Hindawi
Complexity
Volume 2023, Article ID 3988288, 16 pages
https://doi.org/10.1155/2023/3988288

Research Article

A Modified Horse Herd Optimization Algorithm and Its Application in the Program Source Code Clustering

Bahman Arasteh,[1] **Peri Gunes**,[2] **Asgarali Bouyer**,[1,3] **Farhad Soleimanian Gharehchopogh**,[4] **Hamed Alipour Banaei**,[5] **and Reza Ghanbarzadeh**[6]

[1]*Department of Software Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Türkiye*
[2]*Department of Computer Engineering, Istanbul Aydin University, Istanbul, Türkiye*
[3]*Department of Software Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran*
[4]*Department of Computer Engineering, Urmia Branch, Islamic Azad University, Urmia, Iran*
[5]*Department of Electronics, Tabriz Branch, Islamic Azad University, Tabriz, Iran*
[6]*Faculty of Science and Engineering, Southern Cross University, Gold Coast, Australia*

Correspondence should be addressed to Farhad Soleimanian Gharehchopogh; farhad.soleimanian@gmail.com

Received 4 May 2023; Revised 20 July 2023; Accepted 27 November 2023; Published 27 December 2023

Academic Editor: Roberto Natella

Maintenance is one of the costliest phases in the software development process. If architectural design models are accessible, software maintenance can be made more straightforward. When the software's source code is the only available resource, comprehending the program profoundly impacts the costs associated with software maintenance. The primary objective of comprehending the source code is extracting information used during the software maintenance phase. Generating a structural model based on the program source code is an effective way of reducing overall software maintenance costs. Software module clustering is considered a tremendous reverse engineering technique for constructing structural design models from the program source code. The main objectives of clustering modules are to reduce the quantity of connections between clusters, increase connections within clusters, and improve the quality of clustering. Finding the perfect clustering model is considered an NP-complete problem, and many previous approaches had significant issues in addressing this problem, such as low success rates, instability, and poor modularization quality. This paper applied the horse herd optimization algorithm, a distinctive population-based and discrete metaheuristic technique, in clustering software modules. The proposed method's effectiveness in addressing the module clustering problem was examined by ten real-world standard software test benchmarks. Based on the experimental data, the quality of the clustered models produced is approximately 3.219, with a standard deviation of 0.0718 across the ten benchmarks. The proposed method surpasses former methods in convergence, modularization quality, and result stability. Furthermore, the experimental results demonstrate the versatility of this approach in effectively addressing various real-world discrete optimization challenges.

1. Introduction

Software inevitably undergoes changes and upgrades to align with evolving user requirements and system features. The average cost associated with software maintenance, including changes and upgrades, is about 60% of total software costs [1–3]. A deeper comprehension of the software structure contributes to reducing software maintenance costs. Utilising the source code to obtain the structural model represents an efficient approach employed prior to entering the maintenance stage. A reverse engineering technique known as software source code clustering organises software modules with similar characteristics into groups. The quantity of connections within and across clusters, known as cohesion, is employed to evaluate the modularization quality (MQ) [3, 4]. The

efficiency of structural models generated by module clustering algorithms is evaluated using the MQ criterion, which posits improved clustering results from greater cohesion (internal links within a cluster) and reduced coupling.

The general research problem of this study includes the optimal partitioning of the software modules. Software module clustering (SMC) methods partition a program's source code into m clusters, each referred to as components or packages. In the formal specification of the SMC problem, S, representing the source code, consists of n modules, $M1$, $M2$, ..., and Mn, where each one encompasses functions, methods, and properties. The set π indicates various potential combinations to divide the n modules into m clusters, with each element of π representing a configuration of clustering. The quantity of potential solutions for clustering the program source code that has n modules into m clusters is denoted by Sn, m. For instance, a source code with five modules presents fifty-two distinct clustering configurations, while a source code with twenty-five modules yields 1,382,958,545 possible clustering combinations. Notably, the clusters do not intersect and the combination of these m clusters encompasses the entire source code (S). Consequently, the problem of SMC is technically categorised as NP-complete [3–5]. Therefore, the application of metaheuristic methods to select the appropriate grouping is required.

The related studies have suggested numerous metaheuristic methods to address SMC, which is considered a discrete optimisation problem [6–11]. However, earlier approaches have shown notable drawbacks, including lower MQ values, slower convergence rates, and reduced stability, especially when dealing with larger-size programs. Some of those methods also suffered from the local optima problem and have had a lower success rate. The main objectives of this study are as follows:

(i) Enhancing modularisation quality

(ii) Enhancing the likelihood of attaining optimal clusters

(iii) Enhancing the SMC method's robustness across multiple executions

(iv) Speeding up convergence for optimal cluster generation

In this study, in order to address the SMC problem, a novel discrete optimisation strategy using a robust metaheuristic algorithm, known as the horse herd optimisation algorithm (HOA), is presented in the current study. The HOA population simulates the behaviour of a group of horses at different ages and experiences. Six features determine the social behaviour of a horse in its herd. Grazing, hierarchy, sociability, imitation, defence, and roaming are the most common horse behaviour patterns used in the HOA. The reaction of horses demonstrates that they have been prey to predators; in such situations, horses attempt to protect themselves by going into the fight-or-flight mode. Horses compete for water and food to eliminate competitors. Furthermore, these animals avoid dangerous locations where dangers such as wolves exist instinctively. Each solution obtained by the HOA indicates a clustered model of software modules in the proposed method. A solution's fitness (the fitness of the generated structural model) demonstrates its MQ value. The discrete HOA may also be applied to solve a wide range of real-world discrete optimisation problems. The following are the main contributions of the current study:

(i) The discrete version of HOA was developed to address discrete optimization problems. This versatile algorithm is introduced to tackle a variety of discrete optimization challenges effectively.

(ii) The proposed method demonstrates proficiency in addressing a wide spectrum of graph-based optimization challenges.

(iii) Clustering software modules, which is considered a problem of graph partitioning in nature, is addressed.

The remainder of this paper is organised as follows. Section 2 reviews significant SMC-related research in the literature. Section 3 discusses the proposed algorithm applied in addressing the SMC problem. Section 4 discusses the tools and platforms used in executing the introduced method and includes information about the evaluation criteria and datasets relevant to SMC methods. This section also presents and analyses the current study's findings. Section 5 includes the conclusion of the paper, where recommendations for further research are also provided.

2. Related Works

Particle swarm optimisation (PSO) for selecting the best clusters of program modules was introduced in [5]. In this algorithm, every particle represents the software modules' organisational structure and is characterised by the following two essential attributes: the location vector and the speed vector. The particle's current locations in each direction are influenced by the speed vector. As the particles accelerate, their positions change. The initial population comprises these particles. To address the challenges posed by SMC, PSO was employed to manage particle locations and speed. A modified PSO approach was implemented to address the SMC problem, incorporating a local search to refine the position of particles. Test findings demonstrate that the proposed approach created higher-quality software clusters. This approach was successfully examined on a range of real-world software programs and outperformed the previous state-of-the-art method with regards to MQ, accomplishing this in remarkably swift computation complexity. However, the primary limitation of this method is its less optimal performance when applied to software products with a substantial number of module connections.

The firefly algorithm (FA) is founded on the principles of swarm intelligence and has been used to solve SMC problem [6]. Each firefly represents a clustering array with a specific brightness, signifying its quality of clustering. A fitness function (MQ) can be utilised to compute the brightness

level or fitness. To attain the optimal clustering configuration and the most precise timings, fireflies attempt to approach superior fireflies and adjust their positions. Although FA can be employed to address general problem-solving tasks, it can also be applied to specific challenges, such as the SMC problem. According to the experiment results, the FA outperforms hill climbing (HC) and the genetic algorithm (GA) on most benchmarks. The primary limitation of this approach, especially within extensive software systems, is the possibility of local optima.

The development of clustered models of the source code using an HC-based methodology is presented in [7]. The module-dependency graph (MDG) can be divided into N initial clusters due to its N module count. Each module is allocated to a cluster on a random base before the start of the HC iterations. The MQ value of the generated clusters is evaluated by the fitness function. This tactic aims to create clusters with the least number of connections and the highest degree of cohesiveness. Every hill climber aims at reaching the subsequent adjacent cluster with a higher MQ value at each level. Upon encountering a new neighbour within the emerging cluster, the hill climber attempts to locate a high-quality climber. Over time, the hills in the initial stage progressively combine to create a hill sequence. However, this approach carries the risk of converging towards local optima, making the search for the global optimum, or the optimal clustering, more challenging.

GA has effectively addressed the primary limitations of the HC approach in solving the SMC problem [8]. When employing direct search methods such as the HC algorithm, it is impossible to obtain the optimal grouping. In addition, these algorithms struggle with extensive search spaces. Conversely, the GA executes the search process simultaneously with the initial population's chromosomes. In the GA, every chromosome represents a specific clustering arrangement, randomly selected to form the initial population. During each iteration, the fitness function assesses the chromosome's quality, with those exhibiting high cohesiveness and low coupling considered high-quality entities. The application of crossover and mutation operators to update these relevant chromosomes expands the search space [9]. The results demonstrate that the GA effectively addresses the problem of SMC in small to medium software clusters.

In [10], a hybrid PSO-GA strategy was proposed to select the optimal program clusters. This method addresses issues encountered in former methods, including lower convergence, poor MQ, reduced stability, and a lower rate of success. It combines the advantages of both heuristic approaches into a single method. This hybrid strategy achieves fast convergence and enhances clustering quality in comparison with the standard PSO and GA algorithms. Crossover and mutation were utilised to enhance particle positions by updating all particles' velocity vectors. Experimental results conducted with ten popular benchmark MDGs show that the PSO-GA method performs better than the conventional approach in 90% of cases. Moreover, each strategy achieved a unique rate of success in 30% of the benchmarks, while PSO-GA exhibited superior stability compared to PSO and GA in 60% of the programs.

In [11], the software components were clustered using the ant colony optimisation (ACO) technique, where the modules within each cluster (subsystem) exhibit significant interconnections. ACO leverages swarm intelligence to solve a wide range of search-based optimisation problems. In this approach, every ant is a result of clustering. The strongest coherence and the least coupling are found in high-quality clustering. Several benchmark datasets have been used to examine this method, and the results show that ACO performed better in addressing the SMC problem. Furthermore, the ACO typically outperforms PSO and GA regarding the MQ value and the speed of convergence. ACO has the potential to solve the SMC problem in stability; however, it has lower performance for large software products.

In [12], a new method called "Bölen" was introduced for software module clustering by integrating the shuffled frog-leaping algorithm (SFLA) with GA techniques. This approach offers several advantages, including enhanced MQ, improved stability, and a higher rate of success. It employs MDG to show the interconnections between software modules in conjunction with other methods. In the context of the SMC problem, each frog represents a clustering array. The best frogs within each memeplex are created using the crossover operator applied to the least successful ones. Furthermore, each memeplex member undergoes the mutation operator intending to maximise the member with the lowest fitness. The SFLA-GA outperforms previous algorithms regarding average MQ in 80% of the benchmarks. In 90% of the cases, SFLA-GA converges to the optimal solution more rapidly than PSO, GA, and HC.

A hybrid single-objective method for solving the SMC problem was developed by combining the gray wolf optimizer (GWO) with GA [13]. This approach blends elements of evolutionary algorithms with swarm-based techniques. The traditional GWO was adapted and discretised to address the SMC problem. Experimental results conducted on fourteen widely recognised benchmarks demonstrated the superiority of this hybrid single-objective strategy over PSO, GA, and PSO-GA methods in the context of SMC. Notably, it exhibits advantages, including improved MQ and faster convergence, especially in the case of larger software programs. Several chaos-inspired algorithms were developed in [14] to address the SMC problem, including metaheuristic algorithms such as the cuckoo optimization algorithm (COA), bat algorithm (BA), black widow optimizer (BWO), teaching-learning-based optimisation (TLBO), and grasshopper optimization algorithm (GOA) methods. The effect of the chaos theory on the efficacy of different algorithms within this context has also been investigated. Real-world application results show that PSO, BWO, and TLBO approaches outperform others when tackling the SMC problem. In addition, it was observed that when these algorithms generated their initial populations employing the "logistic chaos" method, their performance improved. In the provided benchmark dataset, the average MQ values for clusters generated by PSO, BWO, and TLBO are 3.1200, 3.1550, and 2.7780, respectively.

In [15], an independent approach known as "Savalan" was introduced for addressing the SMC problem, employing a multiobjective algorithm with a unique set of objective functions. The study's main goal was to concurrently advance various objectives, including coupling, cohesion, modularisation quality, cluster size, and quantity. The optimisation objectives in this research were categorised into six distinct objective criteria. The algorithm employed in this method is the Pareto envelope-based selection algorithm (PESA), which is a multiobjective algorithm. Both small and large projects can benefit from the utilisation of this approach. The principal advantage of this strategy, as evidenced by the results from the analysis of fourteen benchmarks, is its ability to enhance all clustering objectives simultaneously. The findings indicate that Savalan improves each clustering requirement in a unified manner. In large software projects, Savalan outperforms comparable techniques such as Bunch [8], CIA [16], and Chava [17], generating more efficient clusters. This tool was developed using JavaScript.

Table 1 provides an overview of the contemporary research on complex software systems. The primary disadvantages of the previous methods, which include slow convergence, getting trapped in local optima, and inadequate stability, have been addressed in the suggested method.

3. Proposed Method

The current study introduces a modified and discretised variant of the horse herd optimisation algorithm (HOA) to tackle the SMC problem. The modified HOA can effectively address problems of discrete NP-hard optimisation, including SMC. SMC stands out as a particularly challenging optimisation problem in the software engineering domain. The ultimate objective in this context is to create the most efficient structural model from the program source code. Formally, the task of clustering n modules into m clusters can be a combinatorial problem, which involves the following two main stages: first, the construction of the MDG based on the source code and then the utilisation of the modified HOA for clustering the program's modules using the MDG. The resulting structural model helps reduce software maintenance costs.

3.1. Horse Herd Optimisation Algorithm

3.1.1. The Structure of HOA. HOA drew inspiration from the behaviours of horses at different age groups, categorising horse activities into six primary divisions with similar patterns as follows: grazing, hierarchy, socialising, imitation, defensive mechanisms, and roaming [22]. Figure 1 illustrates the flowchart of the HOA.

HOA simulates horse movement according to equation (1) in each iteration. In equation (1), $X_m^{\mathrm{Iter,AGE}}$ shows the position of the m^{th} horse, *AGE* and $\overrightarrow{V}_m^{\mathrm{Iter,AGE}}$ signify the range of age and velocity of the current horse, and *Iter* indicates the current iteration. Horses' average life span is 25–30 years, and they exhibit diverse behavioural tendencies as they age. Horses' lives are split into four age categories in the HOA as follows:

(i) δ: horses at ages 0–5
(ii) γ: horses at ages 5–10
(iii) β: horses at ages 10–15
(iv) α: horses ages 15 and older

$$X_m^{\mathrm{Iter,AGE}} = \overrightarrow{V}_m^{\mathrm{Iter,AGE}} + X_m^{(\mathrm{Iter}-1),\mathrm{AGE}}, \mathrm{AGE} = \alpha, \beta, \gamma, \delta. \quad (1)$$

HOA ranks the horses based on their best replies, and as a consequence, the 1st ten percent of horses from the top of a sorted matrix are chosen as horses. The remaining horses are divided into the following horse groups: twenty, thirty, and forty percent. Six behaviours of the horses in the herd are implemented to identify the velocity vector, and the following equation can be utilised to describe the velocity vector associated with horses (of varying ages) in each cycle of the method.

$$\begin{aligned}
\overrightarrow{V}_m^{\mathrm{Iter},\alpha} &= \overrightarrow{G}_m^{\mathrm{Iter},\alpha} + \overrightarrow{D}_m^{\mathrm{Iter},\alpha},\\
\overrightarrow{V}_m^{\mathrm{Iter},\beta} &= \overrightarrow{G}_m^{\mathrm{Iter},\beta} + \overrightarrow{H}_m^{\mathrm{Iter},\beta} + \overrightarrow{S}_m^{\mathrm{Iter},\beta} + \overrightarrow{D}_m^{\mathrm{Iter},\beta},\\
\overrightarrow{V}_m^{\mathrm{Iter},\gamma} &= \overrightarrow{G}_m^{\mathrm{Iter},\gamma} + \overrightarrow{H}_m^{\mathrm{Iter},\gamma} + \overrightarrow{S}_m^{\mathrm{Iter},\gamma} + \overrightarrow{I}_m^{\mathrm{Iter},\gamma} + \overrightarrow{D}_m^{\mathrm{Iter},\gamma} + \overrightarrow{R}_m^{\mathrm{Iter},\gamma},\\
\overrightarrow{V}_m^{\mathrm{Iter},\delta} &= \overrightarrow{G}_m^{\mathrm{Iter},\delta} + \overrightarrow{I}_m^{\mathrm{Iter},\delta} + \overrightarrow{R}_m^{\mathrm{Iter},\delta}.
\end{aligned} \quad (2)$$

Horses are grazing animals that graze at any age throughout their lives. As a consequence of the algorithm modeling the grazing zone around each horse with a "g" coefficient, every horse grazes in specified areas. The following equations show how to execute this horse behaviour mathematically.

$$\overrightarrow{G}_m^{\mathrm{Iter,AGE}} = g_{\mathrm{Iter}}\left(\check{u} + \rho\check{l}\right) + \left[X_m^{(\mathrm{Iter}-1)}\right], \mathrm{AGE} = \alpha, \beta, \gamma, \delta, \quad (3)$$

$$g_m^{\mathrm{Iter,AGE}} = g_m^{(\mathrm{Iter}-1),\mathrm{AGE}} \times \omega_g. \quad (4)$$

In equations (3) and (4), $\overrightarrow{G}_m^{\mathrm{Iter,AGE}}$ signifies the motion parameter of the i^{th} horse, demonstrating the tendency of the i^{th} horse to graze. In each cycle, this component reduces linearly with g. Upper and lower grazing space bounds are u and l, which are advised to be 1.05 and 0.95, respectively. g, the coefficient, is also advised to be 1.5 for all four age groups. Horses live their lives intending to follow a leader, who might be the most experienced and powerful horse in their herd or even a human. HOA considers this propensity to be a hierarchy, and it is represented by the coefficient h. Horses have been seen to obey the rule of hierarchy during the middle ages (aged 5–15 years). The following equations are used to represent this behaviour quantitatively.

$$\overrightarrow{H}_m^{\mathrm{Iter,AGE}} = h_m^{\mathrm{Iter,AGE}}\left[X_*^{(\mathrm{Iter}-1)} - X_m^{(\mathrm{Iter}-1)}\right], \mathrm{AGE} = \alpha, \beta \text{ and } \gamma, \quad (5)$$

$$h_m^{\mathrm{Iter,AGE}} = h_m^{(\mathrm{Iter}-1),\mathrm{AGE}} \times \omega_h. \quad (6)$$

Table 1: Prior methods and tools and their specification.

Reference	Method	Fitness function
[5]	PSO	Single objective
[6]	Firefly	Single objective
[7]	Hill climbing	Single objective
[8]	GA	Single objective
[9]	Two-archive genetic	Multiobjective
[10]	PSO-GA	Single objective
[11]	ACO	Single objective
[12]	SFLA-GA	Single objective
[13]	Hybrid gray wolf	Hybrid single objective
[14]	Chaos-based metaheuristic method	Chaos-based single objective
[15]	PESA-GA	Multiobjective
[17]	Java reverse engineering tool	Single objective
[18]	Hyper-heuristic method	Multiobjective
[19]	Object-oriented method	Multiobjective
[20]	Two-archive ACO	Multiobjective
[21]	Sand Cat swarm	Single objective

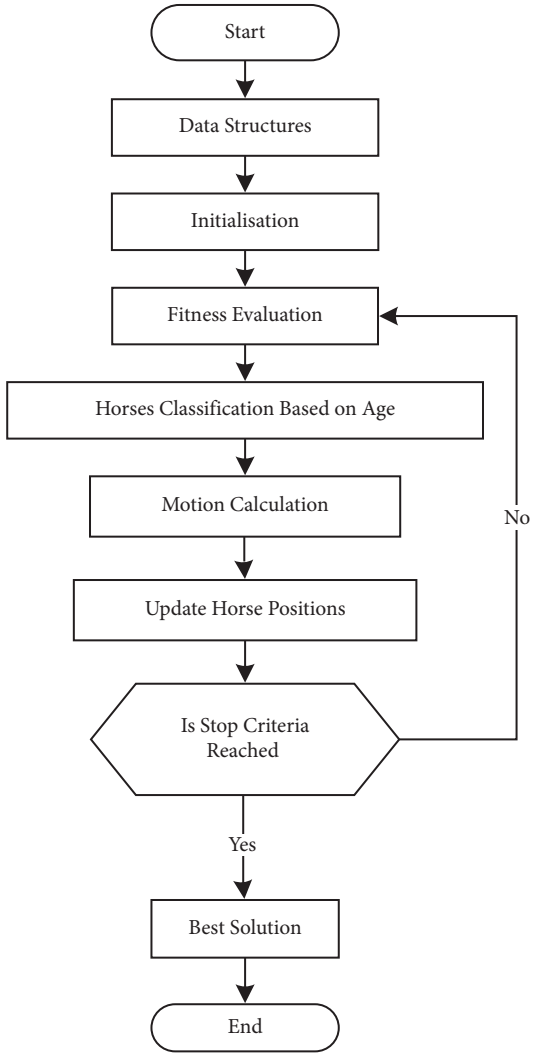


Figure 1: Flowchart of HOA.

In equations (5) and (6), $\overrightarrow{H}_m^{\text{Iter,AGE}}$ and $X_*^{(\text{Iter}-1)}$ are the impact of the best horse's location on the velocity and the best horse's location, respectively. Horses naturally need social contact and often coexist with other animals. The presence of a herd provides protection against predators, increasing their chances of survival and making them less vulnerable. Occasionally, horses may engage in conflicts with each other due to their social nature and their individuality contributes to occasional irritability. Observations have shown that horses tend to thrive when in the company of other animals. Horses between the ages of 5 and 15 typically prefer staying with the herd. This behaviour is characterised by a tendency to move towards the average positions of other horses, and HOA simulates and reflects this tendency with the variable S, as described in the following equations:

$$\%\ \overrightarrow{S}_m^{\text{Iter,AGE}} = s_m^{\text{Iter,AGE}}\left[\left(\frac{1}{N}\sum_{j=1}^{N} X_j^{(\text{Iter}-1)}\right) - X_m^{(\text{Iter}-1)}\right], \text{AGE} = \beta, \gamma, \tag{7}$$

$$S_m^{\text{Iter,AGE}} = s_m^{(\text{Iter}-1),\text{AGE}} \times \omega_s. \tag{8}$$

In equations (7) and (8), $\overrightarrow{S}_m^{\text{Iter,AGE}}$ and $s_m^{\text{Iter,AGE}}$ are social motion vector of i^{th} horse and its orientation towards the herd in the Iter^{th} iteration, respectively. With the ω_s component, the orientation towards the herd reduces in each iteration. The number N is the total quantity of horses in the herd. Horses copy one another and pick up both positive and bad behaviours and routines, such as looking for the greatest grazing place. Juvenile horses copy others, and this behaviour persists until they reach maturity. This behaviour, which is likewise motivated by HOA and shown by i, may be modeled using the following equations:

$$\overrightarrow{I}_m^{\mathrm{Iter,AGE}} = i_m^{\mathrm{Iter,AGE}}\left[\left(\frac{1}{pN}\sum_{j=1}^{\mathrm{pN}}\widehat{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right], \mathrm{AGE} = \gamma, \tag{9}$$

$$i_m^{\mathrm{Iter,AGE}} = i_m^{(\mathrm{Iter}-1),\mathrm{AGE}} \times \omega_i. \tag{10}$$

In equations (9) and (10), $\overrightarrow{I}_m^{\mathrm{Iter,AGE}}$ indicates the movement vector of i^{th} horse in the top horse's direction with $\widehat{X}$ locations. pN is the quantity of horses located in the best locations. p is considered as ten percent of the total quantity of horses, and ω_i indicates a decreasing factor per cycle for i_{Iter}. Horses display a fight-or-flight reaction in response to danger. Their primary instinct is to run. They also buck when caught. Horses instinctively compete for food and water to avoid competition and potentially dangerous environments, such as wolves. According to equations (11) and (12), horses' defence mechanism acts in the HOA by fleeing away from people who exhibit incorrect behaviours and the d factor describes it. This behaviour is presented as a negative coefficient to keep horses away from inappropriate postures.

$$\overrightarrow{D}_m^{\mathrm{Iter,AGE}} = -d_m^{\mathrm{Iter,AGE}}\left[\left(\frac{1}{qN}\sum_{j=1}^{\mathrm{pN}}\check{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right], \mathrm{AGE} = \alpha, \beta \text{ and } \gamma, \tag{11}$$

$$d_m^{\mathrm{Iter,AGE}} = d_m^{(\mathrm{Iter}-1),\mathrm{AGE}} \times \omega_d. \tag{12}$$

In equations (11) and (12), $\overrightarrow{D}_m^{\mathrm{Iter,AGE}}$ signifies the i^{th} horse's escape factor from the average of horses with the worst position presented by the X vector. qN is the total quantity of horses that have the worst position. ω_d is the decrement factor per d_{Iter}, and q is recommended to be considered twenty percent of the total quantity of horses. The sixth horse behaviour influenced by HOA is their wandering attitude. A horse may occasionally choose to graze in a different spot. Horses are inquisitive creatures who want to explore new pastures and get familiar with their environment. Stable walls are often built so that horses can see each other and satisfy their curiosity. Roaming occurs in young horses, and this behaviour fades as they develop. According to the following equations, the HOA simulated this behaviour as a random movement, which is denoted by factor r.

$$\overrightarrow{R}_m^{\mathrm{Iter,AGE}} = r_m^{\mathrm{Iter,AGE}} pX^{(\mathrm{Iter}-1)}, \mathrm{AGE} = \gamma \text{ and } \delta, \tag{13}$$

$$r_m^{\mathrm{Iter,AGE}} = r_m^{(\mathrm{Iter}-1),\mathrm{AGE}} \times \omega_r. \tag{14}$$

In equations (13) and (14), $\overrightarrow{R}_m^{\mathrm{Iter,AGE}}$ is the i^{th} horse's random velocity vector to local search. ω_r is the decrement factor of $r_m^{\mathrm{Iter,AGE}}$ per iteration. The velocity vector is determined by applying equations (3)–(14) and (2), and the velocities of the δ, γ, β, and α horses are computed using equations (15)–(18), respectively.

$$\begin{aligned}\overrightarrow{V}_m^{\mathrm{Iter},\delta} = {} & \left[g_m^{(\mathrm{Iter}-1),\delta}\,\omega_g\,(\breve{u}+\rho\breve{l}) + \left[X_m^{(\mathrm{Iter}-1)}\right]\right] + \left[i_m^{(\mathrm{Iter}-1),\delta}\,\omega_i\left[\left(\frac{1}{pN}\sum_{j=1}^{\mathrm{pN}}\widehat{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right]\right] \\ & + \left[r_m^{(\mathrm{Iter}-1),\delta}\,\omega_r pX^{(\mathrm{Iter}-1)}\right],\end{aligned} \tag{15}$$

$$\begin{aligned}\overrightarrow{V}_m^{\mathrm{Iter},\gamma} = {} & \left[g_m^{(\mathrm{Iter}-1),\gamma}\,\omega_g\,(\breve{u}+\rho\breve{l}) + \left[X_m^{(\mathrm{Iter}-1)}\right]\right] + \left[h_m^{(\mathrm{Iter}-1),\gamma}\,\omega_h\left[X_*^{(\mathrm{Iter}-1)} - X_m^{(\mathrm{Iter}-1)}\right]\right] \\ & + \left[s_m^{(\mathrm{Iter}-1),\gamma}\,\omega_s\left[\left(\frac{1}{N}\sum_{j=1}^{N}X_j^{(\mathrm{Iter}-1)}\right) - X_m^{(\mathrm{Iter}-1)}\right]\right] + \left[i_m^{(\mathrm{Iter}-1),\gamma}\,\omega_i\left[\left(\frac{1}{pN}\sum_{j=1}^{\mathrm{pN}}\widehat{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right]\right] \\ & - \left[d_m^{(\mathrm{Iter}-1),\gamma}\omega_d\left[\left(\frac{1}{qN}\sum_{j=1}^{\mathrm{pN}}\check{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right]\right] + \left[r_m^{(\mathrm{Iter}-1),\gamma}\,\omega_r pX^{(\mathrm{Iter}-1)}\right],\end{aligned} \tag{16}$$

$$\overrightarrow{V}_m^{\mathrm{Iter},\beta} = \left[g_m^{(\mathrm{Iter}-1),\beta}\,\omega_g\,(\check{u}+\rho\check{l}) + \left[X_m^{(\mathrm{Iter}-1)}\right]\right] + \left[h_m^{(\mathrm{Iter}-1),\beta}\,\omega_h\left[X_*^{(\mathrm{Iter}-1)} - X_m^{(\mathrm{Iter}-1)}\right]\right] + \left[s_m^{(\mathrm{Iter}-1),\beta}\,\omega_s\left[\left(\frac{1}{N}\sum_{j=1}^{N} X_j^{(\mathrm{Iter}-1)}\right) - X_m^{(\mathrm{Iter}-1)}\right]\right] - \left[d_m^{(\mathrm{Iter}-1),\beta}\omega_d\left[\left(\frac{1}{qN}\sum_{j=1}^{\mathrm{pN}} \check{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right]\right], \tag{17}$$

$$\overrightarrow{V}_m^{\mathrm{Iter},\alpha} = \left[g_m^{(\mathrm{Iter}-1),\alpha}\,\omega_g\,(\check{u}+\rho\check{l}) + \left[X_m^{(\mathrm{Iter}-1)}\right]\right] - \left[d_m^{(\mathrm{Iter}-1),\alpha}\omega_d\left[\left(\frac{1}{qN}\sum_{j=1}^{\mathrm{pN}} \check{X}_j^{(\mathrm{Iter}-1)}\right) - X^{(\mathrm{Iter}-1)}\right]\right]. \tag{18}$$

3.2. Adaptation of HOA to Address the Problem of SMC

3.2.1. Specification of the Problem. The discrete HOA generated in this study was employed in the second phase to address the SMC problem. The proposed method organises related modules using the MDG derived from the program source code of the benchmark application. Therefore, the input for the HOA consists of MDGs extracted from the program source code of the benchmark. Figure 2 illustrates the extracted MDG from the program source code. The nodes in Figure 2 signify the software modules, and the edges represent their connections, including calls, inheritance, and associations. The figure also shows six software product components along with the corresponding dependency matrix. Values within the matrix entries indicate the module connections. The dependency matrix is constructed based on the information gathered from the MDG. Various integrated development environments such as Visual Studio, VS Code, and Eclipse can automatically generate the MDG from source code. To transform the MDG text file into a visual graph model, the Graphviz library can be employed.

Each array, referred to as a clustering array, corresponds to a horse in HOA. Figure 3 illustrates the structure of the horse's involvement in the problem of SMC, along with the associated clustering MDG. Each array's length matches the quantity of modules in the source code. The array's indices represent module numbers, and each index's values indicate the SMC technique's cluster assignment. As depicted in Figure 3, the clustering array is designed for clustering a source code with twenty-two modules consisting of the same number of cells. Modules "*M*1," "*M*14," "*M*18," and "*M*22" are grouped together in cluster 1 due to module dependencies within the source code. Cluster 2 of the MDGs contains "*M*2," "*M*13," "*M*16," and "*M*21." The SMC techniques aim to group those modules that share similarities and dependencies into the same cluster. Consequently, any modifications made to a module's code within a cluster will probably influence other modules within the same cluster. This can help program developers manage the propagation of impact when modifying the source code of a module.

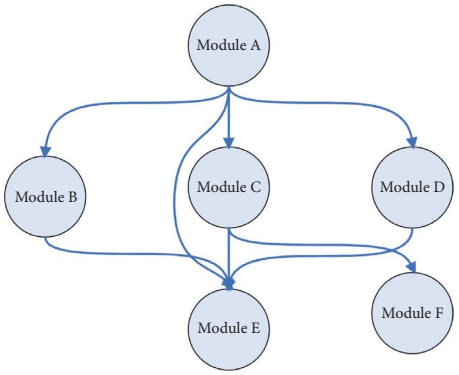
3.2.2. The Objective Function. Using the recommended strategy, MQ was employed as a quality factor to guide the population. The HOA utilises this factor to direct its search for optimal clusters. The authors in reference [8] introduced this factor as a metric for assessing the quality of clustering. The high-quality clusters show strong cohesion and minimum coupling, which is an indication of a well-designed cluster with module components that are tightly interconnected. The technique of evaluating clustering quality (MQ) for a specific cluster, denoted as k, is presented in equation (19), where i is the count of internal connections within the cluster and j signifies the count of external connections. Equation (20) is utilised to assess all created clusters' overall quality, with m indicating the number of clusters. The MQ function strives to strike a balance between intracluster cohesion and intercluster coupling, thereby assessing the quality of clustering. Enhancing the cohesion of individual modules within a cluster is essential while maintaining an optimal level of coupling. In an ideal scenario, a single cluster encompassing all modules might be preferred but a tradeoff between coupling and cohesion is necessary in practice.

$$\mathrm{MFk} = \begin{cases} 0, & \text{if } i = 0, \\ \dfrac{i}{i + (1/2)j}, & \text{if } i > 0, \end{cases} \tag{19}$$

$$\mathrm{MQ} = \sum_{k=1}^{m} \mathrm{MFk}. \tag{20}$$

4. Evaluation System

4.1. Experimental Platform. To assess the performance of the new algorithm, a large quantity of tests was conducted on the developed platform. The suggested HOA-based approach, along with the other compared approaches, PSO, GA, PSO-GA, COA, and SCSO, were implemented in MATLAB to facilitate comparison, and throughout the testing, the parameters of all of these algorithms were adjusted. Table 2 outlines the optimal values of these parameters concerning the problem of SMC. All experiments were executed on the same platform, encompassing both software and hardware, to achieve reliable results. The testing was carried out using ten conventional and real-world benchmark MDGs. The benchmark programs' parameters are detailed in Table 3. These benchmarks were chosen to represent real-world complexities regarding module nodes and their



Dependency Matrix Created From MDG

	A	B	C	D	E	F
A	0	1	1	1	1	1
B	0	0	0	0	1	0
C	0	0	0	0	1	1
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Figure 2: The MDG and its dependency matrix.

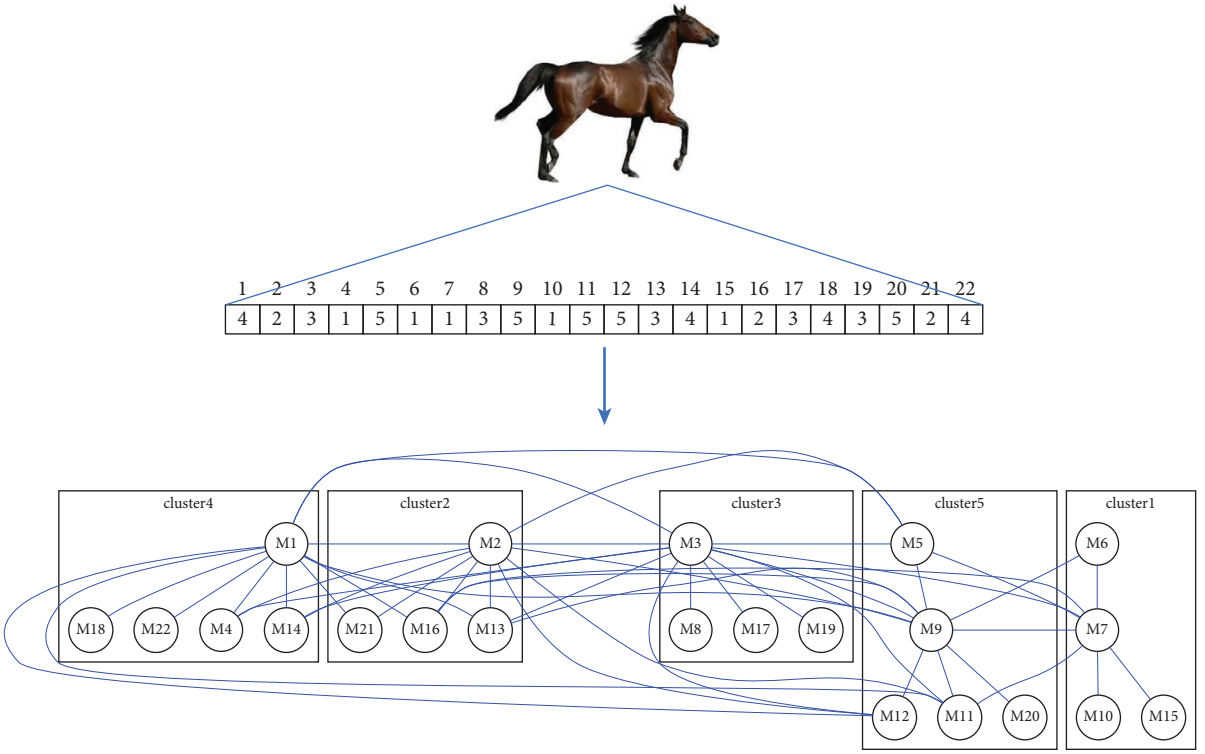


Figure 3: The structure of a horse in HOA used to cluster a program with 22 modules into 5 clusters.

connections (edges). Figure 4 visually represents the MDG for the *cia* benchmark used in this study. This software consists of 38 modules interconnected by 636 connections, with closely related modules grouped together. The fitness function, defined by equation (20), assesses the degree of similarity among modules. The proposed technique aims to cluster modules with the highest degree of similarity into the same cluster. The MQ, cohesion, and coupling are 2.6089, 49, and 119, respectively, as depicted in Figure 4. It is worth noting that a higher MQ factor indicates a better quality of clustering.

Throughout the experiments, various criteria of performance were assessed. The primary performance measure for SMC approaches is modularization quality (MQ), which serves as the key indicator of cluster quality. All SMC algorithms aim to achieve clusters that have the highest MQ value. Equation (20) was applied to evaluate the MQ value of a cluster created for an MDG. MQ tends to be higher in clusters that have lower coupling and greater cohesion. The determination of the optimal number of clusters is typically based on empirical evidence. Another critical performance aspect examined is the convergence speed, which refers to the length of time taken by an SMC technique to identify the optimal grouping, directly influencing the rate of convergence. The rate of success is the other significant performance requirement for SMC methods, representing the method's ability to identify the best clusters. This is determined by executing each SMC approach ten times for each program and computing the rate of success by dividing

Table 2: Various SMC algorithms' parameters that are adjusted experimentally.

Algorithms	Parameters	Value
GA	Quantity of chromosomes	40
	Chromosome length	Depends on the quantity of modules
	Rate of crossover	0.80
	Rate of mutation	0.05
	Quantity of iterations	100
PSO	Quantity of particles	40
	Inertia weight	0.80
	Damping ratio of inertia weight	0.99
	Particle.C1 and Particle.C2	[1.5, 1.7]
	Number of iterations	100
SCSO	Quantity of cats	40
	Range of sensitivity (rG)	[0, 2]
	Range of phases control (R)	[−2rG, 2rG]
	P_c	0.80
	P_m	0.04
COA	Quantity of nests	40
	Lavy distribution parameter	1.50
	Length of step	0.01
	Number of iterations	100
HOA	Quantity of horses	40
	h_β	0.90
	h_y	0.50
	S_β	0.20
	S_y	0.10
	d_β	0.20
	S_y, R_δ	0.10
	R_y	0.05

Table 3: Programs' specifications as benchmarks.

Programs	Number of modules	Number of connection	Size
"*mtunis*"	20.0	57.0	Small
"*spdb*"	22.0	16.0	Small
"*ispell*"	24.0	97.0	Small
"*rcs*"	29.0	155.0	Mid
"*bison*"	37.0	117.0	Mid
"*cia*"	38.0	216.0	Large
"*dot*"	42.0	248.0	Large
"*php*"	62.0	636.0	Large
"*grappa*"	86.0	252.0	Large
"*incle*"	174.0	360.0	Large

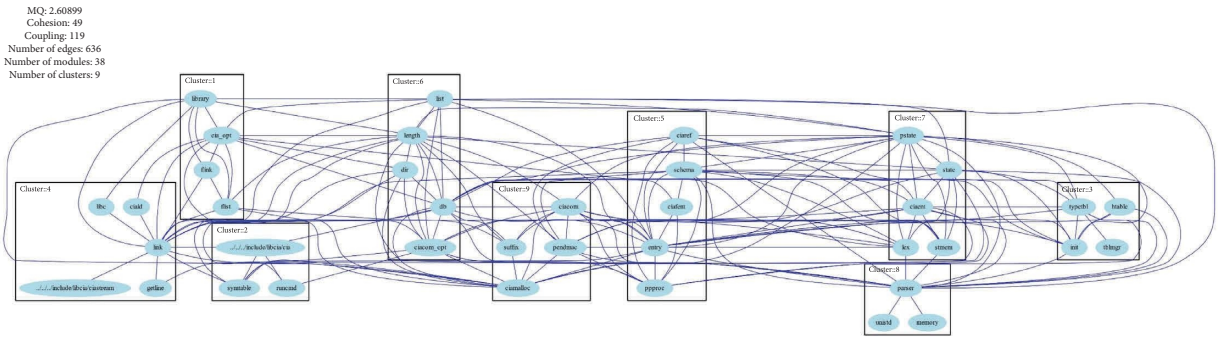


Figure 4: The clustered MDG of *cia* with 38 modules and 636 connections between the modules.

the count of times the ideal solution is achieved by 10. In addition, stability is considered a reliability factor in SMC techniques. Given the utilisation of various metaheuristic algorithms in SMC techniques, the standard deviation among the results achieved from multiple executions of the SMC algorithms is utilised to assess the method's stability. A lower standard deviation value indicates a more stable system.

4.2. Results. Ten distinct benchmark applications were employed to evaluate the proposed methodology. The quality of the clusters produced by the suggested strategy was examined using the MQ criteria. Each of the techniques was executed ten times for every program. Determining the optimal quantity of clusters for each program is a crucial parameter in the problem of SMC, and this necessitates empirical investigation. Several factors, including the number of modules and the intermodule communication, influence the determination of the appropriate number of clusters. Therefore, empirical data are essential. The HOA was applied to each program with a variable number of clusters, and the results were compared to those of the other five algorithms (PSO, GA, PSO-GA, SCSO, and COA). In the context of the problem of SMC, the techniques were assessed on the basis of criteria such as the best MQ value, average MQ value, convergence speed, success rate, and stability. The MQ values of clusters generated by various SMC algorithms for different datasets are depicted in Figures 5–7. Each approach was executed twenty times on each benchmark dataset.

Figure 5 displays the MQ values for the *mtunis*, *rcs*, *spdb*, and *ispell* benchmarks obtained through various SMC approaches. *mtunis* is a small MDG that has 30 modules and 57 connections, on which all of the SMC approaches perform similarly. All approaches converge to optimal solutions (clusters with the best MQ value) in the 1st iteration, except for COA. *Spdb*, the other smaller MDG, consists of 21 modules with 16 connections. As shown in Figure 5, HOA and SCSO excel in MQ and the rate of success, with the success rate of HOA and SCSO in the *spdb* benchmark reaching nearly 100%. The other midsize benchmark frequently used as a standard in SMC benchmarks is *ispell*, composed of 24 modules and 97 connections. The HOA significantly outperforms other SMC algorithms, with the lowest and highest MQ values obtained by it for *ispell* being 2.2675 and 2.2922, respectively. Notably, the best MQ values achieved by other algorithms are lower than HOA's worst result. The last benchmark MDG, *rcs*, includes 29 modules and 155 connections (edges). According to Figure 5, the HOA shows better performance than the other SMC approaches in the value of MQ in this benchmark.

Figure 6 presents SMC algorithms' performance on the *bison*, *cia*, *dot*, and *php* benchmarks. *Bison*, the second millennium development goal benchmark, comprises 37 modules and 167 connections. HOA significantly outperforms the other SMC methods, achieving an MQ of 2.6554 in this benchmark, surpassing the MQ of the other techniques. GA proves to be the second efficient SMC algorithm in this test. Another benchmark, *cia*, has 38 modules and 166 connections. The best and worst MQs for this benchmark obtained by HOA in ten executions are approximately 2.5565 and 2.4171, respectively. *dot* is the other benchmark with 42 modules and 248 connections. In line with the previous benchmarks, HOA performs significantly better than the other methods on *dot* benchmark. The *php* benchmark, containing 62 modules and 163 connections, is one of the key benchmarks used in the study. Figure 7 illustrates that HOA excels significantly in one of major benchmarks (*incle*), with the MQ of clusters generated by HOA reaching 4.093 in the best case, surpassing the MQ of other techniques. Following HOA, PSO-GA and SCSO also produce substantial MQ results. *grappa*, featuring 86 modules and 295 connections, serves as another benchmark used to test the performance of HOA. In the *grappa* benchmark, HOA performs impressively, in line with its performance in other MDGs.

Figure 8 presents the best MQ value achieved by the five compared SMC algorithms and the MQ achieved by HOA. Each method was executed 10 times on each dataset to determine the average MQ for MDGs. For small software packages (mtunis, ispell, and rcs), all SMC algorithms generate clusters of comparable quality in terms of MQ. As shown in Figure 8, all algorithms produce clusters with the same quality in the *mtunis* and *rcs* benchmarks although there may be slight differences among the created clusters with the same MQ, which presents a challenge in software engineering when selecting the best-clustered model among them. HOA surpasses other strategies in terms of the average MQ values obtained from 10 executions, with a significant difference in large benchmarks. The discrete HOA demonstrates its potential in creating structural models for real-world software that has a substantial amount of code. According to experimental results, HOA consistently outperforms the other five algorithms in all benchmarks. Figure 9 illustrates the worst MQ achieved by HOA and the best MQ obtained by the other SMC methods. Each method was executed 10 times on each MDG dataset. For small software packages (such as *mtunis*, *ispell*, and *rcs*), nearly all SMC algorithms produce clusters of equivalent quality in terms of MQ. However, HOA's worst-case MQ surpasses the other SMC algorithms in the *ispell*, *bison*, *cia*, *php*, *grappa*, and *incle* benchmarks. Therefore, HOA outperforms the other techniques based on the worst MQ values obtained from ten executions.

Figure 10 depicts the MQ value of HOA, along with the best MQ among the six SMC methods on average. For small software packages (*mtunis*, *spell*, and *rcs*), all SMC algorithms generate clusters of comparable quality in terms of MQ, and all SMC algorithms perform similarly in the small benchmarks. In *bison*, *cia*, *dot*, *php*, *grappa*, and *incle*, clusters created by HOA exhibit significantly higher MQ values compared to those generated by the other algorithms. On average, HOA showed a better performance than the other algorithm based on the average values of MQ obtained from ten executions. The difference in the average MQ values between HOA and the other algorithms is particularly notable in large benchmarks.

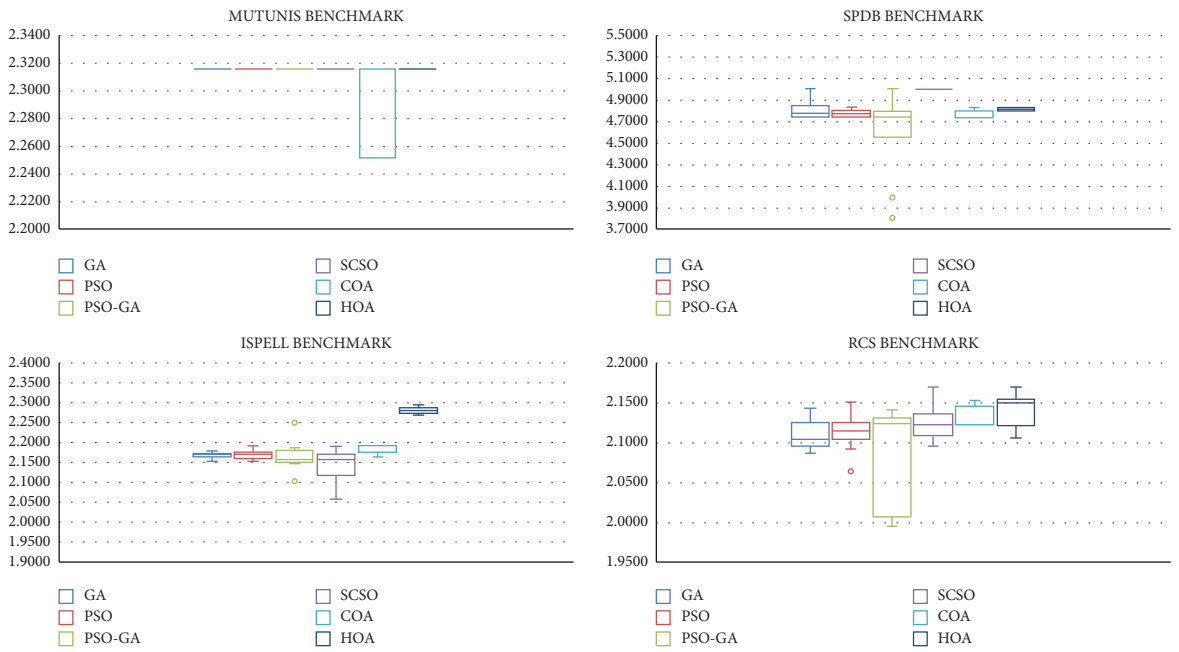


Figure 5: MQ values of various SMC methods for *mtunis*, *spdb*, *ispell*, and *rcs*.

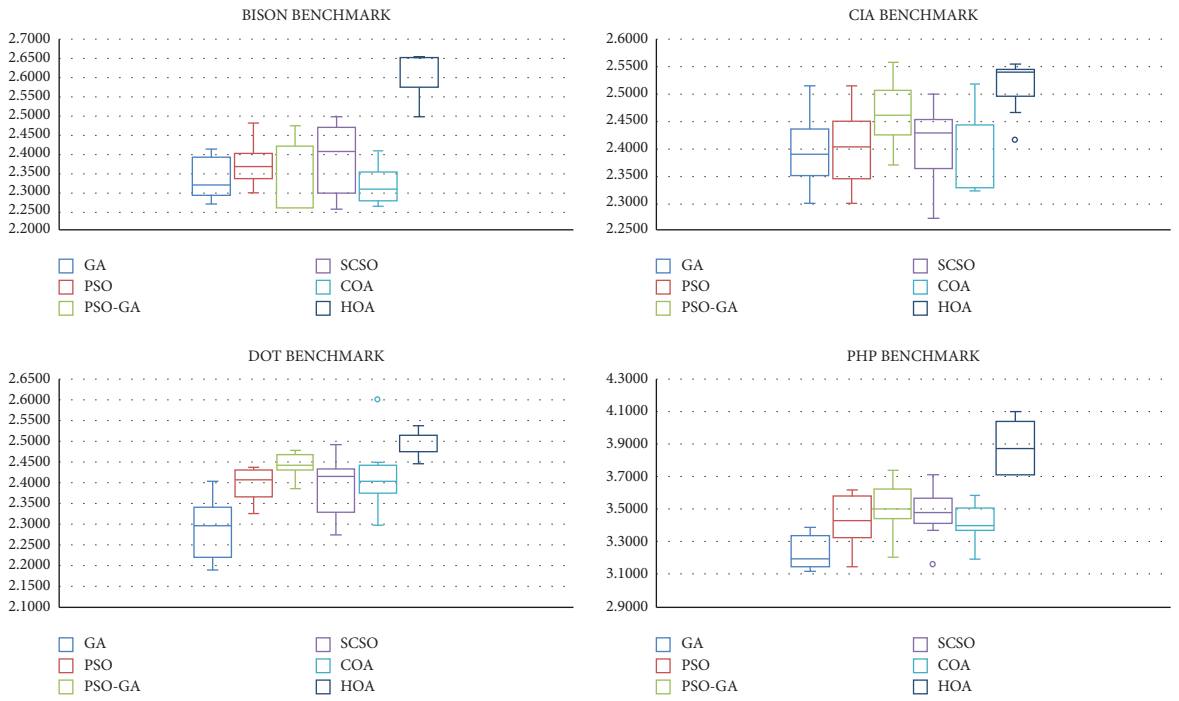


Figure 6: MQ values of various SMC methods for *bison*, *cia*, *dot*, and *php*.

Another important performance consideration is the reliability of results obtained from metaheuristic algorithms, which are inherently indeterministic. In many cases, a metaheuristic method with a higher MQ may generate low-quality clusters. Therefore, it is essential to take into account the standard deviation between values obtained during successive executions. Figure 11 illustrates the standard deviation of MQs derived from 10 iterations of each method. In most benchmarks (excluding *php*), the standard deviation among the results produced by HOA is lower than that of the other five methods. A smaller standard deviation indicates a more reliable algorithm. The average standard deviation values for GA, PSO, PSO-GA, SCSO, COA, and HOA are, 0.07839, 0.07678, 0.11536, 0.09159, 0.08013, and 0.07183, respectively. As demonstrated, HOA exhibits a lower standard deviation compared to the other methods. The key

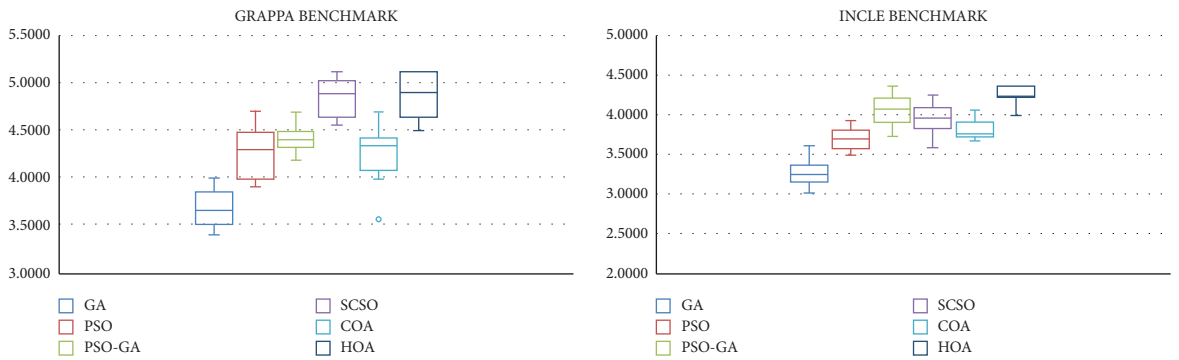


Figure 7: MQ values of various SMC methods for *grappa* and *incle*.

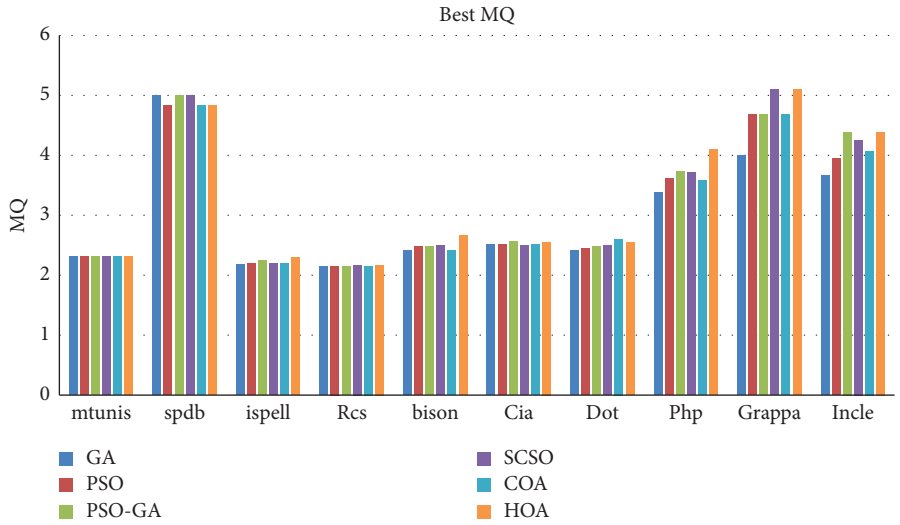


Figure 8: The best values of MQ obtained by 10 executions of various SMC algorithms.

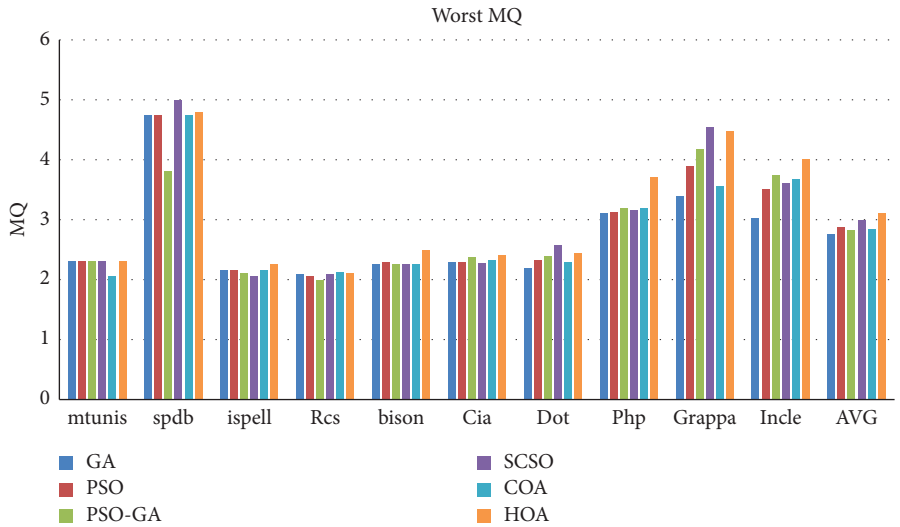


Figure 9: The worst values of MQ obtained by 10 executions of various algorithms.

advantage of HOA in addressing the SMC problem is its ability to produce higher-quality clustered models as well as improved reliability (lower standard deviation).

Figure 12 displays the average MQ values achieved by different SMC algorithms for all clusters. The average MQ values for PSO, GA, PSO-GA, SCSO, COA, and HOA are

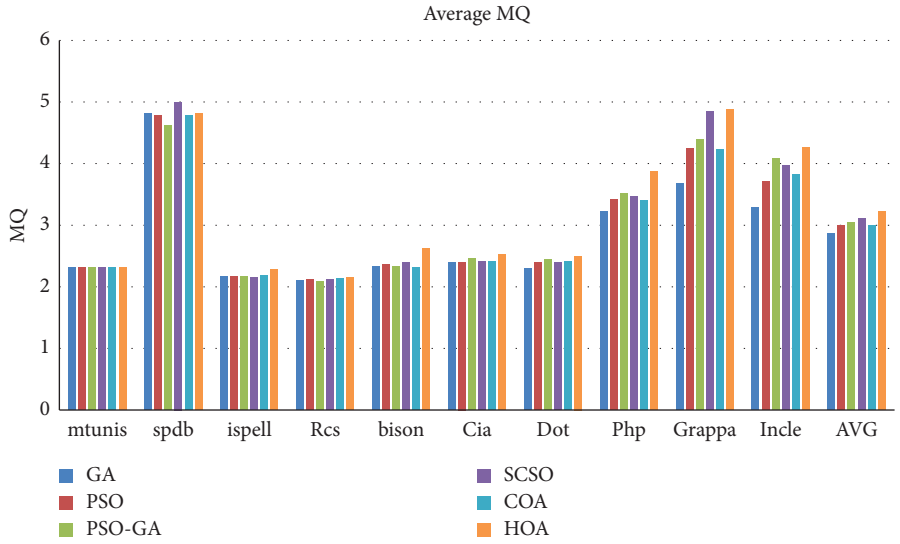


Figure 10: The average values of MQ obtained by 10 executions of various algorithms.

2.993, 2.869, 3.042, 3.107, 3.002, and 3.219, respectively. As depicted in Figure 12, HOA consistently generates the highest MQ value across all benchmarks. HOA's performance in the problem of SMC remains independent of the size of MDGs. In all benchmarks, the best, worst, and average MQ values obtained by HOA surpass those of other approaches. As a discrete method, HOA outperforms other discretised algorithms in the SMC problem, such as PSO, SCSO, and COA. The structural models generated by HOA exhibit superior quality, characterised by lower coupling and higher cohesion, compared to models generated by the other SMC approaches. On average, in all benchmarks, HOA outperforms competing algorithms with regards to average, best, and worst performance.

Additional experiments were conducted to assess the performance of HOA in comparison to existing SMC tools. Bunch [8] was selected as the automated SMC tool for a comparative analysis with HOA. This tool employs HC and GA algorithms to cluster software modules, taking the MDG matrix as an input and producing the clustered model using those two algorithms. The MQ value of models created by both Bunch and HOA was compared. As shown in Figure 13, HOA consistently exhibits higher average MQ values across all benchmarks, outperforming Bunch in each case. The average MQ for Bunch is 2.138, while HOA achieves an average MQ of 3.219. HOA's ability to generate more effective and comprehensible structural models offers software developers valuable tools for the maintenance phase. It is worth noting that Bunch employs both GA and HC as SMC tools, with the GA outperforming HC in most benchmarks. Researchers and developers have widely used Bunch's results for clustering software modules. In a previous study [6], a modified version of the firefly algorithm was developed as an SMC algorithm and its results were compared to Bunch. In those experiments, the firefly algorithm demonstrated a higher MQ than Bunch (HC and GA). Furthermore, the results obtained by HOA were compared to those of Bunch (GA), and as indicated in Figure 13, the proposed HOA exhibited superior performance in terms of MQ compared to the Bunch tool.

Table 4 presents the MQ values provided by various SMC methods along with their associated standard deviations. The proposed HOA, which is a discrete optimisation algorithm that leverages swarm intelligence, outperforms the other algorithms, as indicated in Table 4. HOA achieves a higher MQ while maintaining a lower standard deviation compared to its counterparts. A lower standard deviation across multiple executions signifies greater result stability. The reliability of results from a heuristic algorithm hinge on its stability, and HOA's lower standard deviation translates into higher stability. Therefore, HOA demonstrates superior reliability compared to the other algorithms across multiple executions. In summary, in the context of the problem of SMC, which is a graph-based discrete optimisation challenge, HOA consistently outperforms other effective algorithms, including PSO, GA, PSO-GA, SCSO, and COA.

The computational complexity of HOA is a measure of how long it takes for this algorithm to solve the SMC problem. It considers factors such as the quantity of search agents and the maximum quantity of iterations to evaluate the complexity cost of HOA. HOA strikes a fine balance between exploration and exploitation, effectively reducing the computational complexity of the problem of SMC by utilising the six elements in the movement of the horses. By implementing a sorting mechanism within a global matrix, HOA employs an effective strategy to prevent getting stuck in local optima. This global matrix is constructed by combining the positions (X) of the horses and their respective costs ($C(X)$), where X represents the positions and $C(X)$ signifies the associated costs for each position. In addition, d and m denote the dimension quantity of the problem and the quantity of horses, respectively. In the subsequent step, the global matrix is sorted on the basis of the last column, which contains the

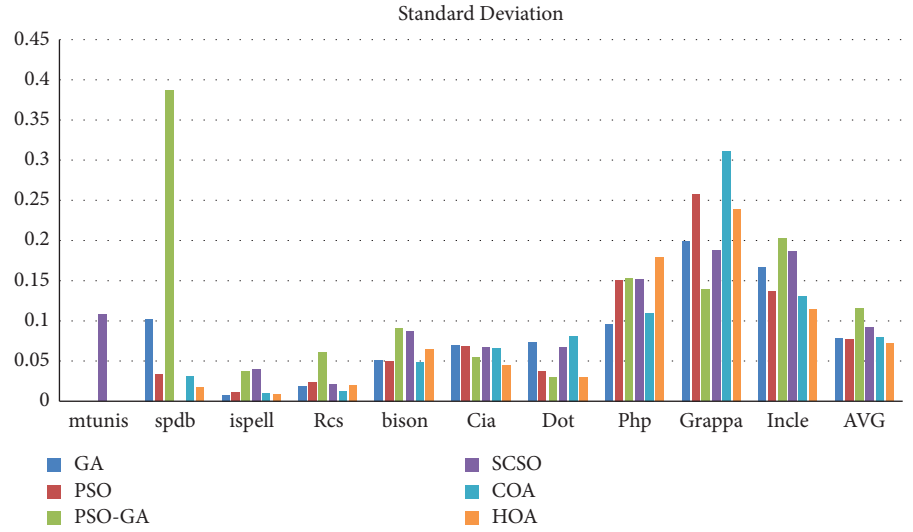


Figure 11: The standard deviation among the MQ values generated as a result of ten executions of various algorithms.

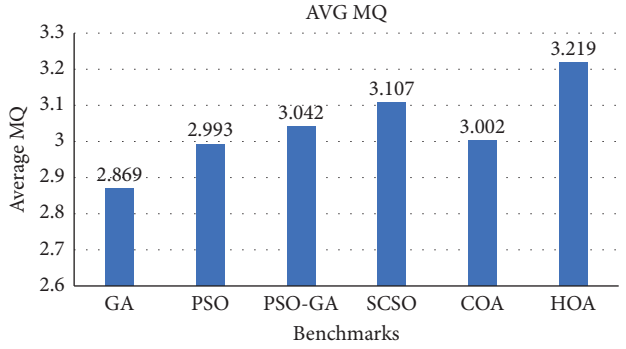


Figure 12: The average MQ values of generated clusters for all benchmarks by various algorithms.

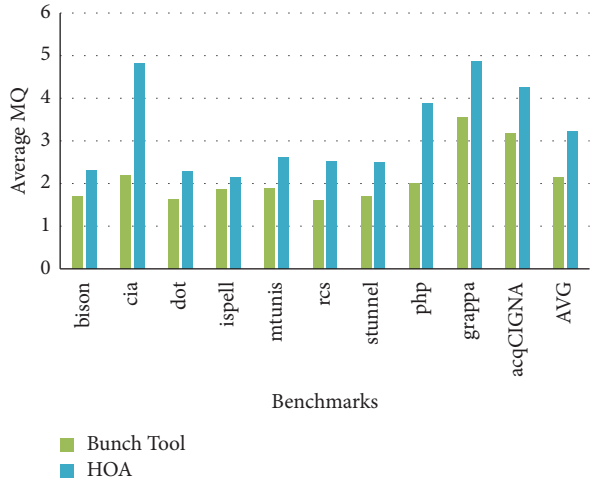


Figure 13: Performance comparison of HOA with the Bunch SMC tool.

costs. HOA takes advantage of a swift matrix sorting system; as a result, the computational cost (O) during the sorting step varies between $O\ (m \times \log\ (m))$ in the best-case scenario and $O\ (m^2)$ in the worst-case scenario. The execution times of various SMC algorithms are provided in Table 5.

Table 4: The average MQ values obtained by various methods and the standard deviation (STDV) among them.

	PSO		PSO-GA		SCSO		COA		HOA	
	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
spdb	4.7744	0.0330	4.6180	0.3870	5.0000	0.0000	4.7844	0.0314	4.8142	0.0168
ispell	2.1690	0.0111	2.1660	0.0603	2.1520	0.0400	2.1800	0.0100	2.2790	0.0088
rcs	2.1170	0.0231	2.0900	0.0910	2.1260	0.0215	2.1380	0.0129	2.1434	0.0204
bison	2.3640	0.0500	2.3300	0.0546	2.39.5	0.0871	2.3213	0.0483	2.6169	0.0644
cia	2.4026	0.0682	2.4639	0.0294	2.4106	0.0669	2.4108	0.0661	2.5220	0.0454
dot	2.4009	0.0370	2.4461	0.1527	2.3949	0.0672	2.4158	0.0813	2.4994	0.0300
php	3.4232	0.1507	3.5141	0.1391	3.4743	0.1513	3.4013	0.1099	3.8761	0.1790
grappa	4.2498	0.2580	4.4001	0.2023	4.8431	0.1874	4.2284	0.3109	4.8721	0.2393
incle	3.7151	0.1367	4.0812	0.2023	3.9690	0.1863	3.8288	0.1305	4.2615	0.1142
AVG	2.9930	0.0767	3.0423	0.1153	3.1074	0.0915	3.0022	0.0801	3.2198	0.0718

Table 5: Average execution time of SMC algorithms in ten iterations (in ms).

Programs	PSO-GA	PSO	GA	HOA
"rcs"	9.170	8.000	6.150	5.310
"incl"	16.000	14.500	9.000	9.140
"grappa"	16.040	11.080	7.150	6.400
"modulizer"	6.040	5.190	3.810	5.610
"Compiler"	3.830	3.100	2.000	3.000
"*mtunis*"	4.710	2.990	2.610	3.670
"*bison*"	5.900	5.430	3.990	3.170
"boxer"	3.520	3.000	1.790	2.950
"*acqcigna*"	13.230	10.970	8.150	7.620
"*ispell*"	4.700	4.190	3.520	3.910

5. Conclusion

The source code of a complex program may not inherently reveal its structure. Identifying the affected code segments within a program's source code during maintenance is among the most difficult challenges in software engineering. Clustering the program's modules can reduce maintenance costs by enhancing the code's comprehensibility. This research developed a discrete variant of the HOA as a metaheuristic approach for creating a clustered design model for source code programs. A selection of real-world software applications served as benchmark programs. The proposed HOA-based method employs local and global search operators to explore the solution space. It is specifically designed to avoid the local optima and maintain a balance between cohesion and coupling. Along with HOA, five distinct clustering algorithms were devised and assessed for their performance. Multiple tests were conducted on ten different software source codes to evaluate the proposed method's effectiveness. HOA showed better performance in comparison with other algorithms, including PSO, GA, PSO-GA, SCSO, and COA, in terms of MQ, cohesion, coupling, and stability, particularly in the context of larger software projects. A potential avenue for future research is the development of algorithms that remain effective regardless of the software product size. In addition, it might be worthwhile to explore the impact of chaotic equations on the effectiveness of HOA further. Another promising direction for future work is the integration of various swarm and evolutionary-based methods into the problem of SMC, which could yield improved results. Researchers are encouraged to explore enhancements to the fitness function to accommodate novel software metrics. It is worth noting that while recent studies have introduced global modules as universal criteria, they are not currently incorporated into the MQ metric. Global modules are modules that receive calls from multiple independent modules but do not initiate any calls. Furthermore, potential future research could involve investigating optimisation methods proposed in [23–25] within the context of SMC techniques. The development of novel data clustering algorithms could be explored as approaches to module clustering. Finally, creating a new experimental platform for investigating additional evaluation criteria is suggested as part of future studies.

Data Availability

The data used to support the findings of this study can be accessed via the following link: https://drive.google.com/drive/folders/1jRUhLXzlofWJaujB2_FRs06v73KtAKu-?usp=sharing.

Disclosure

The data used in the current research do not belong to any other individual or third party and have been prepared and generated by the authors during the study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

The horse herd optimization algorithm was developed and discretised by Bahman Arasteh and Asgarali Bouyer. The designed algorithm was implemented and coded by Bahman Arasteh. The implemented HOA code was adapted and benchmarked by Bahman Arasteh and Farhad Soleimanian Gharehchopogh. The generation of the MDG matrix from the source code of the benchmark programs was performed by Bahman Arasteh and Asgarali Bouyer.

The data and results analysis were performed by Reza Ghanbarzadeh and Farhad Soleimanian Gharehchopogh. The manuscript of the paper was written by Bahman Arasteh and Hamed Alipour Banaei and proofread by Farhad Soleimanian Gharehchopogh.

References

[1] Amarjeet and J. K. Chhabra, "Harmony search based remodularization for object-oriented software systems," *Computer Languages, Systems and Structures*, vol. 47, pp. 153–169, 2017.

[2] J. Sun and B. Ling, "Software module clustering algorithm using probability selection," *Wuhan University Journal of Natural Sciences*, vol. 23, no. 2, pp. 93–102, 2018.

[3] J. K. Chhabra, "Improving the modular structure of software system using structural and lexical dependency," *Information and Software Technology*, vol. 82, pp. 96–120, 2017.

[4] J. Yuste, A. Duarte, and E. G. Pardo, "An efficient heuristic algorithm for software module clustering optimization," *Journal of Systems and Software*, vol. 190, Article ID 111349, 2022.

[5] A. Prajapati and J. K. Chhabra, "A particle swarm optimization-based heuristic for software module clustering problem," *Arabian Journal for Science and Engineering*, vol. 43, no. 12, pp. 7083–7094, 2018.

[6] A. Mamaghani and M. Hajizadeh, "Software modularization using the modified firefly algorithm," in *Proceedings of the 8th.Malaysian Software Engineering Conference (MySEC)*, Langkawi, Malaysia, September 2014.

[7] K. Mahdavi, M. Harman, and R. M. Hierons, "A multiple hill climbing approach to software module clustering," in *Proceedings of the International Conference on Software Maintenance, ICSM*, IEEE, Amsterdam, The Netherlands, September 2003.

[8] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. B. Gansner, "A clustering tool for the recovery and maintenance of software system structures," in *Proceedings IEEE International Conference on Software Maintenance 1999 (ICSM'99)*, Oxford, UK, September 1999.

[9] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264–282, 2011.

[10] B. Arasteh, S. Razieh, and A. A. Keyvan, "A software modules clustering method using the combination of particle swarm optimisation and genetic algorithms," *Intelligent Decision Technologies*, vol. 14, pp. 449–462, 2020.

[11] E. Hatami and B. Arasteh, "An efficient and stable method to cluster software modules using ant colony optimization algorithm," *The Journal of Supercomputing*, vol. 76, no. 9, pp. 6786–6808, 2020.

[12] B. Arasteh, R. Sadegi, and K. B. Arasteh, "Bölen: software module clustering method using the combination of shuffled frog leaping and genetic algorithm," *Data Technologies and Applications*, vol. 55, no. 2, pp. 251–279, 2021.

[13] B. Arasteh, M. Abdi, and A. Bouyer, "Program source code comprehension by module clustering using combination of discretized gray wolf and genetic algorithms," *Advances in Engineering Software*, vol. 173, Article ID 103252, 2022.

[14] B. Arasteh, "Clustered design-model generation from a program source code using chaos-based metaheuristic algorithms," *Neural Computing & Applications*, vol. 35, no. 4, pp. 3283–3305, 2022.

[15] B. Arasteh, A. Fatolahzadeh, and F. S. Kiani, "Savalan: multi objective and homogeneous method for software modules clustering," *Journal of Software: Evolution and Process*, vol. 34, no. 1, p. e2408, 2022.

[16] Y. Chen, "Reverse engineering," in *Practical Reusable Unix Software*, B. Krishnamurthy, Ed., pp. 177–208, John Wiley & Sons, Hoboken, NJ, USA, 1995.

[17] J. Korn, Y. Chen, and E. Koutsofios, "Chava: reverse engineering and tracking of java applets," in *Proceedings of the Working Conference on Reverse Engineering*, Atlanta, GA, USA, October 1999.

[18] A. C. Kumari, K. Srinivas, and M. Gupta, "Software module clustering using a hyper-heuristic based multi-objective genetic algorithm," in *Proceedings of the 3rd IEEE International Advance Computing Conference (IACC)*, IEEE, Ghaziabad, India, February 2013.

[19] J. K. Chhabra, "Improving package structure of object-oriented software using multi-objective optimisation and weighted class connections," *Journal of King Saud University-Computer science*, vol. 29, pp. 349–364, 2017.

[20] A. J. K. Chhabra, "TA-ABC: two-archive artificial bee colony for multi-objective software module clustering problem," *Journal of Intelligent Systems*, vol. 27, pp. 619–641, 2018.

[21] B. Arasteh, A. Seyyedabbasi, J. Rasheed, and M. Abu-Mahfouz, "Program source-code Re-modularization using a discretized and modified sand cat swarm optimisation algorithm," *Symmetry*, vol. 15, no. 2, 2023.

[22] F. MiarNaeimi, G. Azizyan, and M. Rashki, "Horse herd optimisation algorithm: a nature-inspired algorithm for high-dimensional optimisation problems," *Knowledge-Based Systems*, vol. 213, Article ID 106711, 2021.

[23] M. S. Mahmud, J. Z. Huang, V. Ruby, A. Ngueilbaye, and K. Wu, "Approximate clustering ensemble method for big data," *IEEE Transactions on Big Data*, vol. 9, no. 4, pp. 1142–1155, 2023.

[24] M. S. Mahmud, J. Z. Huang, R. Ruby, and K. Wu, "An ensemble method for estimating the number of clusters in a big data set using multiple random samples," *Journal of Big Data*, vol. 10, no. 1, p. 40, 2023.

[25] M. Behera, A. Sarangi, D. Mishra et al., "Automatic data clustering by hybrid enhanced firefly and particle swarm optimization algorithms," *Mathematics*, vol. 10, no. 19, p. 3532, 2022.