Fitness for Solving SMCP Using Evolutionary Algorithm

Neetu Gupta¹, Ajay Rana¹ and Sumit Gupta²

¹Amity Institute of Information Technology, Amity University Uttar Pradesh, Noida, India
²Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, India

E-mail: neetugupta78@gmail.com

Abstract. Search based software engineering is a paradigm with prime focus to apply search techniques and meta-heuristics for solving various software engineering open NP hard problems. Evolutionary meta-heuristics techniques are already proven to provide optimized solutions to other software engineering problems like automated software test data generation, project estimation, class responsibility assignment to name few. Software module clustering is such an open problem of software engineering that cannot be solved in definite manner. For a given module dependency graph of software system, there exists large number of possible partitions. Identifying a good partition to cluster all modules of software system is an exhaustive search that cannot be carried out in finite manner. With search based techniques applying evolutionary algorithms, an optimized solution can be identified with evaluating goodness of a given partitions. Identifying a fitness function that can direct the search towards optimal solution is very critical. This paper discusses various types of fitness applicable to solve software module clustering problem.

1. Introduction

Search based software engineering (SBSE) is a well-established field that aims to solve important problems in domain of software engineering those are infinite in nature with meta-heuristic optimization techniques. One of such well known open problems in software engineering is the problem of module clustering of a software application popularly referred as Software Module Clustering Problem (SMCP). The concept of module clustering is to group modules (or elements) of an application to find a non-intersecting set of subsystems (i.e., cluster) based upon relationships among them in such a manner that it helps to design a software system that is easy to implement, fix bugs, maintain and enhance [1][2][3][4][5].

A software system comprises of multiple components (or modules) those interact with each other to accomplish a task. This interaction brings dependency among these components of the system. With these dependencies, a small change in a single component sometimes may call for more ripple and cascading changes in multiple dependent components if dependencies and clustering of modules is not proper and optimized one. In last two decades, software module clustering is taken up as a search problem [6][7][8][9]. Consider a given software system S with N different modules, a huge set of possible partitions is possible that group these N modules into different clusters. To evaluate the goodness of each partition not practically possible in a definite time. Search based partitioning is an approach that emphases on the automation of partitioning process of a software system based on dependencies among its modules by applying different optimization heuristic techniques. Different meta-heuristic evolutionary algorithms like Genetic Algorithm (GA) [1][2][6][7][8][9][10], Particle...
Swarm Optimization (PSO) [11][12][13][14], Harmony Search (HS) [15], Hill Climbing (HC) [4][16] etc. are applied in past research works for solving SMCP. The results of these different works are claimed to be promising and guide the path ahead in this area.

Two critical and main factors in implementing evolutionary based search based technique for any problem are – 1) to identify a suitable heuristic approach and its encoding with respect to the problem to be solved; 2) to identify a suitable fitness function that guides the search towards a better solution. A fitness function guides the search of optimal solution that is applied to evaluate the fitness value of available solutions. On the basis calculated fitness, superiority of one solution is claimed over others solutions. Hence fitness function is a tool that guides the search process. Therefore, it is critical to identify and define an ideal fitness function in context of the problem to be solved. It is observable that for every new problem that is to be solved, a suitable fitness function must be identified. In context of SMCP, various fitness functions have been devised and evaluated by researchers. It includes single objective fitness as suggested by Doval and later many multi-objective fitness functions.

The study conducted in this research work mainly emphasises on different fitness functions suitable to solve software module clustering problem. The attempt is to elaborate and simplify the understanding of SMCP related fitness. The work is categorized in different sections. Sections 2 and 3 discuss about work as available in literature by different researchers in past to solve the SMCP and contextual background respectively. Sections 4 and 5 are dedicated to the detailed and tabular discussion on different types of fitness functions and their implementation.

2. Related Work

This section elaborates detailed study of previous work done by researchers in the field of SMCP is discussed. This explorative study shows that SMCP has been a problem of interest to many researchers in the field of evolutionary algorithm and search based software engineering.

S. Mancoridis, B. S. Mitchell, C. Rorres [2][3] considered module clustering of a complex software as an optimization problem and applied genetic algorithm in search of an optimal partition solution. This work defines a high-quality partition as the one with less inter-cluster and high intra-cluster dependency. Consequently, defines Modularization Quality (MQ) as fitness for optimization and is calculated as inter-connectivity and intra-connectivity trade-off. A tool called as Bunch [4] is developed for software structure maintenance and enhances Bunch by integrating designer’s knowledge. Bunch applies two search algorithms - GA and Hill Climbing with MQ as objective function. It defines MQ as (write this in paper in which it is proposed first) [5]. Implementation of multiple hill climb [16], with using parallel computing to execute 23 climbs simultaneously is done. It applied fitness as MQ that is sum of modularization factors. It claims that MHC guides the search to higher peaks in subsequent executions.

A [1] multi-objective approach based solution is introduced with proposed Equal-size cluster approach (ECA) and Maximizing cluster approach (MCA). For unweighted graphs, Bunch HC technique showed better results whereas MCA and ECA outperformed with weighted graphs. EAC outperformed at cost of more computational.

The work of Praditwong [6] suggests an implementation of Group Genetic Algorithm (GGA) to partition software units into groups. Chromosomes are of variable length where each chromosome represents a cluster of lists consisting of sets of modules. A comparison is of average MQ values id performed among GGA and GNE over 17 different weighted and non-weighted problems. For weighted graphs, GGA outperformed than GNE in t-Test. Further [7]. A work implements Hyper-heuristic GA (HypGA) to achieve an equilibrium among exploration and exploitation of search space. A set of twelve low level heuristics were used using variations in selection, crossover and mutation. And reinforcement principle is implemented to increase/decrease the weight of low-level heuristics. It claimed that MHygGA is better than two-Archive algorithm in terms of MQ and computational cost.

Chhabra [17] based his work on eight directional relations between classes with weights to calculate connection strength. This work implements Non-dominated Sorting Genetic Algorithms (NSGA-II) and claimed that it worked better than SA and HC of Bunch in terms of modularization merit factor and modularization quality.
Amarjeet [11] devised PSOMC a PSO based technique for module clustering. PSOMC is redesigned with velocity and particle position calculation in context of modular clustering. It claimed the outperformance of PSOMC over SA, HC and GGA based solutions in terms of Modularization Quality, coupling, cohesion and non-extreme distribution (NED). The author [18] explores structural similarity (MS) as a new measure to evaluate modularization quality and compares it with MQ devised by Macrodnis. This work compares MS and MQ using HA, GA and Multi Agent EA [19] on basis of software design rules Freverse and Fdirection.

The work in [20] explores hierarchical clustering techniques for architecture recovery and modularization of software systems. Test subjects include four applications - CVA, BASH, Mosaic, and Xfig. Abadeen [21] works on minimizing the cyclic connectivity among packages to automatically optimize an existing software modularization by moving classes among classes. This approach is based on Simulated Annealing (SA) [22] a neighborhood search technique. A dependency might be method calls, class access, or class inheritance. It measures modularization quality in terms of Cohesion Quality, Coupling Quality, Package Cyclic Dependency Quality and Package Cyclic Connections Quality. Harmony Based Re-modularization Algorithm (HGRA) and its four variations [23] are implemented. Fitness is based on package coupling, package cohesion, Package Count Index and Package Size Index [24][25][26]. They experimented to compare HBRA with GA, HC, SA, ABC and DE in terms of MQ, NED values to claim its outperformance.

A multi-objective [27] search based re-modularization approach is explored using NSGA-III [28]. A set of re-modularization operations and their Similarity Score is defined prior. A vector of re-modularization operations is used to represent a solution. A random solution length is generated between lower and upper bounds. And then an \(i^{th}\) operation (feasible) is chosen from pool of operations. Fitness is computed using 7 objectives usefulness. The work focus on providing solution to improve metrics such as classes per package (NCP), packages count (NP), number of inter-edges (NIE), number of intra-edges (NAE), manual precision (MP), cohesion, coupling, improve semantic coherence and minimize code changes.

Harman [8] improves the approach of Macrodnis with a unique representation (lookup table use) and crossover operator to avoid multiple representations of same modularization to limit search space. It applies modified crossover operator-based GA i.e., GA+ and compares its results with standard GA (with single point crossover). Random approach and HC.

Bavota and his team [29] suggest R3 a refactoring tool to enhance software modularization quality. This approach suggests moving class operations into suitable packages to perform refactoring operations and avoids whole new re-modularization. R3 approach is capable to reduce coupling among modules by 10% to 30%. The information-flow-based coupling (ICP) [29] is a metric to capture the dependencies between classes present in software system. For a class \(C\), it measures the in and out of information flow through sum of number of parameters passed during invocation of a method. The information-flow-based coupling between a pair of classes \(C_i\) and \(C_j\) is measured as the number of method invocations in the class \(C_i\) to methods in the class \(C_j\), weighted by the number of parameters of the invoked methods.

A multi-objective scheme is devised [17] with different eight types of relations defined. Different weights are assigned to these relations. This work applies NSGA-II [24] heuristic and compares it with HC and SA in terms of improved percentage of MQ. The work of Harman [30] compares effectiveness of two objective functions – MQ and clustering fitness EVM [31]. The test data consists of six real programs, three randomly generated module dependency graphs (MDG) and 3 perfect MDGs. The research of Lutz [32] is towards evolutionary approach to search better hierarchical module decompositions (HMD) of a system using a variation of GA in which individual mates with a fittest neighbor. System S is represented as HMD with leaves as basic nodes and internal nodes are modules of S. For GA implementation, one HMD is an individual represented as a tree. With the help of concatenation definition of HMDs and module trees, defines new crossover and mutation operator in context on HMDs. The principle of minimum description length [33] is used to choose best HMD.

A variation of GA is implemented for graph partitioning with a new Knowledge-Based Non-Uniform Crossover (KNUX) operator [34]. A comparison of KNUX-GA with Index-Based Partition and Recursive Spectral Bisection [9] applies GGA [6] with a modified crossover and three mutations –
split and join, elimination and Adoption. Initial population is generated based on Kruskal’s algorithm. For fitness, uses cohesion, coupling, complexity, cycles and bottlenecks. It tries to evaluate these five fitness parameters on a case study of JHotDraw application.

3. Background
The main aim of software module clustering is to group software modules as disjoint set of sub-systems (i.e., cluster). It is a search based optimization problem with a reason that a software system can be divided into an exhaustive set such partitions but aim to find a best partition that produces high quality software.

3.1. Module Dependency Graph
Search based approach to solve SMCP models software application as a module dependency graph (MDG). The MDG represents the software system as a graph $G = (N, E)$, where N is the set of nodes representing modules and E is the set of edges corresponds to dependencies among them. The MDG can be modelled in both ways - unweighted and weighted. An unweighted graph has an edge $E_i$ among two modules $M_a$ and $M_b$ having minimum one dependency. Similarly, weighted graph assigns a weight $w$ to an edge $E_{iw}$ between two modules such that the value of $w$ represents the strength of dependencies among these two modules.

For a MDG $G = (N, E)$, the goal of SMCP is to divide the set $N$ (i.e., modules) into a partition set $P = \{C_1, C_2, C_3, C_m\}$ having m clusters such that $C_i \cap C_j = \Phi$, and $\cup C_i = N$. Figure1 represents a sample MDG of a software system with six modules showing their dependencies. Figures 2 and 3 demonstrate two possible partitions of MDG in figure 1.

![Figure 1. An MDG Example](image1)

![Figure 2. Partition1 of MDG](image2)

![Figure 3. Partition2 of MDG](image3)

3.2. Search Based Approach
Search based approach is applied as explained in algorithm 1. A step wise approach to apply any evolutionary algorithm is described in algorithm 2.
**Algorithm 1: Search Based Approach**

1. Choose a meta-heuristic algorithm $A$
2. Decide an encoding or presentation in context to SMCP
3. Decide a fitness function $F(x)$ suitable to SMCP
4. Apply the algorithm $A$ to search an optimal solution

**Algorithm 2: Evolutionary Algorithm**

1. Initialize the population
2. Evaluate the fitness
3. While (condition)
   4. $t = t + 1$
   5. Select $P(t)$ from $P(t-1)$
   6. Apply recombination and mutation operators on $P(t)$
   7. Evaluate the fitness of $P(t)$
   8. end while

4. **Single Objective fitness for module clustering**

Single objective based on fitness focuses to optimize only one criterion relevant to the problem. A single fitness function $f$ is defined that is to be either minimized or maximized. The single objective fitness further can be determined based on single factor or some combination of multiple factors. For any available $k$th clustering solution, a fitness possessing single objective is represented as equation (1)

$$f(k) = \min \text{ or } \max (f(k) \forall k)$$

Many features are identified those are primarily considered to define quality of a software application. Modularization of software is one of these features. Few considerable factors affecting modularization (or clustering) are coupling, cohesion, granularity, no of modules, no of clusters etc. Given a set of possible modularization solutions, fitness is calculated to choose better solution from available ones. Many different ways to calculate fitness of a modularization solution are suggested and implemented to resolve the SMCP. Single objective based module fitness can is calculated in based on – i) single factor ii) multiple factors. These are listed in Table 1.

4.1. **Single factor based single objective fitness**

Single factor based function tries to optimize only one suitable factor responsible for module quality to search an optimal clustering solution.

- Module quality (MQ) is a basic and simple single objective fitness [3][4], Dove [2]. It is calculated as difference of average intra-connectivity and interconnectivity. It defines MQ as in eq (2). Here, $A_i$ is intra-connectivity within $i$th module and $E_{ij}$ is the inter-connectivity between any two modules. These are calculated as eq 3) and 4) respectively.

$$MQ = \left\{ \begin{array}{ll}
\frac{1}{k} \sum_{i=1}^{k} A_i - \frac{1}{k(k-1)} \sum_{i,j=1}^{k} E_{i,j}, & \text{if } k_i > 1 \\
A_1, & \text{if } k = 1
\end{array} \right. \quad (2)$$

$$A_i = \frac{\mu_i}{N_i^2} \quad (3)$$

$$E_{i,j} = \frac{\beta_{ij}}{2+N_i+N_j} \quad \text{for all } i \neq j \quad (4)$$

Here $\mu_i, N_i$ denotes actual and maximum possible intra edges respectively in $i$th cluster. And the value of $\beta_{ij}$ is inter edges between $i$th and $j$th cluster.

- MQ based fitness is also defined as the sum of modularization factors (MF) [1][16][6][7][35]. A value of MF is calculated for each cluster in given clustering solution. For a given clustering solution of software with $n$ possible clusters, the values of MQ and MF are defined as in equations 5 and 6. Here, values of $i$ and $j$ are weights of intra-edges and inter edges for the $k$th cluster.
\[(MF_i) = \begin{cases} 0, & \text{if } i = 0 \\ \frac{i}{i+2^i}, & \text{if } i > 0 \end{cases} \] \hspace{1cm} (5)

\[MQ = \sum_{k=1}^{n} MF_k \hspace{1cm} (6)\]

### 4.2. Multiple factor based fitness

A single objective fitness is also based on multiple quality factors related to modularization of software application. Usually fitness is an aggregate or multiplicative function of different factors contributing to fitness of a modularization solution.

- Fitness based on three components – cohesion, coupling and granularity is computed and applied [8]. **Cohesion fitness** is calculated as sum of cohesion of each module in `S`. **Coupling fitness** is the inverse of coupling unfitness that is ratio of inter-module associations by total possible associations. **Granularity** is calculated using actual granularity and target granularity of a modularization. All three components are given equal weight while computing overall fitness of the system as shown in table 1.

#### Table 1. Single Objective Fitness Functions for SMCP

| Year | Year | Fitness Function |
|------|------|------------------|
| 1 [3] 1998 | MQ = 1 \[ \sum_{i=1}^{k} A_i - \frac{1}{k(k-1)} \sum_{i,j=1}^{k} E_{ij}, k_i > 1 \] | Intra-connectivity within module inter-connectivity between two modules |
| 2 [6] 2011 | MQ = \[ \sum_{k=1}^{n} MF_k \] | Sum of Modularization Factors (MF) for each kth cluster MF for kth cluster is ratio of weights of its intra-edges and total weights of edges |
| 3 [8] 2002 | fitness = A function of cohesion \(C(S)\), coupling unfitness \(CU(S)\), granularity(G) with all three given equal weightage | C \((S) = \sum C(m)/k \) \(CU(S) = \sum \text{inter-module associations} / \) \(\sum \text{associations in } S \) G depends on actual and target granularities |
| 4 [11] 2018 | \(f(x) = \left( \frac{MD_{\text{intra}}}{MD_{\text{intra}} + MD_{\text{inter}}} \right)^a \left( \frac{1}{\text{No of Clusters}} \right)^b + \left( \frac{\text{No of clusters}}{\text{No of clusters}} \right)^c \) | An aggregate function using four quality criteria – Intra-module dependencies (maximized), Inter-module dependencies (minimized), No of clusters (minimized), No of modules per cluster (minimized) Values of a, b, c lies between [0,1] |
| 5 [15] 2017 | \(f(x) = (P_{\text{coup}})^a + (P_{\text{cohe}})^b + (PCI)^c + (PSI)^d \) | A multiplicative function of four factors - Package Coupling, Package Cohesion, Package Count Index, Package Size Index |
Multi Objective fitness for module clustering

A multi objective fitness focuses to optimize more than one objective. To solve SMCP using multi objective fitness, a set of objective functions are identified. Let’s consider there are k functions representing fitness as $f_1, f_2, f_3, \ldots, f_k$. A set of more than one non-dominated modularizations are identified from the available list of possible solutions. Given two modularizations $M_a$ and $M_b$, the solution $M_a$ is said to dominate $M_b$ if $M_a$ exceeds in terms of at least one of k fitness(s) and not lesser in other fitness(s). This is represented in equations 7.

$$f_i(M_a) \geq f_i(M_b) \text{ for all } i \in (1, \ldots, k) \quad (7)$$

5.1. Multi Objective SMCP fitness

- Derived from design objectives of software application, one such fitness of clustering solution is calculated as an aggregate function of three parts. This fitness is derived from four quality criteria [11]. These criteria are – total of intra-module dependency; total of inter-module dependency; total clusters (NC); count modules per cluster. The first part of fitness is calculated as total intra-module dependency divided by total possible dependencies. Second and third parts are inverse of NC and inverse of modules per cluster respectively as these are to be minimized.

- Based on the coupling and cohesion, package based fitness is defined as a multiplicative function four important design factors of software [15][24][26]. These factors include Package Cohesion (PCohe), Package Coupling (P_coup), Package Count Index (PCI) and Package Size Index (PSI). The value of $P_{Cohes}$ is sum of package coupling measurement (PCS) of all clusters that is calculated as coupling strength between all pairs of classes within a package is divided by the total coupling strength in that package.

- An another calculation of fitness of module F(M) is suggested based on introducing two factors - Dependency Quality (DQ) and Connection Quality (CQ) as in eq. 10 [21]. A set of different metrics are well-defined to calculate the values of DQ and CQ for a module M (cluster) - Class Outgoing Dependencies (Cout), Class Incoming Dependencies (Cin), Package External Dependencies (PExt,D), Package Internal Dependencies (Pint,D), Package Out Connections (POut.Con), Package Incoming Connections (PInc.Con), POut.Cyc.D, PInc.Cyc.D.

- The use of Evaluation Metric Function (EVM) [31] defines [30] fitness that rewards high intra-module dependency. It defines a mechanism to evaluate score of a cluster. This score value is incremented for each intra-module relationship and decremented for absent possible relationship. Though not directly but this fitness in a way tries to minimize coupling.

5. Multi Objective fitness for module clustering

An average of two factors for a module M

$$DQ(M)\text{ is weighted average of Common Closure Quality and Acyclic Dependencies Quality}$$

$$CQ(M)\text{ is weighted average of Common Reuse Quality and Acyclic Connections Quality}$$

$$f(M) = \frac{(DQ(M) + CQ(M))}{2}$$

The fitness is sum of scores of clusters i.e. $h(M_i)$ for $i = 1$ to $m$

$$EVM(M) = \sum h(M_i)$$

The fitness is sum of scores of clusters i.e. $h(M_i)$. Here $c_{ij}$ refer to the jth element of the ith cluster of C.

$$L(c_{xy}, c_{pq}) = \begin{cases} 1, & \text{if relation b/w } c_{xy} \text{ to } c_{pq} \\ -1, & \text{otherwise} \end{cases}$$

$$h(m_i) = \left\{ \begin{array}{ll}
\sum_{\alpha=1}^{k_i-1} \sum_{\beta=\alpha+1}^{k_i} L(m_{i\alpha}, m_{i\beta}), & k_i > 1 \\
0, & \text{otherwise}
\end{array} \right.$$
Two approaches based on different set of objectives are Maximum Cluster Approach (MCA) and Equal-size Cluster Approach (ECA) [1]. The MCA focuses on good clustering attributes like cohesion (maximize), coupling (minimize), cluster count (maximize), modular quality (MQ) (maximize), and isolated cluster count (minimize). The main focus of ECA is to generate equal size clusters to ensure the possibility of no isolated or large size clusters. Its objectives are similar to MCA other than cluster count. ECA includes the difference between minimum and maximum number of modules in cluster (minimize).

A suitable multi modal based fitness is computed by adding up multiple weighted individual objective fitness values [9]. Overall fitness $fit(S)$ is derived from five different individual fitness functions – cohesion, coupling, complexity, cycles and bottlenecks. The cohesion function estimates overall cohesion of application as the sum of cohesion of all subsystem divided by number of subsystems. Similarly, the coupling function is sum of coupling values of each subsystem where coupling of an individual subsystem $S_i$ is the number of dependency edges outside $S_i$ divided by total edges in $S$. The complexity function uses McCabe’s control flow complexity due to its fuzzy shape. The value of cycle function is sum of size of highly connected components. And the bottleneck fitness of a subsystem $S_i$ is calculated by dividing minimum of in/out degree of $S_i$ by highest in/out degree.

A MOSMCP approach uses multi-objective fitness is based on two objectives - MQ and Average Reversed Edge Number (AREN) between clusters. The metric AREN is based on the unidirectional performance among clusters. It is based on the fact that a system with consistent direction of call relationship between methods is good as it results in a system with independent function block. [13]. Table 2 lists multiple objective fitness functions.

### Table 2. Multiple Objective Fitness Functions for SMCP

| Year | Fitness                                                                 | Dependency Factor / Explanation |
|------|-------------------------------------------------------------------------|---------------------------------|
| 1    | [1] 2011                                                                | MQ, Cohesion, Coupling          |
|      | Total of intra-edges in clusters (Maximize)                            |                                 |
|      | Total of inter-edges in clusters (Minimize)                            |                                 |
|      | Cluster Count (Maximize)                                                |                                 |
|      | Modularization Quality (Maximize)                                       |                                 |
|      | Isolated clusters Count (Minimize)                                     |                                 |
| 2    | [13] 2018                                                               |                                 |
|      | $f_{dir} = \frac{1}{m+1} \sum_{i=1}^{m} \min (L(v_i, v_i), L(\bar{v_i}, v))$ | $L(v_i, v_j)$ is count of inter-edges among clusters -i and j |
|      | $L(v_i, \bar{v_i})$ is count of inter-edges among cluster-i and the other clusters |

### 6. Conclusion

The quality of software depends on modularization of different components into subsystems (clusters) based on dependencies among them. The well modularised software reduces the effort spent in the process of bug fixing and maintenance. The problem of software module clustering is to group modules into clusters that may contribute significantly to software quality. Search based solutions to solve SMCP are widely applied and tested. Different evolutionary algorithms like GA, PSO, HC, Harmony etc. are tested and result in good solutions. The effective working of evolutionary algorithms primarily depends on fitness function to direct the search towards an optimal solution. In context of SMCP, various fitness functions are identified and evaluated. Discovery of related fitness function can be based on one objective or may be derived using few multi-objective functions. This study
conducted in presented work emphases on various fitness functions suitable to solve software module clustering problem.

7. References

[1] Praditwong, K., Harman, M. and Yao, X., 2010. Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, 37(2), pp.264-282.

[2] Doval, D., Mancoridis, S. and Mitchell, B.S., 1999, September. Automatic clustering of software systems using a genetic algorithm. In *STEP'99. Proceedings Ninth International Workshop Software Technology and Engineering Practice* (pp. 73-81). IEEE.

[3] Mancoridis, S., Mitchell, B.S., Rorres, C., Chen, Y. and Gansner, E.R., 1998, June. Using automatic clustering to produce high-level system organizations of source code. In *Proceedings. 6th International Workshop on Program Comprehension. IWPC’98* (Cat. No. 98TB100242) (pp. 45-52). IEEE.

[4] Mancoridis, S., Mitchell, B.S., Chen, Y. and Gansner, E.R., 1999, August. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).*Software Maintenance for Business Change*(Cat. No. 99CB36360) (pp. 50-59). IEEE.

[5] Mitchell, B.S. and Mancoridis, S., 2002. A heuristic search approach to solving the software clustering problem. Philadelphia, PA, USA: Drexel University.

[6] Praditwong, K., 2011, May. Solving software module clustering problem by evolutionary algorithms. In 2011 *Eighth International Joint Conference on Computer Science and Software Engineering* (JCSSE) (pp. 154-159). IEEE.

[7] Kumari, A.C., Srinivas, K. and Gupta, M.P., 2013, February. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (pp. 1045-1051).

[8] Tasgin, M., Herdagdelen, A. and Bingol, H., 2007. Community detection in complex networks using genetic algorithms. arXiv preprint arXiv:0711.0491.

[9] Jiaze, S. and Beilei, L., 2018. Density PSO-based software module clustering algorithm. *The Journal of China Universities of Posts and Telecommunications*, 25(04), pp.38-47.

[10] Sun, J., Xu, Y. and Wang, S., 2018. PSO with Reverse Edge for Multi-Objective Software Module Clustering. *International Journal of Performability Engineering*, 14(10), pp.2423-2431.

[11] Praditwong, K., Harman, M. and Yao, X., 2010. Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, 37(2), pp.264-282.
[18] Huang, J. and Liu, J., 2016. A similarity-based modularization quality measure for software module clustering problems. *Information Sciences*, 342, pp.96-110.

[19] Liu, J., Zhong, W. and Jiao, L., 2006. A multiagent evolutionary algorithm for constraint satisfaction problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(1), pp.54-73.

[20] Maqbool, O. and Babri, H., 2007. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11), pp.759-780.

[21] Abdeen, H., Ducasse, S., Sahraoui, H. and Alloui, I., 2009, October. Automatic package coupling and cycle minimization. In *2009 16th Working Conference on Reverse Engineering* (pp. 103-112). IEEE.

[22] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P., 1983. Optimization by simulated annealing. *Science*, 220(4598), pp.671-680.

[23] Chidamber, S.R. and Kemerer, C.F., 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6), pp.476-493.

[24] Balasubramanian NV.  Object-oriented metrics. In: *Software Engineering Conference*; 1996:30–34.

[25] Gupta, V. and Chhabra, J.K., 2009. Package coupling measurement in object-oriented software. *Journal of computer science and technology*, 24(2), pp.273-283.

[26] Prajapati, A. and Chhabra, J.K., 2018. A particle swarm optimization-based heuristic for software module clustering problem. *Arabian Journal for Science and Engineering*, 43(12), pp.7083-7094.

[27] "K. Deb and H. Jain. 2014. An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, Part I: Solving problems with box constraints. *IEEE Trans. Evol. Comput.*, 18, 4, 577.

[28] Bavota, G., Gethers, M., Oliveto, R., Poshvyvanyk, D. and Lucia, A.D., 2014. Improving software modularization via automated analysis of latent topics and dependencies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(1), pp.1-33.

[29] Harman, M., Swift, S. and Mahdavi, K., 2005, June. An empirical study of the robustness of two module clustering fitness functions. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (pp. 1029-1036).

[30] Tucker, A., Swift, S. and Liu, X., 2001. Variable grouping in multivariate time series via correlation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 31(2), pp.235-245.

[31] Lutz, R., 2001. Evolving good hierarchical decompositions of complex systems. *Journal of systems architecture*, 47(7), pp.613-634.

[32] Rissanen, J., 1978. Modeling by shortest data description. *Automatica*, 14(5), pp.465-471.

[33] Maini, H., Mehrtra, K., Mohan, C. and Ranka, S., 1994, November. Genetic algorithms for graph partitioning and incremental graph partitioning. In *Supercomputing’94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing* (pp. 449-457). IEEE.

[34] Clarke, J., Dolado, J.J., Harman, M., Hierons, R., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M. and Shepperd, M., 2003. Reformulating software engineering as a search problem. *IEEE Proceedings-software*, 150(3), pp.161-175.