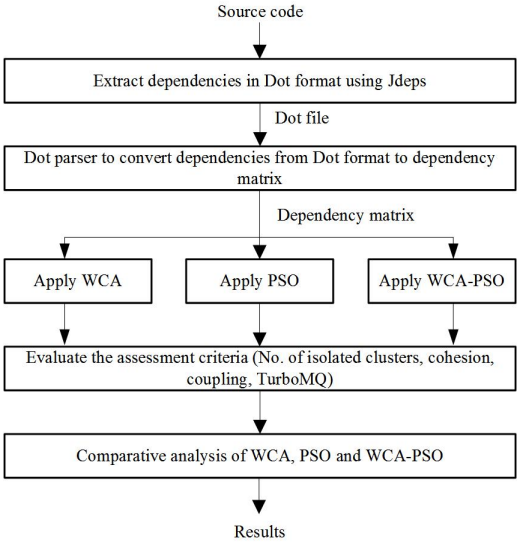

Fig. 1. Experimental methodology followed in our work.

V. EXPERIMENTAL RESULTS AND ANALYSIS

For each test system, total 30 trials of each algorithm are executed. The clustering providing the highest TurboMQ is selected to be the best clustering solution for each execution and then an average is taken over all the executions.

978-1-5090-0082-1/16/$31.00 ©2016 IEEE

The values of number of isolated clusters formed, number of clusters, cohesion, coupling and TurboMQ are shown in Table III. In case of both WCA and WCA-PSO, analysis is performed over a range of iterations; however iterations giving best TurboMQ are shown in Table III. Increasing the number of iterations of PSO is not affecting the overall results; hence we decided to execute 100 iterations of PSO.

A. Number of Clusters

Table III shows number of isolated clusters formed by all the three algorithms under study. It is clear from Table III that WCA-PSO does not form any isolated clusters for the 3 out of 5 test systems i.e. Launch4J, MidiQuickFix and JHawk. But WCA forms isolated clusters for all test systems except JHawk. PSO also forms isolated clusters for 2 test systems i.e. Launch4J and MidiQuickFix.

B. Cohesion and Coupling

Table III shows that WCA-PSO results in stronger cohesion in comparison to that resulted by WCA for all the test systems. WCA-PSO steadily increases the number of clusters formed (or atleast keeps it constant in the worst case), except for Launch4J, which is a good sign for better software modularization. PSO also increases cohesion for all test systems except for Sellwin.

As shown in Table III, WCA-PSO results in higher coupling as compared to WCA for 3 out of 5 test systems (i.e. Launch4J, JHawk, and Sellwin). PSO also does the same for 2 out of 5 test systems (i.e. JHawk, Sellwin).

C. TurboMQ

It is clear from Table III that WCA-PSO gives highest TurboMQ values for all test systems except Graphviz.

It is evident that where WCA-PSO reduces isolated clusters significantly and increases cohesion, it increases coupling slightly in some cases e.g. Launch4J, JHawk and Sellwin. However this increase in coupling can be ignored as long as TurboMQ is on the higher end. Because in such cases, an increasing TurboMQ can be attributed to proportionate increase in cohesion in individual clusters, which is a sign of good modularization.

In case of WCA, once clusters are formed, it does not change the position of classes and keeps on merging clusters (singleton/others).Towards the ending phase of algorithm, it combines all the clusters in a single cluster which is disastrous for software modularization. But in case of WCA-PSO, PSO optimizes the results of WCA by changing classes' positions according to global optima, and hence helps provide better overall clustering.

It is clear from the results that PSO alone also reduces number of clusters and increases cohesion than the WCA. But it reduces TurboMQ as well. So it answers our research question that when we optimize WCA using PSO, better software clustering is achieved. The results also provide evidence that rather than applying optimization techniques alone, combination of clustering techniques promise better results.

VI. THREATS TO VALIDITY

The major threat to any research in software engineering is related to generalizing the research results to a wider range of software systems. Hence a threat to this study is linked to number and type of test systems. We have considered five test systems for our study from a variety of domains to make the empirical study more applicable. However, to make our results more affordable, we have considered only open source object-oriented software systems. Using limited features (only static dependencies) for the proposed approach could be considered as another threat, however as the results are satisfactory with these limited features, the utilization of more features could be deferred. In case of optimization technique, selecting parameters' values is very crucial and important for producing better results. To mitigate this challenge, we have experimented with each possible value of parameters used in PSO and selected the best ones.

TABLE III. COMPARISION OF CLUSTERING SOLUTIONS FOUND BY WCA, PSO AND WCA-PSO

Test Systems	*Algorithm*	*Iterations*	*Clusters*	*Isolated Clusters*	*Cohesion*	*Coupling*	*TurboMQ*
Graphviz	WCA	26	3	1	15	49	0.410
	PSO	100	4	0	48	19	**1.934**
	WCA-PSO	26+100	3	1	17	46	0.541
Launch4J	WCA	72	10	12	89	147	2.790
	PSO	100	12	2	156	118	1.385
	WCA-PSO	72+100	8	0	104	165	**3.060**
MidiQuickFix	WCA	125	8	10	78	189	0.744
	PSO	100	10	2	138	179	1.217
	WCA-PSO	125+100	8	0	130	185	**1.241**
JHawk	WCA	263	10	0	343	764	2.922
	PSO	100	11	0	249	990	1.572
	WCA-PSO	263+100	14	0	391	843	**3.168**
Sellwin	WCA	237	22	30	194	655	3.215
	PSO	100	20	0	99	934	1.302
	WCA-PSO	237+100	25	1	236	790	**3.750**

978-1-5090-0082-1/16/$31.00 ©2016 IEEE

VII. Conclusions and Future Work

This study presents a multi-objective technique for software modularization. It proposes a PSO based optimization of hierarchical clustering algorithm WCA and compares the results with WCA and PSO. The comparison is based on four main objectives TurboMQ, isolated clusters, cohesion and coupling. The results indicate that proposed approach (hierarchical algorithm integrated with optimization technique) produces better software clustering than individual algorithms. Future work could consider additional features of entities like inheritance, global variables, user defined types etc. It should also consider effect of other optimization techniques such as Artificial Bee Colony (ABC) optimization, Ant Colony Optimization (ACO) etc. for improving hierarchical clustering based software modularization. Another future direction is to consider the use of dynamic dependency information for software modularization.

References

[1] K. Sartipi, K. Kontogiannis, and F. Mavaddat, "Design Recovery using Data Mining Techniques," *In Fourth European Conference Software Maintenance and Reengineering*, Zurich, 2000, pp. 129-140.

[2] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 13, no. 3, pp. 264-323, 1999.

[3] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, "Using Automatic Clustering to Produce High-Level System Organizations of Source Code," *In International Workshop on Program Comprehesion, Los Alamitos,California, USA, 1998,* pp. 45-53.

[4] K. Sartipi and K. Kontogiannis, "A User-Assisted Approach to Component Clustering," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 4, pp. 265-295, 2003.

[5] A. Shokoufandeh, S. Mancoridis, T. Denton, and M. Maycock, "Spectral and Meta-Heuristic Algorithms for Software Clustering," *Journal of Systems and Software*, vol. 77, no. 3, pp. 213-223, 2004.

[6] R. Naseem, O. Maqbool, and S. Muhammad, "Cooperative Clustering for Software Modularization," *Journal of Systems and Software*, 2013.

[7] N. Anquetil, Lethbridge, "Comparative Study of Clustering Algorithms and Abstract Representations for Software Remodularisation," *In IEE Proceedings Software*, vol. 150, no. 3, pp. 185–201, 2003.

[8] M. Saeed, O. Maqbool, H. A. Babri, S. Hassan, and S. Sarwar, "Software Clustering Techniques and the use of Combined Algorithm," *In International Conference Software Maintenance and Reengineering*, 2003, pp. 301–306.

[9] O. Maqbool and H. A. Babri, "The Weighted Combined Algorithm: A Linkage Algorithm for Software Clustering," *In International Conference Software Maintenance and Reengineering*, 2004, pp. 15– 24.

[10] P. Andritsos and V. Tzerpos, "Information Theoretic Software Clustering," *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 150–165, 2005.

[11] O. Maqbool, Babri, H.A., "Hierarchical Clustering for Software Architecture Recovery," *IEEE Transactions on Software Engineering*, vol. 33 , no. 11, pp. 759–780, 2007.

[12] Y. Wang, P. Liu, H. Guo, H. Li, X. Chen, "Improved Hierarchical Clustering Algorithm for Software Architecture Recovery," *In International Conference onIntelligent Computing and Cognitive Informatics*, Kuala Lumpur, 2010, pp. 247–250.

[13] R. Naseem, O. Maqbool, and S. Muhammad, "Improved Similarity Measures for Software Clustering," *In 15th European Conference on Software Maintenance and Reengineering*, Oldenburg, 2011, pp. 45-54.

[14] S. Muhammad, "Experimental Evaluation of Relationships for the Modularization of Object-Oriented Software Systems," Mphil. Thesis, Quaid-i-Azam University, Islamabad, 2010.

[15] S. Muhammad, O. Maqbool, A. Abbasi, "Evaluating Relationship Categories for Clustering Object-Oriented Software Systems," *IET Software* , vol. 6, no. 6, pp. 260–274, 2012.

[16] V. Tzerpos, R.C. Holt, "ACDC: An Algorithm for Comprehension-Driven Clustering," *In Conference on Reverse Engineering*, Brisbane, Queensland, 2000, pp. 258–267.

[17] D. Doval, S. Mancoridis, and B.S. Mitchell, "Automatic Clustering of Software Systems using a Genetic Algorithm," *In International Conference on Software Tools and Engineering Practic*, Pittsburgh, PA, 1999.

[18] M. Harman, S. Swift, K. Mahdavi, "A New Representation and Crossover operator for Search-Based Optimization of Software Modularization," *In Genetic and Evolutionary Computation Conference*, 2002, pp. 1351-1358.

[19] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A Clustering Tool for the Recovery and Maintenance of Software," *In IEEE International Conference on Software Maintenance*, Oxford, 1999, pp. 50-59.

[20] B.S. Mitchell, S. Mancoridis, "Using Heuristic Search Techniques to Extract Design Abstractions from Source CDE," *In Genetic and Evolutionary Computation Conference*, 2002, pp. 1375–1382.

[21] K. Mahdavi, M. Harman, and R. M. Hierons, "A Multiple Hill Climbing Approach to Software Module Clustering," *In IEEE International Conference on Software Maintenance*, Los Alamitos, California, USA, 2003, pp. 315– 324.

[22] K. Praditwong, M. Harman, Xin Yao, "Software Module Clustering as a Multi-Objective Search Problem," *IEEE Transactions on Software*, vol. 37, no. 2, pp. 264-282, 2011.

[23] K. Praditwong and Xin Yao, "A New Multi-objective Evolutionary Optimisation Algorithm Praditwong, "The Two-Archive Algorithm," *In International Conference on Computational Intelligence and Security*, Guangzhou, China, 2006, pp. 286–291.

[24] M. Barros, "An Analysis of the Effects of Composite Objectives in Multiobjective Software Module Clustering," *In FourteenthInternational Conference on Genetic and Evolutionary Computation Conference*, 2012, pp. 1205–1212.

[25] A.C. Kumari, K. Srinivas, M. P. Gupta, "Software Module Clustering using a Hyper-Heuristic based Multi-Objective Genetic Algorithm," *In Advance Computing Conference*, Ghaziabad,2013.

[26] I. Hussain, A. Khanum, A. Q. Abbasi, and Y. Javed, "A Novel Approach for Software Architecture Recovery using Particle Swarm Optimization," *International Arab Journal of Information Technology*, 2014.

[27] H. Izakian, B. Ladani, A. Abraham, and V. Snasel, "A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling", *International Journal of Innovative Computing Information and Control*, vol. 6, no. 9, 2010.

[28] X. Cui, T.E. Potok, and P. Palathingal, "Document Clustering using Particle Swarm Optimization", *In IEEE Swarm Intelligence Symposium*, pp. 185-191, 2005.

[29] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *In IEEE International Conference on Neural Networks*, 1995, pp. 1942-1945.

[30] Y. Shi, R.C. Eberhart, "A Modified Particle Swam Optimizer," *In IEEE Word Congress on Computational Intelligence*, 1998, pp. 69-73.

[31] B. S. Mitchell, "A Heuristic Search Approach to Solving the Software Clustering Problem," PhD dissertation, Drexel University, 2002.

[32] B.S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool", *IEEE Transaction on Software Engineering*, vol. 32, no. 3, pp.193 -208, 2006.

[33] D.H. Hutchens and V.R. Basili, "System Structure Analysis: Clustering with Data Bindings," *IEEE Transactions Software Engineering*, vol. 11, no. 8, pp. 749-757, 1985.

[34] M. Shtern and V. Tzerpos, "A Framework for the Comparison of Nested Software Decompositions," *In 11th IEEE Working Conference Reverse Engineering*, 2004, pp. 284-292.

[35] Graphviz. http://www.graphviz.org/Download.php.

[36] SourceForge. http://sourceforge.net/projects/launch4j/files/launch4j-3/.

[37] OSDN.http://en.osdn.jp/projects/sfnet_midiquickfix/downloads/midiquickfix/MidiQuickFix1.4.0/MidiQuickFix.jar/.

[38] JHawk. http://www.virtualmachinery.com/jhdownload.htm.

[39] SourceForge. http://sourceforge.net/projects/sellwincrm/.

[40] Oracle.http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

[41] M. Harman, S. Swift, K. Mahdavi, "An Empirical Study of the Robustness of Two Module Clustering Fitness Functions," *In Genetic and Evolutionary Computation Conference*, 2005, pp. 1029–1036.

978-1-5090-0082-1/16/$31.00 ©2016 IEEE