A Novel Approach for Software Architecture Recovery using Particle Swarm Optimization

Ibrar Hussain[1], Aasia Khanum[2], Abdul Qudus Abbasi[3] and Muhammad Younus Javed [4]
[1,4]College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan
[2]Department of Computer Science, Forman Christian Collage Lahore, Pakistan
[3]Department of Information Technology, Quaid-i-Azam University, Pakistan

Abstract: *Software systems evolve and change with time due to change in business needs with the result that at some stage, the original design and architecture descriptions may not give exact representation of the actual software system. Accurate understanding of software architecture is very important for software maintenance because it helps in estimating scope of change, re-usability, cost, and risk involved in change. In some cases, for instance in legacy systems, an accurate architectural description may not even exist and it becomes necessary to extract the same from source code. Software clustering is the process of decomposing large software system into sub-systems on the basis of similarity between units in the sub-systems, essentially a depiction of the architecture. Software clustering, however, is an NP-hard problem that can be efficiently handled with help of meta-heuristic approaches. Particle Swarm Optimization (PSO) is an evolutionary meta-heuristic search based on flocking behavior of biological species and can be used to solve software clustering problem. This paper provides a novel framework for software clustering using PSO. The proposed algorithm is examined using three industrial software systems. Comparison of results with another mainstream meta-heuristic shows that the PSO approach performs better in terms of computational effort, consistency and quality of results.*

Keywords: *Software clustering, software architecture, software maintenance, software evolution, search based software engineering, PSO.*

Received September 21, 2012; accepted September 18, 2013; published online April 17, 2014

1. Introduction

Architecture of a software system is defined as "the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution" [18]. Software architecture encapsulates higher level design of software, defining its various sub-systems and their relationships. Knowledge of software architecture is needed in various phases of software lifecycle e.g., maintenance, evolution and reuse [22]. However, for many systems this architectural knowledge is not so readily available and the software managers have to incur extra efforts in recovering the underlying architecture from source code. Manual methods can be considered as last resort measure for architecture recovery, but in the face of large size and complexity of today's legacy software these measures prove costly and time-consuming. It is now generally recognized that in order for software architecture recovery to be viable, it must be handled by automatic or semi-automatic tools [34, 43].

Clustering is the process of decomposing large system into smaller manageable subsystems in such a way that entities within the subsystem are similar to one another and different from those in other subsystems. The similarity and difference is measured based on presence and absence of some features [32]. The terms entities and features are commonly used. Entities are usually files, classes or functions whereas features are the attributes such as number of function calls of one class within another class [39].

Clustering was first adopted for software architecture recovery in 1980's and is now a well-established approach for automated recovery of software architecture. Historically, the adoption of clustering methodology in software domain has progressed along multiple strategies. Sometimes, classical clustering techniques have been applied directly to the software, sometimes classical techniques have been adapted according to the special characteristics of the software domain and sometimes entirely new clustering techniques have been devised specifically for software applications [13, 15]. Techniques used for software clustering can be grouped into graph theoretic techniques, Information Retrieval (IR) based techniques, data mining based techniques, pattern matching based techniques and meta-heuristic approaches. Owing to the NP-hard nature of the problem, meta-heuristics based techniques are of special relevance due to their efficiency in finding near-optimal solutions. Many meta-heuristics have been used for software clustering including Genetic Algorithms (GA) and evolutionary strategies. This paper presents a novel approach based on a relatively new and more efficient meta-heuristic

i.e., the Particle Swarm Optimization (PSO) for software clustering.

The remainder of the paper is organized into five sections. In section 2, literature review is presented. Section 3 discusses theoretical framework of the proposed approach. Section 4 presents experimental results. Section 5 is the concluding part.

2. Literature Review

In the past, several different techniques have been employed for clustering software. We can broadly divide these techniques into five categories: Graph theoretic techniques, IR based techniques, data mining based techniques, pattern matching based techniques, and meta-heuristic approaches. Representing the software as a graph (where nodes are software entities like classes or modules and edges are relationships like function calls or inheritance etc.,), the graph theoretic techniques try to find the best clusters in the graph using mathematical graph-theoretic concepts [33]. Some of the earliest clustering algorithms belong to this category e.g., single-link [42] and complete-link [26]. Single link clusters are sub-graphs of the Minimum Spanning Tree (MST) [14] whereas complete-link clusters are maximal complete sub-graphs of the MST [4]. Similarly, min-cut is a well-known graph-partitioning technique that can be applied to software graphs. Mancoridis *et al.* [31] used graph search for software clustering with modification to min-cut in the form of a Modularization Quality (MQ) measure which depicts cohesion and coupling among software modules. They presented an optimal clustering algorithm that finds the best clustering of software's Module Dependency Graph (MDG) by finding all partitions of the MDG and selecting the best k-partition on MQ basis. The algorithm's complexity, however, is exponential.

Spectral graph theory deals with the algebraic representations of graphs and their properties like adjacency matrix, laplacian matrix etc., Spectral graph partitioning [10, 11] aims to partition a graph on basis of eigen-vectors. Shokoufandeh *et al.* [41] applied spectral graph theory to find clusters in the MDG. Modeling the problem as a relaxed optimization problem with objective function trading off cohesion and coupling between nodes of the MDG, they were able to propose a polynomial-time solution within a known factor of the optimal solution. However, their worst case complexity is quite high (i.e., $\theta(n^4)$ [41]). Further, the quality of the solution and running time deteriorates with larger size of the graph [7].

Problem with graph theoretical methods is that the search-space grows exponentially as the size of the software system increases [36]. Most of these methods assume that good software engineering principles have indeed been followed during construction of software so that inter-related components will have high cohesion and low coupling; however, there is no guarantee of truth of this assumption [3]. Reliance on high cohesion also makes these methods less suitable for identifying utility clusters which do not general possess this attribute.

IR deals with disciplined ways of extracting information from wide data sources. Frakes and Nejmeh [12] were an early approach that used IR for performing keyword and expression based searches on source code which was treated as a natural language corpus.

Latent Semantic Analysis (LSA) [5] is a statistical IR technique that is used for representing and comparing natural language text as real valued vectors. The vector represents semantic meaning of the text in a quantitative manner that encapsulates corpus-derived knowledge about the context of a particular word of text. Maletic and Marcus [29] used LSA for automatic software clustering. Documents correspond to modules, files, components and classes whereas words correspond to identifier names, operator, operands in source code and other words in documentation. Document/ module is added to a cluster if its cosine similarity measure with other documents/ modules is above a specified threshold. This method can not only discover semantically related modules but can also find stand-alone modules.

Kuhn *et al.* [27] introduced "Semantic Clustering" where in Latent Semantic Indexing (LSI) is used to cluster entities (e.g., methods, classes etc.,) on the basis of similar vocabulary. LSI is used again to derive labels for the clusters with their most relevant terms. Finally, the distribution of concepts in the software is analyzed to determine how it is structured. The LSI approach has the advantage of language independence. One problem with this approach is that it is dependent on the quality of naming conventions (e.g., Java Naming Conventions, Hungarian Notation etc.,) used in the software under analysis. The system fails if the programmers consistently followed bad naming habits.

Data mining is a field of study that aims to process information present in large databases and convert it to a meaningful structure usually in the form of association rules. For instance, Montes and Carver [38] used data mining concepts for extracting sub-systems. Specifically, association rule mining is used to identify data-related subsystems that use the same data.

Pattern matching approaches are based on the observation that there is a general vocabulary of patterns that are seen across many software architectures. Describing software clustering in terms of these patterns can be more helpful to the software maintainer than structure based approaches. For instance ACDC [44] identifies common patterns found in large systems. ACDC finds clusters according to various types of patterns, including source file pattern, directory structure pattern, body-header pattern, leaf-collection pattern, support library pattern, central dispatcher pattern and sub-graph dominator pattern.

Sartipi used a graph pattern matching [40] approach for architecture recovery of legacy software. Architecture Query Language (AQL) is introduced to describe architectural patterns including abstract

modules and interactions. AQL patterns are mapped to Attributed Relational Graphs (ARG) which are used to query an entity-relation graph obtained by parsing the source code. Nodes in the graph represent heterogeneous entities like variables, functions, types etc., and edges represent entity relationships like call, update, define, set and declare. Query satisfaction is an in-exact graph matching process based on finding the optimal sequence of graph edit operations (insert edge, delete edge and delete node) on subsets of the source code graph that render the source code graph most similar to the query graph. Even after finishing, the algorithm may leave some unresolved entities that require user involvement for placing them in appropriate modules. The system is computationally heavy as it requires finding suitable query matches in the graph.

Due to computational complexity of exact solutions in high-dimensional spaces, approximate or heuristic solutions are a favored option. Heuristic methods, however, provide only locally optimum solutions in the search space; they cannot guarantee global optimization. Meta-heuristic approaches, in general, are used to optimize the performance of heuristic approaches in search and optimization problems. Meta-heuristics are heuristic procedures used to tune, control, guide and allocate computational resources or reason about object-level problem solvers in order to improve their quality, performance or efficiency [6]. Well-known meta-heuristics include Evolutionary Algorithms (EA), GA, Artificial Neural Networks (ANN), Fuzzy Logic (FL), Simulated Annealing (SA), PSO, and Artificial Immune Systems (AIS).

Search Based Software Engineering (SBSE) [16] recognizes the central role of meta-heuristic search in software engineering problems. SBSE, a term first used by Harman and Jones [16] aims to formulate software engineering problems as search based optimization problems. Meta-heuristic search, as advanced by SBSE, is particularly suited for software engineering in general and software clustering in particular because of the following peculiarities of the software domain:

- There are too many competing and interacting constraints that have to be balanced against each other.
- These problems have high computational complexity and hence cannot be solved by traditional optimization techniques like linear programming or dynamic programming etc.
- There are too many interacting parameters.
- There are a large number of solutions. No precise definition of the best solution exists, yet it is possible to recognize a good solution from a bad one.
- The constraints are vaguely/ poorly defined.

Mancoridis *et al.* [31] used GA to find clusters in a File Dependency Graph (FDG) extracted from source code. Kazem and Lotfi [21] used MDG and concepts of inter-and intra-cluster connectivity as parameters in a GA. The GA search is guided by MQ measure. Khan *et al.* [24] presented Evolutionary Strategy Based Automatic Software Clustering Approach (ESBASCA). Wang *et al.* [45] combined FL with a richer feature set to obtain a LIMBO Based Fuzzy Hierarchical Clustering (LBFHC).

The approach presented in this paper utilizes a more recent meta-heuristic algorithm, the PSO for software clustering and architecture recovery. It is a population based parallel meta-heuristic inspired from collective intelligence of swarms of particles. We have mapped the software clustering problem to PSO formalism. Further, we also show how the PSO based approach performs better than the other state of the art approach i.e., GA.

PSO approach has not been widely used to solve software clustering problem where the input is the number of classes contained in the software system and relationships between those classes. Li and Yang [28] used PSO to track multiple peaks based on nearest neighborhood search strategy. The experimental results show good performance on different test problems. Khanesar *et al.* [25] used binary PSO to find global minimum and the experimental results are quite satisfactory. Hassan *et al.* [17] performed comparative study and applied PSO and GA on Banana function and Eggcrate function. The experimental results verify the computational efficiency of PSO. Izakian *et al.* [19] applied discrete PSO to job scheduling problem to minimize the makespan and flowtime and compared the results with GA, Ant Colony Optimization (ACO), Fuzzy PSO and Continuous PSO. Experimental studies illustrate that the Discrete PSO is more efficient. Cui *et al.* [9] used hybrid PSO (combined with K-means clustering) to solve document clustering problem. Their experimental results illustrate that the use of hybrid PSO produces best clusters as compared to using PSO and K-means alone. Alam *et al.* [2] applied PSO to data clustering problem and compared the results with K-means clustering. Experimental results verify the effectiveness of PSO against K-means clustering.

3. Theoretical Framework of Proposed Approach

PSO is a nature inspired optimization algorithm introduced by Kennedy and Eberhart [23]. It is a probabilistic technique based on social behavior of bird flocking and fish schooling [8]. In PSO, potential solutions are called particles. These particles fly through the problem space by following the best positions found by neighbor particles and by themselves [28]. Every particle keeps record of its best position achieved so far. This value is called *lbest* [8]. There is another value *gbest* which is the best position obtained so far by any particle in the swarm [8]. A swarm is similar to population as in GA and particle is analogous to an individual in GA [19]. PSO is considered as collective and iterative method due to its

emphasis on cooperation [8]. Main advantage behind PSO idea is to combine local search and global search in terms of lbest and gbest respectively.

Each particle has position and velocity. Initially positions and velocities of all the particles are randomly initialized [46]. On each iteration, first the velocity of the particle is updated and then its position. The PSO algorithm uses lbest and gbest for adjusting the velocity of the particle. This process continues until desired fitness level is achieved. In other words, PSO algorithm is mainly composed of three steps; velocity update, position update and fitness calculation until desired convergence level is achieved [17].

Equations 1 and 2 are used to update velocity and position of the particle [46] as described in Figure 1.

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

$$x_{t+1} = x_t + v_{t+1} \quad (2)$$

Where, c_1 and c_2 are self-confidence factor and swarm-confidence factor respectively [17], r_1 and r_2 are uniform random numbers in the range [0, 1].

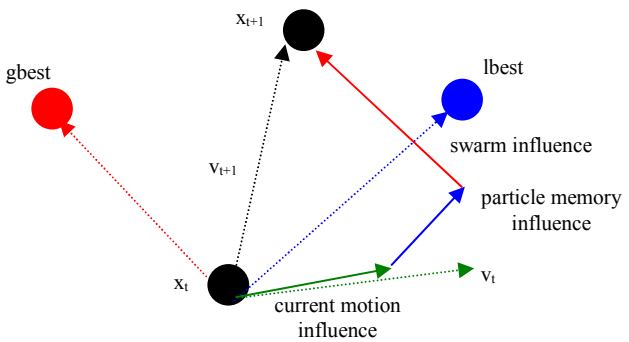


Figure 1 [17]. Velocity and position updates in PSO.

Equations 1 and 2 represent what is called real-valued PSO [25] because velocities and positions are represented using real values. The other version is the discrete version also proposed by Khanesar *et al*. [25] which is used to make Boolean decision. New position of the particle is decided using sigmoid function [17].

$$x_{t+1} = \begin{cases} 1 & if\ r < sig(v_{t+1}) \\ 0 & otherwise \end{cases} \quad (3)$$

Where, r is a uniform random number in the range [0, 1] and

$$sig(v_{t+1}) = 1 / (1 + exp(-v_{t+1})) \quad (4)$$

4. Software Clustering Problem with PSO

Suppose in a system there are m classes $\{C_1, C_2, \ldots, C_m\}$ that we want to cluster into k clusters $\{K_1, K_2, \ldots, K_k\}$ as a depiction of the system's architecture. Quality of the resulting architecture can be evaluated in terms of its cohesion and coupling. Cohesion is a feature that measures how interrelated the modules in a cluster are. Coupling on the other hand is an inter-cluster concept that measures how inter-dependent two given clusters are. We always strive for high cohesion and low coupling in architecture. This ensures desirable architectural characteristics like understandability, maintainability, reusability etc.

As described by [31, 37] when software architecture having n clusters is represented as a MDG with modules as nodes and their relationships as edges, coupling and cohesion can be expressed in terms of inter-and intra-connectivity of the clusters respectively. Inter-connectivity $\varepsilon_{i,j}$ between cluster i and cluster j with N_i and N_j nodes (i.e., modules) and $E_{i,j}$ inter-edges (edges that start and end in different clusters) is generally given by [30]:

$$\varepsilon_{i,j} = \begin{cases} 0 & if\ \ i = j \\ \dfrac{E_{i,j}}{2 \times N_i \times N_j} & if\ \ i \neq j \end{cases} \quad (5)$$

On the other hand, cohesion (or intra-connectivity) μ_i of cluster i with N_i nodes and μ_i intra-edges (edges that start and end in the same cluster) is in general calculated as [30]:

$$\mu_i = \frac{M_i}{N_i^2} \quad (6)$$

We can encapsulate both cohesion and coupling in a fitness function and try to optimize it with PSO such that cohesion is maximized and coupling is minimized. Fitness function or quality of clustered architecture is expressed in form of MQ [31, 37]. We use a modified version of MQ, called *TurboMQ* defined as [35]:

$$TurboMQ = \sum_{i=1}^{k} CF_i \quad (7)$$

Where, k is the number of clusters, and Component Factor (CF) is defined as:

$$CF_i = \begin{cases} 0 & if\ \ \mu_i = 0 \\ \dfrac{\mu_i}{\mu_i + \frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & otherwise \end{cases} \quad (8)$$

Where, CF is based on cohesion and coupling for cluster [31].

TurboMQ can assume arbitrarily large values depending on cohesion and coupling. In order to contain the MQ values between 0 and 1, we modify the MQ formula as [1]:

$$N - TurboMQ = MQ / k \quad (9)$$

A potential solution to the clustering problem is represented with the help of a particle. A swarm of N particles is represented by $\{P_1, P_2, \ldots, P_k\}$. Reference [19] suggests a way of representing discrete problems with PSO particle structure. In the present work, using similar approach, each particle is represented as a $k \times m$ matrix where each entry may be either 0 or 1. For a

specific particle P_i if $P_i[r, c]=1$ it means that class c is assigned to cluster r, and opposite meaning is inferred if $P_i[r, c]=0$. At any step t during PSO iteration the entries in P_i represent the current position of P_i and are denoted by P_i^t. An example of a particle as shown in Table 1 where k=3 and m=5. According to the above-explained interpretation, classes 1 and 3 belong to cluster 2, class 4 belongs to cluster 1 and classes 2 and 5 belong to cluster 3.

Table 1. Particle representation.

	Class 1	Class 2	Class 3	Class 4	Class 5
Cluster 1	0	0	0	1	0
Cluster 2	1	0	1	0	0
Cluster 3	0	1	0	0	1

Associated with each particle P_i is a P_i^{best} indicating the best position experienced by the particle in terms of fitness function. Globally, the best of all P_i^{best} is maintained as G^{best}.

Similarly, the velocity matrix of particle P_i is a $k \times m$ matrix with entries in the range $[-V_{max}, V_{max}]$.

Based on above problem formulation, the PSO algorithm for software clustering is shown in Algorithm 1. Termination criterion of the algorithm is either to have no improvement in fitness value for consecutive T (where T is an integer) iterations or having fitness value reach 1.

Algorithm 1: PSO algorithm for software clustering

```
For each particle:
    Initialize particle velocity
    Initialize Particle position
    Set lbest to 0
End
  Set gbest to 0
  Do:
    For each particle:
        a. Calculate fitness value using Equation 7.
        b. If fitness value is better than lbest fitness value
           Set lbest to current position of particle.
    End
      Set gbest to lbest having maximum fitness value
    For each particle:
        a. Calculate velocity using Equation 1.
        b. Calculate position using Equation 2.
    End
  While maximum iterations criteria is not attained or
  fitness value ≠ 1
```

5. Experimentation, Results, and Discussion

5.1. Test Environment

The system has been implemented using Visual C++. Entire process for architecture recovery as shown in Figure 2. We use a custom source code analyzer [1] to extract inter-module relationships. This paper describes only the working of clustering algorithm in the figure. For testing, we used 2.26 GHz Intel Core i3 machine with 3 GB RAM running Windows 7 Professional 32-bit operating system.

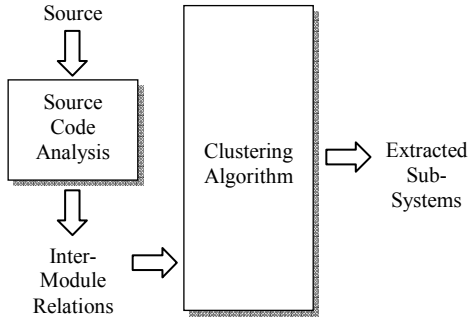


Figure 2. Architecture recovery process.

5.2. Benchmark Algorithm

We compare our results with another state-of-the-art meta-heuristic for software clustering, the GA. We have selected GA for comparison because just like PSO it is also a population based search in which an entire population of potential solutions iterates repeatedly to converge to an optimal solution. While PSO is inspired from flocking behavior of various species, GA gets its inspiration from reproductive behavior and "survival of the fittest". GA represents a potential solution as a chromosome which evolves through successive generations to empower the desirable characteristics and weaken the undesirable ones. Unlike PSO, GA involves additional functions that mimic natural selection process; e.g., mutation and crossover. Another difference lies in the information sharing mechanisms of the two approaches; while particles in PSO share only lbest and gbest, chromosomes in GA share their contents completely with each other [20]. One advantage of PSO over GA is that PSO has memory of previous best known solutions whereas GA does not have such memory.

5.3. Performance Measures and Test Systems

The two meta-heuristics are compared in terms of computational effort and quality of solution. We define computational effort as the number of iterations required or the time required to converge. Quality of solution is defined as the fitness value at convergence, with higher fitness value meaning better quality solution.

Three test systems are used for experimentation:

- *Test System 1. Statistical Analysis Visualization Tool* (*SAVT*): This application helps statistical analysts in analysis and visualization of statistical data. It provides complete support of user interface for input and visualize along with the saving and loading data files.
- *Test System 2. Print Language Converter* (*PLC*): This application is a sub-system of a large software system. It provides conversion support from intermediate data structures to a well-known printer language.

- *Test System 3. Power Economic Dispatch System* (*PEDS*): This system is related to electrical power systems. It solves economic power dispatch problem using conventional and evolutionary computing techniques [1]. It uses MFC document viewer architecture.

Table 2. Test systems.

Test System	Alias	No. of Classes
1	SAVT	97
2	PLC	69
3	PEDS	41

5.4. Experimental Setup

For each test system, we applied PSO and GA and performed comparison between decompositions of each algorithm. The comparisons are based on fitness values, rate of convergence and computation time. Fitness values are in the range *0* ~ 1. Zero specifies the worst decomposition while 1 indicates best decomposition [1].

Table 3 gives the parameter settings for the experiments. These settings were derived empirically by running several experiments and selecting those settings that gave the best results.

Table 3. Parameter settings.

	PSO	GA
Swarm/ Population Size	10 × No. of classes (Swarm Size)	10 × No. of classes (Population Size)
No. of Clusters	Average of three expert decompositions	Average of three expert decompositions
Termination Criteria	MQ=1 or No Improvement in MQ for the last 1000 iterations	MQ=1 or No Improvement in MQ for the last 1000 iterations
Algorithm-Specific Parameters	c1=2 c2=2	Selection: Tournament Crossover Probability: 100% Mutation Probability: 2%

5.5. Results and Discussion

For each test system, total 10 trials are taken (each continuing till termination criteria are satisfied), 5 trials using GA and 5 trials using PSO. On the basis of these readings, comparative analysis is performed. Time is shown in "HH:mm:ss" format.

- *Test System 1* (*SAVT*): Results of both approaches for SAVT are shown in Table 4. There is 9.89% improvement in fitness value.

Table 4. SAVT results.

Trial No.	PSO		GA	
	Fitness Value	Time	Fitness Value	Time
1	0.505032	1:04:17	0.474990	2:05:43
2	0.505032	1:03:11	0.434460	2:05:40
3	0.505032	1:03:14	0.461820	2:04:46
4	0.506467	1:04:56	0.435390	2:05:29
5	0.505032	1:04:51	0.492440	2:04:37
Average	0.505319	1:03:54 or 3834 sec.	0.459820	2:05:24 or 7524 sec.

PSO quickly converged to maximum fitness value within first 500 iterations. Analyzing fitness values Figure 3 reveals that results of PSO are more stable and consistent because of the minimum variations in the fitness values.

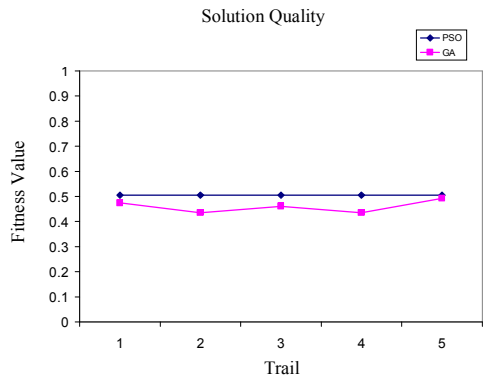


Figure 3. PSO algorithm for software clustering comparison of solution quality (fitness values) for SAVT.

Comparison of computational effort of the two approaches as shown in Figure 4. There is 49.04% decrease in computational time for PSO in comparison to GA. The figure also reveals consistency of computational effort in PSO.

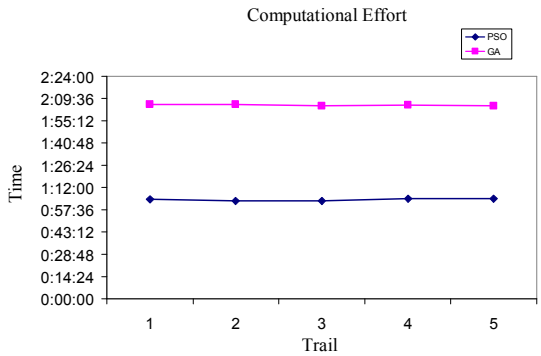


Figure 4. Comparison of computational effort for SAVT.

- *Test System 2* (*PLC*): Results of both approaches for PLC are shown in Table 5.

Table 5. PLC results.

Trial No.	PSO		GA	
	Fitness Value	Time	Fitness Value	Time
1	0.688900	0:10:43	0.640430	0:42:21
2	0.688900	0:10:01	0.652140	0:43:24
3	0.688900	0:09:28	0.640430	0:43:22
4	0.688900	0:09:58	0.652140	0:44:42
5	0.688900	0:09:47	0.652140	0:45:39
Average	0.688900	0:09:59 or 599 sec.	0.647456	0:43:54 or 2634 sec.

There is 6.4% improvement in fitness value. PSO quickly converged to maximum fitness value within first 500 iterations whereas GA consumed much more time. Analyzing fitness values Figure 5 reveals that results of PSO are more stable and consistent because of the minimum variations in the fitness values.

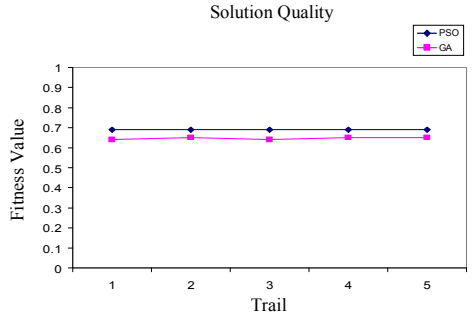


Figure 5. Comparison of solution quality (fitness values) for PLC.

Comparison of computational effort of the two approaches is shown in Figure 6. There is 77.26% decrease in computational time of PSO as compared to GA.

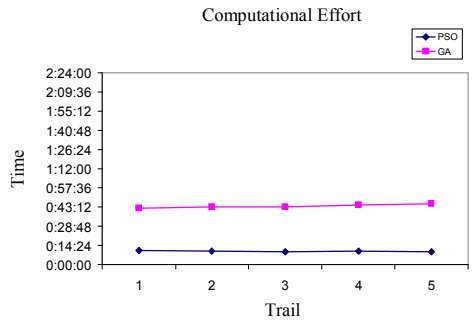


Figure 6. Comparison of computational effort for PLC.

- *Test System 3.* (*PEDS*): Results of both approaches for PEDS are shown in Table 6. There is 0.76% improvement in fitness value using PSO. PSO quickly converged to maximum fitness value within first 500 iterations.

Table 6: PEDS results.

Trial No.	PSO		GA	
	Fitness Value	Time	Fitness Value	Time
1	0.805339	0:02:15	0.773470	0:09:55
2	0.805339	0:02:12	0.770880	0:09:50
3	0.824819	0:03:24	0.858480	0:09:45
4	0.824819	0:03:00	0.773470	0:09:50
5	0.805339	0:02:11	0.858480	0:09:39
Average	0.813131	0:02:36 or 156 sec.	.806956	0:09:48 or 588 sec.

Analyzing fitness values Figure 7 reveals that results of PSO are more stable and consistent because of the minimum variations in the fitness values.

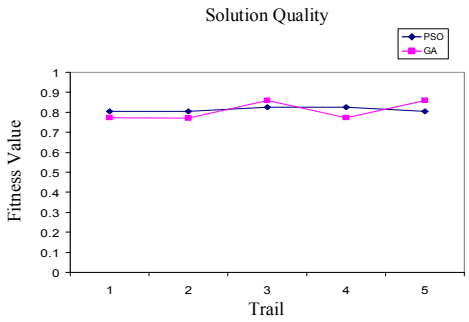


Figure 7. Comparison of solution quality (fitness values) for PEDS.

Comparison of computational effort of the two approaches is shown in Figure 8. There is 73.47% decrease in computational time for PSO in comparison to GA. The figure also reveals consistency of computational effort in PSO.

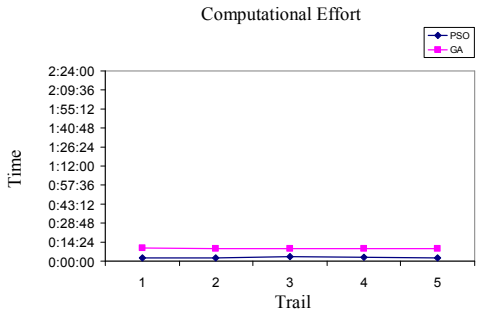


Figure 8. Comparison of computational effort for PEDS.

A comprehensive view of the performance of the two algorithms is shown in Figure 9. Looking at the almost horizontal graph of solution quality ratio, it can be concluded that both approaches provide almost the same solution quality. In terms of computational effort, PSO is less expensive approach and computational savings vary from test system to test system. This is because PSO does not involve genetic operators like selection or mutation etc., Still, because of its provision for remembering the best known solutions it can give comparable solution quality to GA.

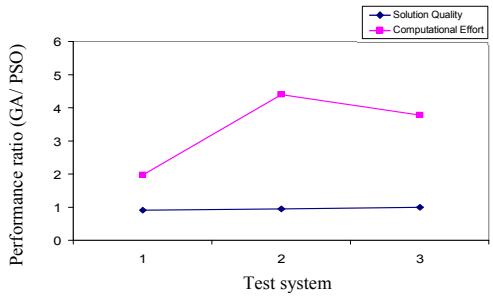


Figure 9. Comparison of ratios of mean computational effort and mean quality of solution.

Another measure performance an algorithm is the consistency of results. In this regard a comparison of standard deviation of fitness values of PSO and GA is shown in Figure 10. It is evident that PSO results are much more consistent (lower standard deviation) than GA.

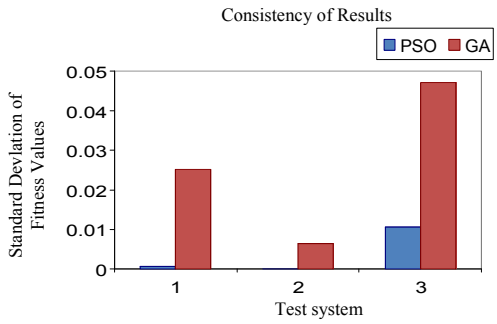


Figure 10. Comparison of consistency of results.

6. Conclusions and Future Work

The paper presents the use of PSO algorithm to solve the software clustering problem. We tested the PSO algorithm on three different software test systems and compared the results with GA. Simulation results show that the PSO approach has quick convergence. It requires small computational time and low computational effort compared to GA. Further work is required to find the optimum values of PSO parameters because a good parameter setting is generally very important factor in the performance of the algorithm.

Although, PSO is less expensive in terms of computational effort as compared to GA, in future we want to work to stabilize the computational effort by identifying the influential attributes of test system.

References

[1] Abbasi A., "Application of Appropriate Machine Learning Techniques for Automatic Modularization of Software Systems," *M-Phil Thesis*, Quaid-i-Azam University Islamabad, Pakistan, 2008.

[2] Alam S., Dobbie G., Riddle P., and Naeem A., "Particle Swarm Optimization Based Hierarchical Agglomerative Clustering," *in Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Toronto, Canada, vol. 2, pp. 64-68, 2010.

[3] Andritsos P. and Tzerpos V., "Software Clustering Based on Information Loss Minimization," *in Proceedings of the 10th Working Conference on Reverse Engineering*, Victoria, Canada, pp. 334-344, 2003.

[4] Backer F. and Hubert L., "A Graph Theoretic Approach to Goodness-Of-Fit in Complete-Link Hierarchical Clustering," *Journal of the American Statistical Association*, vol. 71, no. 356, pp. 870-878, 1976.

[5] Berry M., Dumais S., and O'Brien G., "Using Linear Algebra for Intelligent Information Retrieval," *Society for Industrial and Applied Mathematics Review*, vol. 37, no. 4, pp. 573-595, 1995.

[6] Bonissone P., "Soft Computing and Meta Heuristics: Using Knowledge and Reasoning to Control Search and Vice-versa," available at: http://homepages.rpi.edu/~bonisp/NASA-course/MH. pdf, last visited 2004.

[7] Chatzigeorgiou A., Tsantalis N., and Stephanides G., "Application of Graph Theory to OO Software Engineering," *in Proceedings of International Workshop on Interdisciplinary Software Engineering Research*, Shanghai, China pp. 29-36, 2006.

[8] Clerc M., *Particle Swarm Optimization*, ISTE Ltd, available at: www.onlinelibrary.wiley.com/ doi/10.1002/9780470612163.fmatter/pdf, last visited 2010.

[9] Cui X., Potok E., and Palathingal P., "Document Clustering using Particle Swarm Optimization," *in Proceedings of IEEE Swarm Intelligence Symposium*, California, USA, pp. 185-191, 2005.

[10] Donath W. and Hoffman A., "Lower Bounds for the Partitioning of Graphs," *IBM Journal of Research and Development*, vol. 17, no. 5, pp. 420-425, 1973.

[11] Fiedler M., "A Property of Eigenvectors of Non-Negative Symmetric Matrices and Its Applications to Graph Theory," *Czechoslovak Mathematical Journal*, vol. 25, no. 4, pp. 619-633, 1975.

[12] Frakes W. and Nejmeh B., "Software Reuse Through Information Retrieval," *ACM SIGIR Forum*, vol. 21, no. 1-2, pp. 30-36, 1987.

[13] Girard J., Koschke R., and Schied G., "Comparison of Abstract Data Type and Abstract State Encapsulation Detection Techniques for Architectural Understanding," *in Proceedings of the Working Conference on Reverse Engineering*, Amsterdam, Netherlands, pp. 66-75, 1997.

[14] Gower J. and Ross G., "Minimum Spanning Trees and Single-Linkage Cluster Analysis," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 18, no. 1, pp. 54-64, 1969.

[15] Harman M. and Hierons R., "A New Representation and Crossover Operator for Search-Based Optimization of Software Modularization," *in Proceedings of the Genetic and Evolutionary Computation Conference*, New York, USA, pp. 1359-1366, 2002.

[16] Harman M. and Jones B., "Search-Based Software Engineering," *Information and Software Technology Journal*, vol. 43, no. 14, pp. 833-839, 2001.

[17] Hassan R., Cohanim B., Weck O., and Venter G., "A Comparison of Particle Swarm Optimization and the Genetic Algorithm," *in Proceedings of the 46th American Institute of Aeronautics and Astronautics/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Texas, USA, pp. 1-13, 2005.

[18] IEEE Std 610.12-1990., "IEEE Standard Glossary of Software Engineering Terminology," available at: www.dis.unal.edu.co/~icasta/ggs/ Documentos/Normas/610-12-1990.pdf, last visited 1990.

[19] Izakian H., Ladani B., Abraham A., and Snasel V., "A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling," *International Journal of Innovative Computing, Information and Control*, vol. 6, no. 9, pp. 1-15 2010.

[20] Jones K., "Comparison of Genetic Algorithm and Particle Swarm Optimization," *in Proceedings of International Conference on Computer Systems and Technologies-CompSysTech*, Varna, Bulgaria, 2005.

[21] Kazem A. and Lotfi S., "An Evolutionary Approach for Partitioning Weighted Module Dependency Graphs," *in Proceedings of the Innovations in Information Technology*, Dubai, UAE, pp. 252-256, 2007.

[22] Kazman R., Woods S., and Jeromy Carriere S., "Requirements for Integrating Software Architecture and Reengineering Models: CORUM II," *in Proceedings of the 5th Working Conference on Reverse Engineering*, Hawii, USA, pp. 154-163, 1998.

[23] Kennedy J. and Eberhart R., "Particle Swarm Optimization," *in Proceedings of the IEEE International Conference on Neural Networks*, Perth, Western Australia, vol. 4, pp. 1942-1948, 1995.

[24] Khan B., Sohail S., and Javed M., "Evolution Strategy Based Automated Software Clustering Approach," *in Proceedings of the Advanced Software Engineering and its Applications*, China, pp. 27-34, 2008.

[25] Khanesar M., Teshnehlab M., and Shoorehdeli M., "A Novel Binary Particle Swarm Optimization," *in Proceedings of the 15th Mediterranean Conference on Control and Automation*, Athens, Greece, pp. 1-6, 2007.

[26] King B., "Step-Wise Clustering Procedures," *Journal of the American Statistical Association*, vol. 62, no. 317, pp. 86-101, 1967.

[27] Kuhn A., Ducasse S., and Gˆırba T., "Semantic Clustering: Identifying Topics in Source Code," *Journal on Information Systems and Technologies*, vol. 49, no. 3, pp. 230-243, 2006.

[28] Li C. and Yang S., "A Clustering Particle Swarm Optimizer for Dynamic Optimization," *IEEE Congress on Evolutionary Computation*, Trondheim, Norway, pp. 439-446, 2009.

[29] Maletic J. and Marcus A., "Using Latent Semantic Analysis to Identify Similarities in Source Code to Support Program Understanding," *in Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence*, Vancouver, Canada, pp. 46-53, 2000.

[30] Mancoridis S., Mitchell B., Chen Y., and Gansner E., "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures," *in Proceedings of the IEEE International Conference on Software Maintenance*, Hawaii, USA, pp. 50-59, 1999.

[31] Mancoridis S., Mitchell B., Rorres C., Chen Y., and Gansner E., "Using Automatic Clustering to Produce High-Level System Organizations of Source Code," *in Proceedings of the Workshop on Program Comprehension*, Ischia, Italy, pp. 45-52, 1998.

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

[33] Mirkin B., *Mathematical Classification and Clustering*, Kluwer Academic Publishers, London, UK, 1996.

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

[35] Mitchell B., "A Heuristic Search Approach to Solving the Software Clustering Problem," *PhD Thesis*, Drexel University, Philadelphia, PA, 2002.

[36] Mitchell B., "Clustering Software Systems to Identify Subsystem Structures," available at: https://www.cs.drexel.edu/~bmitchell/research/survey.pdf, last visited, 2006.

[37] Mitchell B., Mancoridis S., and Traverso M., "Search Based Reverse Engineering," *in Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy, pp 431-438, 2002.

[38] Montes C. and Carver D., "A Visual Representation Model for Software Subsystem Decomposition," *in Proceedings of the 5th Working Conference on Reverse Engineering*, Hawaii, USA, pp 231-240, 1998.

[39] Saeed M., Maqbool O., Babri H., Hassan S., and Sarwar S., "Software Clustering Techniques and the Use of Combined Algorithm," *in Proceedings of the 7th European Conference*, Germany, pp. 301-306, 2003.

[40] Sartipi K. and Kontogiannis K., "A Graph Pattern Matching Approach to Software Architecture Recovery," *in Proceedings of the IEEE International Conference on Software Maintenance*, USA, pp. 408-419. 2001.

[41] Shokoufandeh A., Mancoridis S., Denton T., and Maycock M., "Spectral and Meta-heuristic algorithms for Software Clustering," available at:https://www.cs.drexel.edu/~spiros/papers/JSS05.pdf, last visited 2004.

[42] Sneath P. and Sokal R., *Numerical Taxonomy*, Freeman, London, UK, 1973.

[43] Sulaiman S., Idris B., and Sahibuddin S., "Enhancing Cognitive Aspects of Software Visualization using DocLike Modularized Graph," *the International Arab Journal of Information Technology*, vol. 2, no. 1, pp. 1-9, 2005.

[44] Tzerpos V. and Holt R., "ACDC: An Algorithm for Comprehension-Driven Clustering," *in*

Proceedings of the 7th Working Conference on Reverse Engineering, Brisbane, Australia, 2000.

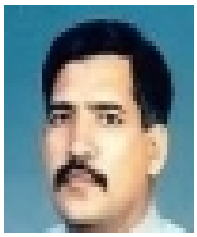
[45] Wang Y., Liu P., Guo H., Li H., and Chen X., "Improved Hierarchical Clustering Algorithm for Software Architecture Recovery," *International Conference on Intelligent Computing and Cognitive Informatics*, UK, pp. 247-250, 2010.

[46] Weise T., "Global Optimization Algorithms-*Theory and Application*", available at: http://www.it-weise.de/, last visited 2009.

Ibrar Hussain received MS degree in software engineering from National University of Sciences and Technology, Pakistan in 2012. He has been in the software industry since 2005. He is currently working as a project manager in Japan based company, Komatsu Pakistan Soft (Pvt.) Ltd. His research areas include software code refactoring, machine learning, reverse engineering, and software architecture recovery.

Aasia Khanum did PhD in Software Engineering from National University of Sciences and Technology, Pakistan. Currently, she is assistant professor at Department of Computer Science, Forman Christian Collage Lahore, Pakistan. She has more than 40 research publications. Her current research interests include application of intelligent techniques to information retrieval, pattern recognition, and software engineering.

Abdul Qudus Abbasi is an Assistant Professor at Quaid-e-Azam University (QAU), Islamabad, Pakistan. He did MPhil in Information Technology from QAU in 2008. His current research interests include information retrieval, soft computing, and search based software engineering.



Muhammad Younus Javed has PhD in computer engineering from UK. Currently, he is serving as Dean at College of Electrical and Mechanical Engineering, National University of Sciences and Technology (CEME-NUST), Pakistan. He has more than 150 publications in international conferences and journals. His current research interests are in areas of algorithms and software engineering.