Search based Software Modularization Using Evolution Algorithm

Divya Sharma[1*], Shikha Lohchab[2]

Abstract

To comprehend a Software system, Software modularization strategies are used. The goal of modularization is to break down a software system into meaningful and intelligible sub-systems from its source-code (modules). Because software classification modularization is an NP-hard task, evolutionary methods produce better modularization quality rather than avaricious algorithms. All available transformative techniques for software modularization only take into account structural aspects reliant on programming language syntax. Because most computer languages lack a mechanism for extracting structural characteristics, they cannot be modularized.

A novel heuristic is proposed in this work. with several objectives that accomplishes both in orderto lead optimization algorithms towards a proper decomposition of software systemsautomatically, structural (e.g.; calling dependence and in-heritance dependency) and non- structural (e.g.; semantics in code comments and identifier names) aspects are used. It is analyzed using 3 optimization plans, viz; global-based-search, combining global and local search, and Estimation of Distribution (EoD) to upgrade it. According to outcomes on Mozilla Firefox, suggested optimization algorithm based on EoD and the newly developed MOF function exceed those so it use structural-based objective functions in finding more understandable modules, as well as guiding the optimization procedure. In the lack of a unique concept, structure, the originaldesign can be identified by using the source code of the disturbed software. Effective software maintenance depends on the concept of software system.

One of most powerful techniques in software clustering is the ability to divide enormous Software systems into workable subsystems with modules of identical characteristics, thereby reducing thecomplexity of the system. A meta-heuristic optimization imperialist competitive system has emerged algorithm, genetic algorithm, and their combination is examined for software clustering in this paper. When it comes to value, of clustering, the number of epochs required for convergence, and the standard unconventionality found at the end of repeated application of these algorithms, it appears that recursive application is the most effective for achieving the best performance.

Key Words: Search-based Software Engineering, Software Module Clustering, Genetic Algorithm, Evolutionary Algorithm, Software Maintenance.

DOI Number: 10.14704/nq.2022.20.5.NQ22240 **NeuroQuantology 2022; 20(5):822-831**

Introduction

Boehm [1] claims upkeep expenses might up to 10 folds the cost of early improvement, while Parikh and Zvegintzov [2] estimate because upkeep is expensive half of all computing and human resources. Software developers are increasingly concerned with maintaining systems because of their increasing complexity and scale. Unless and if programme dearth sufficient documentation of the modifications made throughout system evolution, the situation becomes dire. As a result, it will be harder to maintain the software in the course of time. Replacing such Software with fresh software is also not a viable option because it not ensures security and full utility of system.

The approaches as re-verse-engineering and re - engineering evolved to manage such software systems.

Corresponding author: Divya Sharma
Address: [1*]Research Scholar, Department of Computer Science, G.D Goenka University, Gurugram, India; [2]Assistant Professor, Department of Computer Science, G.D Goenka University, Gurugram, India.
E-mail: [1*]divya.07sharma@gmail.com; [2]shikha.lohchab@gdgu.org
Relevant conflicts of interest/financial disclosures: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.
Received: 28 March 2022 **Accepted:** 30 April 2022

Because the source code is the sole precise copy of re-verse engineering community is working on approaches in order to retrieve excellent dimensional data directly from source code, making it available to software creators and maintainers. Like techniques are tendency to focusing on recuperation of design via software clustering; programme slicing, and source code analysis, among other things. The practice of organizing software modules forming bunches in a quiet way about modules, heavily dependent are merged together is known as software module clustering [12, 15, 19].

Clustering aids in a better understanding of the system and facilitates future maintenance. The connections between the modules provide the basis for this breakdown. Module Dependency Graphs (MDGs) are commonly used to depict these interactions; Modules are represented as nodes, with edges connecting the nodes defining their interrelationships. As a result, clustering may be viewed together with network partitioning problem, Doval & others. [5] took on the problem of identifying great clustering as they addressed the challenge by searching the huge solution space of all potential MDG partitions with a Genetic Algorithm. Technique's utility is identified by its usage on medium-sized system.

Harman & others [6] presented a novel cross-over operator to enhance the production, confinementof building blocks, and discovered this crossover technique is more suited to genetic techniques than regular cross-over. [24]

Dependency on Artifacts the input to modularization approaches is a graph (ADG). An artefact can be any part of a programme, such as a class, method, function, or file, depending on the kind of software. The reliance on or resemblance between comments and other objects is also known as artefact dependency.

The greatest assembly/similarity should be found between artefacts from the same module, whereas the least should be found between artefacts from other modules [15]. As a result, the majority of software modularization methods rely on cohesion and coupling requirements. Cohesion is the degree of intermodular connectivity, whereas coupling is the degree of coherenceof a module.

As a result, the majority of modularization solutions rely on criteria for cohesion and coupling. The degree of connection [14, 17] between modules is referred to as coupling, whereas cohesion refers to the coherence of a module. Candela et al. (2016)'s paper "Employing Coherence and Coupling for known to be NP-hard. As a result, typical optimization strategies are unable to tackle it effectively. As a result, search-basedtechniques to software module clustering are developed. Despite the fact that they cannot guarantee optimum answers, in a reasonable amount of time, produce near-optimal answers [21, 23].

Mancoridis & others. [3] Proposed the first search-based solution to software module clustering, which used hill climbing as the principal search strategy. Their research intended to automaticallyretrieve a system's modular structure from its source code.

They tested various systems and assessed the outcomes based on system designers' input, which they considered to be satisfactory. They then created Bunch [4], a clustering programme that automatically develops system decay by considering clustering as an optimization issue. Following that, numerous academics used a variety of optimization strategies to experiment with automated module clustering.

Software Re-modularization: Will That Be enough?" is a fantastic piece of work. These criteria alone were found to be insufficient, and further criteria were necessary. It's worth noting that they didn't argue on which criteria were necessary.

This study addresses the questions of the study mentioned:

1) What aspects in the source code might lead to modularization that resembles that of the domain expert?
2) Is it possible to enhance the worth of modularization by applying multi-objective fitness function entails structural and non-structural aspects into account?
3) Can the estimate of distribution-based approach provide superior to other search-based modularization methods in terms of quality?

To accomplish modularization, most modularization strategies rely on structural features like cohesion and coupling. We will look at how inheritance links, as well as non-structural features like identifier names and comments, affect modularization noteworthy of a system, as well as cohesion and coupling.

Search-based and evolutionary strategies, viz; genetic algorithm, more productive over others' methods since the issue of modularization is NP-hard (Isazadeh. 2017).

There are two types of search methods in search-based algorithms: Search both globally and locally. Based on a global search techniques analyze search space in order to find a suitable solution. Particular techniques feature an operator that allows you to search the search space for new locations. Genetic Algorithm, a widely used search algorithm uses global search method.

To ameliorate merit of present solutions, local search algorithms make advantage of the ability to exploit. These algorithms begin solution and move constantly over search space, spreading local modifications from solution still in use to neighboring solution (as yet undetermined).

One of most frequent local search techniques is hill-climbing. Currently, we use genetic algorithmalone as well as two distinct combinations with the hill-climbing algorithm to evaluate a multi- objective function that has been suggested; in the hopes of integrating the benefits of these two search methodologies. The performance of the genetic algorithm is influenced by the mutation and crossover operators, as well as their likelihood [23, 25, 26].

The primary flaw in a genetic algorithm is that it does not keep track of the building blocks and key arrangements that emerge throughout the process of evolution. After using crossover and mutation operators, the building blocks can be destroyed. This will delay down the process of identifying the best option. The architectural pieces are maintained Furthermore, in leading optimization techniques for identifying more understandable modules, the proposed multi- function with an objective, which integrates structural and non-structural features, beats structure-based goal functions.

Data also demonstrate that the supply strategy estimate technique is better to the other approaches to software modularization. [12]

Related Work

If the software architecture is thoroughly documented, the conceivability of a system evolves efficiently & effectively. Unfortunately, if programmers do not keep up with the changes, this type of documentation may become obsolete [13]. Furthermore, the worth of software architectures has a significant impact on the success of the programme generated. [14]

using techniques based on Estimation of Distribution (EoD). The EoD-based algorithms establish (basic) population and build a probabilistic model taken away it to solve a problem.

The following generation is built using the probabilistic model that has been produced, with the goal of preserving valuable building blocks. One of the issues with genetic algorithms is calculating the rates of mutation and crossover.

To build a new population, EoD-based algorithms don't employ notions viz; mutations and crossings. As a result, deleting the good replies from the responses will be minimized to the greatest extent feasible.

This work suggested a novel multi-objective fitness function for modularization entails into account both structural characteristics (viz; calling and inheritance dependencies) and latent semantic in non-structural elements (identifier names and comments).

Machine learning techniques were used to extract non-structural features of their underlying meaning. Three optimization procedures have been adjusted to increase performance of this objective function: global-based-search, combining global and local search, & distribution assessment. 3, 6, and 17] Each algorithm has its own set of characteristics.

In a large-scale application like Mozilla Firefox, the results suggest that modularization based on structural features [17] beats modularization based on non-structural factors.

Various reverse (decrypting) engineering methods have been proven to be effective in extracting architecture of systems in this setting [15-16].

Clustering methods are used in the literature, the bulk of reverse engineering tools are suggested.[17]. ACDC [18] is a widely used software clustering tool. This is a sequence software clustering approach recovers sub-systems through incremental clustering. Technique entails establishing a framework that supports patterns across subsystems, and then placing each anew introduced resource in sub-system that appears to be the most appropriate.

Harman & others. [16] suggested a completely unique encoding and crossover operator to enhance the GA technique utilized in BUNCH.

Based on both publications and citations, Figure 1 below depicts the top software modularization scholars over the previous decade.

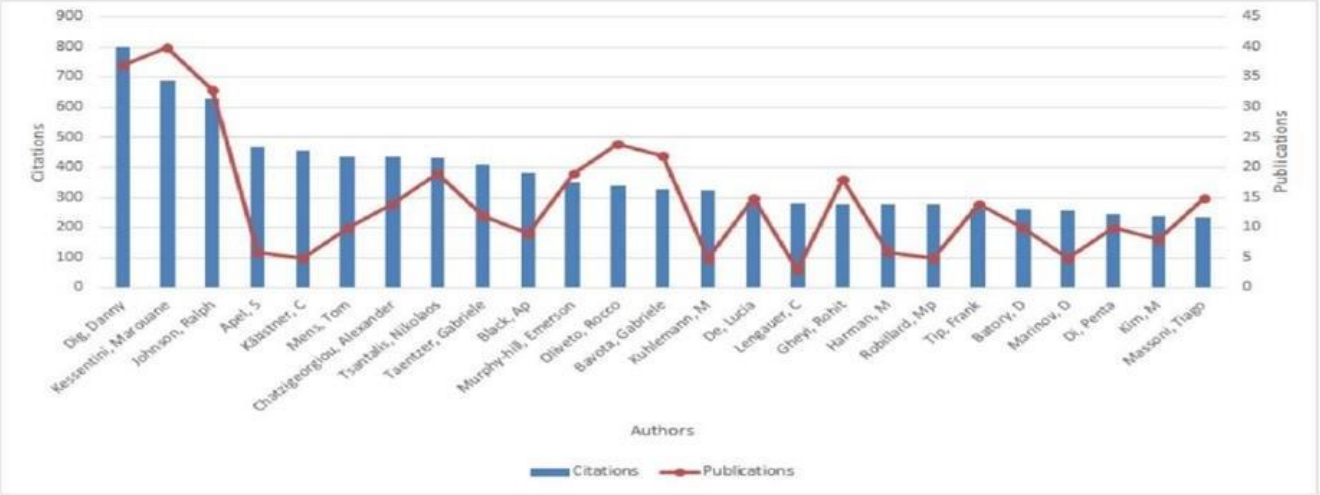

Figure 1. Data of citation and publications of different authors

Bunch [19], another popular tool, tries to spot a decomposition that maximizes the TurboMQ quality score, predicated on strong cohesion and low coupling, a well-known tool that uses search-based strategies viz; hill climbing and GA, among others, during work, we compare the outcomes of our method to those of this tool.

By enabling just one representation per modularization, this method changed the way GA was used previously. Although it outperforms the conventional crossover operator, it usually gets trapped at local optima.

The authors of a new paper [2 0] exploited MQ-based co-operative clustering for software clustering. Because the dimension of the problem grows larger, this strategy degrades performance. Particle Swarm Optimization (PSO) being employed for unravel matter of softwareclustering [21], using MQ because the optimization goal.

Data clustering techniques and graph clustering procedures are divided into two classes within the literature. (Abualigah 2017a, b, 2018, Alswaitti 2018, Liu & others. 2017; Alswaitti & others. 2018; Liu et al. 2017; Alswaitti & others. 2018; Liu & others. 2017; Alswaitti & others 2018; Liu& others. 2017; Alswaitti & others 2018; Liu & others).

The subsequent are some samples of how data clustering methods are used: Graph-based clustering techniques could also be utilized in a spread of applications, including social network analysis, protein complex identification, picture segmentation, and software clustering. the subsequent are some graph clustering methods: Chang & others., 2017; Wen & others.,., 2017; Wen & others., 2017). There are two sorts of software modularization (or software clustering) strategies: hierarchical and non-hierarchical (including search-based approaches and greedy algorithms).

- The objectives employed in E.C.A.
- Total number of edges in all modules should be as high as possible.
- Overall number of edges in all modules should be lessened.
- There'd be more modules enhanced.
- MQ must be amplified.
- Quantity of mono modules be lessened are the objectives employed in ECA.

The M.C.A goals are equal to the E.C.A goals, with the exception as opposed to very last one, "the distinction among the most and minimal variety of artefacts module need to be reduced" [13, 16, 19, 24].

These multi-goal algorithms entirely remember structural houses and forget about non-structural ones. The algorithms for hierarchical modularization approaches, nearby seek algorithms, and international seek algorithms are proven in Tables 1 and 2. Most modularization approaches, in step with the literature, use seek-primarily based totally algorithms to modularize a program system.

In addition, maximum search-primarily based totally approaches, in step with Table 1 and 2, use structural topographies like calling relationships to gain modularization and forget about the have an impact on of structural traits like inheritance relationships and non-structural factors like remarks and identifier names.

Table 1. Modularization techniques based on local search

Approaches	Taking into account non-structural characteristics	Multi-objective single-objective	Disadvantages
Algorithm for hill climbing (Mitchell and Mancoridis 2008)	4	Single- objective	Trapping in optimization algorithm would not work if there is no call graph.
SA. algorithm (Isazadeh & others.2017)	4	Single- objective	Trapping in optimization algorithm would not work if there is no call graph.
Climbing method including severalhills (Mahdavi 2005)	4	Single- objective	Trapping in optimization algorithmwould not work if there is no call graph.
Learning- Automata (Izad Khan &others 2016)	5	Single- objective	Trapping in optimization algorithm would not work if there is no call graph.
Huge neighborhood search applied (Monc¸ores & others 2018)	4	Single- objective	Trapping in optimization algorithm would not work if there is no call graph.

Table 2. Hierarchical algorithms

Approaches	Examiningthe non- structural features	Dis-advantages
Single-link-age (Maqbool & Babri 2007)	7	Arbitrary decision, greedy
Complete-linkage (Maqbool & Babri 2007)	7	Arbitrary decision, greedy
Average-linkage (Maqbool & Babri 2007)	7	Arbitrary decision, greedy
CA algorithm (Saeed 2003)	7	Arbitrary decision, greedy
WCA algorithm (Maqbool & Babri 2007)	7	Arbitrary decision

Software Clustering

Clustering of Software program modules changed into previously studied as a single-goal seek problem, however it has lately been reformed as a multi-goal seek problem. In this part, we will undergo each version.

Software module clustering is a way about grouping program modules into awesome clusters to enhance programme shape and make renovation easier. With nodes representing modules and edges indicating interactions among modules, the MDG is utilized because the seek algorithm's input. MDGs may be weighted or unweighted [21,23,25].

An un-weighted M.D.G. represents connection amongst modules with aid of using the presence or absence of an edge, while a weighted MDG depicts the connection amongst modules with the aid of using an edge's presence or absence.

To map modules to clusters, an easy array is hired because the illustration strategy. The module is represented with the aid of using the array index, and the array content material identifies the cluster in which the module is delivered.

Modularization Quality (MQ) metric is the number one goal in any single-goal formula of a program module clustering.

The foremost intention of a software program module clustering is to create excellent program clusters with most terms of inter (cohesion) and inter-connectivity at a minimum (coupling). Termsof inter - connectivity refers back to number of connections among modules in a unmarried cluster.

Because the often-established modules are clustered in to equal cluster, excessive intra-connectivity suggests a success clustering. The number of links between modules in diverse clusters is measured with the aid of using inter-connectivity. Because the clusters are extensively impartial of 1 another, low interconnectivity suggests wonderful clustering.

The Modularization Quantity (MQ) [22] investigates the trade-off among intra- and inter- connectivity, profitable cohesiveness at same time as penalizing coupling.

A. Software Clustering with a Variety of Goals

Since improving cohesion while lowering coupling are two conflicting goals, Praditwong & others [10] reinterpreted module clustering as a multi-objective issue and created 2 multi-objective approaches:
The Maximizing Cluster Approach (M.C.A.) and the Equal-size Cluster Approach (E.C.A.). [26].

- Maximizing the total of all bunch intra-edges
- Minimizing of total of inter-edges of all bunches
- Maximum of amount of bunches
- Maximization of MQ
- Minimization of number of sequestered bunches are objectives stated under M.C.A. technique

The main purpose of the M.C.A. method is to generate good partitions with high cohesion and lowcoupling while increasing quantity of clusters and decreasing the numeral of solitary clusters.
E.C.A. method fosters the creation of clusters of identical size and the breakdown of the system into clusters of roughly comparable size.

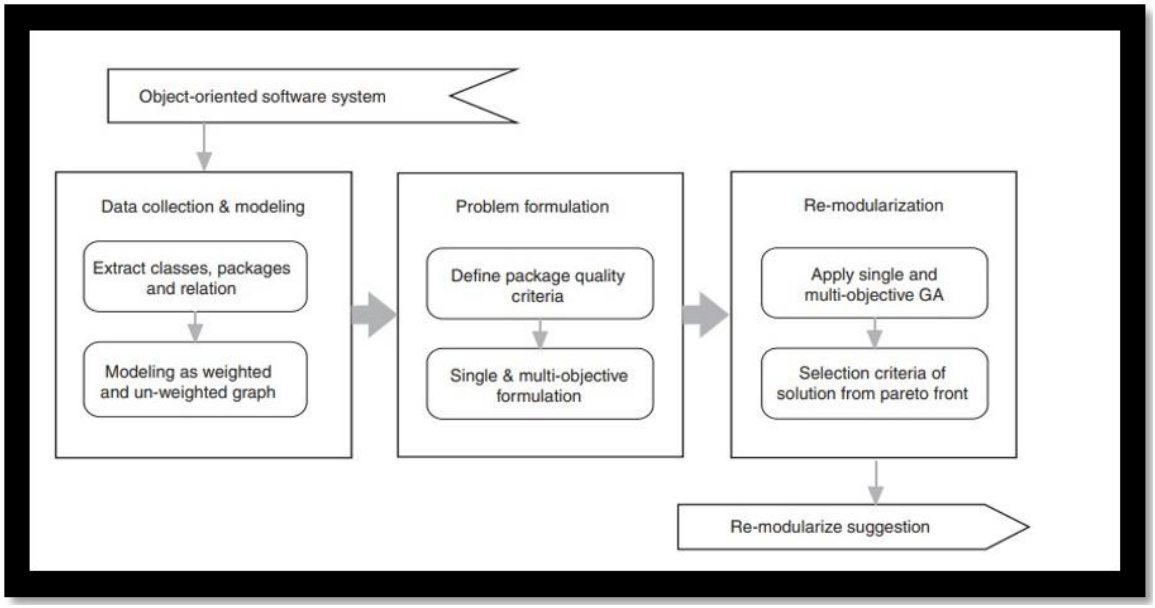

Figure 2. System Structure Representation of Software Remodularization

Because that is a multi-goal optimization, the idea of optimality shifts due to the fact the answers are trade-offs or excellent meddles the various goals [12].
To generate those trade-off answers, an idea as Edgeworth-Pareto optimality is adopted, states strategy to a multi-goal hassle is Pareto best if there no different possible answer that improves one criterion without degrading as a minimum an extra criterion on the identical time.
When this belief is used, it almost usually yields a fixed of non-ruled answers called the Pareto top-rated set. The complete waft is depicted in Figure 2 as a gadget structure.

Proposed Approach and Results

Our suggested strategy is outlined in this section. The typical order of low-level strategies discovered is E.A./selection/cross-over/mutation. Selection implies to procedure through parents are chosen for goal of having children.
Selection methods utilized are rand-to-best. Both parents are selected at random by population in rand, whereas one parent is chosen at random from the population in contrast to other is an elite parent in rand-to-best (the best one).
Three sorts of cross-over operators are adopted to make off-spring. Uniform crossover is the 1st operator, which creates children by picking a gene from each parent at random. A hybridcrossover1 is the second operator (hc1), which described as a mixture of uniform and single-pointcrossover.
After choosing a crossover point, the children are the cross-over point is attained by duplicating each gene from the parent to the off-spring until the cross-over moment arrives.

After that, the remaining genes are arbitrarily chosen from either parent. 3rd operator is a hybrid crossover2 (hc2), has been wrapped by a 2-point cross-over identical cross-over hybridization.

Offspring is generated by mixing the DNA of the parents until the first crossing point is achieved.The genes are then chosen at random from any one of the parents until the 2nd cross-over point. The remaining genes are then passed on to the child from the parent. There are two forms of mutations: copy and exchange.

Two genes are chosen at random in the first mutation operator, and the second gene is cloned intothe first. Two genes are picked at random and their positions are switched in the second mutation operator.

The suggested hyper-heuristic consists of two stages. Choosing which mutation to employ is the first step (copy or ex-change). This is completed at random, involving both groups having an equal probability. C. R. is arbitrary integer chosen by a homogenous distribution on the unit interval to pick an E.A. model with an equal chance of copy or exchange mutation.

Both set at the start of specified hyper-heuristics at a low-level notation are indicated by values ofhb and he. In the 2nd phase, low-level heuristic is chosen from underneath defined group.

To choose a specific model of EA, this stage employs a reinforcement learning technique with flexible weights and a roulette wheel. Algorithm 1 contains the pseudo code for the MHypEA.

Evolutionary Algorithm with Multi-Objective Hyper-heuristics (Algorithm 1) (MHypEA)

1: Build population of parents
2: Assess the fitness of the parental population
3: Choose low-level heuristic that is based on collection apparatus (not as a termination condition)
4: On the parent population, use the low-level heuristic that you've chosen to generate the descendants' population.
5: Assess the population of offspring
6: Combine the populations of the parents and offspring
7: Sort the combined population using non-dominated sorting and choose persons from the most promising fronts for the next version
8: come to an end.

Conclusively, if the MoJo value is low and the MoJoFM value is large, the resulting modularization is comparable to domain expert modularization (ground-truth modularization).

The parameter settings for the trials are shown in Table 2, with n denoting the number of futuristics. In genetic algorithms, notional of generations typically higher comparatively to population size.

As a consequence, we used technique in the recommended algorithms and treated value linearly as a co-efficient of the several of artifacts.

Adopting recommended methodologies, twenty different combinations of mutation and cross-over probability were tested on the Db folder to determine likelihood of mutation and crossover operators.

Figure 3 and Figure 4 depicts the call dependency graph evaluated from the modularization/re-modularization data set folder extracted through visual studio code.

Figure 3. Call Dependency Graph Extraction

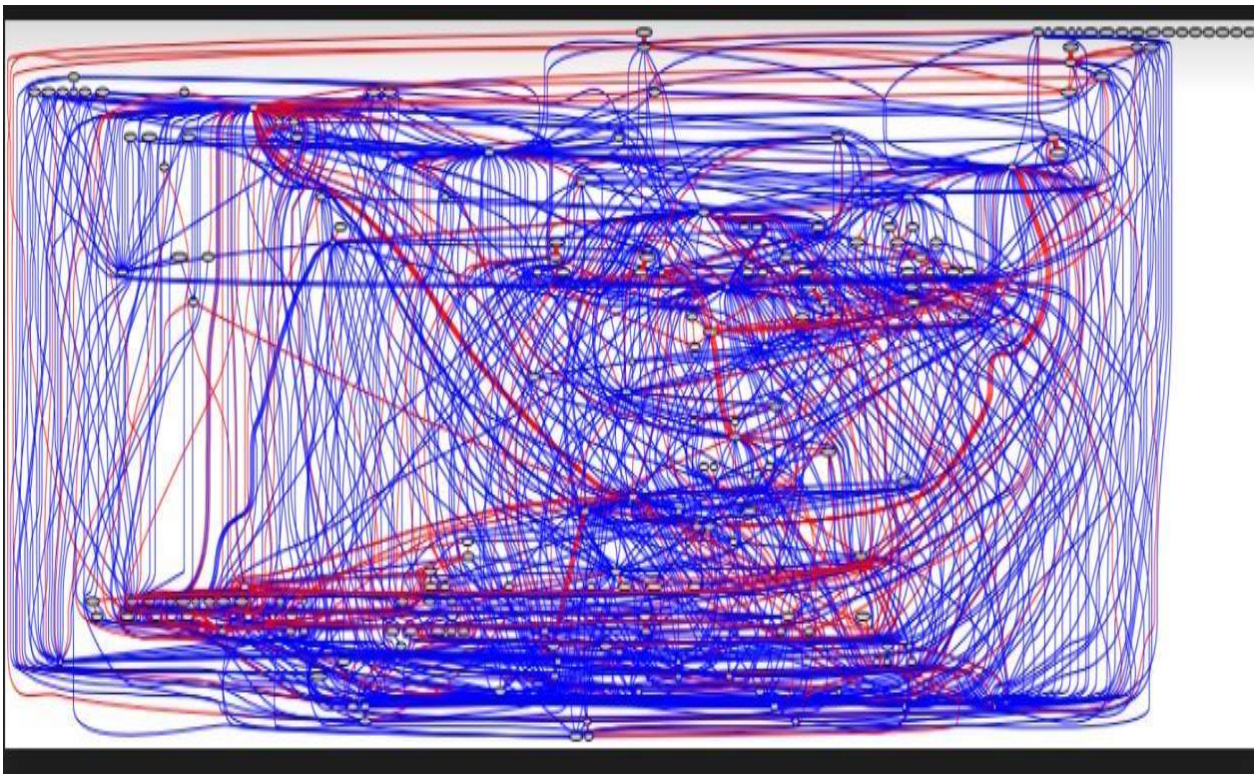

Figure 4. Call Dependency Graph Extraction from the Data Set

When using function-call dependencies to drive code modularization, first create a function-dependency graph (derived from function dependencies), with nodes representing single functions(or files), and edges indicating func-tion-call interdependence or other inter-function/inter-file features.

As a result, MQ measure in PMQ is redefined by increasing the P D ("Perturbation Degree").

The PD is a consequence of MoJoFM measurement [6] and is well-known in terms of the remodularization process's movement and joining.

To guide the GA's optimization method to a higher level of modularization with the least amount of modification to the current modular structure, an adequate fit-ness function is necessary.

Throughout this paper, designers provide the PMQ fitness function, which penalizes the modularity excellence criterion, such as the MQ metric, with the MoJoFM metric.

The P.M.Q metrics are identical to the MQ measures:

MQ: The goal is to assess programmers in a modular fashion.

It's expressed as an amount of MFs ("Modularization Factors"), with each MF having a restrictionon the amount of connection between and within packages.

$$M Q = \sum_{k=1}^{n} MFk \text{ where } MFk = \Sigma\, 0|I/\, i+1/2\, j \text{ if } i > 0 \quad \text{(i)}$$

The total no. of packages is n, and intra- and inter-package connectivity is I and j.The trade-off around cohesion and coupling is visible in MQ.

Another measure, such as elementary MQ [5], can be used to evaluate modularity.

$$\text{“P M Q = MQ×PD where PD=1−mno (M new, Morg)”/max (m no (∀M new, Morg))} \quad \text{(ii)}$$

To modify the modularization response Mnew to the modularization resolution, the fewest frequency of Join or Move procedures are necessary Morg is mno (M new, Morg), and max (mno(M new, Morg)) is the largest possible distance around any novel modularization key M new and the unique modularization result Morg. The MoJoFM measure determines if two modularization outcomes are linked.

Figure 5 illustrates a Visual Studio code extract from the main.py folder of the modularization dataset.

The below extracted snippet includes class names, file path of the generated system, MoJo generation, range of search center in module names.

This has been pulled out from the code to describe the main class and generating MoJo call dependency graph through the below mentioned code snippet.

```
def Search_Center(Class, File_Path):
    idx = -1
    for i in range(len(File_Path)):
        if Class == File_Path[i][1] or Class == File_Path[i][2]:
            if idx == -1:
                idx = i
            else:
                return -1
    return idx

def Generate_Mojo_CDG(Module_names, Module_paths, Directory):
    l = []
    for i in range(len(Module_names)):
        # Search Center in Module Names
        idx = Search_Center (Module_names[i], Module_paths)

        if idx == -1:
            print("invalid position of the Module ", Module_names[i])
            return
        l.append([Module_paths[idx][0], Module_names[i]])
    l.sort()

    f = open (Directory + "/MoJoFormat.txt", "w")
    for i in range(len(Module_names)):
        MoJoFormat =  "contain " + "package" + str(i) + " " + l[i][1] + "\n"
        f.write(MoJoFormat)
    f.close()
```

Figure 5. Code snippet extracted from the main.py folder of modulzarition data set from visual

Conclusion and Future Work

In Software maintenance duties, architecture recovery is becoming increasingly important. Open- source assimilation (source code understanding) is a critical maintenance task that comprises howdevelopers maintain current source code. To help source code understanding and recover programme architecture, many modularization technologies were used to split a system into expressive sub-systems automatically. The purpose of this study was to create recent multi-objective fitness function that would allow system's source code to be modularized. Evolutionary algorithms can use provided objective function to modularize source code while taking into account both architectural and quasi components. The research included structural aspects like call dependency graphs and inheritance linkages, as well as non-structural data like text, identifier names, and comments.

From a supply or source code, we furnished the formulae for calculating those sceneries. Non-structural traits also are subjected to dimensionality discount to make statistics interpretation easier. 5 algorithms, including genetic algorithms with encodings, a distribution estimationalgorithm, and combination of genetic & hill mountaineering algorithms, are tailored to optimize provided multi-goal health feature in order to modularize a software programme device that usesthose features.

The cautioned algorithms' consequences are assessed the usage of quite a few metrics, which includes Mojo and MoJoFM. The checks discovered that the modularization generated through the distribution estimation method is much like the taught modularization. We additionally checked out the findings for merging and stability.

For future works, the following proposals are made:

- Providing semantic and trivial dependency networks to enable the modularization of several-programming oft-ware techniques;
- Establishing thesaurus when it comes to programming languages and how to use them to compare comments and identifier assignments;
- Objective function is being studied using information theory.
- Extracting features from source code using feature range approaches such as those found in (Abu aligah and Khader 2017; Abualigah. 2016).

The probabilistic model that has been built determines quality of solutions in estimation of distribution algorithms. The Gaussian probability distribution function, we feel, can improve

correctness of the probability model we've created.

References

Boehm, B.W. (1975). "The High Cost of Software", In Horowitz E., Practical Strategies for Developing Large Software Systems", Addison Wesley.

Parikh, G., Zvegintzov, N. (1983) "Tutorial on Software Maintenance", IEEE Computer Society press, Silver Spring Maryland.

Spiros Mancoridis, Brian S. Mitchell, C. Rorres, Yih-Farn Chen, and Emden R. Gansner (1998) "Using automatic clustering to produce high-level system organizations of source code". In International Workshop on Program Comprehension (IWPC'98), pages 45–53, Los Alamitos, California, USA. IEEE Computer Society Press.

Spiros Mancoridis, Brian S. Mitchell, Yih-Farn Chen, and Emden R. Gansner (1999) "Bunch:A clustering tool for the recovery and maintenance of Software system structures". In Proceedings of IEEE International Conference on Software Maintenance, pages 50–59. IEEE Computer SocietyPress.

Doval, D., Mancoridis, S., and Mitchell, B.S. (1999) "Automatic clustering of Softwaresystems using a genetic algorithm". In International Conference on Software Tools and Engineering Practice (STEP'99), Pittsburgh, PA, 30 August - 2 September.

Harman, M., Hierons, R., and Proctor, M. (2002) "A new representation and crossover operator for search-based optimization of Software modularization. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, 9-13 July 2002, MorganKaufmann Publishers, pp. 1351–1358.

Kiarash Mahdavi, Mark Harman, and Robert Mark Hierons (2003) "A multiple hill climbing approach to Software module clustering". In IEEE International Conference on Software Maintenance, pages 315– 324, Los Alamitos, California, USA, September 2003. IEEE Computer Society Press

Bilal Khan, Shaleeza Sohail and M. Younus Javed (2008) "Evolution Strategy Based Automated Software Clustering Approach"; 2008 International Conference on Advanced Software Engineering & Its Applications (ASEA 2008) held on December 13 - 15, Hainan, China.

Praditwong, K. (2011) "Solving Software Module Clustering Problem by Evolutionary Algorithms", Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE 2011), 11-13 May 2011, Nakhon Pathom, Thailand, pp. 154-159.

Praditwong, K., Harman, M., Xin Yao (2011) "Software Module Clustering as a Multi- Objective Search Problem", IEEE Transactions on Software Engineering, Volume 37(2), 2011, pp.264-282.

Praditwong, K., and Xin Yao (2006) "A New Multiobjective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm". In Proceedings of the 2006 International Conference on Computational Intelligence and Security, volume 1, pages 286–291, Guangzhou, China.

Deb, K. (2001) "Multi-Objective Optimization using Evolutionary Algorithms", WileyChichester, UK.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003) "Handbook of metaheuristics", chapter 16, "Hyper-heuristics: an emerging direction in modern search technology", pp. 457–474, Kluwer Academic Publishers.

Cowling, P.I., Kendall, G., Soubeiga, E. (2001) "Hyperheuristic Approach to Scheduling a Sales Summit", Selected papers of Proceedings of the Third International Conference of Practice and Theory of Automated Timetabling, Springer LNCS vol. 2079, pp. 176-190.

Kaelbling, L.P., Littman, M.L., Moore, A.W. (1996) "Reinforcement learning: a survey", Journal of Artificial Intelligence Research 4, 237–285.

Nareyek, A. (2003) "Choosing search heuristics by nonstationary reinforcement learning", In Metaheuristics: Computer decision-making, pp. 523–544. Kluwer Academic Publishers,Dordrecht.

O. Maqbool and H. Babri, "Hierarchical clustering for Software architecture recovery," Software Engineering, IEEE Transactions on, vol. 33, pp. 759-780, 2007.

V. Tzerpos and R. C. Holt, "ACDC: An algorithm for comprehension-driven clustering," in ware, 2000, p. 258.

S. Mancoridis. 03 January 2014). Bunch tool.

A. Ibrahim, D. Rayside, and R. Kashef, "Cooperative based Software clustering on dependency graphs," in Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on, 2014, pp. 1-6.

I. Hussain, A. Khanum, A. Q. Abbasi, and M. Y. Javed, "A Novel Approach for Software Architecture Recovery using Particle Swarm Optimization," International Arab Journal of Information Technology (IAJIT), vol. 12, 2015.

M.T. Ziabari, A.R. Sahab, and S.N.S. Fakhari, "Synchronization New 3D Chaotic System Using Brain Emotional Learning Based Intelligent Controller," International Journal of Information Technology and Computer Science (IJITCS), vol. 7, p. 80, 2015.

V. Khorani, F. Razavi, and A. Ghoncheh, "A New Hybrid Evolutionary Algorithm Based on ICA and GA: Recursive-ICA-GA," in IC-AI, 2010, pp. 131-140.

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 Conferenceon Program Comprehension, 1998, pp. 45-45.

K. Mahdavi, M. Harman, and R. M. Hierons, "A multiple hill climbing approach to Software module clustering," in Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, 2003, pp. 315-324.

S. Mancoridis. (2002, 10 September 2013). Sample MDGs.

Available: https://www.cs.drexel.edu/~spiros/bunch/ [27] D. E. Goldberg, Genetic algorithms: Pearson Education India, 2006.