Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

A bioinspired discrete heuristic algorithm to generate the effective structural model of a program source code

Bahman Arasteh [a,*], Razieh Sadegi [b], Keyvan Arasteh [a], Peri Gunes [c], Farzad Kiani [d], Mahsa Torkamanian-Afshar [e]

[a] *Department of Software Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Turkey*
[b] *Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran*
[c] *Department of Software Engineering, Faculty of Engineering, Yıldız Technical University, Istanbul, Turkey*
[d] *Computer Engineering Dept., Faculty of Engineering, Fatih Sultan Mehmet Vakif University, 34445 Istanbul, Turkey*
[e] *Computer Engineering Department, Faculty of Engineering and Architecture, Nisantasi University, 34398, Istanbul, Turkey*

ARTICLE INFO

Article history:
Received 17 May 2023
Revised 18 June 2023
Accepted 12 July 2023
Available online 18 July 2023

Keywords:
Olympiad optimization algorithm
Software module clustering
Cohesion
Modularization quality

ABSTRACT

When the source code of a software is the only product available, program understanding has a substantial influence on software maintenance costs. The main goal in code comprehension is to extract information that is used in the software maintenance stage. Generating the structural model from the source code helps to alleviate the software maintenance cost. Software module clustering is thought to be a viable reverse engineering approach for building structural design models from source code. Finding the optimal clustering model is an NP-complete problem. The primary goals of this study are to minimize the number of connections between created clusters, enhance internal connections inside clusters, and enhance clustering quality. The previous approaches' main flaws were their poor success rates, instability, and inadequate modularization quality. The Olympiad optimization algorithm was introduced in this paper as a novel population-based and discrete heuristic algorithm for solving the software module clustering problem. This algorithm was inspired by the competition of a group of students to increase their knowledge and prepare for an Olympiad exam. The suggested algorithm employs a divide-and-conquer strategy, as well as local and global search methodologies. The effectiveness of the suggested Olympiad algorithm to solve the module clustering problem was evaluated using ten real-world and standard software benchmarks. According to the experimental results, on average, the modularization quality of the generated clustered models for the ten benchmarks is about 3.94 with 0.067 standard deviations. The proposed algorithm is superior to the prior algorithms in terms of modularization quality, convergence, and stability of results. Furthermore, the results of the experiments indicate that the proposed algorithm can be used to solve other discrete optimization problems efficiently.

1. Introduction

Change and evolution of software based on changes in user needs and system features are inevitable. Software maintenance (Change and evolution) accounts for 60% of software expenditures on average (Amarjeet and Chhabra, 2017; Sun and Ling, 2018; Chhabra, 2017; Yuste et al., 2022). One of the most difficult tasks in software maintenance is determining the impact of the applied change on the rest of the software. The action of examining the likely impacts of a change to limit unexpected side effects is known as impact analysis. When the source code is the only product available, program understanding has a substantial influence on software maintenance costs. The main goal in code comprehension is to extract information that is used in the software maintenance stage. The primary maintenance task, which accounts for around 50% of the expenditures, is program comprehension. The generated structural model from the source code helps the software developers to make the desired maintenance in the software system with a limited amount of cost.

* Corresponding author.
E-mail addresses: b_arasteh2001@yahoo.com, bahman.arasteh@istinye.edu.tr (B. Arasteh).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

The structural model, as an architectural model of a software product, consists of the modules (components, classes, methods) in the software product and the static relationships that exist between them. This model describes the related codes (modules) in the program source code and display the organization of a software system in terms of the components. As a result, when confronted with large and complex software source code, extracting, and interpreting the software structural models from source code is essential. Before making source-code changes, one can learn about the design structure of software by reverse engineering the source code components. As a reverse engineering approach, software source-code clustering groups software modules (components, classes, and methods) with similar qualities after extracting them from the source code. As a metric, modularization quality (MQ) is evaluated based on the number of connections inside (cohesion) and between clusters (coupling). The MQ criterion is used to assess the performance of module clustering algorithms. According to this criterion, optimal clustering results from maximal cohesion (the ties between modules in a cluster) and minimal coupling (the links between different clusters).

The software module clustering (SMC) methods divide the source code of a program into m clusters (components or packages). Assume S (a source code) is made up of n modules, M1, M2, …, Mn, each of which includes methods, functions, and properties. Set π represents the available combinations for dividing n modules into m clusters. Each item of π indicates a potential combination of clustering options. The number of possible solutions for clustering a program with n modules into m clusters is represented by Sn, m (Stirling numbers) that is calculated by Eq. (1). A program with five modules, for example, has 52 distinct clustering combinations, but a 25-module program has 1,382,958,545 possible clustering combinations. The cluster intersection is empty, while the union of the m clusters equals all source code (S). As a result, the SMC problem is technically classified as an NP-Complete problem (Chhabra, 2017; Yuste et al., 2022; Prajapati and Chhabra, 2018). This forces academics to use heuristic ways to choose the appropriate grouping.

$$\begin{aligned} &S(n,m) = S(n-1,m-1) + m \times S(n-1,m) \\ &n : number\ \ of\ \ modules\ \ in\ \ the\ \ software\ \ product \\ &m : number\ \ of\ \ clusters \end{aligned} \tag{1}$$

The primary goals of the study are as follows:

- Improving the quality of modularization (MQ).
- Increasing the chances of obtaining optimal clusters.
- Improving the SMC method's stability across several runs.
- Accelerating convergence to get the best clusters.

Different heuristic and machine-learning algorithms are now being employed to solve various computer science and engineering challenges. The SMC has been approached as a discrete search-based optimization issue using a variety of search-based heuristic approaches. The primary shortcomings of the prior techniques are as follows:

- Lower MQ.
- Reduced success rate and falling into the local optimum.
- Poor stability.
- Slower convergence speed in finding the best clustering, particularly in big software applications.

This research proposed a novel discrete optimizer strategy for dealing with the SMC problem. The proposed technique is based on the process of group teaching and learning of class members (students) in preparation for an Olympiad test (Olympiad Optimization Algorithm). In the Olympiad Optimization Algorithm (OOA), the population mimics the behavior of a group of students that desire to enhance their knowledge by learning from their peers. The students try to improve their knowledge by learning from the other students. Therefore, students are competing to gain knowledge from each other. The OOA splits and conquers an optimization problem by using both local and global search techniques. Each iteration divides the population into student groups, and teaching and learning take place among the students. No teachers are participating in the teaching and learning process at the OOA. In the OOA, population evolution is based on swarm intelligence and group-based learning. The local search operation is carried out by each group. Each solution delivers software module clustering; a solution's fitness demonstrates the solution's MQ. OOA may also be used to solve discrete optimization problems. The followings are the study's main contributions:

- In this work, a unique discrete optimization algorithm (OOA) for tackling discrete optimization problems was developed. The suggested method is a swarm-based algorithm that simulates the learning process of a class of students preparing for the Olympiad test. The steps of teaching and learning amongst population members (students) in each iteration produce population evolution. The suggested algorithm employs a divide-and-conquer strategy, as well as local and global search methodologies. Individuals are separated into groups, and each group searches in a distinct part of the total solution space using a specific imitation strategy.
- The suggested OOA can handle the majority of graph-based optimization challenges. Another contribution of this study is the use of the OOA to tackle the software module clustering problem as a graph partitioning problem.

The following five sections comprise the paper: The second section provides an outline of major SMC research. Section 3 illustrates and explains the recommended OOA method and its application in SMC problem. Section 4's first half discusses the platform and tools that were utilized to carry out the proposed approach. This section also includes evaluation criteria and data sets related to the SMC methods. The study's findings are presented and discussed in the second part of Section 4. Section 5 concludes the paper and recommends future research in the field of SMC.

2. Related works

In Yuste et al. (2022) a PSO-based approach for selecting the optimum clusters for software application modules was presented. Each particle depicts the structure of software components (modules). Each particle possesses two critical properties: its location vector and its speed vector. The speed vector, on the other hand, affects the current positions of the particles in each direction. As the particles travel faster, their locations alter. The starting population consists of particles. In response to the SMC task's challenges, the particle locations and speed of the PSO technique were created. A modified version of the PSO method was utilized for the SMC issue, which applies a local search technique to enhance particle position. The results of the tests reveal that the recommended method produced higher-quality software clusters. Chhabra (2018) created a one-of-a-kind heuristic method based on a Greedy Random Adaptive Search Procedure using Variable Neighborhood Descent. The method has been tested successfully on a set of real-world software projects and exceeds the prior state-of-the-art approach in terms of Modularization Quality in extremely fast computation times.

XinShe-Yang introduced the firefly algorithm in Mamaghani and Hajizadeh (2014) (FA). Its foundation is based on swarm intelligence. In this method, each firefly symbolizes a clustering combination with a specified light intensity (clustering quality). The degree of light intensity may be quantified using a fitness function (MQ). Fireflies try to approach optimal fireflies and change their positions to get the best clustering combination and even the most accurate timings. To address difficulties, the Firefly algorithm, a non-discrete optimization approach, is utilized. It may, however, be used for specific problems such as the SMC problem. Firefly outperforms HC and GA on most benchmarks, according to experimental data. The approach's principal restrictions, particularly in big software systems, are local optimum likelihood and sluggishness.

A hill climbing (HC)-based approach for building clustered structural models of software source code was reported in Mahdavi et al. (2003). Because it includes N modules, MDG can generate up to N initial clusters. Before the first hill climber starts, each software module is assigned to a random cluster. A fitness function is used to assess the MQ of the created clusters. The purpose of this strategy is to create clusters with the least amount of connection and the greatest level of cohesion. At each stage of the method, each hillclimber seeks to reach the next neighboring cluster with a greater MQ. When the hillclimber notices another neighbor in the new cluster, it goes in quest of another (climber with a higher MQ). When none of the clustering's nearest neighbors can find a optimal MQ value, the search activity is ended. The first stage's hills finally join together to form a hill sequence. This strategy has the potential to slip into the local optimum, searching for the global optimum more difficult (the optimal clustering).

The genetic algorithm is a heuristic search technique that is evolutionary in nature (GA). The GA has solved the HC approach's key flaws in the SMC problem. There is no way to get the best grouping when using direct search techniques like the HC algorithm. Such algorithms are incapable of dealing with huge search spaces. In contrast, the search procedure in GA happens concurrently with the beginning population's chromosomes. Each chromosome refers to a clustering combination in GA. The starting population consists of chromosomes selected at random (random clustering combinations). The fitness function analyzes the chromosomes in each iteration, intending to construct chromosomes with a high degree of cohesion and a low degree of coupling. The search is then enlarged by updating the chromosomes of interest with the crossover and mutation operators (Mancoridis et al., 1999; Praditwong et al., 2011). According to the findings, the GA is an appropriate solution for the SMC problem in small and medium-sized software clusters.

For selecting the best software application clusters, Arasteh and colleagues proposed a hybrid PSO-GA approach (Arasteh et al., 2020). This approach seeks to address shortcomings in previous methods (low convergence, inadequate MQ, low stability, and inadequate success rate). This approach incorporates the advantages of both heuristic methods. When compared to PSO and GA algorithms, this hybrid approach enhances clustering quality and provides rapid data convergence. Throughout the level, crossover and mutation were used to update and improve particle position by updating the speed vectors of all particles. Experiments on ten well-known benchmark Module Dependency Graphs (MDGs) reveal that the PSO-GA technique beats the standard strategy 90% of the time. Furthermore, in 30% of the benchmarks, all three approaches obtained the same success rate. In 60% of the benchmark programs, the PSO-GA technique outperforms the PSO and GA algorithms in terms of stability. The code that was used is freely accessible for download.

In Hatami and Arasteh (2020), the ant colony optimization (ACO) approach was used for clustering the components of the software. The modules of each cluster (subsystem) are considerably connected. Swarm intelligence is used by the ACO algorithm to solve a variety of search-based optimization issues. Each ant in the proposed approach is a clustering solution. A high-quality clustering has the highest coherence and the lowest coupling. This method was tested on a variety of benchmark datasets. The findings indicate the higher performance of ACO in the SMC problem. Furthermore, regarding the convergence speed and MQ value, the ACO-based approach typically beats the GA and PSO. In regard to stability, all three solutions have the potential to overcome the SMC problem.

By combining the shuffle frog leaping algorithm (SFLA) with GA approaches, an SMC method (Bölen) for clustering software modules was proposed in Arasteh et al. (2021). This method offers several advantages, like faster data convergence, greater modularization quality (MQ), higher success rate, and improved stability. MDG, like the other techniques, is used to demonstrate how various software components (modules) are linked. SFLA covers both local and international searches. In the SMC problem, each frog in SFLA is viewed as a clustering array (clustering combination). Using the crossover operator, the best frogs in each memeplex were generated from the poorest frogs. The mutation operator is also performed for every memeplex individual, with the option of optimizing the memeplex member with the lowest strength. In terms of average MQ, the SFLA-GA methodology outperformed the other strategies in 80% of the benchmark instances. The SFLA-GA approach, according to the convergence criteria, converges to the optimal solution faster than that of the HC, GA, and PSO algorithms in 90% of circumstances.

In Arasteh et al. (2022), a hybrid single objective strategy that merges the gray wolf algorithm (GWOA) and GA was developed for the SMC problem. The suggested method combines a swarm-based and an evolutionary algorithm. For the SMC issue, the conventional GWOA was discretized and tweaked. Experiments on 14 prominent benchmarks reveal that this single-objective hybrid approach outperforms the GA, PSO, and PSO-GA strategies in the SMC problem; greater MQ and quicker convergence speed are significant benefits of this method, especially in big software packages. Many chaos-based heuristic algorithms for the SMC issue, such as the Bat, Cuckoo, Teaching-Learning-Based, Black Widow, and Grasshopper heuristic algorithms, were developed in (Arasteh, 2022). The effects of chaos theory on the effectiveness of various algorithms in this context have also been examined. The BWO, PSO, and TLB methods outperform the other methods in the SMC issue, according to real-world application findings. Furthermore, when the initial populations of these algorithms were generated using the logistic chaos method, their performance increased. The average MQ values for clusters formed by BWO, PSO, and TLB in the supplied benchmark set are 3.155, 3.120, and 2.778, accordingly.

In Arasteh et al. (2022), an autonomous strategy (Savalan) for the SMC issue was proposed, that is based on a multi-objective genetic algorithm and a special combination of goal functions. The fundamental purpose of this study is to improve all clustering goals at the same time (coupling, cohesion, cluster size, modularization quality, and number). Six separate objective criteria were considered as optimization objectives in this study. As the multi-objective genetic algorithm in the proposed approach, the Pareto envelope-based selection algorithm (PESA) was used. This method is useful for both little and major projects. Based on the results of the 14 benchmark programs, the primary advantage of this approach is that it improves all clustering goals at the same time. The results reveal that the Savalan technique concurrently improves all clustering criteria. Savalan generates higher-quality clusters than similar technologies such as Bunch (Mancoridis et al., 1999), CIA (Chen, 1995), and Chava (Korn et al., 1999), and it outperforms them in large software systems (MQ). Savalan was released as free software for researchers and developers in (Available online: http://savalan-smct.com/). As a result, the software industry will benefit from both the theoretical and practical implications of our research. This tool was built using the JavaScript programming language. Table 1 summarizes the key components of contemporary research and creative techniques in complex software systems. The main disadvantages of the previous approaches include slow convergence, local optimum, and poor stability.

3. Proposed method

In this study, a novel discrete heuristic algorithm was proposed. Introducing, designing, and implementing a new discrete heuristic algorithm is the first innovation of this research. The proposed algorithm (OOA) can be used for solving discrete NP-hard optimization problems. Adapting the OOA for SMC problem is the other contribution of this study. SMC is one of the challenging optimization problems in the field of software engineering. Producing the most effective structural model from a source code is the optimal solution to the SMC problem. Clustering the *n* modules of a program into *m* clusters is formally considered a combination problem. In this problem, the module dependency graph (MDG) of the source code is generated (first stage) and then the OOA clusters the modules of the program using MDG. The generated structural model alleviates the cost of software maintenance.

3.1. Olympiad algorithm

3.1.1. Algorithm structure

For tackling the SMC problem, a novel discrete heuristic technique was presented in this work. To solve the SMC problem, the Olympiad optimization algorithm (OOA) is presented and implemented as a discrete swarm algorithm. The suggested OOA uses the swarm-based imitation technique as its local and global search methods. OOA is a population and group-based heuristic method that solves the optimization problems in divide and conquer form. Each member of the population in the OOA simulates the behavior of a student in the class. The proposed OOA mimics the learning process among students of a class who are preparing for the Olympiad exam.

The steps of teaching and learning between the members of the population (students) in each iteration cause the evolution of the population. The proposed algorithm is a divide-and-conquer-based algorithm with local and global search strategies. As shown in Fig. 1, the individuals (students) of the whole population are sequentially divided into equal size subgroups. Line 9 of Algorithm 1 indicates the implemented MATLAB code for the population division. The individuals are divided into subgroups, and each subgroup uses a specific imitation approach to search in a different region of the overall solution space. Fig. 1 shows the general workflow of the OOA that can be used to solve an optimization problem. There is competition among the individuals (students) to learn from other students. Each student has a memory for storing their learning rate (position of the student in terms of learning). Each student in the OOA was designed and implemented as a numeric array. The student population is made up of solutions. As depicted in Fig. 1, the partitioning of the student population into n teams is the first stage of the OOA after sorting. Each team includes m students. The first team is the global best team, and the last team is the global worst team. Each team, comprised of the students, explores its local solution space. The first student of the first team is the global best student, and the first student of each team is its local best student. Algorithm 1 illustrates the pseudocode of the main part of introduced OOA.

Algorithm 1. The pseudocode of the main part of the proposed OOA.

```
1.  Initialize population size, iteration, number of teams and size of teams;
2.  nPop=Number of populations.
3.  n=number of Teams.
4.  m=Size of Team.
5.  itr=0;
6.  While (itr < iteration)
7.    {
8.      Sort(population); //parallel bubble sort algorithm
9.      Team=Partition (population, n, m);
10.     globalBest=1;
11.       for i=2 to n
12.         {
13.           localBest=i-1;
14.           localWorst=i;
15.           for j=1 to m do in parallel
16.             {
17.               NewStd=Learn(Team[localBest][j], Team[localWorst][j]); //1st step
18.               if(NewStd.cost < Team[localWorst][j].cost)
19.                 Team[localWorst][j]= NewStd;
20.               else
21.                 NewStd= Mutate(Team[localBest][1])); //2nd step
22.                 if(NewStd.cost < Team[localWorst][j].cost)
23.                   Team[localWorst][j]= NewStd;
24.                 else
25.                   NewStd=Learn(Team[globalBest][j], Team[localWorst][j]); //3rd step
26.                 if(NewStd.cost < Team[localWorst][j].cost)
27.                   Team[localWorst][j]= NewStd;
28.                 else
29.                   NewStd= Mutate(Team[globalBest][1])); //4th step
30.                   if(NewStd.cost < Team[localWorst][j].cost)
31.                     Team[localWorst][j]= NewStd;
32.                   else
33.                     NewStd= StdRandGeneration();
34.                     if(NewStd.cost < Team[localWorst][j].cost)
35.                       Team[localWorst][j]= NewStd;
36.             }
37.         }
38.       Evaluate the fitness of the population using equations (2-3);
39.       itr=itr + 1;
40.         } //end of while
41. Return Team[globalBest][1]; //return the global best student as the best solution.
```

3.1.2. Population evolution by learning procedure

The students are divided into teams of the population and try to learn from the best student of the adjacent team (the local best student) or the global best team. The proposed learning operator implements the local and global search of OA. Indeed, learning is the main operator of the OOA to find the optimal solution to the problem. The learning operator was introduced to improve the population's knowledge (fitness). The individuals (students) were sorted based on their knowledge (fitness function). As shown in Fig. 2, OOA tries to improve the knowledge of the population by the introduced learning operator. Learning operator includes four steps. In the first step, the students of each team learn from the students of their adjacent team. As a result of the first step of learning the operator, the knowledge of the first team is transferred to other teams as a bubble.

Fig. 3 shows the first step of the proposed learning operator. In this step, the knowledge of a team is transferred to the next adjacent team. The procedure of learning, like the bubble sort algorithm, is performed serially from one team to the adjacent team. As shown in Fig. 3, all students of a learner team learn from the corresponding students in the adjacent team (left side team). If there is no improvement in the knowledge of the weak students of the weak team, the second stage of the learning process is performed. In the second improvement step, the best student of the learner's team (right side team) is mutated to make diversity. The mutation operator implements a local search on the best student of the weak team. If no improvement was made in the second step, then the third step is executed. In the third step, all students of the learner team (weak team) learn from the corresponding students of the global best team (first team).

If the global-best team's students could not teach the worst team's students, then the mutation operator is performed on the globally best student. The mutation of the global best may avoid the local optimum by making minor diversity. Finally, students (search agents) from different teams are combined to produce a new population. The learning operator is performed iteratively on the student population. Algorithm 2 illustrates the pseudocode learning operator that is conducted in each iteration of the OOA. In the proposed method, learn operator (imitation operator) was implemented by crossover operator. Some of bits (cells) in the worst individual are replaced by the corresponding elements of best individual. The LearnCount indicates the number of bits from the weak element that should be replaced by the equivalent bits in the strong element. The best value of LearnCount is determined experimentally. In this study, the optimal value of LearnCount is 30% of the length of clustering array.

Algorithm 2. The pseudocode of the learning operator in the OOA.

```
Function studentType Olampiyad_Learn(BestStd,
WorstStd)
{
  nVar= length(Student_array);
  LearnCount=ceil((30 * nVar)/100); %imitation
count
  count=0;
  i = 0;
  NewStd=WorstStd;
  while (count<=LearnCount and i < nVar)
    {
    aL=randi(nVar);
    if (WorstStd(aL)!=BestStd(aL))
      {
        NewStd(aL)=BestStd(aL);
    count=count+1;
      }
    {i = i + 1;
    }
Return(NewStd);
}
```

3.2. Adapting the olympiad algorithm to SMC problem

3.2.1. Problem specification

In the second phase of this study, the proposed innovative algorithm (OOA) was used to solve the SMC problem. The recommended OOA uses the MDG of the program source code to arrange related modules together. As a result, the OOA's input is the MDGs of the benchmark source code. Fig. 4 depicts the MDG extracted from the source code. The nodes depict the software modules, while the edges reflect the connections between them (calls, inheritance, and association). Fig. 4 displays the software product's six components as well as the related dependency

Table 1
Specifications of prior methods and tools.

Researcher	Publication Year	Algorithm Type	Fitness Function
Prajapati (Prajapati and Chhabra, 2018)	2018	PSO based Method	Single-objective
Mamaghani (Mamaghani and Hajizadeh, 2014)	2014	Firefly based Method	Single-objective
Mahdavi (Mahdavi et al., 2003)	2003	Hill Climbing based Method	Single-objective
Mancoridis (Mancoridis et al., 1999)	1999	GA based Tool	Single-objective
Praditwong (Praditwong et al., 2011)	2010	Two Archive Genetic based Method	Multi-objective
Arasteh (Arasteh et al., 2020)	2020	PSO-GA based Method	Single-objective
Hatami (Hatami and Arasteh, 2020)	2020	ACO based Method	Single-objective
Arasteh (Arasteh et al., 2021)	2019	SFLA-GA based Method	Single-objective
Arasteh (Arasteh et al., 2022)	2022	Hybrid Gray Wolf based Method	Hybrid Single Objective
Arasteh (Arasteh, 2022)	2022	Chaos-based Metaheuristic Method	chaos-based Single Objective
Arasteh (Arasteh et al., 2022)	2022	PESA-GA based Method	Multi-objective
Arasteh (Arasteh et al., 2020)	2022	web based and Multi objective based tool	Multi-objective
Korn (Korn et al., 1999)	1999	Java Reverse Engineering Tool	Single Objective
Kumari (Kumari et al., 2013)	2013	Hyper-heuristic Method	Multi-objective
Chhabra (Chhabra, 2017)	2017	Object Oriented Method	Multi-objective
Chhabra (Chhabra, 2018)	2018	Two-Archive ACO based Method	Multi-objective
Arasteh (Arasteh et al., 2023)	2022	Sand cat swarm based Method	Single-objective

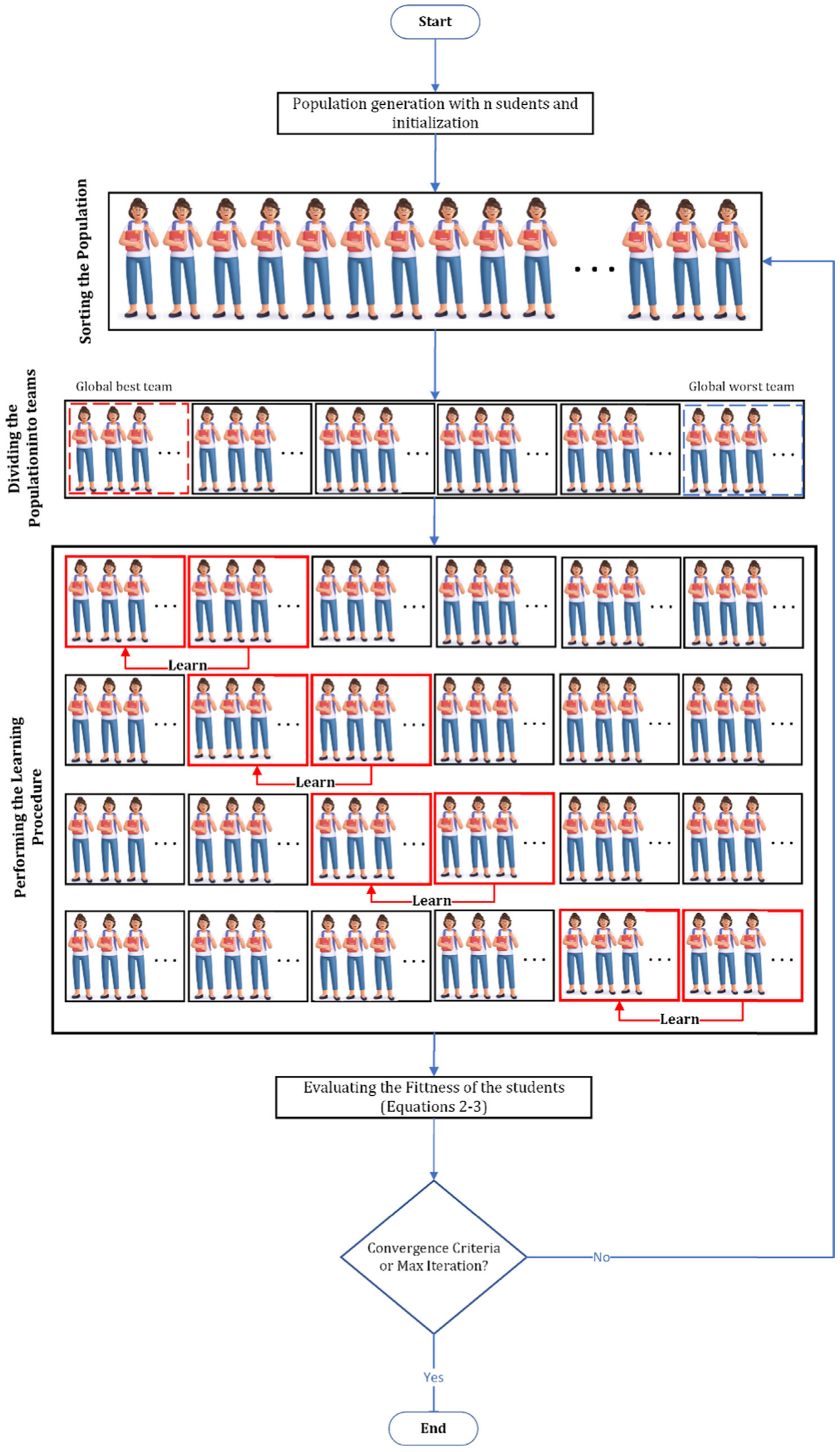

Start
Population generation with n sudents and initialization
Sorting the Population
Dividing the Populationinto teams
Global best team
Global worst team
Performing the Learning Procedure
Learn
Learn
Learn
Learn
Evaluating the Fittness of the students (Equations 2-3)
Convergence Criteria or Max Iteration?
No
Yes
End

Fig. 1. The proposed OOA workflow.

matrix. The values of the matrix elements represent the module linkages. The MDG is used to generate the dependency matrix. Different programming environments, such as visual studio, visual studio code, and eclipse, may produce the MDG of a source code automatically. The graphviz libraries can be used to convert the MDG text file to the visual graph model (Jalali et al., 2013).

Each student in the SMC issue is represented by a numeric array (clustering array). Fig. 5 depicts a student's structure in the SMC

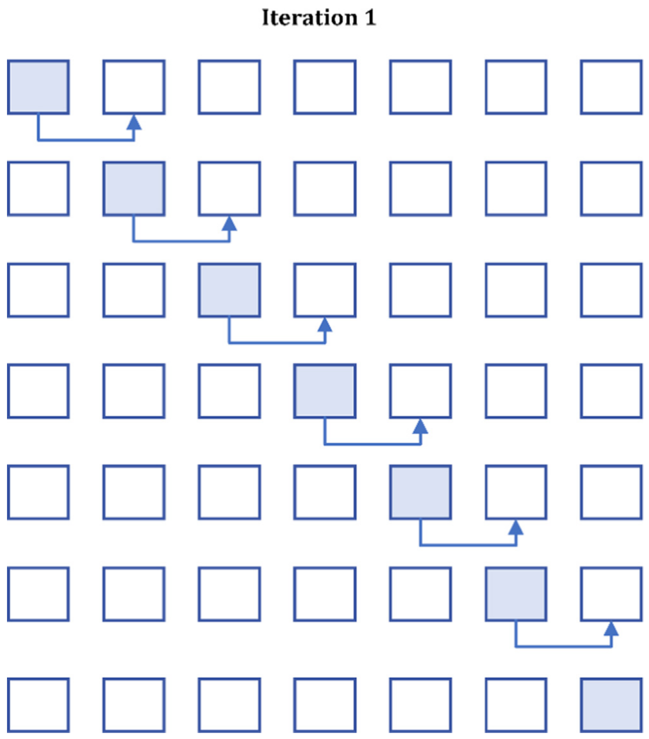


Fig. 2. The first step of the proposed learning algorithm.

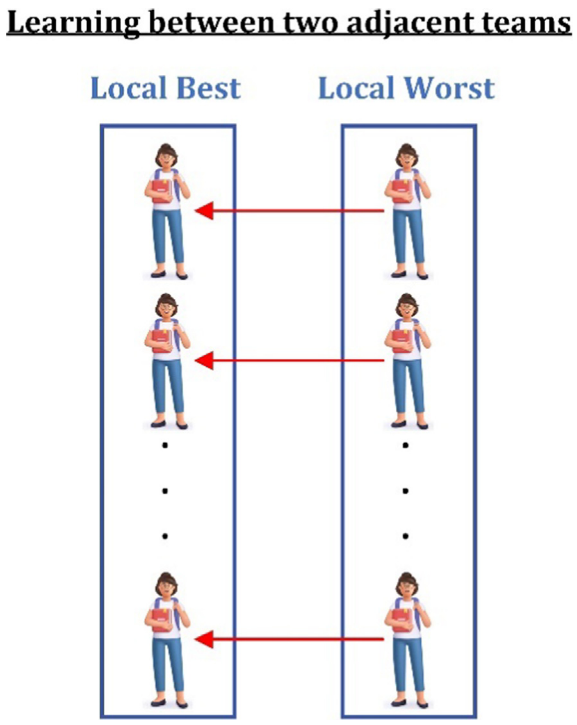


Fig. 3. The learner team's students learn from the adjacent team's students.

problem as well as the associated clustered MDG. The array's length equals the number of modules in the program source code. The indices denote the module number, and the value of each index in the clustering array represents the SMC method's cluster allocation. As seen in Fig. 5, the clustering array was designed to cluster a program source code with 22 modules; hence, it has 22 cells. In terms of module dependencies in the relevant source code, modules M1, M14, M18, and M22 were all grouped (cluster 1). Cluster 2 of the clustered MDGs depicted in Fig. 5 comprises M2, M13, M16, and M21. The SMC approaches attempt to organize the modules that are the most comparable and connected into the same cluster. As a result, any change to the code of a cluster module will most likely affect the other modules in the same cluster. This finding assists the program developer in managing the impact propagation when updating a module's source code.

3.2.2. Objective function

As a quality criterion, the modularization quality (MQ) was used to direct the population via the suggested method. This criterion is used by the OOA algorithm to guide its search for the finest software clusters. This criterion was introduced by Mancoridis et al. as a measure of clustering quality (Prajapati and Chhabra, 2018). High cohesion (internal connection) and low coupling characterize high-quality clusters (external connection). Because cluster modules are densely connected, a cluster with significant cohesion (internal connection) suggests an effective clustering design. Eq. (2) depicts the approach for determining the clustering quality (MQ) for cluster k. In Eq. (2), the variable i represents the number of internal connections (coupling) and variable j represents the number of exterior connections (cohesion) for a given cluster. The overall quality of all clusters formed is calculated using Eq. (3); in this equation, m denotes the number of clusters. MF_k indicates the modularization factor for cluster k. MQ indicates the sum of MF for all created clusters. The MQ function depicts the trade-off between intra-connectivity and interconnectivity (coupling). This function assesses clustering quality by optimizing coupling and cohesion. Individual module cohesiveness within a cluster must be strengthened. The higher the MQ, the higher clustering quality. If the coupling is regarded undesirable, the "optimal" system would be a single cluster that housed all modules. As a result, there must be a balance between coupling and cohesion.

$$MF_k = \begin{cases} 0 if i = 0 \\ \frac{i}{i+\frac{1}{2}j} if i > 0 \end{cases} \tag{2}$$

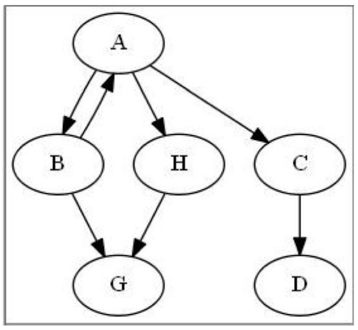
$$MQ = \sum_{k=1}^{m} MF_k \tag{3}$$

4. Experiments and results

4.1. Experiment platform

To evaluate the performance of the recommended OOA, a significant number of tests were run on the developed platform in MATLAB. The proposed technique, as well as the GA, PSO, PSO-GA, Cuckoo optimization algorithm (COA), Sand cat swarm algorithm (SCSO), and GWO, were developed and implemented in MATLAB in the same software platform. The calibration parameters of GA, PSO, PSO-GA, COA, SCSO, GWO, and OOA were adapted during the tests in this study. The optimal values for these parameters were determined during the experiments and are shown in Table 2. To achieve valid results, all the experiments were run on the same hardware platform, same software platform and under equal conditions. Ten conventional and actual benchmark MDGs were employed in the tests.

Table 3 displays the specifications of the benchmark MDGs (datasets). Each benchmark MDG indicates the modules dependency graph of a real-world program. Each MDG is a text representation of the call graph of the related real-world program. The MDGs were automatically extracted from the source code by



Dependency Matrix Created From MDG

	A	B	C	D	G	H
A	0	1	1	0	0	1
B	1	0	0	0	1	0
C	0	0	0	1	0	0
D	0	0	0	0	0	0
G	0	0	0	0	0	0
H	0	0	0	0	1	0

Fig. 4. The MDG and the dependency matrix of a software product with six modules and their relations.

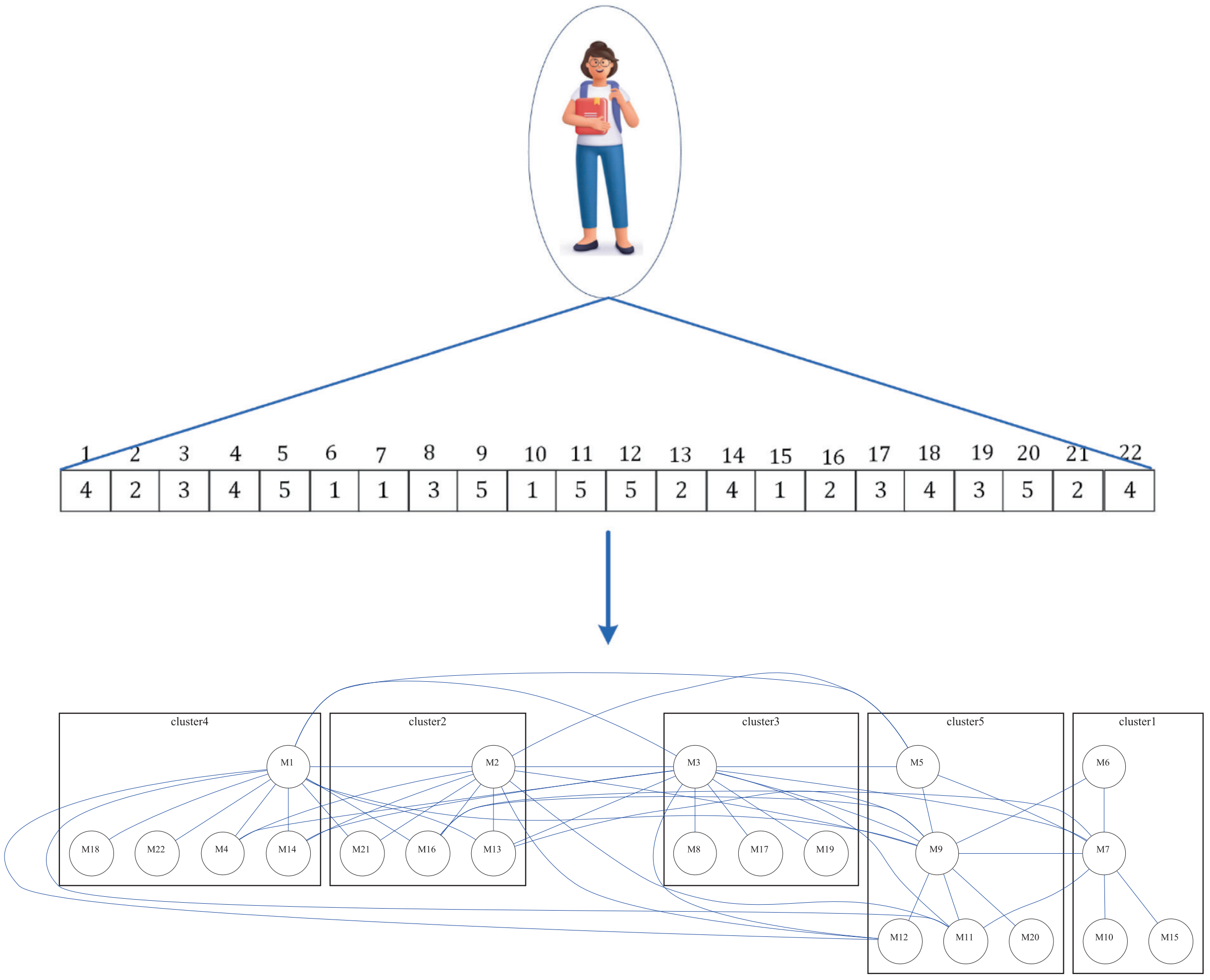


Fig. 5. The structure of a student in OOA is used to cluster a program with twenty-two modules into five clusters.

different programing IDEs (such as Visual Studio). The MDGs are simply converted to the adjacency matrix by the developed code in this study. The proposed OOA takes the adjacency matrix (shown in Fig. 4) of a MDG, as input, and finds its best clustering model. The number of rows and columns of the matrix is equal to the number of modules of the related program. The input of the proposed method is a matrix (shown in Fig. 4) and its out is in the form of a linear array (shown in Fig. 5). The benchmark programs were chosen to reflect real-world complexity in terms of nodes and connections (edges) between modules. The MDG of the mtunis benchmark (as a small benchmark) utilized in the research is depicted in Fig. 6. This software is made up of 20 modules linked together by 57 connections. The modules that are the most closely related have been grouped. The fitness function determines how comparable the modules in this study are (Eequation 3). The proposed technique seeks to group modules that are most similar into the same cluster. Fig. 6 demonstrates that the modularization quality (MQ), cohesiveness, and coupling are 1.858, 36, and 21 correspondingly. The higher the MQ criterion, the greater the clustering quality.

In this study, the graph (MDG) is generated automatically and directly from the source code using the programing platforms tools

Table 2
Parameters of SMC algorithms that have been adjusted experimentally.

Algorithms	Parameters	Value
Genetic Algorithm(GA)	Number of chromosomes	40
	Length of chromosome	Depends to the number of modules
	Rate of Ccossover	0.8
	Rate of mutation	0.05
	Number of iterations	100
Gray Wolf Algorithm (GWA)	Number of wolves	50
	a	Random value from [0, 2]
	C	Determined by GWO movement relations
	A	Determined by GWO movement relations
	r1, r2	[0–1]
Particle Swarm Optimization Algorithm (PSO)	Number of particles	40
	Inertia Weight	0.8
	Inertia Weight Damping Ratio	0.99
	Particle.C1 and Particle.C2	[1.5, 1.7]
	Number of iterations	100
(SCSO)	Number of Cats	40
	Sensitivity range (rG)	[0, 2]
	Phases control range (R)	[-2rG, 2rG]
	P_c	0.8
	P_m	0.04
(COA)	Number of Nests	40
	Lavy Distribution Parameter	1.5
	Step Length	0.01
	Number of iterations	100
Olympiad Optimization Algorithm(OOA)	Number of students	40
	Number of teams	10
	Size of teams	4
	Learning rate	Random values between [0.2–0.8]
	Imitation count	1
	Number of iterations	100

Table 3
The specifications of the programs as benchmarks.

Programs	# Modules	# Connections among modules	Size
mtunis	20	57	Small
spdb	22	16	Small
ispell	24	97	Small
Rcs	29	155	Mid
bison	37	117	Mid
Cia	38	216	Large
Dot	42	248	Large
Php	62	163	Large
Grappa	86	252	Large
Incle	174	360	Large

and libraries. Each node of MDG indicates a module in the program source code and each edge represents the connection among the nodes (call or data access). In MDG, the number of nudes and edges are deterministic; in this graph, |N| represents the number of nodes and |E| the number of edges. In the MDG (as deterministic graph) an edge between two vertices denotes a certain link in the corresponding source code and a node denotes a certain module in the corresponding source code. In contrast, in the SMC problem, the belonging of program modules to a program function (cluster) is an uncertain phenomenon. A module may play a role in different functions of the program with different probabilities. It means that a module does not definitely belong to a cluster. Hence, the created clusters are considered as an uncertain outcome due to the internal uncertainty of a module belonging to a cluster.

Many assessment factors were investigated during the tests. MQ is the fundamental performance criterion for SMC techniques. The MQ value is used to assess the quality of SMC-generated clusters. SMC methods all seek clusters with the highest MQ. Eq. (2) is used to calculate the MQ of a produced cluster for an MDG. Clusters with reduced coupling and more cohesiveness have higher MQ. Empirical evidence is used to determine the best number of clusters. Convergence speed was another performance factor that was examined. The speed of convergence is proportional to the time it takes an SMC approach to determine the best grouping. The success rate is another performance criterion for SMC algorithms. The success rate of an SMC technique in determining the best clusters is revealed. To do this, each SMC technique was run ten times upon every benchmark program. The success rate is calculated by dividing the number of times the SMC technique finds the optimal solution by ten. In SMC approaches, stability is also examined as a reliability criterion. SMC approaches employ several heuristic algorithms. To assess the method's stability, the standard deviation (STDV) of the results produced from numerous SMC algorithm executions is employed. The lower the STDV number, the higher the system's stability.

4.2. Results

The suggested technique was tested using 10 different benchmark applications. The MQ criterion was used to assess the clusters of the proposed strategy. The MQ of the clusters generated by each MDG program is shown in Tables 4, 5, and 6. For each benchmark, the approaches were run ten times. One of the critical parameters in the SMC issue is the number of clusters in each benchmark. Experimentation is used to find the optimal cluster number. The number of clusters is influenced by several parameters, such as the number of modules, and the connection between the modules. As a result, empirical evidence is necessary. As shown in Tables 4, 5, And 6, the OOA was tested on each benchmark with a different number of clusters. The findings of the other six algorithms (GA, PSO, PSO-GA, SCSO, COA, and GWA) were studied and compared to those of the OOA. The approaches were compared in the SMC issue in terms of best MQ, average MQ, convergence speed, success

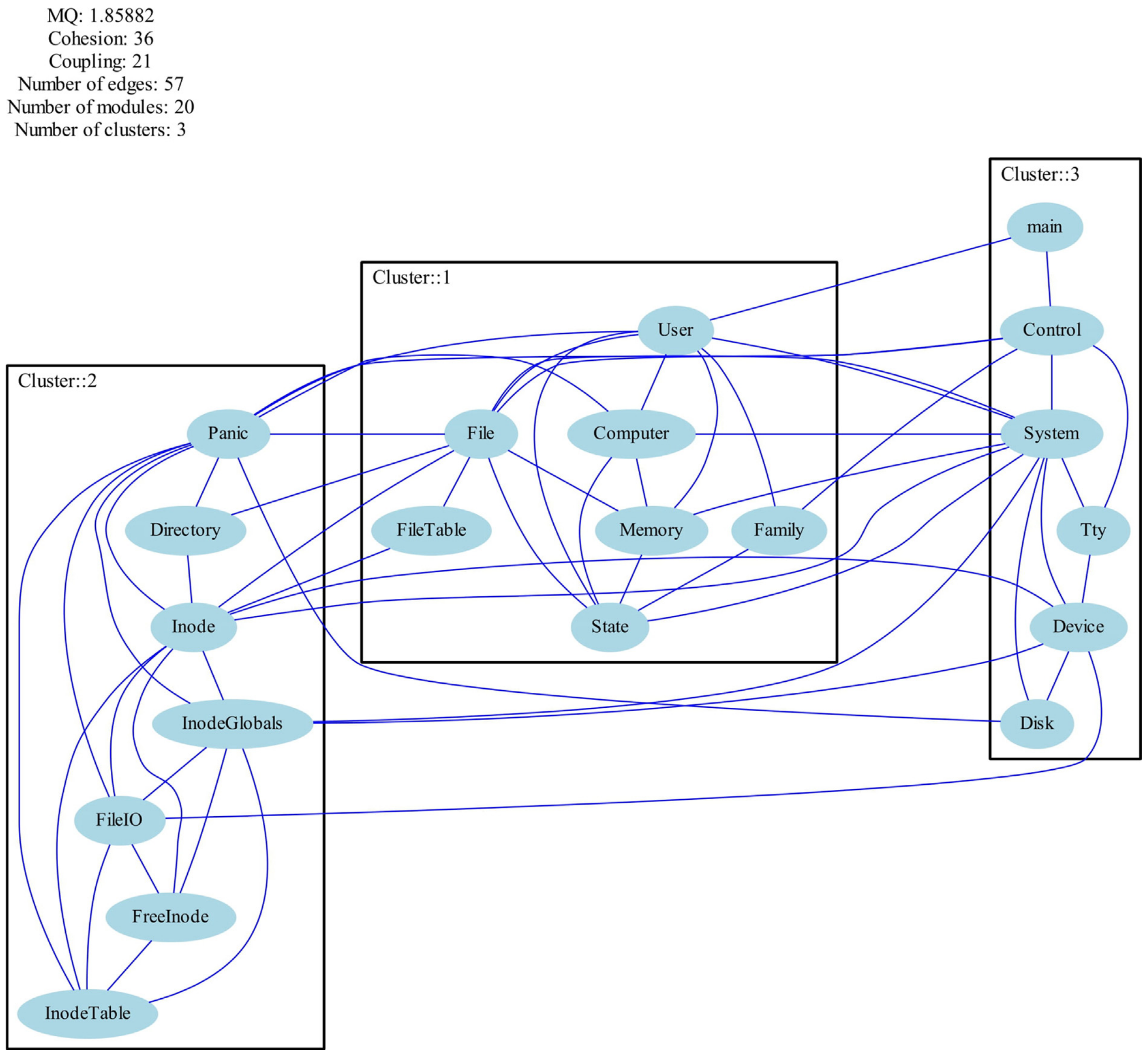

MQ: 1.85882
Cohesion: 36
Coupling: 21
Number of edges: 57
Number of modules: 20
Number of clusters: 3

Fig. 6. The clustered MDG of the mtunis software with 20 modules and 57 relations among the modules.

Table 4
The generated clusters' MQ for mtunis, SPDB, ispell and rcs by different SMC algorithms.

Benchmark		Mtunis			SPDB			ispell			rcs		
Num. of Cluster		2	3	5	2	3	5	3	5	7	3	4	5
Runs Number	1	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9369	2.1851	2.499	1.7454	1.8980	2.1499
	2	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9117	2.1943	2.2747	1.7454	1.8900	2.1420
	3	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9369	2.1851	2.2753	1.7454	1.8980	2.1233
	4	1.5788	2.0810	2.3145	2.0000	3.0000	5.0000	1.9369	2.1769	2.2922	1.7454	1.8980	2.1492
	5	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9369	2.1943	2.2675	1.7454	1.8980	2.0958
	6	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9163	2.1943	2.2747	1.7454	1.8980	2.1256
	7	1.5788	2.0579	2.3145	2.0000	3.0000	5.0000	1.9369	2.1943	2.2805	1.7454	1.8980	2.1413
	8	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9369	2.2021	2.2922	1.7454	1.8900	2.1233
	9	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9369	2.2021	2.2675	1.7454	1.8980	2.1492
	10	1.5788	2.1248	2.3145	2.0000	3.0000	5.0000	1.9369	2.2021	2.2825	1.7454	1.8980	2.1393

rate, and stability. Figs. 7, 8, and 9 demonstrate the MQ of the clusters formed by different SMC methods for different datasets. On each benchmark dataset, each method was run twenty times. Fig. 7 depicts the MQ values achieved by several SMC methods for the benchmarks mtunis, spdb, ispell, and rcs. In the manuscript, the term cost refers to the fitness (MQ) of the solution created the SMC algorithms; the fitness (MQ) is calculated by Eqs. (2) and (3).

Mtunis is a tiny MDG with 30 modules and 57 connections between them. In this benchmark, all SMC methods perform similarly. Except for COA, all methods converge to optimum solutions (clusters with optimal MQ) in the first iteration. Spdb is another little MDG used to assess the performance of the SMC algorithms. This benchmark contains 21 modules and 16 connections between them. Fig. 7 shows that OOA and SCSO outperform the other

Table 5
The generated clusters' MQ for Bison, cia, Dot and php by different SMC algorithms.

Benchmark		Bison			cia			Dot			php		
Num. of Cluster		4	6	8	4	6	8	5	8	10	5	8	10
Runs Number	1	2.1681	2.5344	2.6554	2.0274	2.3909	2.6344	2.0126	2.3694	2.5387	2.6302	3.6659	5.9414
	2	2.1681	2.5344	2.6534	2.0130	2.3102	2.5996	2.0126	2.3759	2.5643	2.8772	3.7034	5.7947
	3	2.1574	2.5344	2.6518	2.0860	2.3986	2.5660	2.0840	2.3786	2.5575	2.8615	3.6752	5.9175
	4	2.1497	2.5344	2.6539	2.0322	2.3696	2.6344	2.0113	2.4056	2.5169	2.6848	3.6254	5.4259
	5	2.1574	2.5344	2.6434	2.0322	2.3722	2.6344	2.0840	2.3103	2.5162	2.7567	3.6806	5.8425
	6	2.1427	2.5344	2.6539	2.0130	2.3696	2.6344	2.0023	2.3236	2.5120	2.8772	3.6065	6.0000
	7	2.168	2.5344	2.6052	2.0130	2.3888	2.4690	2.0451	2.42932	2.4823	2.7774	3.7072	5.8880
	8	2.1367	2.5344	2.6539	2.0860	2.3716	2.5431	2.0559	2.4006	2.4580	2.8632	3.6603	5.8126
	9	2.1380	2.5344	2.6539	2.0322	2.3179	2.5045	2.0024	2.3525	2.5393	2.4931	3.6965	5.7963
	10	2.1467	2.5344	2.6554	2.0861	2.3722	2.5565	2.0752	2.3189	2.4476	2.7890	3.6662	5.8875

Table 6
The generated clusters' MQ for grappa and Incle by different SMC algorithms.

Benchmark		grappa			Incle		
Num. of Cluster		6	9	12	6	9	12
Runs Number	1	3.8679	5.9414	8.7495	4.5038	5.5619	7.0074
	2	3.9232	5.7947	8.2371	4.2762	5.9407	7.3179
	3	3.8737	5.9175	8.4581	4.3794	5.8205	6.9797
	4	3.9048	5.4259	8.7945	4.0651	6.0689	7.2736
	5	4.0000	5.8425	8.6803	4.4647	5.6535	6.9109
	6	3.8412	6.0000	8.4771	4.2762	6.0140	7.2584
	7	3.9263	5.8880	8.4657	4.2281	5.9131	7.1733
	8	3.8232	5.8126	8.5986	4.2223	6.1532	6.9856
	9	3.8309	5.7963	8.4657	4.4358	5.9605	7.6701
	10	3.9210	5.8875	8.6979	4.9295	5.7104	7.5419

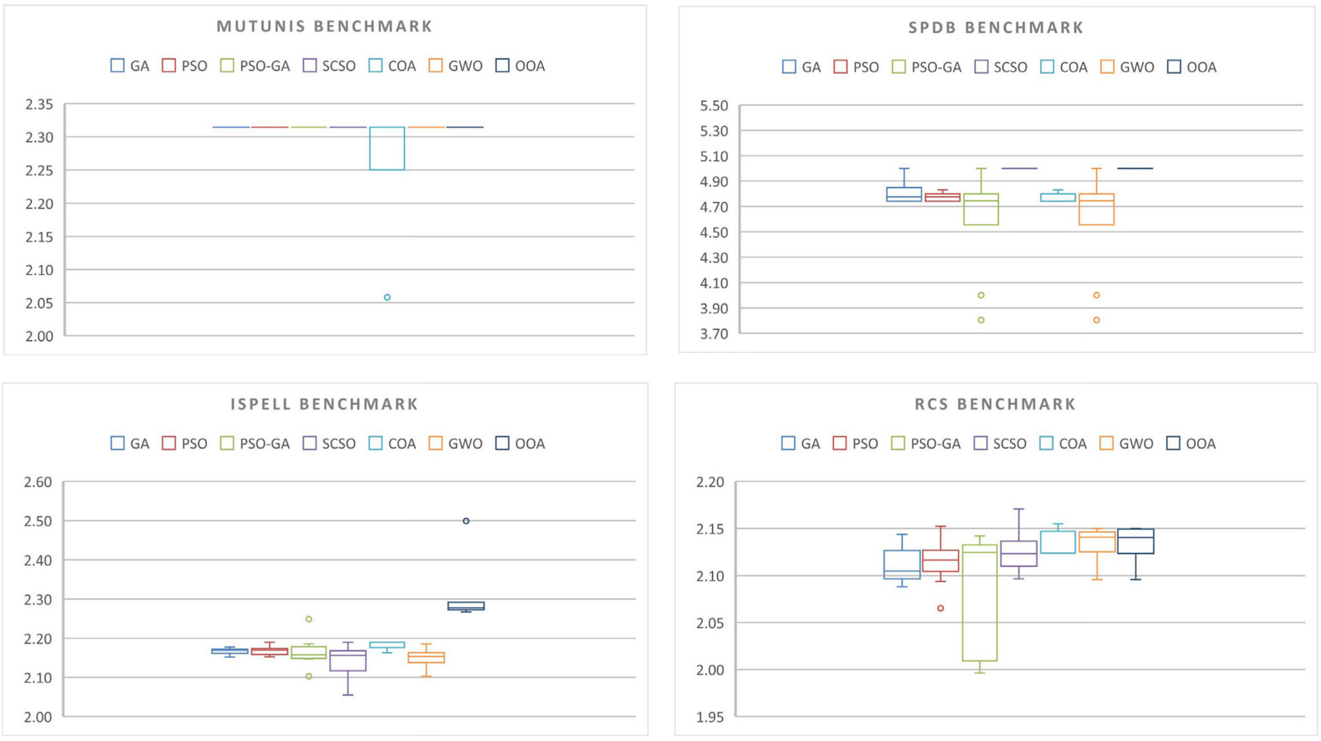


Fig. 7. The MQs of different SMC algorithms for mtunis, spdb, ispell and rcs MDGs.

algorithms in terms of MQ and success rate. In the spdb benchmark, the rate of success by the OOA and SCSO is close to 100%. Ispell is the other midsize benchmark that is used as a standard SMC benchmark. This MDG includes 24 modules and 97 connections. In this benchmark, OOA is considerably superior to the other SMC algorithms. The minimum and maximum values of MQ obtained by OOA for spell are respectively 2.267 and 2.4990. Whereas the best MQ values provided by the other algorithm are lower than the worst value of OOA. Rcs is the other benchmark MDG that consists of 29 modules and 155 connections (edges). As shown in Fig. 7, OOA outperforms the other SMC algorithms according to the MQ.

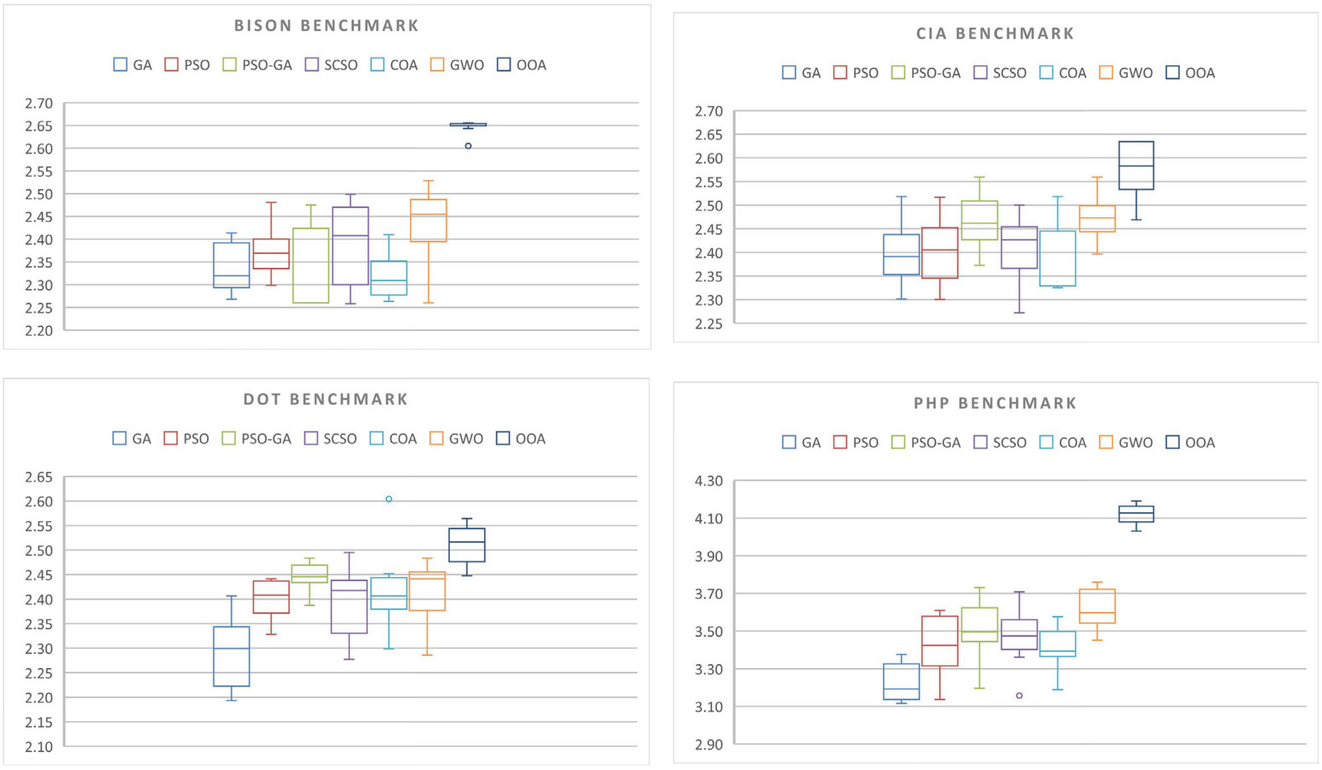


Fig. 8. The MQs of the different SMC algorithms for bison, cia, dot and php MDGs.

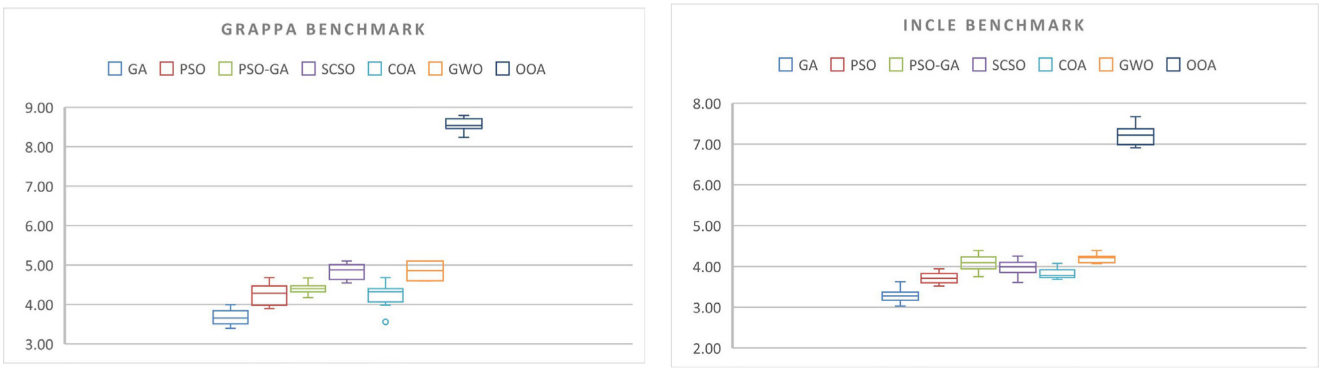


Fig. 9. The MQs of the different SMC algorithms for grappa, and incle MDGs.

Fig. 8 illustrates the performance of the SMC algorithms on the bison, cia, dot, and php benchmarks. bison, as the other MDG benchmark, includes 37 modules and 167 connections. OOA is considerably superior to the other SMC algorithms. In this benchmark, the provided MQ by the OOA is 2.6554 which is higher than the other algorithms. In this benchmark, GWA is the most efficient SMC algorithm. The other benchmark (cia) consists of 38 modules and 166 connections among the modules. The MQ of the generated clusters for the cia MDG is 2.6345. The worst MQ obtained during 10 executions is about 2.4691. Another benchmark is dot which includes 42 modules and 248 connections. Like the other benchmarks, OOA considerably outperforms the other algorithms. The average value of MQ after 10 times executions is 2.5133. One of the large benchmarks that have been used in the experiments is php; this MDG includes 62 modules and 163 connections. As shown in Fig. 8, OOA has considerably higher performance in the large benchmarks. The MQ of the generated clusters by OOA in the best case is 4.1904 which is considerably higher than the MQ of the other algorithm. After OOA, GWO could generate the highest MQ (37591). The other benchmark used for the evaluation of the OOA is grappa which includes 86 modules and 295 connections among the modules. Like the other MDG, OOA has outstanding performance in the grappas benchmark. The obtained MQ by the OOA is about 1.7 times the MQ obtained by the GWO. As showsn in Fig. 9, incle as the other benchmark MDG was used to evaluate the SMC algorithms. This MDG consists of 164 modules and 360 connections. In this benchmark, OOA generated the optimal clusters with the highest MQ.

Fig. 10 shows the average MQ obtained by the six SMC algorithms along with the MQ of the OOA. To evaluate the average MQ, each algorithm was executed ten times on each dataset (MDG). The clusters generated for the modules of the small software products by all SMC algorithms have similar quality (MQ). As shown in Fig. 10, in the mtunis and rcs benchmarks, all the algorithms generate the same quality clusters. Analysis of the results indicates that the generated clusters with the same MQ may be

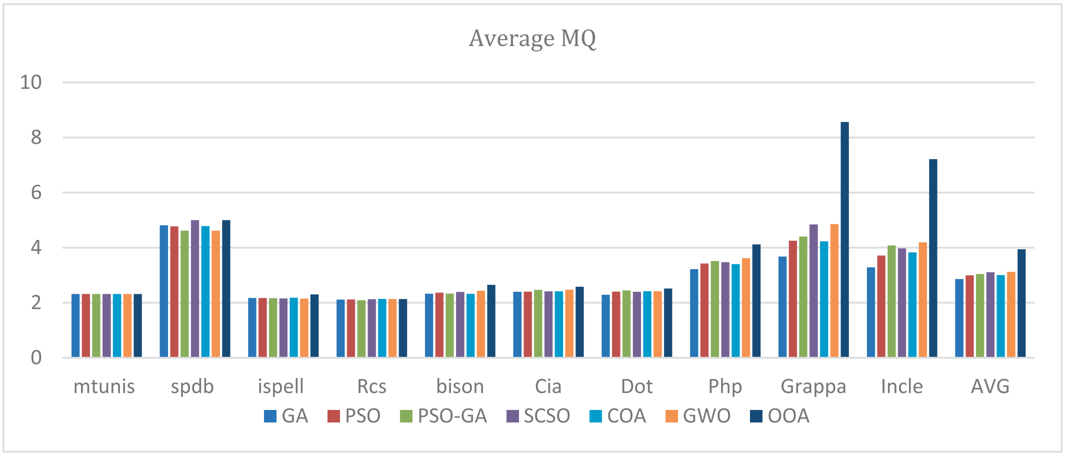


Fig. 10. The average MQ obtained by ten executions of different algorithms for ten benchmark programs.

slightly different. Determination of the best-clustered model (as a design model) from the generated clusters with the same MQ is one of the challenging problems of software engineering. The average MQ values computed from 10 executions show that OOA outperforms the other techniques. The difference in average MQ between the OOA and other methods is significant, especially in big benchmarks. The suggested OOA may be used to construct the structural model from real-world software with a significant number of code lines. Regarding the results of experiments, in all benchmarks, OOA generates more high-quality structural models than the other six algorithms.

Reliability of the results generated by the heuristic algorithms (as indeterministic algorithms) is the other performance criterion that should be considered into account. The heuristic algorithm with higher MQ may generate low-quality clusters in most cases. Hence, the standard deviation (STDV) among the generated results during different executions should be taken into consideration. To this end, the STDV of the MQs obtained from ten executions of each algorithm were calculated and shown in Fig. 11. In most benchmarks (except for incle), the STDV among the generated results of OOA is lower than the other six algorithms. The lower the STDV, the higher the reliability of the algorithm. The STADV of the results obtained by GA, PSO, PSO-GA, SCSO, COA, GWO, and OOA are respectively 0.07839, 0.07678, 0.11536, 0.09159, 0.08013, 0.1027 and 0.0752. As shown the STDV of the OOA is lower than the other algorithms. Generating high-quality clustered models and higher reliability (lower STDV) are the main merits of the OOA in SMC problem.

Table 7 displays the best MQ produced by various SMC methods across 10 runs. As seen in Table 7, OOA produces the best MQ across all benchmarks. Indeed, the OOA's performance in the SMC problem is independent of the number of MDGs. Indeed, the best MQ of the OOA is greater than the other methods in all benchmarks. In the SMC problem, OOA is an essentially discrete algorithm that outperforms the other discretized algorithms (PSO, SCSO, COA, and GWO) (as a discrete problem). The OOA-derived structural models are of greater quality (higher cohesion and lower coupling) than the models created by the other SMC methods. Table 8 demonstrates the worst-case performance of the SMC algorithms. The worst MQ of each algorithm from the generated MQs during ten executions is shown in Table 8 Except for dot benchmark in all benchmarks the MQ of the OOA is superior to the other algorithms. In all benchmarks (except for dot benchmark), the performance of OOA in the best, worst, and average modes is more than other algorithms.

The next experiments have been carried out to compare the performance of the OOA with the existing SMC tools. Bunch (Prajapati and Chhabra, 2018) and Savalan (Zadahmad et al., 2011) have been selected as the automatic SMC tools to compare with the OOA. Bunch uses GA and HC to cluster the modules of software; it takes the matrix of MDG as input and generates the clustered model by the selected algorithm (GA or HC). Savalan is

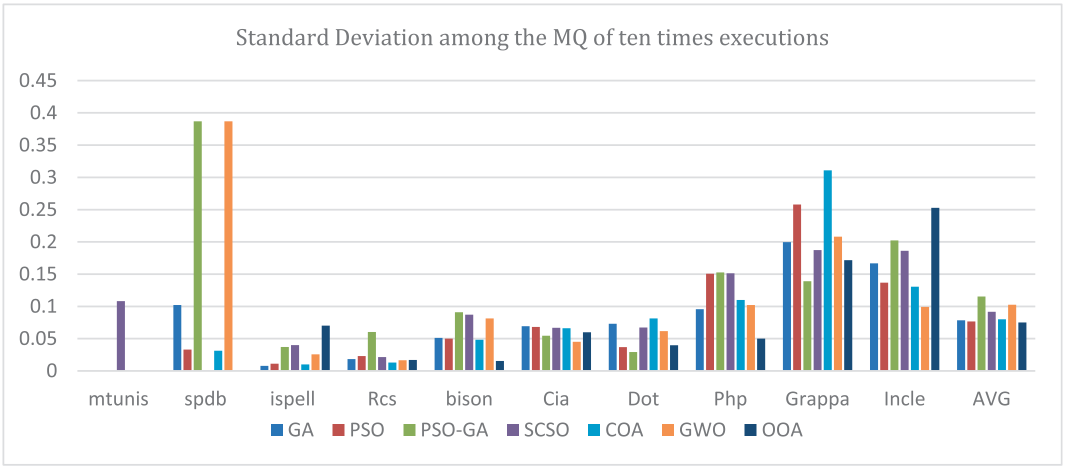


Fig. 11. The standard deviation among the MQs generated from ten times executions of different algorithms.

Table 7
The MQ of different SMC algorithms for different benchmarks in the best cases during ten executions.

	mtunis	spdb	ispell	Rcs	bison	Cia	Dot	Php	Grappa	Incle	AVG
GA	2.3140	5,0000	2.1774	2.1430	2.4135	2.5180	2.4060	3.3764	3.9930	3.6666	3.0007
PSO	2.3140	4.8300	2.1899	2.1523	2.4808	2.5160	2.4410	3.6097	4.6810	3.9411	3.1155
PSO-GA	2.3140	5,0000	2.2491	2.1420	2.4752	2.5590	2.4830	3.7310	4.6769	4.3889	3.2019
SCSO	2.3140	5,0000	2.1899	2.1708	2.4982	2.5000	2.4950	3.7083	5.1012	4.2540	3.2231
COA	2.3140	4.8300	2.1899	2.1549	2.4099	2.5182	2.6041	3.5759	4.6815	4.0731	3.1351
GWO	2.3140	5,0000	2.1899	2.1499	2.5284	2.5594	2.4836	3.7591	5.1012	4.3889	3.2474
OOA	2.3140	5,0000	2.4990	2.1499	2.6554	2.6345	2.5644	4.1904	8.7945	7.6701	4.0472
Best Alg.	All	Except for PSO, COA	OOA	COA	OOA	OOA	COA, OOA	OOA	OOA	OOA	OOA

the other web base automatic tool that was developed in JavaScript programing language. Savalan takes the MDG file and automatically generates the adjacency matrix; then clusters the software modules using multi-objective GA. Savalan generates the more practical output in the form of a clustered graph. The output of the Savalan is more understandable by the software developers. The output of the Savalan is a structural model of the input source code. The MQ of the generated models by Bunch, Savalan, and OOA was compared with each other. As shown in Fig. 12, in 80% of the benchmarks, OOA has a higher MQ value. In the php and grappa benchmarks, Savalan is superior to Bunch and OOA. The average MQ of the Bunch, Savalan, and OOA are 3.1904, 6.1977, and 6.3634 respectively. Indeed, the average performance of the OOA is higher than that of Bunch and Savalan. OOA can generate more effective and understandable structural models that can be used by software developers during the maintenance phase.

The other performance criterion of the heuristic algorithms in optimization problems is their convergence speed. The convergence speed is one of the important criteria that have been used to evaluate the performance of all heuristic algorithms in finding the optimal solution of an optimization problem. In Fig. 13, the x-axis indicates the number of repetitions of the algorithm and the y-axis indicates the fitness of the answer obtained from the algorithm. An effective algorithm obtains the best answer in the least repetition. An algorithm may fall into local optimality and cannot make progress in calculating the optimal solution. The ability of a heuristic algorithm to converge to the optimal solution is one of the evaluation criteria. Fig. 13 shows the convergence of the swarm-based and efficient algorithms (PSO, GWO, and OOA) for the small MDGs. Approximately, in the mtunis, spdb, and rcs all algorithms have the same performance in terms of MQ. In the ispell benchmark, the OOA converges to the higher MQ; indeed, in the ispell benchmark, the generated clustered model by the OOA has higher modularization quality. Overall, the performance of the proposed method in the worst case is equal to the performance of the PSO and GWO in the small program. Fig. 14 shows the convergence of the PSO, GWO, and OOA in the larger benchmark MDGs. As explained in Table 2 the used MDGs are related

Table 8
The MQ of different SMC algorithms for different benchmarks in the worst cases during ten executions.

	mtunis	spdb	ispell	Rcs	bison	Cia	Dot	Php	Grappa	Incle	AVG
GA	2.3140	4.7410	2.1516	2.0882	2.2680	2.3000	2.1934	3.1161	3.3965	3.029	2.75978
PSO	2.3140	4.7410	2.1512	2.0653	2.2986	2.3000	2.3281	3.1367	3.8985	3.519	2.875242
PSO-GA	2.3140	3.8045	2.1024	1.9963	2.2602	2.3720	2.3871	3.1963	4.1747	3.748	2.83555
SCSO	2.3140	5.0000	2.0548	2.0966	2.2580	2.2720	2.5771	3.1577	4.5456	3.6061	2.98819
COA	2.0579	4.7412	2.1627	2.1237	2.2633	2.3252	2.2970	3.1897	3.5570	3.6841	2.84018
GWO	2.3140	3.8045	2.1024	2.0958	2.2602	2.3965	2.2859	3.4512	4.5945	4.0677	2.93727
OOA	2.3140	5.0000	2.2675	2.0966	2.6052	2.4910	2.4476	4.0307	8.2371	6.9109	3.83998
Best Alg.	All	SCSO, OOA	OOA	SCSO, OOA	OOA	OOA	SCSO, OOA	OOA	OOA	OOA	OOA

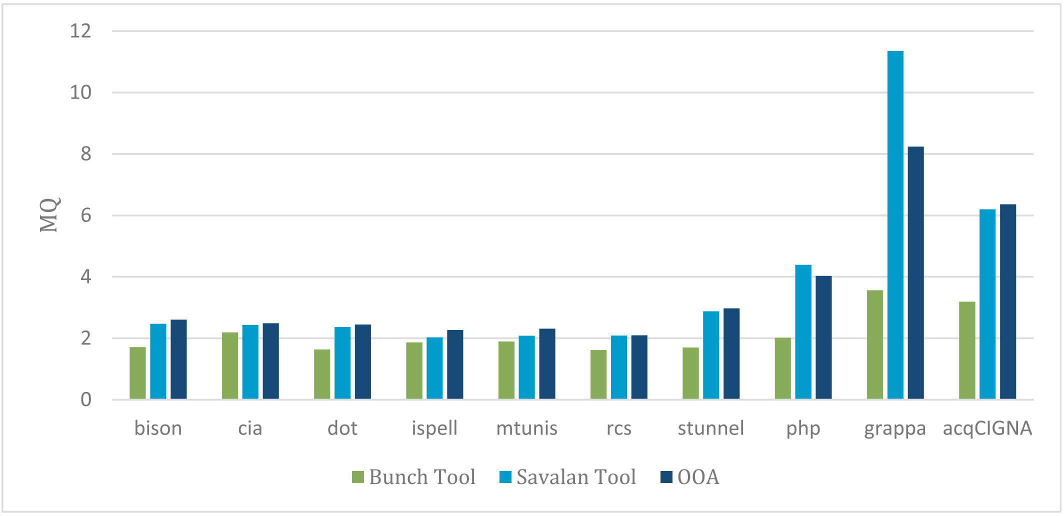


Fig. 12. Comparing the performance of the OOA with the single-objective and multi-objective SMC tools.

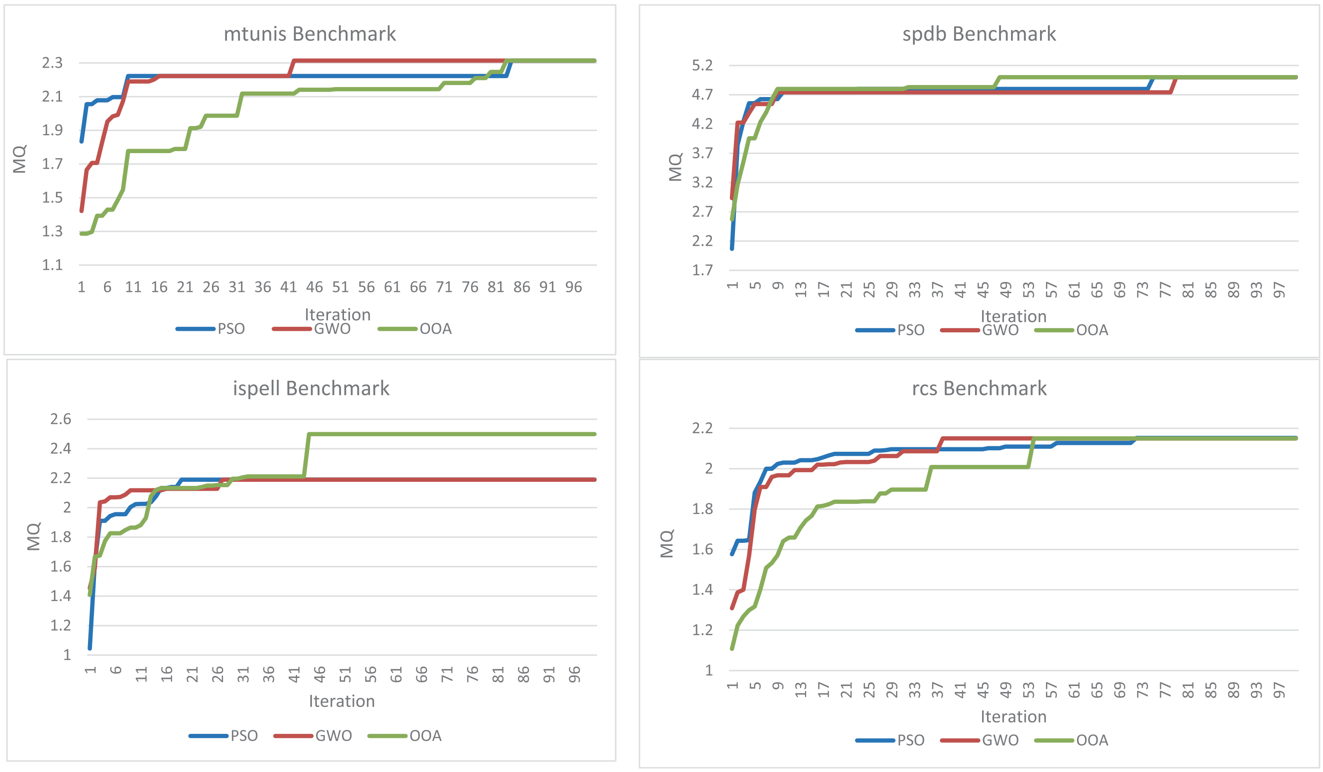


Fig. 13. Comparing the convergence speed of the PSO, GWO and OOA as three efficient swarm-based Algorithms for mtunis, spdb, ispell and rcs.

to real-world software products. In the bison benchmark, the OOA provides optimal solutions (structural models) sooner than the other algorithms. in this benchmark, the OOA attains the optimal MQ in the 41st iteration. GWO is one of the effective continuous heuristic algorithms in many of the optimization problems; but, has lower performance in the SMC problem. Similar results have been obtained in the cia benchmark. In this benchmark, the OOA converges to the optimal solution before iteration 45. Whereas the PSO and GWO never attain the solution generated by OOA. The other hard-to-understand benchmark is dot. This benchmark includes 248 connections among 42 modules. The source code and MDG of this program are hard to understand and consequently

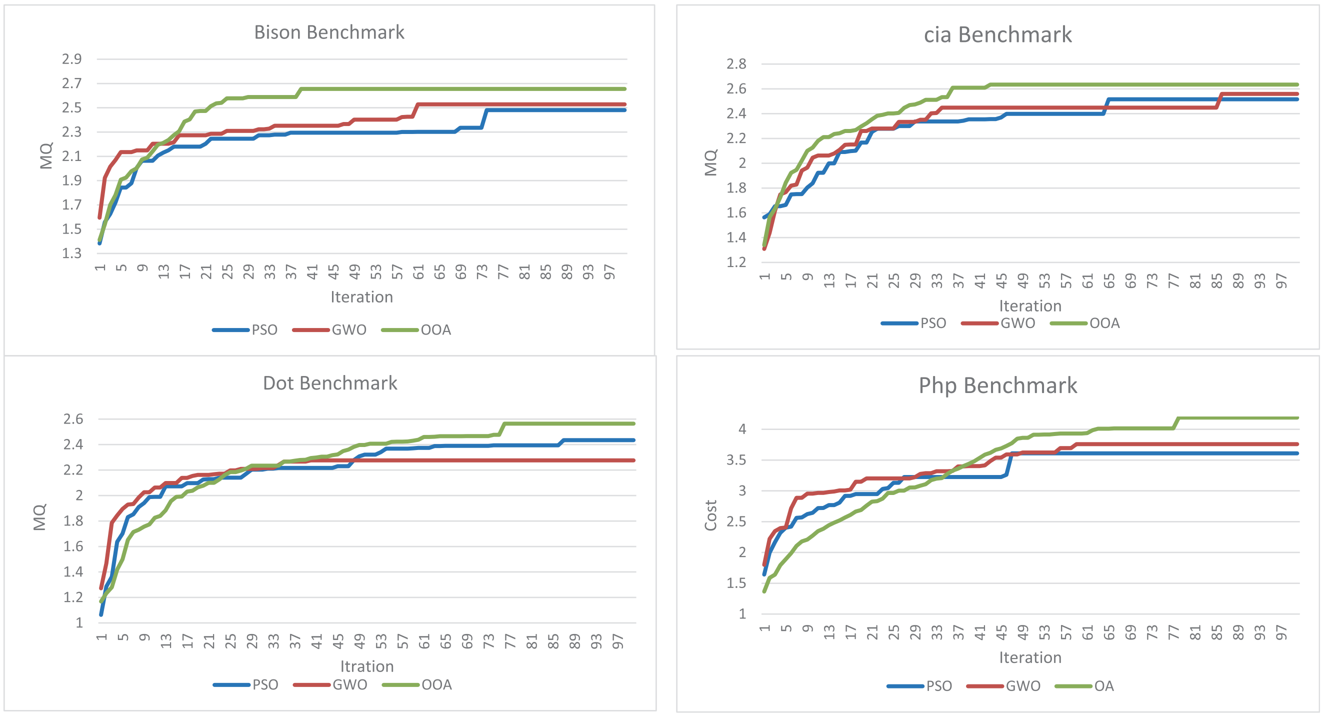
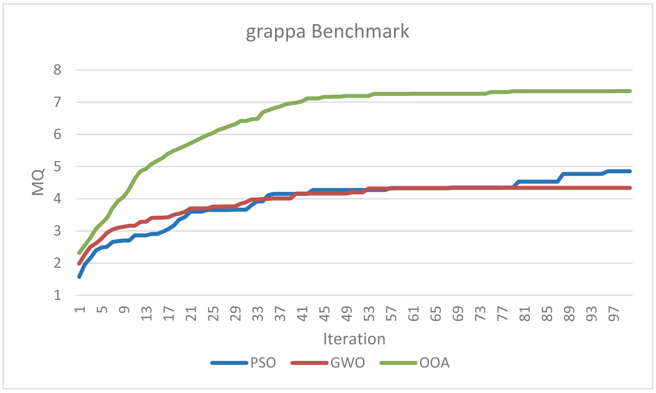


Fig. 14. Comparing the convergence speed of the PSO, GWO, and OOA as three efficient swarm-based Algorithms for bison, cia, dot and php.



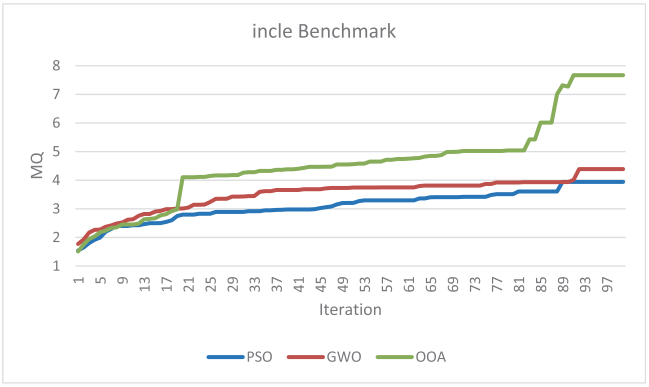


Fig. 15. Comparing the convergence speed of the PSO, GWO and OOA as three efficient swarm-based Algorithms for grappa and incle.

Table 9
The MQ and standard deviation of the generated clusters for the MDGs by the SMC Algorithms.

	PSO		PSO-GA		SCSO		COA		GWO		OOA	
	MQ	STDV	MQ	STDV	MQ	STDV	MQ	STDV	MQ	STDV	MQ	STDV
mtunis	2.3140	0.0000	2.3140	0.0000	2.3140	0.1082	2.3140	0.0000	2.3140	0.0000	2.3140	0.0000
Spdb	4.7744	0.0330	4.6180	0.3870	5.0000	0.0000	4.7844	0.0314	4.6170	0.3870	5.0000	0.0000
ispell	2.1690	0.0111	2.1660	0.0603	2.1520	0.0400	2.1800	0.0100	2.1500	0.258	2.3000	0.0702
rcs	2.1170	0.0231	2.0900	0.0910	2.1260	0.0215	2.1380	0.0129	2.1350	0.0165	2.1340	0.0171
bison	2.3640	0.0500	2.3300	0.0546	2.39.5	0.0871	2.3213	0.0483	2.4357	0.0814	2.6480	0.0154
cia	2.4026	0.0682	2.4639	0.0294	2.4106	0.0669	2.4108	0.0661	2.4735	0.0452	2.5777	0.0599
dot	2.4009	0.0370	2.4461	0.1527	2.3949	0.0672	2.4158	0.0813	2.4167	0.0617	2.5133	0.398
php	3.4232	0.1507	3.5141	0.1391	3.4743	0.1513	3.4013	0.1099	3.6171	0.1022	4.1184	0.050
grappa	4.2498	0.2580	4.4001	0.2023	4.8431	0.1874	4.2284	0.3109	4.8535	0.2080	8.5625	0.1716
incle	3.7151	0.1367	4.0812	0.2023	3.9690	0.1863	3.8288	0.1305	4.1925	0.0992	7.2119	0.2528
AVG	**2.99.0**	**0.0767**	**3.0423**	**0.1153**	**3.1074**	**0.0915**	**3.0022**	**0.0801**	**3.1205**	**0.1027**	**3.9379**	**0.6768**

hard to modify. OOA generates the optimal clustered model for this benchmark before iteration 75. Regarding the results of conducted experiments, -one of the algorithms can solve the proposed algorithm (OOA). The results of experiments on the php MDG indicate the superiority of the OOA over the other algorithms in terms of convergence. The MDGs of grappa and incle are the most complex benchmark used in this study. As shown in Fig. 15, the results obtained by OOA have considerably higher MQ than the generated models by the other algorithms.

Table 9 shows the provided MQ by different SMC algorithms and the related standard deviation. The suggested OOA is inherently discrete optimization algorithm that solves the problem in divide-and-conquer form. In this algorithm, the population is divided into subpopulations (teams) and a specific local search (learn) operator was carried out into the created teams. In global search, the individuals (students) imitate (learn) from the individuals of the best team. In each iteration, the global best team's students are the best individuals of the whole populations. The learn operator is performed among the students of the best team. At the end of each iteration, the conquer operator combines the local best students and a more elite population is produced. Hence, the knowledge (fitness) of the best students transfers to the other students like the bubble. Indeed, the bubble transmits knowledge among the population. Furthermore, the knowledge of the best students is improving by the global learn operator. The local and global learn navigates the population to the optimal solutions. Regarding the structure of the OOA, the steps of the algorithm are parallelizable. The design and implementation of the parallel OOA (POOA) is considered as the future study. In the SMC problem, as a discrete and graph-based optimization problem, OOA is considerably superior to other effective heuristic algorithms like GWO. PSO, PSO-GA, COA and SCSO. Hence, solving the other discrete graph-based optimization problems in different fields of science and industry is the other future study.

5. Conclusions

A sophisticated software system's source code may not always show its structure. Among the most difficult tasks in software engineering is identifying the impacted code parts of a source code throughout the maintenance process. By making it easier to understand how a program is put together, clustering the modules can reduce maintenance expenses. The study's purpose was to develop clustered structural models that surpassed previous techniques in terms of cohesion, coupling, and MQ value. OOA was introduced as an evolutionary discrete method for creating the clustered design model of a source code program. As benchmark programs, all small and large software applications were chosen. Six alternative clustering algorithms were created in tandem with the OOA to test their performance. The suggested OOA, as an intrinsic discrete and divide-and-conquer-based algorithm, uses local and global search operators to explore the solution space (the potential combinations of the modules clustering). In the OOA, the population was separated into subpopulations (teams) and then united with the subpopulations' selected solutions. The best individuals (students) of the created teams navigate the search process. The OOA avoids the local optimum explicitly and makes a compromise between cohesiveness and coupling. Extensive experiments were conducted utilizing ten distinct software source codes. OOA outperformed other algorithms (GA, PSO, PSO-GA, SCSO, COA, and GWO) in general, especially in big software projects in terms of MQ, cohesion, coupling, and stability. One of the future research projects that has been recommended is to build the algorithms in a way that is independent of the size of the software product. The impact of chaotic equations on OOA effectiveness may be studied further. Incorporating different swarm and evolution-based tactics into the SMC problem may yield optimal outcomes. It is suggested that future research on improving the fitness function to account for new software metrics be conducted. Even though global modules have been used as universal criteria in a recent study, they are not included in MQ. Global modules receive calls from more than two independent modules but do not make any calls themselves. Solving the other discrete graph-based optimization problems in different fields of science and industry is the other future study. Finally, the optimization methods proposed in (Arasteh et al., 2014; Keshtgar and Arasteh, 2017; Zadahmad et al., 2011; Bouyer et al., 2007; Arasteh et al., 2023; Jalali et al., 2013; Ghaemi and Arasteh, 2020; Arasteh et al., 2020; Tutsoy, 2022) can be investigated in SMC techniques.

Ethical and informed consent for data used

The data used in this research does not belong to any other person or third party and was prepared and generated by the researchers themselves during the research. The data of this research will be accessible by other researchers.

Data Availability Access

The data relating to the current study is available on the google drive and can be freely accessed by the following link: https://drive.google.com/drive/folders/1TmZWpEzmkDqWI-yjJ0EJ_fyKMMzFUB78?usp=share_link.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Amarjeet, Chhabra, J.K., 2017. Harmony search-based modularization for object-oriented software systems. Comput. Lang. Syst. Struct. 47, 153–169. https://doi.org/10.1016/j.cl.2016.09.003.

Arasteh, B., 2022. Clustered design-model generation from a program source code using chaos-based metaheuristic algorithms. Neural Comput. Appl., 1–23 https://doi.org/10.1007/s00521-022-07781-6.

Arasteh, B., Pirahesh, S., Zakeri, A., Arasteh, B., 2014. Highly available and dependable E-learning services using grid system. Procedia-Social Behav. Sci. 143, 471–476. https://doi.org/10.1016/j.sbspro.2014.07.519. ISSN 1877-0428.

Arasteh, B., Razieh, S., Keyvan, A., 2020. ARAZ: A software modules clustering method using the combination of particle swarm optimization and genetic algorithms. Intell. Decis. Technol. 14 (4), 449–462.

Arasteh, B., Razieh, S., Keyvan, A., 2020. ARAZ: A software modules clustering method using the combination of particle swarm optimization and genetic algorithms. Intell. Decis. Technol. 14, 449–462.

Arasteh, B., Sadegi, R., Arasteh, K., 2021. Bölen: Software module clustering method using the combination of shuffled frog leaping and genetic algorithm. Data Technol. Appl. 55, 251–279.

Arasteh, B., Fatolahzadeh, A., Kiani, F., 2022. Savalan: Multi objective and homogeneous method for software modules clustering. J. Softw. Evol. Proc. 34 (1), e2408.

Arasteh, B., Abdi, M., Bouyer, A., 2022. Program source code comprehension by module clustering using a combination of discretized gray wolf and genetic algorithms. Adv. Eng. Softw. 173. https://doi.org/10.1016/j.advengsoft.2022.103252. ISSN 0965-9978.

Arasteh, B., Karimi, M.B., Sadegi, R., 2023. Düzen: generating the structural model from the software source code using shuffled frog leaping algorithm. Neural Comput. Appl. 35, 2487–2502. https://doi.org/10.1007/s00521-022-07716-1.

Arasteh, B., Seyyedabbasi, A., Rasheed, J., Abu-Mahfouz, M.A., 2023. Program source-code re-modularization using a discretized and modified sand cat swarm optimization algorithm. Symmetry. 15 (2), 401. https://doi.org/10.3390/sym15020401.

Available online: http://savalan-smct.com/ (accessed on).

Bouyer, A., Arasteh, B., Movaghar, A., 2007. A new hybrid model using case-based reasoning and decision tree methods for improving speedup and accuracy. In: IADIS International Conference of Applied Computing.

Chen, Y., 1995. Reverse engineering. In: Krishnamurthy, B. (Ed.), Practical Reusable Unix Software. John Wiley & Sons, Hoboken, NJ, USA, pp. 177–208 (Chapter 6).

Chhabra, J.K., 2017. Improving the modular structure of software system using structural and lexical dependency. Inf. Softw. Technol. 82, 96–120.

Chhabra, J.K., 2017. Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. J. King Saud Univ. Comput. Inf. Sci. 29, 349–364. https://doi.org/10.1016/j.jksuci.2015.09.004.

Chhabra, A.J.K., 2018. TA-ABC: Two-archive artificial bee colony for multi-objective software module clustering problem. J. Intell. Syst. 27, 619–641. https://doi.org/10.1515/jisys-2016-0253.

Ghaemi, A., Arasteh, B., 2020. SFLA-based heuristic method to generate software structural test data. J. Softw. Evol. Proc. 30, e2228.

Hatami, E., Arasteh, B., 2020. An efficient and stable method to cluster software modules using ant colony optimization algorithm. J. Supercomput. 76, 6786–6808.

Mansour Jalali, Asgarali Bouyer, Bahman Arasteh, Maryam Moloudi, 2013. The effect of cloud computing technology in personalization and education improvements and its challenges, Proc. - Social Behav. Sci. 83, 655–658, ISSN 1877-0428, https://doi.org/10.1016/j.sbspro.2013.06.124.

Keshtgar, A., Arasteh, B., 2017. Enhancing software reliability against soft-error using minimum redundancy on critical data. Int. J. Comput. Netw. Inf. Secure. 9, 51. https://doi.org/10.5815/ijcnis.2017.05.03.

Korn, J., Chen, Y., Koutsofios, E., 1999. Chava: Reverse Engineering and Tracking of Java Applets. In: Proceedings of the Working Conference on Reverse Engineering, 6–8 October 1999; Atlanta, GA, USA.

Kumari, A.C., Srinivas, K., Gupta, M., 2013. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In": Proceedings of the 3rd IEEE International Advance Computing Conference (IACC), 22–23 February 2013, Ghaziabad, India; IEEE.

Mahdavi, K., Harman, M., Hierons, R.M., 2003. A multiple hill climbing approach to software module clustering. In: Proceedings of the International Conference on Software Maintenance, ICSM 2003, Amsterdam, The Netherlands 22–26 September 2003, IEEE, 2003.

Mamaghani, A., Hajizadeh, M., 2014. Software Modularization Using the Modified Firefly Algorithm, 8th. 7. Malaysian Software Engineering Conference (MySEC).

Mancoridis, S., Mitchell, B.S., Chen, Y., Gansner, E.R., 1999. Bunch: A clustering tool for the recovery and maintenance of software system structures. In: Proceedings IEEE International Conference on Software Maintenance 1999 (ICSM'99), 30 August–3 September 1999, Oxford, UK.

Praditwong, K., Harman, M., Yao, X., 2011. Software module clustering as a multi-objective search problem. IEEE Trans. Softw. Eng. 37, 264–282. https://doi.org/10.1109/tse.2010.26.

Prajapati, A., Chhabra, J.K., 2018. A particle swarm optimization-based heuristic for software module clustering problem. Arab. J. Sci. Eng. 43, 7083–7094. https://doi.org/10.1007/s13369-017-2989-x.

Sun, J., Ling, B., 2018. Software module clustering algorithm using probability selection. Wuhan Univ. J. Nat. Sci. 23, 93–102.

Tutsoy, O., 2022. Pharmacological, non-pharmacological policies and mutation: an artificial intelligence based multi-dimensional policy making algorithm for controlling the casualties of the pandemic diseases. IEEE Trans Pattern Anal Mach Intell. 44 (12), 9477–9488. https://doi.org/10.1109/TPAMI.2021.3127674. Epub 2022 Nov 7 PMID: 34767503.

Yuste, J., Duarte, A., Pardo, E.G., 2022. An efficient heuristic algorithm for software module clustering optimization. J. Syst. Softw. 190, 111349. https://doi.org/10.1016/j.jss.2022.111349. ISSN 0164-1212.

Zadahmad, M., Arasteh, B., Yousefzadeh Fard, P., 2011. A pattern-oriented and web-based architecture to support mobile learning software development. Procedia Soc. Behav. Sci. 28, 194–199. https://doi.org/10.1016/j.sbspro.2011.11.037.