A hybrid approach of metaheuristic algorithms and Group Method of Data Handling (GMDH) to Predict Source Code Testability

Xiaobo Wu
School of Business, Lingnan Normal University, Zhanjiang 524048, Guangdong, China
E-mail: zuoye_wu@163.com

Keywords: testability, group method of data handling, colliding bodies optimization, firefly algorithm, ant colony optimization

Received: August 10, 2025

Testability is a critical software quality attribute that enables developers to efficiently test and improve code throughout the development lifecycle. This study investigates the prediction of software testability using machine learning algorithms, focusing on hybrid models that integrate the Group Method of Data Handling (GMDH) with metaheuristic optimization techniques. A dataset of 16,165 software classes with 17 static source-code metrics was used, split into 80% training and 20% testing samples. GMDH parameters were optimized using three metaheuristic algorithms: Colliding Bodies Optimization (CBO), Firefly Algorithm (FA), and Ant Colony Optimization (ACO), forming hybrid predictive models. Comparative baseline regressors included SVM, Random Forest, AdaBoost, MLP, and GLM. All models were evaluated using eight metrics: RMSE, MAE, R, RAE, Std, VAF, PI, and Max Error. The results show that among baselines, GLM achieved RMSE = 0.2357, MAE = 0.1871, and R = 0.5505, while SVM, RF, AdaBoost, and MLP achieved slightly higher errors. The GMDH hybrids improved performance, with GMDH–FA achieving RMSE=0.209, R=0.671, GMDH–ACO RMSE =0.209, R = 0.673, and GMDH–CBO reaching the best overall performance with RMSE=0.208, and R=0.674. These results indicate that metaheuristic-optimized GMDH models, particularly GMDH–CBO, provide a robust and accurate approach for predicting software testability.

Povzetek: Študija kaže, da hibridni modeli GMDH, optimizirani z metahevrističnimi algoritmi, natančneje napovedujejo testabilnost programske opreme kot klasični regresijski modeli, pri čemer je najboljše rezultate dosegel model GMDH–CBO.

1 Introduction

By increasing the complexity of software applications, software design becomes a challenging activity, and quality and reliability become more controversial [1]. One of the most important aspects of a good software design is estimating the functional and non-functional needs of a system and testing it in a specific test scenario [1]. A software artifact's ability to facilitate testing in a particular test context is one of the crucial aspects of the software development process [2]. However, not all systems are equally testable. Testing some is simpler. Furthermore, some of them are quite complicated [2]; testability is a quality agent that may be anticipated by defining the system in a way that demonstrates the system's ease (or quality) of testing. In general, six factors play a role in the testability of software, which are [2]: (1) specification of the representation, (2) specification of the performance, (3) built-in test capabilities, (4) test cases, (5) the test support environment, (6) the software process in which testing is conducted.

Testability prediction in the context of recognizing software components that require effort can help developers improve software quality. It can be applied at any stage of the expansion of software [3].

Over the last years, data-driven and machine learning-based solutions have gained more and more impact on software testability prediction. Zakeri -Nasrabadi (2024) showed that structural metrics are a software design analysis method that predicts and enhances testability, which offers a systematic method to identify weaknesses likely at the design phase [4].

Based on this, Talakola (2024) examined how artificial intelligence and machine learning methods can be applied to improve the efficiency and effectiveness of a testing process and demonstrated that ML is capable of automating test planning and minimizing human input [5].

Likewise, Nascimento (2024) has used ML models to prioritize tests, choosing those modules that are at high risk to minimize the effort and cost of testing, which demonstrates the usefulness of predictive models in resource allocation in practice [6].

Building on more general tendencies, Pandy et al. (2024) evaluated the emergent methods of AI and ML integration, emphasizing how they can be used to automate testing, gain more coverage, and better understand faults, as well as the opportunities of integrated structures in the context of current software testing [7].

Dwelling on the quality of tests, Lin, Liu, and Tahvildari (2024) suggested a machine learning-based test flaky categorization system and showed that the ML methods could help to improve the predicting accuracy of the defects, but also reliability and consistency of the test [8]. As part of metric development, Guglielmo et al. (2024) used automated test generation with mutation analysis to bring a new measure of effective test generation difficulty to complement the traditional structural metrics and give a more precise picture of testing effort [9]. Regarding human-centered view, Sharma et al. (2024) analyzed the knowledge of developers in understanding testability and found that the perception of developers has a substantial impact on maintenance and evolution of software and therefore developers' insight should be considered in predictive models [10].

Lastly, Stocco et al. (2023) discussed the system of applying machine learning to the software testing process and suggested new methods of test automation and test quality evaluation, which are consistent with the objectives of predictive testability, with focus on efficiency and robustness in a real-life setting [11].

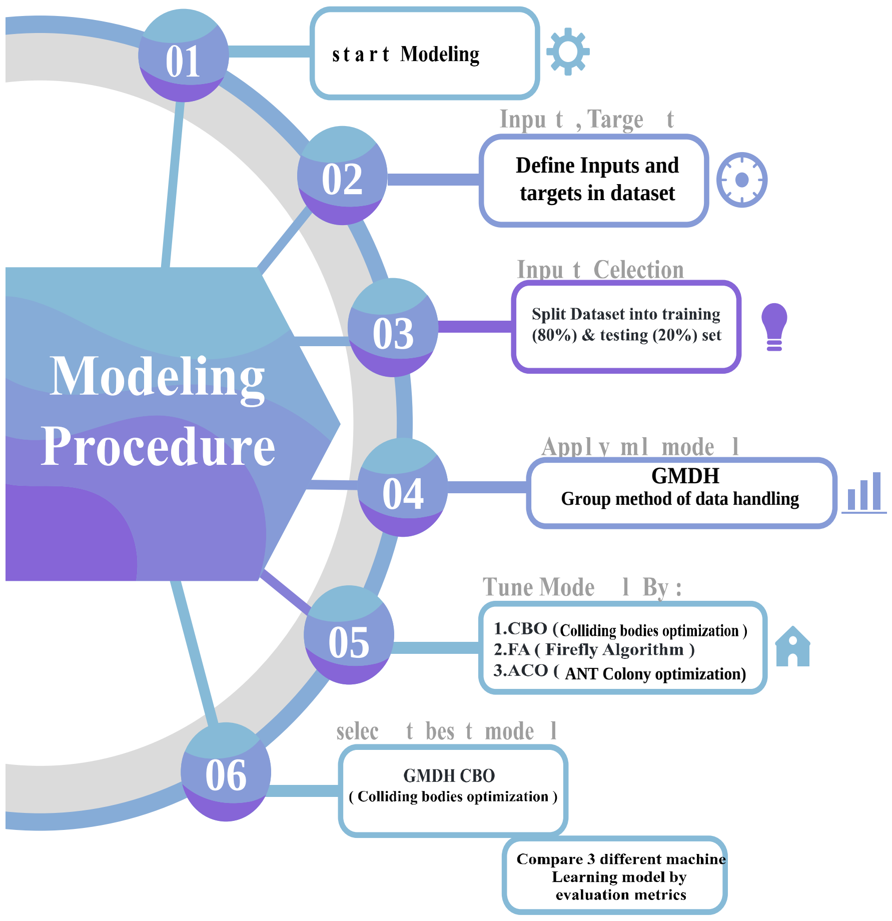
In this paper, a study on testability prediction is conducted based on experimental data. During training, the GMDH network's hyperparameter has been optimized using ACO, CBO, and FA algorithms. The final purpose of this paper is to compare meta-heuristic algorithms with each other, choose the superlative algorithm to offer an accurate hybrid algorithm, and predict testability.

The structure of this paper is organized as follows: in the section, the methodology is depicted, and the main principles of algorithms and evaluation indicators are discussed. In section 3, the performance of the methods and algorithms is evaluated, and the results are presented. Finally, the conclusions of the analysis are drawn in section 4.

2 Methodology

The method adopted for this study is a polynomial neural network called GMDH that does the prediction of testability. Three selection algorithms of GMDH, namely CBO, FA, and ACO, are reviewed. Tuning each algorithm with the GMDH network has been compared using various statistical indexes.

A flowchart in Fig. 1 illustrates the six primary phases of the modeling process. Modeling begins in step 1. The dataset is then used to define the inputs and targets. In step three, divide the dataset into two classes: testing (20%) and training (80%). Use the GMDH model in step four. The model is then adjusted using the CBO, FA, and ACO algorithms. Ultimately, the optimal model is chosen by contrasting three distinct ML models using assessment measures.



Modeling Procedure
01
start Modeling
02
Inpu t , Targe t
Define Inputs and targets in dataset
03
Inpu t Celection
Split Dataset into training (80%) & testing (20%) set
04
Appl y ml mode l
GMDH
Group method of data handling
05
Tune Mode l By :
1.CBO (Colliding bodies optimization)
2.FA (Firefly Algorithm)
3.ACO (ANT Colony optimization)
06
selec t bes t mode l
GMDH CBO
(Colliding bodies optimization)
Compare 3 different machine Learning model by evaluation metrics

Figure 1: The flow chart of the modeling procedure

2.1 Formal definition of testability

Software testability is the degree to which a software system or unit under test makes support for itself being tested. For a given class, A [3]:

$$T(A) = T_Q(A) \times T_P(A) \quad (1)$$

where $T(A)$ is testability, $T_Q(A)$ is effective, and $T_P(A)$ is efficiency.

By computing the mean of various coverage criteria considered for a test set, the effectiveness of the test $T_Q(A)$ of the class A is obtained:

$$T_Q(A) = \frac{1}{|Criteria|} \sum\nolimits_{c \in Criteria} c^{level}(A) \quad (2)$$

where c^{level} indicates the level of specified criterion that has been covered, c.

The efficiency of the test $T_P(A)$ is thought to be the inverse of test effort, $T_E(A)$:

$$T_P(A) = \frac{1}{T_E(A)} \quad (3)$$

The size of the test suite is the primary metric used in the majority of software testing literature to gauge test effort. As the number of tests rises, the pace at which code coverage grows falls. Specifically, the assumption that adding extra test data has no effect on coverage becomes less valid as the proportion of significant test data, which directly impacts coverage, increases [3].

2.2 Group method of data handling ($\boldsymbol{GMDH}$)

The GMDH is one form of an artificial neural network [12]. GMDH can be useful for solving various problems, including classification and prediction, among others [13]. The GMDH algorithms are featured by an inductive procedure that sorts out gradually complicated polynomial models and selects the best solution by applying some external criterion. Primarily, the idea of constructing GMDH methods as a new ML method was suggested by Ivakhnenko (1971) [14]. As a self-organization method [12] capable of simulating complex and nonlinear systems without requiring hypotheses about internal parameters, Farlow (1981) [14] provides a foundational approach. The GMDH neural network structure includes an input layer, several intermediate layers, and an output layer [13]. In the model developed, GMDH determines the variables that will enter, the number of layers, and the neurons in that hidden layer.

GMDH networks mostly solve this problem by using regression analysis [12]. First, the polynomial types, which are segregated, must be specified. It is possible to express the general relevance between the input and output variables with the Volterra functional series introduced by Wiener in nonlinear analysis. Also, this approach incorporates Kolmogorov-Gabor's ideas.

$$f = \omega_0 + \sum_{i=1}^{m} \omega_i x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} \omega_{ij} x_i x_j \quad (4)$$
$$+ \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} \omega_{ijk} x_i x_j x_k + \cdots$$

Here, f, is the variable or neuron in different positions, ω, is the vector of weights, x, is the input variables, and m, is the number of input variables. The quadratic form of the polynomial is expressed as follows:

$$f = \omega_0 + \omega_1 x_i + \omega_2 x_j + \omega_3 x_i x_j + \omega_4 x_i^2 + \omega_5 x_j^2 \quad (5)$$

Following these procedures, all polynomial terms are combined linearly using variable coefficients. To find the difference between the sample outputs and model predictions, the method minimizes the coefficients to the least sum of squares. The operation of GMDH is based on building sequential layers with intricate connections, which are terms of a polynomial [15]. Regression analysis of the input features is used to generate the first layer, and the most useful characteristics are subsequently selected [15]. Calculating the regressions of the first layer's values yields the second layer. Therefore, the algorithm constructs polynomials, and the best ones are selected by the algorithm [15]. These steps continue until a predetermined selection criterion is met.

GMDH uses normal regression to fit models to the given data, which results in overfitting models being considered [15]. For this reason, methods are used to decrease the quantity of parameters in the models. For this purpose, there are two methods [16]: Stepwise Regression and Stein Estimator. Stepwise regression was presented by Duffy and Franklin. While the term Stein estimator is related to Akaike's recent works [16].

In this research, the CBO, FA, and ACO nature-inspired algorithms are integrated with GMDH to develop a hybrid model. Models are evaluated with the aim of predicting testability. Standard metrics are used to achieve the goal with the least error and best performance.

2.2.1 GMDH-CBO

Inspired by the rules of momentum and energy conservation, CBO is one of the newest meta-heuristic search algorithms. It was first introduced by Kaveh and Mahdavi in 2014 [17], [18], [19]. Each of the numerous CBs in this method is seen as an object with a specified mass and velocity [17]. Normally, CBO is independent of tuning any internal parameters. Each one of the CBs possesses a certain mass defined as [20]:

$$m_k = \frac{\frac{1}{fit(k)}}{\sum_{i=1}^{n} \frac{1}{fit(i)}}, \qquad k = 1, 2, \ldots, n \quad (6)$$

$fit(i)$ denotes the value of the objective function (OF) of representative i, whereas n represents the population size.

In CBO, each solution character, X_i including several variables (i.e. $X_i = \{X_{ij}\}$) as a collision body (CB) [21]. Objects consist of two equal groups of moving and fixed bodies, where the moving objects collide with the fixed ones when they move to follow stationary objects. Two results are obtained from this encounter [21]: (i) recover the positions of moving bodies and (ii) push the stationary bodies into superior positions. Fig. 2 shows the flow chart of the CBO algorithm.

Initial all CBs

Similar to other meta-heuristic algorithms, CBs' beginning placements in the search space are chosen at random:

$$x_i^0 = x_{min} + rand(x_{max} - x_{min}),\quad i = 1, 2, \dots, z, \tag{7}$$

Here, z is the quantity of the CBs, rand is a random vector in the interval [0, 1], x_{max} and x_{min} are the maximum and minimum limits of the design variables, respectively, and x_i^0 is the initial value vector of the ith population.

OFs

The OFs are evaluated, and the value of mass for all populations, $X_{i=1,2,..,z}$, are calculated according to Eq. (6).

The velocities before the collision

There are two equal groups of CBs [17]: (i) the stationary group and (ii) the mobile group.

The pre-collision velocity of sorted populations is zero, and the lowest half of them are immobile:

$$v_i = 0,\qquad i = 1, 2, 3, \dots, z \tag{8}$$

The top half of sorted populations move to the bottom half. The velocity of these bodies is computed using the following equation:

$$v_i = x_i - x_{i-z},\qquad i = z + 1, z + 2, z + 3, \dots, 2z \tag{9}$$

(1) The velocities after the collision

The fixed and mobile bodies' velocities are updated after the impact using Eqs. (10) and (11), respectively:

$$v_i' = \frac{(m_{i+z} + \gamma m_{i+z})v_{i+z}}{m_i + m_{i+z}}, \tag{10}$$

$$i = 1, 2, 3, \dots, z$$

$$v_i' = \frac{(m_i - \gamma m_{i-z})v_i}{m_i + m_{i-z}},\quad i = z + 1, z + 2, z + 3, \dots, 2z \tag{11}$$

where ε shows the coefficient of restitution (COR), and often, its value is linearly between 0 and 1 [19]. Thus, γ is defined as:

$$\gamma = 1 - \frac{iter}{iter_{max}} \tag{12}$$

where $iter$ and $iter_{max}$ are the actual iteration number and the maximum number of iterations, respectively.

Note: v_i is the velocities of the CB_i before the collision and v_i' is the velocities of the CB_i after the collision.

New positions of the CB population

After computing the velocities of populations, new positions of CBs are determined using Eqs. (13) and (14):

$$x_i^{new} = x_i + rand.v_i',\qquad i = 1, 2, 3, \dots, z \tag{13}$$

$$x_i^{new} = x_{i-z} + rand.v_i',\quad i = z + 1, z + 2, z + 3, \dots, 2z \tag{14}$$

where, x_i^{new} and x_i are the new and old positions of the ith CB.

Terminating condition check

If the termination condition is not met, optimization is iterated again from step 2; otherwise, there is no penalty.

Best solution

The optimization process is then stopped if the maximum number of iterations is achieved. Then, the best solution is reported.

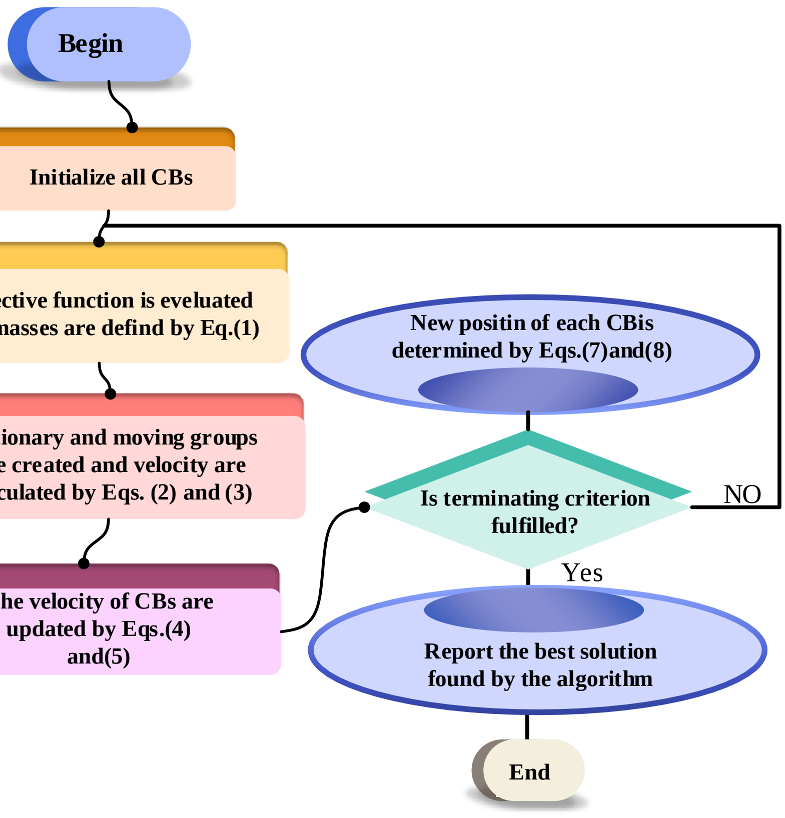

Begin
Initialize all CBs
Objective function is eveluated and masses are defind by Eq.(1)
Stationary and moving groups are created and velocity are claculated by Eqs. (2) and (3)
The velocity of CBs are updated by Eqs.(4) and(5)
New positin of each CBis determined by Eqs.(7)and(8)
Is terminating criterion fulfilled?
NO
Yes
Report the best solution found by the algorithm
End

Figure 2: The CBO algorithm

2.2.2 GMDH-FA

Another meta-heuristic algorithm is FA, which belongs to stochastic algorithms [22]. Xin-She Yang was the first to introduce a nature-inspired meta-heuristic FA algorithm based on flashing patterns and firefly behavior in 2008 [13], [23].

As mentioned, the principal idea of this algorithm is obtained from the behavior of fireflies. A variety of fireflies are scattered at random throughout the space of search. The adaptability value of the optimization problem determines each firefly's light gravity [23]. With increased light intensity, each individual flies in line with the firefly [14]. The final optimization is demonstrated when, after a number of rounds, all individuals converge around the optimal firefly. The framework for this method is simple to develop. However, like other algorithms, the conventional FA also faces problems such as precocious convergence.

According to the explanation above, light intensity and attractiveness are the two primary factors in a typical FA. Higher light intensity and attraction in a given area correlate with improved fitness [22]. On the other hand, light absorption weakens light as one gets farther away from the light source [24]. The OF may be optimized by integrating these phenomena.

Considering the brightness as I and the attractiveness of the firefly as β, the brightness I is defined as follows:

$$I = I_0 e^{-\gamma d_{ij}^2} \qquad (15)$$

where, I_0, is the light source, γ, is the light absorption coefficient, and d represents the remoteness among two fireflies i and j, defined as:

$$d_{ij} = \|s_i - s_j\| \qquad (16)$$

Similarly, attractiveness β is defined as follows:

$$\beta = \beta_0 e^{-\gamma d_{ij}^2} \qquad (17)$$

where, β_0, denotes the attractiveness at $d = 0$.

The movement of a firefly i to a brighter (more attractive) firefly j is based on the following equation:

$$s_i = s_i + \beta_0 e^{-\gamma d_{ij}^2}(s_j - s_i) + \alpha\epsilon_i \qquad (18)$$

where, ϵ_i, is a random number drawn from Gaussian distribution.

In general, three parameters control FA, which are [25]: the absorption coefficient γ, the attractiveness β, and the randomization parameter α. According to the parameter setting, FA determines two asymptotic behaviors. One is that if $\gamma \to 0$, the attractiveness becomes $\beta = \beta_0$. This means that the attractiveness is constant at any point in the search space. Another is that if $\gamma \to \infty$, the second term of Eq. (18) is removed, and the movement of the firefly becomes a random walk.

Fireflies in a small area exchange enough information to influence each other and eventually reach the target [22]. However, in a larger search area, it may be difficult to find some fireflies and attract them to one of the brighter ones for alien distribution. This results in their becoming "failures" at random in their initial position, which has a significant impact on the group's capacity to optimize and leads to premature convergence [22]. As a result, the FA can achieve the global optimization goal only when there is a close connection between the fireflies [24].

FA can also be used for hybridization with other general problem-solver models [26]. The FA was incorporated into the procedure of model training [13]. The training method with FA is the same for all ML methods [13]. Since FA is a kind of population-based optimization algorithm, the number of population and other fixed parameters must be determined before the start of optimization [25]. To start the process, the parameters which should be optimized must be specified. In general, meta-heuristic algorithms work based on trial and error [13]. Also, metrics are considered to evaluate between the trained and tested values.

2.2.3 GMDH-ACO

In this section, the proposed combined GMDH models using ACO are described. ACO meta-heuristic was first presented by Dorigo et al. in 1999 [27]. The ACO is derived from exploring the behavior of a real ant colony [28]. That behavior originates from indirect communication between ants through pheromone chemical trials, which help them find the shortest path between the food source and their nest. In ant species, the pheromone chemicals are released on the ground by walking ants [27]. Due to the release of pheromones, the path of the food source to the nest is determined. Some ants have a skill that can determine the shortest path [27]. This method helps other ants to find food sources by smelling the pheromone [27], [28]. This feature is applied to ACO algorithms to solve problems. The structure of the ACO algorithm is reviewed in the following section.

(1) Initialise parameters

By experimental experience, consider the number of ants as M, the maximum number of repetitions, K, the exploratory factor of pheromone, α, the expected exploratory factor, β, the evaporation coefficient of pheromone, ρ, and the intensity value of pheromone, Q.

(2) Generate global random

Consider a random initial position for each ant individually.

(3) Calculate fitness

The following equation calculates the probability of transition to the next state.

$$p_{i,j}^m = \begin{cases} \dfrac{[\tau_{(i,j)}]^\alpha [\varphi_{(i,j)}]^\beta}{\sum_{\delta \epsilon J_m(i)} [\tau_{(i,\delta)}]^\alpha [\varphi_{(i,\delta)}]^\beta}, & (i,j) \in J_m \\ 0, & ortherwise \end{cases} \qquad (19)$$

where τ represents the pheromone concentration value between two states, φ, represents the distance between the two states, α and β are the control parameters.

(4) Update pheromone

Updating the pheromone trails is needed to improve the quality of the solution, which is divided into local and global updating:

Local trail updating rule: As the ant moves between two states, the pheromone concentration is updated by the following equation:

$$\tau_{il}(t) = (1-\rho)\tau_{ij}(t-1) + \rho\tau_0 \qquad (20)$$

where, τ_0, is the initial value of the pheromone that depends on the node value.

Global trial updating rule: When whole ants find the paths in the ACO algorithm, the nodes passing the shortest trial implement the global update strategy using Eq. (21):

$$\tau_{il}(t) = (1-\rho)\tau_{ij}(t-1) + \frac{\rho}{L^{+}} \tag{21}$$

In which, L^{+}, is the length of the best path.

(5) A new path and applying transition

Evaporate a constant ratio of the pheromone on each path to improve the global search capability. By updating the pheromone, the new path can be achieved, and the transition can be applied.

(6) Iteration

Determine whether it needs to be iterated; if the solution obtained does not meet the need, then return to step two; otherwise, continue to the next step.

(7) End

After all iterations, the best solution is selected.

ACO was developed to optimize many problems. ML is one of these cases. Although many articles have reported the use of meta-heuristic algorithms for improvement [27], hybridization techniques of ACO with a basic model (GMDH) still obtain statistical evaluation results very close to the target.

2.3 Statistics

The data set utilized in the study is 16,165 software classes, which were initially available in the publicly accessible software testability dataset presented in [3]. There are 17 fixed source-code measures in each case, and the last column is the calculated testability measure, the prediction object.

Each statistic code-metric measure such as CSORDAvgLineCodeExe was simply obtained directly by the original software testability dataset obtained by extracting a large Java-based industrial software system using automated tools to analyses the source-code. To achieve the methodological transparency and complete replicability, the dataset and the entire MATLAB implementation of this study are represented as the supplementary materials.

The columns with missing data were dropped before modelling and there were no NaN values left in the data set. MATALAB was used to investigate the outliers in the target variable, where the default settings of the is outlier () function were used; no samples were identified or eliminated. Minimum maximum scaling was used to normalize all the input features to provide equal numerical ranges and allow more stability in training.

After preprocessing, the dataset had 16,165 valid samples, where 80% formed training data (12,932 samples) and 20% formed testing data (3,233 samples).

In order to achieve reliability of the performance evaluation, we made use of a deterministic train-test split of the 80/20 split with a set random seed thus giving fully reproducible results.

A histogram was created along with descriptive statistics to make clear the distribution of the target of prediction. The testability score after preprocessing has a mean value of 0.5038, a standard deviation of 0.2965, a minimum of 0.0007, a maximum of 1.000, a skewness of 0.0329 and a kurtosis of 1.8954.

This information increases the transparency, reproducibility and external validity of the dataset utilized in this study.

To make a fair comparison all three metaheuristic algorithms CBO, FA and ACO were performed in the same experimental conditions. The population size (NP) was determined to 2 and maximum number of iterations (T) was determined to 100. D = 5 is the search space dimension which is the number of GMDH hyperparameters that are optimized by each algorithm and the lower and upper boundaries are set to [1,100]. There was no early-stopping convergence threshold used; each optimizer ended exactly on the completion of 100 iterations. As metaheuristics use stochastic initialization and probabilistic search operators, to ensure replicability, all experiments used a fixed random seed. In this way all the reported results are deterministic, reproducible single-run results with controlled stochasticity.

Table 1 gives a full list of the parameter settings and configurations of the CBO, FA, and ACO algorithms, which gives a full replication of the proposed framework. Min-max scaling was used to normalize all the input features to [0,1] before training. A fixed random seed was used to eliminate stochastic variability and ensure deterministic execution, thus, prompting no repetitions of execution and determining confidence intervals or standard deviations. Computationally, despite the more complicated update functions used in CBO as compared to FA and ACO, the guided collision-based search strategy results in faster convergence and simpler GMDH structures, with fewer layers and neurons, to manage the complexity of the model and decrease training time. Moreover, the training and testing metrics are very close to each other, and the patterns are also consistent across the regression plots and the Taylor diagrams, which means that there is no overfitting in the model, and the model has good generalization ability.

Table 1: Parameter settings and configurations of the CBO, FA, and ACO algorithms used for GMDH optimization.

Algorithm	NP	T	D	Bounds	Specific Parameters	Hyperparameters Optimized
CBO	2	100	5	[1, 100]	–	Max LayerNeurons; Max Layers; α; p Train; Mode
FA	2	100	5	[1, 100]	α=0.25; β_min=0.2; γ=1	Max LayerNeurons; Max Layers; α; p Train; Mode
ACO	2	100	5	[1, 100]	α=1; β=1; Q=1	Max LayerNeurons; Max Layers; α; p Train; Mode

2.3.1 Metrics

To optimize hybrid forecasting models, their accuracy and error should be evaluated using multiple metrics. To compare meta-heuristic algorithms, the eight statistical evaluation indicators examined in this article are the root means square error ($RMSE$), the mean absolute error (MAE), max error (Max error), standard deviation (Std), the variance account factor (VAF), performance index (PI), correlation coefficient (R), and relative absolute error (RAE). The mathematical expression of these metrics is shown in Fig. 3. These statistical evaluation indexes are known as key performance indicators (KPI). Each has its own characteristics, which distinguish it from other indicators. However, all of them are used to evaluate the models and select the optimal one.

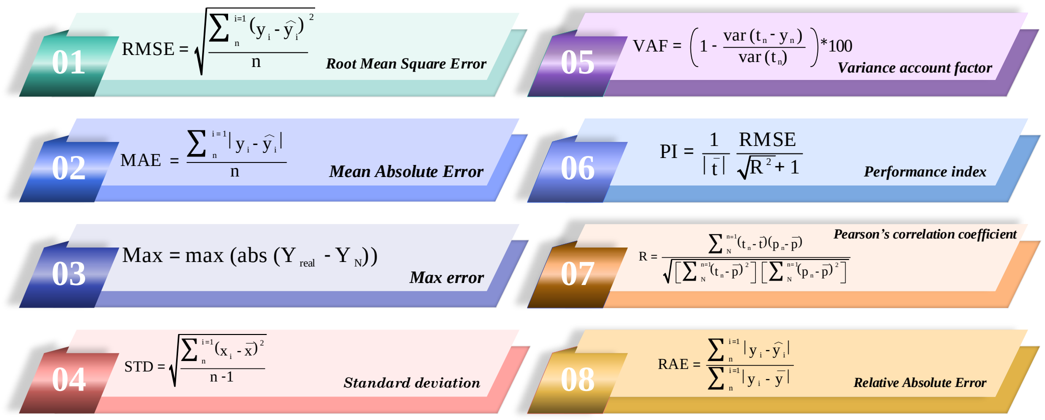

01
$RMSE = \sqrt{\frac{\sum_{n}^{i=1}(y_i - \hat{y}_i)^2}{n}}$
Root Mean Square Error
02
$MAE = \frac{\sum_{n}^{i=1}|y_i - \hat{y}_i|}{n}$
Mean Absolute Error
03
$Max = max\ (abs\ (Y_{real} - Y_N))$
Max error
04
$STD = \sqrt{\frac{\sum_{n}^{i=1}(x_i - \bar{x})^2}{n - 1}}$
Standard deviation
05
$VAF = \left(1 - \frac{var\,(t_n - y_n)}{var\,(t_n)}\right) * 100$
Variance account factor
06
$PI = \frac{1}{|\bar{t}|} \frac{RMSE}{\sqrt{R^2 + 1}}$
Performance index
07
Pearson's correlation coefficient
$R = \frac{\sum_{N}^{n=1}(t_n - \bar{t})(p_n - \bar{p})}{\sqrt{\left[\sum_{N}^{n=1}(t_n - \bar{p})^2\right]\left[\sum_{N}^{n=1}(p_n - \bar{p})^2\right]}}$
08
$RAE = \frac{\sum_{n}^{i=1}|y_i - \hat{y}_i|}{\sum_{n}^{i=1}|y_i - \bar{y}|}$
Relative Absolute Error

Figure 3: The metric equations.

Carl Pearson proposed the "coefficient of correlation" in 1896 [29]. This coefficient (R) is one of the most widely used metrics. The correlation coefficient explains the magnitude relationship between two variables in a linear manner, with values ranging from +1 to -1 [29]. Considerations for the correlation coefficient are considered as follows [29]:

(1) The absence of a linear relationship belongs to 0.

(2) Number + 1 represents the best positive linear relation. Through a precise linear rule, as the values of one variable grow, the values of another variable similarly increase.

(3) Number - 1 represents the best negative linear relation. By increasing one variable in its values, another variable lowers its values through a precise linear rule.

However, the RMSE is another metric that defines the best model. The smaller the value of RMSE, the better the model, i.e., the more accurate the predictions. RMSE is another form of the MAE, but MAE has been found to perform better than RMSE for evaluating mean model accuracy in most positions except Gaussian noisy scenarios [29]. In the continuation of the paper, the results obtained from the statistical evaluation criteria are examined.

3 Result and discussion

In this part, the results captured from the analysis of different algorithms that were adjusted with the GMDH model are reviewed. As previously stated, the data were divided into training and testing sets. Fig. 4 compares the results obtained by the meta-heuristic algorithms with the ideal experimental dataset, showing that the models achieve a reasonably good fit to the observations. By comparing the data, it is clear that the CBO model is closer to the optimal model and the target. In other words, the domain of error changes in this model is the least.

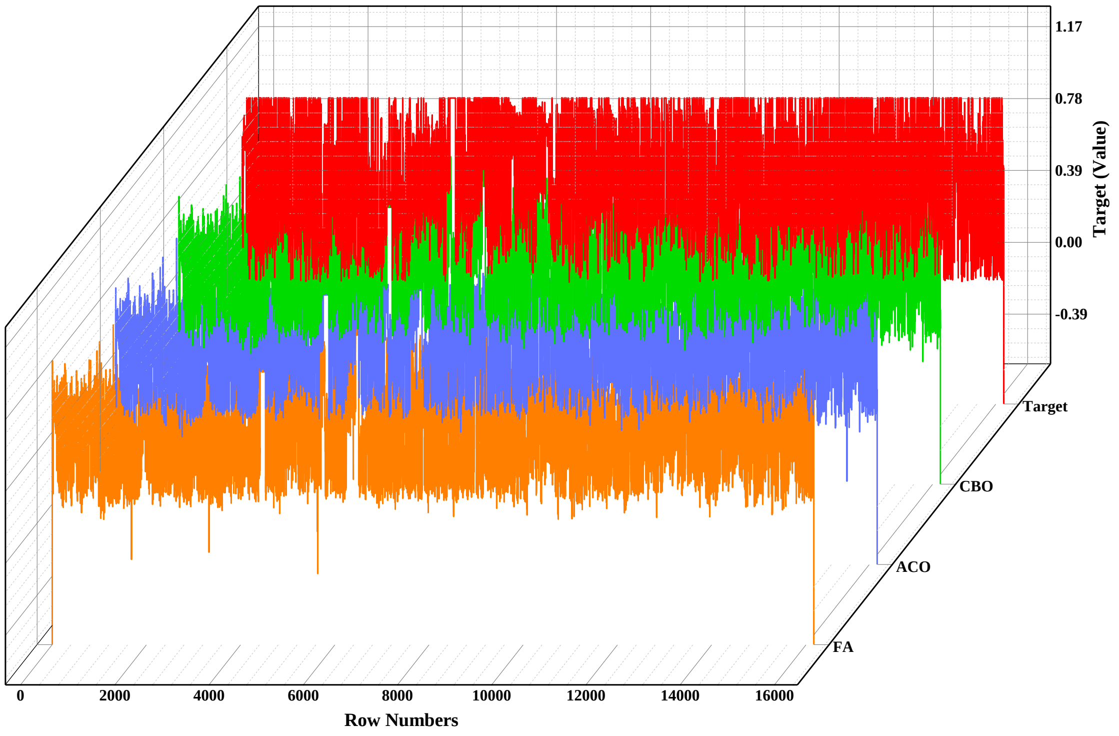


Figure 4: Comparison between obtained data and the optimal experimental data set of meta-heuristic algorithms

A more detailed examination of the error values of the algorithms for training and testing data is displayed through the histogram of Fig. 5. In fact, Fig. 5 shows the error values of GMDH combined models and the error distribution in each algorithm. According to the figure, all hybrid models in both the train data set and the test data set have almost similar fluctuation domain of error changes. By comparing the data obtained from the figure, it is obvious that the combined CBO model shows lower error values than other models. Meanwhile, the results obtained from the FA model show that this combined model has a weaker performance.

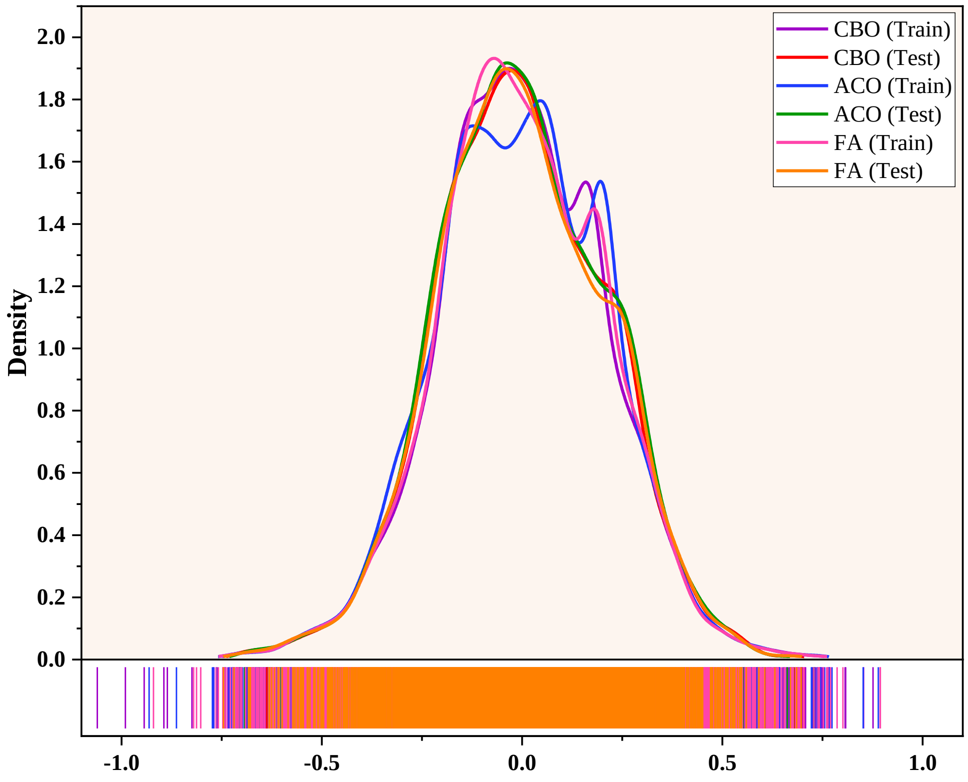


Figure 5: The algorithms' histogram error values.

Fig. 6 shows the values of seven metrics, both qualitatively and quantitatively, for the hybrid models separated by the type of data set. Among statistical evaluation indicators, the correlation coefficient (R) is more reliable and useful than others. The following figure shows the R values generated for the optimal model and other hybrid algorithms. According to the figure, the R values obtained for the CBO hybrid model in both train and test data sets are 0.719 and 0.674, respectively. The R values on the test dataset for the FA and ACO models are 0.671 and 0.673, respectively, showing only a slight difference. Considering that, it is better if the value of R is close to one, so, in conclusion, this model outperforms other models. Also, values of RAE, MAE, and RMSE obtained for the CBO model in the test data set, compared to FA and ACO models, are the lowest. These RAE, MAE, and RMSE values of the CBO model for the test data set are 1.020, 0.167, and 0.208, respectively. The smaller these values are, the more accurate the model will be, and at the same time, it will have fewer errors. By comparing the values obtained from RMSE and MAE metrics in the test data set, it can be seen that FA and ACO algorithms have similar performance. According to Fig. 6, the PI graphs show that the PI value for each model in the training and test datasets is as follows:

In training datasets: GMDH-CBO < GMDH-FA < GMDH-ACO

In testing datasets: GMDH-CBO = GMDH-ACO < GMDH-FA

The Std index has similar results, with the difference that the results obtained from the FA and ACO models are the same and equal to 0.209, while for the CBO model, it is 0.208. Finally, the maximum values of CBO, FA, and ACO hybrid models are 0.710, 0.746, and 0.729 for test datasets, respectively. It is clear that the CBO hybrid model still reports better results.

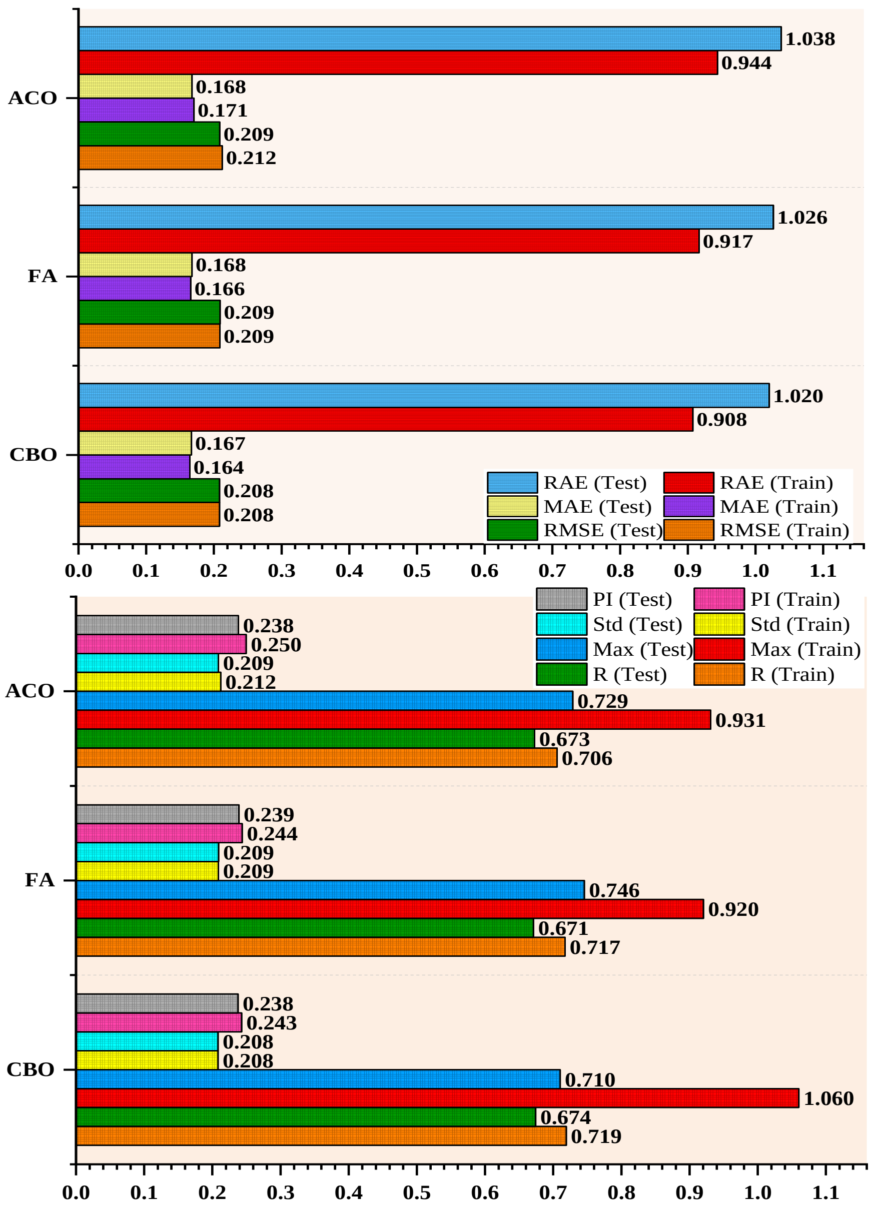


Figure 6: The performance of hybrid models based on seven metrics

The results of the CBO algorithm applied to the GMDH model to predict testability are shown in Fig. 7. According to the figure, among all inputs, the CSORDAvgLineCodeExe has the highest sensitivity. While for CSLEX Number of Return and Print Statements, it is the lowest value.

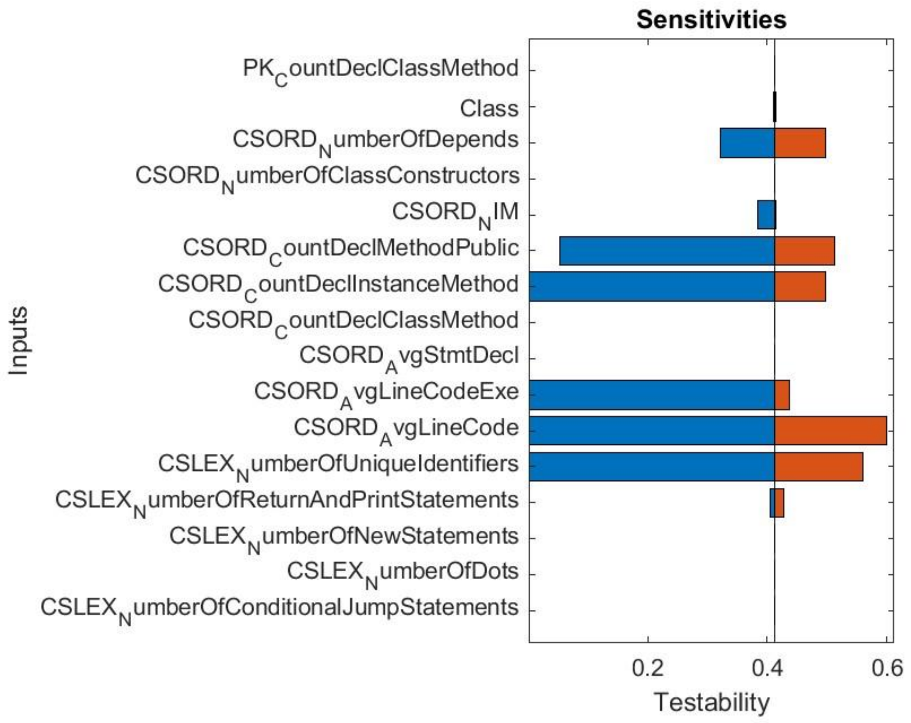


Figure 7: The tornado chart of the optimal hybrid model

To contextualize the performance of the proposed GMDH–CBO model, several widely used regression algorithms, including SVM, Random Forest, AdaBoost, MLP, and GLM, were implemented as baseline benchmarks. Table 2 summarizes a comprehensive comparison of all models using multiple error metrics (RMSE, MAE, R, RAE, Std, VAF, PI, and Max Error) on both training and testing sets. The results indicate that the proposed GMDH–CBO model outperforms all baseline and hybrid alternatives, achieving the lowest prediction errors (RMSE = 0.208, MAE = 0.167) and the highest correlation with the ground truth R = 0.674. In contrast, the best-performing baseline model GLM achieved RMSE = 0.2357 and R = 0.5505 on the test dataset. These findings confirm the superior predictive capability of the proposed optimization-driven GMDH framework compared to conventional machine-learning regressors.

Table 2: Error metrics obtained from the application of the hybrid models.

Model	Split	RMSE	MAE	R	RAE	Max Error	Std	VAF	PI
SVM	Train	0.2418	0.2013	0.6050	0.7837	1.2276	0.2418	35.0569	0.5443
	Test	0.2361	0.1933	0.5749	0.8185	0.7756	0.2361	30.0751	0.5963
RF	Train	0.2350	0.1931	0.6233	0.7519	0.7415	0.235	38.6752	0.495
	Test	0.2394	0.194	0.5318	0.8215	0.6579	0.2394	28.1005	0.5398
AdaBoost	Train	0.5585	0.4745	0.5793	1.8475	0.9840	0.2952	3.2091	1.8099
	Test	0.5700	0.4978	0.5214	2.1078	0.9840	0.278	3.0261	1.9603
MLP	Train	0.2831	0.2326	0.5564	0.9055	0.8505	0.2659	21.4686	0.7557
	Test	0.2866	0.2375	0.4598	1.0055	0.6956	0.2604	14.961	0.8382
GLM	Train	0.2292	0.1837	0.6454	0.7151	1.3188	0.2292	41.6601	0.4642
	Test	0.2357	0.1871	0.5505	0.7923	1.4766	0.2357	30.2906	0.5344
GMDH–CBO (Proposed)	Train	0.208	0.164	0.719	0.908	1.06	0.208	51.708	0.243
	Test	0.2080	0.167	0.6740	1.020	0.710	0.208	45.454	0.238
GMDH–FA	Train	0.2090	0.166	0.7170	0.917	0.920	0.209	51.479	0.244
	Test	0.2090	0.168	0.6710	1.026	0.746	0.209	45.018	0.239
GMDH–ACO	Train	0.2120	0.171	0.7060	0.944	0.931	0.212	49.794	0.25
	Test	0.2090	0.168	0.6730	1.038	0.729	0.209	45.242	0.238

To rigorously assess whether the performance differences between the optimized GMDH models were statistically meaningful, we applied the Wilcoxon signed-rank test to the absolute prediction errors on the test set. As summarized in Table 3, the CBO-based model significantly outperforms both FA ($p = 0.00034$) and ACO ($p = 1.55\times10^{-5}$), while the difference between FA and ACO is statistically insignificant ($p = 0.376$), confirming that the observed improvements of CBO are not due to random variation.

Table 3: Results of the Wilcoxon signed-rank test for pairwise model comparison.

Comparison	p-value	Significant
CBO vs FA	0.00034	Yes
CBO vs ACO	1.55×10^{-5}	Yes
FA vs ACO	0.376	No

A further comparison of the absolute errors is presented in Table 4, which gives the median error, the mean error and the effect size. Although in the pointwise measures of performance like R, RMSE, some differences in numbers are noticed, the statistical test validates that CBO portrays a marginal yet significant bitterness than FA and ACO. On the other hand, the values of accuracy of FA and ACO are very similar. We also observe that some of the individual measures like the maximum error are a little smaller in the case of the FA-based model; hence, the statement of superiority of CBO has been tempered to capture this aspect effectively.

Finally, the testability score used in this study corresponds to the static structural testability metric defined in the original dataset. It should be interpreted as an approximation of real-world testability rather than a direct measurement. The limitations of this proxy metric and its potential gap from practical testing effort have been explicitly discussed in the revised manuscript.

Table 4: Absolute errors of GMDH models with CBO, FA, and ACO.

Comparison	CBO vs FA	CBO vs ACO	FA vs ACO
Median Abs Err Model1	0.1450	0.1450	0.1444
Median Abs Err Model2	0.1444	0.1470	0.1470
Mean Abs Err Model1	0.1667	0.1667	0.1675
Mean Abs Err Model2	0.1675	0.1676	0.1676
P value	0.0003	0.0000	0.3761
Effect Size r	0.0597	0.0733	0.0156

Beyond the numerical gains, the superiority of CBO can be attributed to its suitability for a small-population, mixed-variable search on a noisy, rugged objective. With NP=2, CBO's collision-based updates sustain exploration and generate self-adaptive, fitness-scaled steps that balance diversification and exploitation without heavy parameter tuning. By contrast, ACO depends on pheromone reinforcement that is statistically fragile with very small swarms, often causing early stagnation or noisy trail updates; FA, driven by attractiveness and random walks, tends to over-disperse candidates in this regime and slows exploitation. Consequently, CBO converges faster and reaches lower terminal validation loss, while also exhibiting lower variance across folds when cross-validation is applied evidence of greater robustness to resampling noise and split variability. These dynamics explain the consistent edge of CBO over ACO and FA observed in our experiments.

4 Conclusion

Software testability is a very important technique for flawless software design. This process is necessary to diagnose software bugs. This study was conducted on software testability prediction using the GMDH of the ML model. Besides the original algorithm (GMDH), three meta-heuristic optimizer algorithms, including CBO, FA, and ACO, were integrated with GMDH to create hybrid models. Using meta-heuristic algorithms helps the GMDH network to be more effective. To increase the prediction accuracy and reliability, several statistical evaluation indices were used, namely RMSE, MAE, RAE, R, Max error, Std, VAF, and PI. Finally, by using different considerations, the prediction accuracy of the models was checked. Among all three models, GMDH-CBO was the only hybrid model that obtained the best values of each of the statistical evaluation metrics with an insignificant difference. The results obtained from the evaluation of the CBO model showed values of 0.208, 0.167, and 1.020 for RMSE, MAE, and RAE, respectively, which all are less than others. Overall, all the results of the static indicators obtained to predict software testability indicated that the GMDH-CBO evaluation performed the best. The proposed method is capable of predicting software testability. In other words, the accuracy of this model is higher, and it reports fewer errors.

With more research and studies, the algorithms used in this paper can be programmed singly or combined for different types of software systems existing in the industry and extended to different programming languages. Also, along with the development of prediction models using the preferred algorithms, hyper-parameter tuning of various function parameters can extend the current work.

Acknowledgements

I would like to take this opportunity to acknowledge that there are no individuals or organizations that require acknowledgment for their contributions to this work.

Authors' contributions

Xiaobo Wu: Writing-Original draft preparation, Conceptualization, Supervision, Project administration.

Conflicts of interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Author statement

The manuscript has been read and approved by all the authors, the requirements for authorship, as stated earlier in this document, have been met, and each author believes that the manuscript represents honest work.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Ethical approval

The research paper has received ethical approval from the institutional review board, ensuring the protection of participants' rights and compliance with the relevant ethical guidelines.

References

[1] Zakeri-Nasrabadi, M., S. Parsa and S. Jafari (2024). Measuring and improving software testability at the design level. Information and Software Technology, 174: 107511. https://doi.org/10.1016/j.infsof.2024.107511

[2] Garousi, V., M. Felderer and F.N. Kılıçaslan (2019). A survey on software testability. Information and Software Technology, 108(December 2018): 35–64. https://doi.org/10.1016/j.infsof.2018.12.003

[3] Zakeri-Nasrabadi, M. and S. Parsa (2022). An ensemble meta-estimator to predict source code testability[Formula presented]. Applied Soft Computing, 129: 109562. https://doi.org/10.1016/j.asoc.2022.109562

[4] Zakeri-Nasrabadi, M., S. Parsa and S. Jafari (2024). Measuring and improving software testability at the design level. Information and Software Technology, 174: 107511. https://doi.org/10.1016/j.infsof.2024.107511

[5] Talakola, S (2024). The optimization of software testing efficiency and effectiveness using AI techniques. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(3): 23–34. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I3P104

[6] Nascimento, A.M., G.K.G. Shimanuki and L.A. V Dias (2024). Making More with Less: Improving Software Testing Outcomes Using a Cross-Project and Cross-Language ML Classifier Based on Cost-Sensitive Training. Applied Sciences, 14(11): 4880. https://doi.org/10.3390/app14114880

[7] Pandy, G., V.J. Pugazhenthi and A. Murugan (2024). Advances in software testing in 2024: Experimental insights, frameworks, and future directions. International Journal of Advanced Research in Computer and Communication Engineering, 13(11): 40–44.

[8] Lin, S., R.Z.H. Liu and L. Tahvildari (2024). FlaKat: A Machine Learning-Based Categorization Framework for Flaky Tests. ArXiv Preprint ArXiv:2403.01003. https://doi.org/10.48550/arXiv.2403.01003

[9] Guglielmo, L., L. Mariani and G. Denaro (2024). Measuring software testability via automatically generated test cases. IEEE Access, 12: 63904–63916.

[10] Sharma, T., S. Georgiou, M. Kechagia, T.A. Ghaleb and F. Sarro (2023). Investigating developers' perception on software testability and its effects. Empirical Software Engineering, 28(5): 120. https://doi.org/10.1007/s10664-023-10373-0

[11] Stocco, A., O. Shehory, G. Jahangirova, V. Riccio, G. Barash, E. Farchi and D. Saha (2023). Software testing in the machine learning era: Special issue of the empirical Software Engineering (EMSE) journal. Empirical Software Engineering, 28(3): 74. https://doi.org/10.1007/s10664-023-10326-7

[12] Jahed Armaghani, D., M. Hasanipanah, H. Bakhshandeh Amnieh, D. Tien Bui, P. Mehrabi and M. Khorami (2020). Development of a novel hybrid intelligent model for solving engineering problems using GS-GMDH algorithm. Engineering with Computers, 36(4): 1379–1391. https://doi.org/10.1007/s00366-019-00769-2

[13] Mahdavi-Meymand, A., W. Sulisz and M. Zounemat-Kermani (2022). A comprehensive study on the application of firefly algorithm in prediction of energy dissipation on block ramps. Eksploatacja i Niezawodnosc, 24(2): 200–210. http://dx.doi.org/10.17531/ein.2022.2.2

[14] Mahdavi-Meymand, A. and M. Zounemat-Kermani (2020). A new integrated model of the group method of data handling and the firefly algorithm (GMDH-FA): application to aeration modelling on spillways. Artificial Intelligence Review, 53(4): 2549–2569. https://doi.org/10.1007/s10462-019-09741-4

[15] Srinivasan, D (2008). Energy demand prediction using GMDH networks. Neurocomputing, 72(1–3): 625–629. https://doi.org/10.1016/j.neucom.2008.08.006

[16] Mehra, R.K (1977). Group Method of Data Handling (Gmdh): Review and Experience. Proceedings of the IEEE Conference on Decision and Control, (4): 29–34. https://doi.org/10.1109/CDC.1977.271540

[17] Kaveh, A. and V.R. Mahdavi (2019). Multi-objective colliding bodies optimization algorithm for design of trusses. Journal of Computational Design and Engineering, 6(1): 49–59. https://doi.org/10.1016/j.jcde.2018.04.001

[18] Kaveh, A. and V.R. Mahdavi (2014). Colliding bodies optimization: A novel meta-heuristic method.

Computers and Structures, 139: 18–27. https://doi.org/10.1016/j.compstruc.2014.04.005

[19] Kaveh, A., M. Kamalinejad, H. Arzani and F. Barzinpour (2021). New enhanced colliding body optimization algorithm based on a novel strategy for exploration. Journal of Building Engineering, 43(February): 102553. https://doi.org/10.1016/j.jobe.2021.102553

[20] Kaveh, A. and M. Ilchi Ghazaan (2014). Computer Codes for Colliding Bodies OptimizationandIts Enhanced Version. International Journal of Optimization in Civil Engineering, 4(3): 321–339.

[21] Kaveh, A. and V.R. Mahdavi (2014). Colliding Bodies Optimization method for optimum design of truss structures with continuous variables. Advances in Engineering Software, 70: 1–12. https://doi.org/10.1016/j.advengsoft.2014.01.002

[22] Tong, N., Q. Fu, C. Zhong and P. Wang (2017). A multi-group firefly algorithm for numerical optimization. Journal of Physics: Conference Series, 887(1). DOI: 10.1088/1742-6596/887/1/012060

[23] Ren, H., H. Ren and Z. Sun (2023). HSFA: A novel firefly algorithm based on a hierarchical strategy. Knowledge-Based Systems, 279: 110950. https://doi.org/10.1016/j.knosys.2023.110950

[24] Yang, X.S. and X. He (2013). Firefly algorithm: recent advances and applications. International Journal of Swarm Intelligence, 1(1): 36. https://doi.org/10.1504/IJSI.2013.055801

[25] Fister, I., X.S. Yang and J. Brest (2013). A comprehensive review of firefly algorithms. Swarm and Evolutionary Computation, 13: 34–46. https://doi.org/10.1016/j.swevo.2013.06.001

[26] Johari, N.F., A.M. Zain, N.H. Mustaffa and A. Udin (2013). Firefly algorithm for optimization problem. Applied Mechanics and Materials, 421: 512–517. https://doi.org/10.4028/www.scientific.net/AMM.421.512

[27] Akhtar, A., (2019). Evolution of Ant Colony Optimization Algorithm -- A Brief Literature Review. https://doi.org/10.48550/arXiv.1908.08007

[28] Katiyar, S., Ibraheem and A.Q. Ansari (2015). Ant Colony Optimization : A Tutorial Review Ant Colony Optimization : A Tutorial Review Department of Electrical Engineering Corresponding Author : (Email : aqansari@ieee.org). (August):

[29] Ratner, B (2009). The correlation coefficient: Its values range between 1/1, or do they. Journal of Targeting, Measurement and Analysis for Marketing, 17(2): 139–142. https://doi.org/10.1057/jt.2009.5