Contents lists available at SCCE

Journal of Soft Computing in Civil Engineering

Journal homepage: www.jsoftcivil.com

Comparison of Machine Learning Classifiers for Reducing Fitness Evaluations of Structural Optimization

Tran-Hieu Nguyen[1*], **Anh-Tuan Vu**[2]

1. MSc., Ph.D. Student, Faculty of Building and Industrial Constructions, Hanoi University of Civil Engineering, Hanoi, Vietnam
2. Associate Professor, Faculty of Building and Industrial Constructions, Hanoi University of Civil Engineering, Hanoi, Vietnam

Corresponding author: *hieunt2@nuce.edu.vn*

https://doi.org/10.22115/SCCE.2021.306249.1367

ARTICLE INFO

Article history:
Received: 23 September 2021
Revised: 05 November 2021
Accepted: 22 November 2021

Keywords:
Evolutionary algorithm;
Differential evolution;
Surrogate model;
Machine learning classifier;
AdaBoost.

ABSTRACT

Metaheuristic algorithms have been widely used to solve structural optimization problems. Despite their powerful search capabilities, these algorithms often require a large number of fitness evaluations. Constructing a machine learning classifier to identify which individuals should be evaluated using the original fitness evaluation is a great solution to reduce the computational cost. However, there is still a lack of a thorough comparison between machine learning classifiers when integrating into the optimization process. This paper aims to evaluate the efficiencies of different classifiers in eliminating unnecessary fitness evaluations. For this purpose, the weight optimization of a double-layer grid structure comprising 200 members is used as a numerical experiment. Six machine learning classifiers selected for assessment in this study include Artificial Neural Network, Support Vector Machine, k-Nearest Neighbor, Decision Tree, Random Forest, and Adaptive Boosting. The comparison is made in terms of the optimal weight of the structure, the rejection rate as well as the computing time. Overall, it is found that the AdaBoost classifier achieves the best performance.

How to cite this article: Nguyen TH, Vu AT. Comparison of machine learning classifiers for reducing fitness evaluations of structural optimization. J Soft Comput Civ Eng 2021;5(4):57–73. https://doi.org/10.22115/scce.2021.306249.1367

1. Introduction

Structural optimization has attracted a great deal of interest from both academia and industry. Pioneering works of structural optimization were carried out by Maxwell in 1869 [1] and Mitchell in 1904 [2]. During the 1960s, Mathematical Programming techniques were applied to solve structural optimization problems, beginning with Schmit's work [3]. In the early 1990s, Genetic Algorithm (GA) was firstly used to optimize truss structures [4]. GA is a population-based metaheuristic algorithm developed by John Holland in 1975. In comparison with classical optimization approaches, metaheuristics have some advantages such as easily escaping from local optima, handling discrete variables. During the last four decades, many metaheuristic algorithms have been developed and successfully applied to solve structural optimization, for example, Genetic Algorithm (GA) [4], Evolution Strategy [5], Differential Evolution (DE) [6], Particle Swarm Optimization (PSO) [7], Ant Colony Optimization (ACO) [8], etc.

Despite their advantages as mentioned above, metaheuristic algorithms often require a large number of fitness evaluations to obtain a good solution. In structural optimization, the evaluation of the fitness function is often computationally expensive due to conducting time-consuming finite element analyses. In the following cases, one potential solution is to construct an approximate model that can quickly evaluate fitness function [9–14]. Besides, there is another approach that is to employ machine learning (ML) classifiers to identify which individuals should be evaluated using the original fitness evaluation. Rosso et al. [15] proposed a hybrid method called SVM-PSO for structural optimization in which a Support Vector Machine (SVM) model is integrated into the PSO process with the aim of separating feasible and infeasible solutions, thereby reducing the search space. The proposed method is applied to solve two numerical examples of a simply supported beam and a Warren truss beam. Recently, Nguyen and Vu [16] employed Artificial Neural Network (ANN) to classify the safety state of a structural solution. The ANN classification model is used in conjunction with the objective function comparison for eliminating worse individuals during the DE optimization. An example of a 47-bar planar tower is carried out and the results show that the proposed method saves about 20% of fitness evaluations.

It is obvious that there exist many different ML classification algorithms and each algorithm will achieve its own percentage of rejection. Although there are numerous comparative studies on the performances of ML classifiers in structural problems [17–19], there is still a lack of a thorough evaluation of ML classifiers when embedding them into the optimization process. This is the aim of this study.

For this purpose, six commonly used ML classifiers including ANN, SVM, k-Nearest Neighbor (kNN), Decision Tree (DT), Random Forest (RF), and Adaptive Boosting (AdaBoost) are carefully chosen for the comparison. The weight optimization of a double-layer grid structure of 200 members is used as a numerical experiment. The comparison is made in terms of the optimal weight of the structure, the percentage in eliminating unnecessary fitness evaluations as well as the computing time.

The remainder of this paper is organized as follows. The paper first describes the technique for reducing fitness evaluations using ML classifiers. Six considered ML classifiers are briefly introduced in Section 3. In the following section, the comparison is conducted and the findings are discussed. Section 5 concludes this paper.

2. Reduction of fitness evaluations using machine learning classifiers

2.1. Statement of a structural optimization problem

Structural optimization problems can be formulated as a constrained optimization problem (COP) where the objective function can be the weight or the cost of the whole structure while the constraints are often specified in design standards like stress conditions, buckling conditions, displacement conditions. The formulation of a structural optimization problem is expressed as follows:

$$\begin{aligned} &\text{Find} && \mathbf{x} = \{x_i, i = 1,2,...,n\} \\ &\text{to minimize:} && f(\mathbf{x}) = \sum_{i=1}^{n} \rho_i l_i x_i \\ &\text{subject to:} && \begin{cases} g_j(\mathbf{x}) \le 0, \; j = 1,2,...,m \\ \quad l_i \le x_i \le u_i \end{cases} \end{aligned} \tag{1}$$

where: **x** denotes the design variable vector; x_i is the *i*th design variable; n is the number of design variables; $f(\mathbf{x})$ is the objective function; $g_j(\mathbf{x})$ is the *j*th constraint; m is the number of constraints; l_i, u_i are lower and upper limits of the variable x_i, respectively.

Metaheuristic algorithms are designed for solving unconstrained optimization problems. To handle COPs, a widely-used technique called the penalty method is applied in this study as follows:

$$F(\mathbf{x}) = (1 + \varepsilon_1 \cdot cv)^{\varepsilon_2} \times f(\mathbf{x}) \tag{2}$$

where: $F(\mathbf{x})$ is called the penalty function; cv denotes the degree of constraint violation; ε_1 and ε_2 are two parameters that are carefully selected to ensure a good balance between the exploration and the exploitation during the optimization process. In this study, ε_1 is set 1 and ε_2 is linearly increased from 20 to 40 based on the recommendation of Ref. [20]. The degree of constraint violation cv is determined by taking a sum of all violated constraints as follows:

$$cv = \sum_{j=1}^{m} \max\left(0, g_j(\mathbf{x})\right) \tag{3}$$

The variables can be the cross-sectional areas of structural members for sizing optimization, or the nodal coordinates for shape optimization, or the material distribution for topology optimization. This study is limited to sizing optimization.

2.2. Technique to reduce fitness evaluations using machine learning classifiers

The application of machine learning classifiers to eliminate worse individuals during the optimization process is not new. This technique has been proposed in Ref. [16] and successfully applied to solve the problem of the 47-bar planar tower. The idea behind this technique is based on the observation that during the DE optimization, many trial individuals are worse than their parent individuals and will lose the pairwise tournament at the selection step. A classification model with the ability to detect whether an individual is feasible or not could save many useless fitness evaluations. The present study uses the same technique with few enhancements to improve efficiency.

In particular, the optimization process is split into two stages. At stage I, after initializing a population $\mathbf{P}_0=\{\mathbf{x}_{i,0}|i=1,\ldots,NP\}$, three operators of the original DE algorithm (mutation, crossover, and selection) are sequentially carried out. At the certain iteration g, the current population is $\mathbf{P}_g=\{\mathbf{x}_{i,g}|i=1,\ldots,NP\}$. The mutation operator creates the mutant individual $\mathbf{v}_{i,g}$ while the crossover produces the trial individual $\mathbf{u}_{i,g}$ by taking some components of $\mathbf{v}_{i,g}$ and the rest from $\mathbf{x}_{i,g}$. Next, the selection operator chooses the better one among $\mathbf{u}_{i,g}$ and $\mathbf{x}_{i,g}$ to enter the next iteration. Implementing iterative three operators aims to move the population towards the optimum. The DE is a well-established metaheuristic algorithm that has been introduced in many previous documents. To avoid wordiness, the formulations of the DE algorithm are not presented in this paper. Readers can refer to Ref. [6,14] for more details.

It is noted that each metaheuristic algorithm always comprises the exploration and the exploitation tasks, in which the exploration is the process of expanding bounding regions to find new solutions while the exploitation is the process of improving current solutions. The implementation of the original DE in the first stage ensures the exploration and collects data for training machine learning models. Accordingly, each newly produced individual is evaluated by the original fitness function and it is either labeled "+1" if it is feasible or "−1" otherwise:

$$\begin{cases} y_{i,g}=+1 \text{ if } cv\left(\mathbf{x}_{i,g}\right)=0 \\ y_{i,g}=-1 \text{ if } cv\left(\mathbf{x}_{i,g}\right)>0 \end{cases} \tag{4}$$

All newly evaluated individuals ($\mathbf{x}_{i,g}$, $y_{i,g}$) are saved into the database. After several iterations, a training dataset is collected from historical fitness evaluations and an ML-based classification model is trained with the aim of classifying the feasibility of individuals.

In stage II, the label of a trial individual $\mathbf{u}_{i,g}$ produced by two operators mutation and crossover is predicted using the classification model just trained at the end of the previous stage. Three possible situations can occur as follows: (i) if the predicted label $y_{i,pred}$=+1, the trial individual $\mathbf{u}_{i,g}$ is then evaluated using the original fitness function; (ii) if the predicted label $y_{i,pred}$=−1 and the objective function of the trial individual $f(\mathbf{u}_{i,g})$ is smaller than that of the target individual $f(\mathbf{x}_{i,g})$, it is also evaluated using the original fitness function; (iii) if the predicted label $y_{i,pred}$ =−1 and $f(\mathbf{u}_{i,g})>f(\mathbf{x}_{i,g})$, it is eliminated. Among three situations, there is one case that does not require conducting the original fitness evaluation. As a result, the computational cost is significantly reduced. This technique is called the Classification-Assisted Differential Evolution (CADE). Both flowcharts of the CADE and the DE are schematized in Fig. 1 to clearly illustrate the proposed technique.

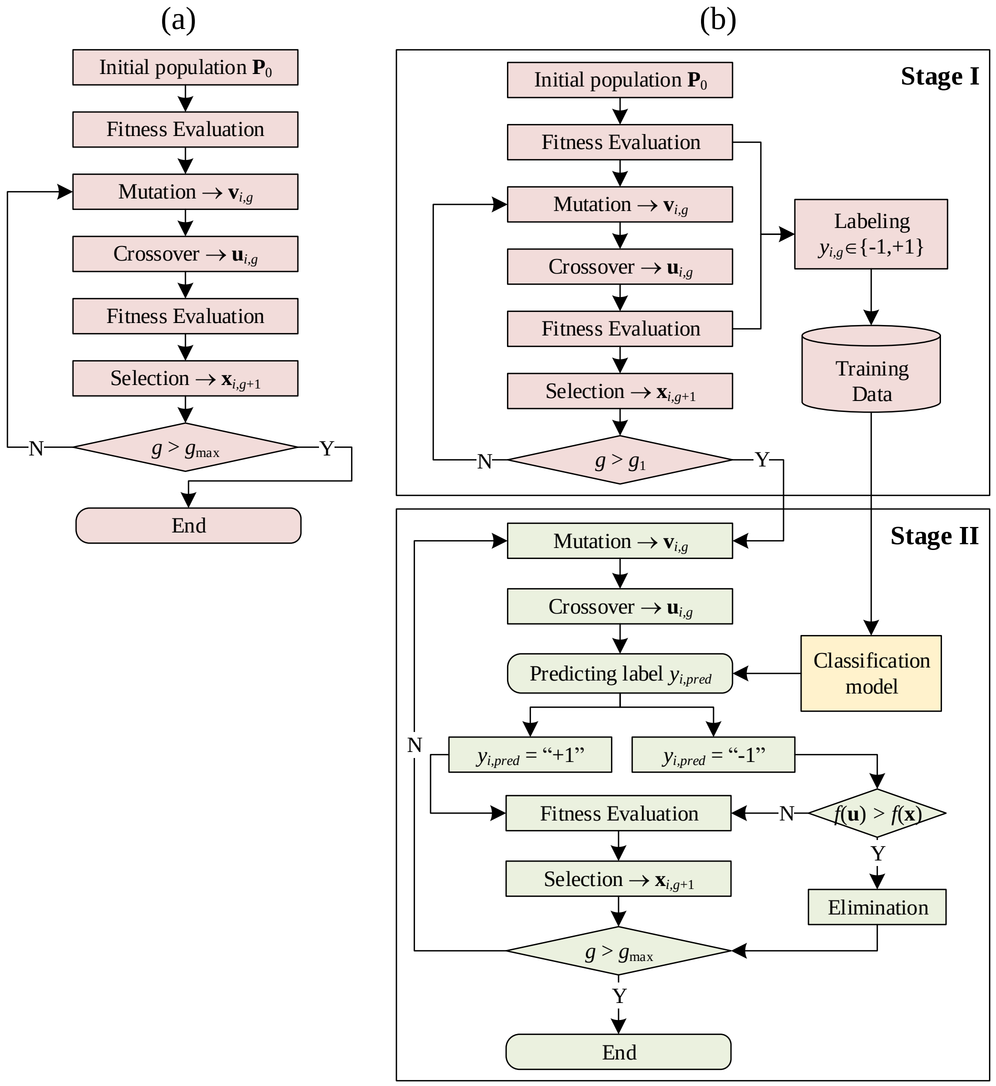
(a)
(b)
Initial population $\mathbf{P}_0$
Fitness Evaluation
Mutation $\rightarrow \mathbf{v}_{i,g}$
Crossover $\rightarrow \mathbf{u}_{i,g}$
Selection $\rightarrow \mathbf{x}_{i,g+1}$
$g > g_{\max}$
N
Y
End
Stage I
Labeling $y_{i,g} \in \{-1,+1\}$
Training Data
$g > g_1$
Stage II
Predicting label $y_{i,pred}$
Classification model
$y_{i,pred}$ = "+1"
$y_{i,pred}$ = "-1"
$f(\mathbf{u}) > f(\mathbf{x})$
Elimination

Fig. 1. Flowcharts: (a) Differential Evolution; (b) Classification-Assisted Differential Evolution.

3. Review of machine learning classifiers

Obviously, any ML classifier can be used in the CADE. In this work, six most commonly used classifiers that are selected for the comparison include Artificial Neural Network, Support Vector Machine, k-Nearest Neighbor, Decision Tree, Random Forest, and Adaptive Boosting. In the following subsections, these classifiers are briefly introduced.

3.1. Artificial neural network (ANN)

The ANN model comprises many nodes which are arranged into three types of layer: input layer, hidden layers, and output layer (Fig. 2). In this model, the weighted sum of outputs from nodes of the previous layer becomes the input for nodes of the current layer. At each node, the input is transformed into the output by the activation function like the sigmoid function, the tanh function, the softmax function, or the ReLU function. These nonlinear functions will allow the model to create a complex mapping between the inputs and the outputs of the data. The model is trained using a technique called the back-propagation algorithm for minimizing the error between the predictions and ground truth values. ANN models can well handle both regression and classification tasks. For classification tasks, the cross-entropy is frequently used to measure the error.

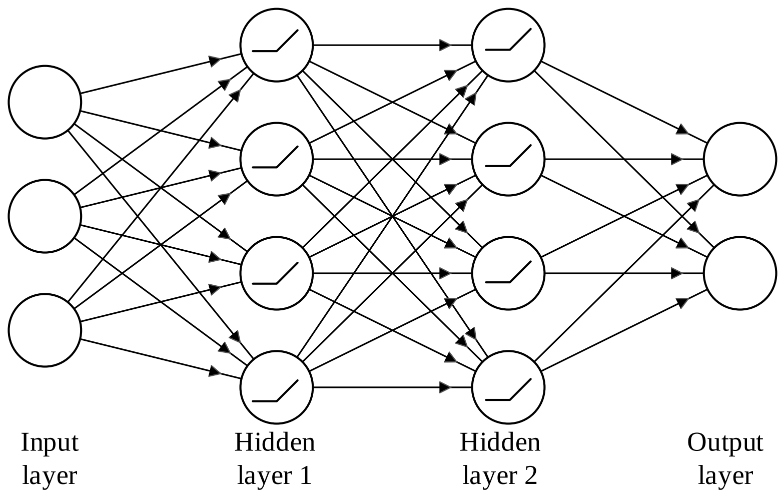

Fig. 2. Artificial Neural Network.

3.2. Support vector machine (SVM)

The goal of this algorithm is to find an optimal hyperplane in multi-dimensional space that can separate data points into distinct regions. Fig. 3 illustrates the SVM model for a binary classification problem with two features. There are, in fact, many hyperplanes that could be detected. The optimal hyperplane is the one having the largest margin.

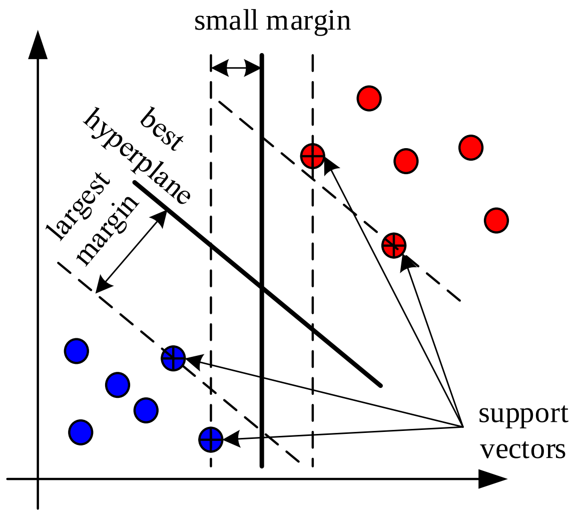

Fig. 3. Support Vector Machine.

3.3. k-Nearest neighbor (kNN)

The kNN is categorized as a "lazy learner" algorithm in which all data points are stored in the database without conducting any training process. When a prediction is requested, the algorithm will search k points that are closest to the considered point, and then the prediction is determined by taking a plurality vote of newly detected neighbors. In addition, a useful technique can be applied to improve the accuracy where distance-based weights are assigned to neighbors. This technique ensures that nearer neighbors will contribute more to the prediction than that of far neighbors. Euclidean, Manhattan, and Minkowski distances are three distance functions that are commonly used in the kNN. Fig. 4 displayed a simply kNN model with two cases of k=3 and k=7.

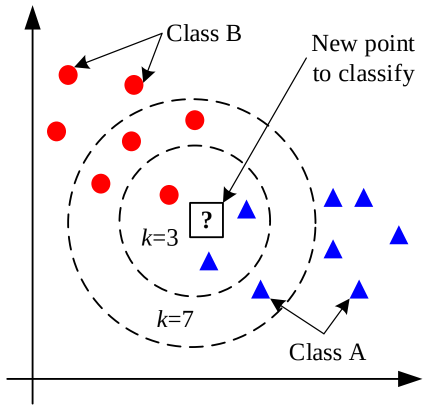

Fig. 4. k-Nearest Neighbor.

3.4. Decision tree (DT)

The DT model is a tree-like graph that is frequently used to split data into smaller subsets. A simple DT model is visualized in Fig. 5, in which, each internal node denotes a test on an attribute, and each branch represents an outcome of the test and each leaf node holds a class label. The quality of a split can be measured using the information gain or the Gini impurity.

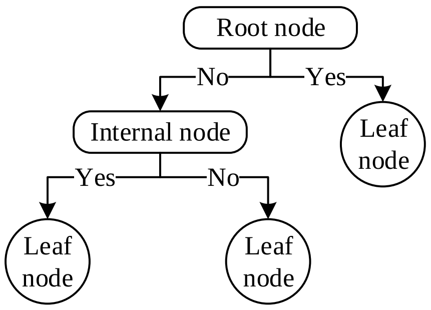

Fig. 5. Decision Tree.

3.5. Random forest (RF)

There is an effective way to enhance the classification accuracy by using many DT models simultaneously. Each DE model is trained by a random subset from the training data. The final output of the prediction is obtained through the majority vote. This technique is called bagging and this is very efficient to improve the stability for unstable algorithms like DT or ANN. One of the most bagging-type methods is the RF where only a few features are randomly chosen to train the DT model. An illustration of the concept of the RF is presented in Fig. 6.

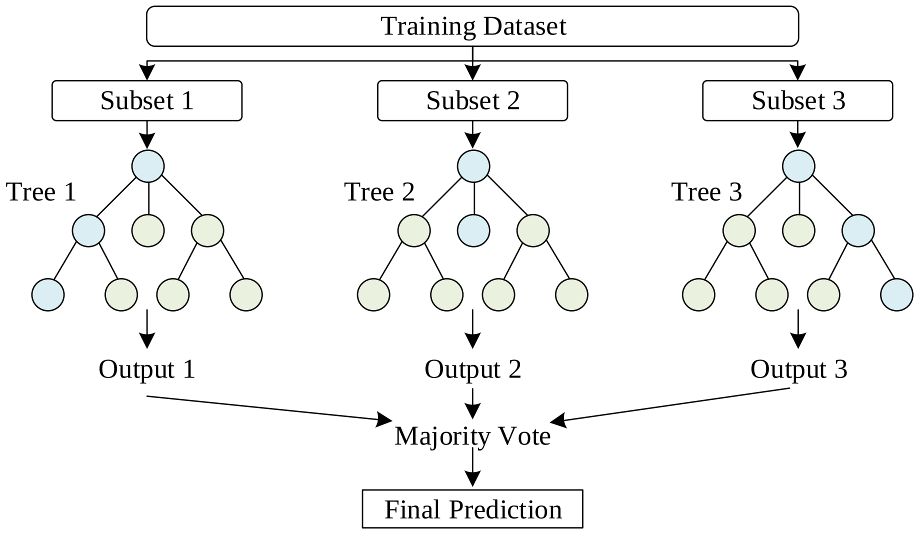

Fig. 6. Random Forest.

3.6. Adaptive boosting (AdaBoost)

Boosting is also a method for reducing the variance and the bias of machine learning models. Like bagging, boosting combines several weak classifiers to create a strong classifier. The main difference between the two methods is that weak classifiers in bagging are built independently while weak models in boosting are trained sequentially. There exist many boosting algorithms, of which the AdaBoost is the first one. This algorithm is proposed by Schapire and Freund in 1997 [21]. Fig. 7 illustrates the initial idea of the AdaBoost.

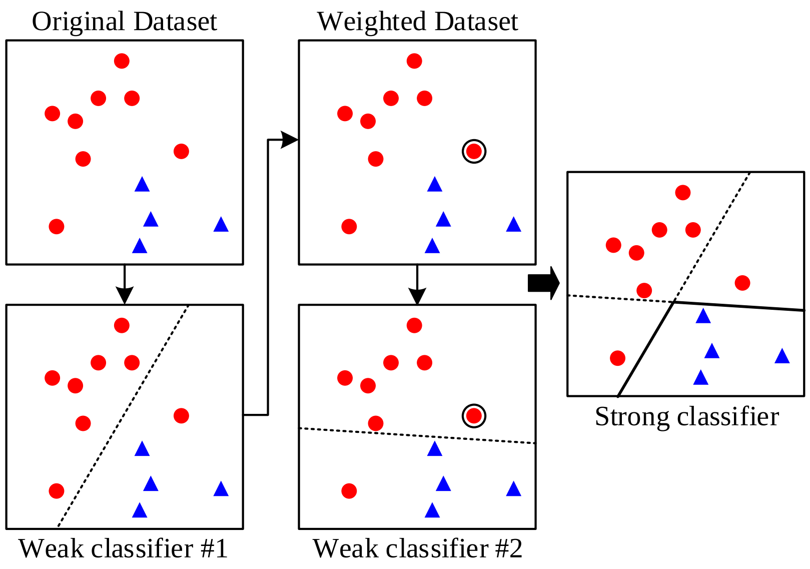

Fig. 7. Adaptive Boosting.

4. Comparison

In this section, a numerical example of a real-size structure is carried out using both the original DE and the CADE. As mentioned above, the CADE can be used in conjunction with any classification algorithm. Six ML classifiers selected in Section 3 will be employed in turn and their efficiencies are compared thoroughly.

4.1. Numerical example

The example considered in this work is a 10x10m double-layer grid structure consisting of 200 bars. The aim of this example is to find the cross-sectional areas of members for minimizing the weight of the structure. The formulation of this problem is expressed as follows:

$$\begin{aligned}
&\text{Find} && \mathbf{A} = \left[A_1, A_2, \ldots, A_{ng}\right] \\
&\text{to minimize:} && W(\mathbf{A}) = \rho \sum_{i=1}^{ng} A_i \sum_{j=1}^{nm(i)} L_j \\
&\text{subject to:} && \begin{cases} g_k(\mathbf{A}) \le 0, k = 1,2,\ldots,nc \\ A_i \in \mathbf{S} \end{cases}
\end{aligned} \tag{5}$$

in which: A_i is the cross-sectional area of members of the ith group; ng is the number of groups; L_j is the length of jth member; $nm(i)$ is the number of members which belongs to the ith group; ρ is the density of the material used in this structure; $g_k(\mathbf{A})$ denotes the kth constraint; nc is the number of constraints; **S** represents the list of available profiles.

In this problem, members are divided into ng=9 groups as presented in Fig. 8. All members are fabricated from the material steel having mechanical properties as follows: the density ρ=7850 kg/m^3; the modulus of elasticity E=2.1E+10 kg/m^2; and the yield strength F_y=3.515E+07 kg/m^2. This structure subjects to a load of q=250 kgf/m^2 on the top surface. The list of available profiles S is described in Table 1. There are three types of constraints: tensile stress constraint, compressive stress constraint, slenderness ratio constraint, and displacement constraint. The tensile stress constraint is as follows:

$$\sigma_t \le \phi_t F_y \qquad \phi_t = 0.9 \tag{6}$$

where: σ_t is the tensile stress of the member; F_y is the yield strength.

The compressive stress constraint is as follows:

$$\sigma_c \le \phi_c F_{cr}; \qquad \phi_c = 0.85 \tag{7}$$

where: σ_c is the compressive stress of the member; F_{cr} is the critical stress of the corresponding member which is calculated based on the following formula:

$$F_{cr} = \begin{cases} \left(0.658^{F_y/F_e}\right) F_y & \text{if } KL/r \le 4.71\sqrt{E/F_y} \\ 0.877 F_e & \text{if } KL/r > 4.71\sqrt{E/F_y} \end{cases} \tag{8}$$

where: $F_e = \pi^2 E/(KL/r)^2$; K is the effective length factor which equals 1.0 in this case; L is the length of the member; and r is the radius of gyration of the member's cross-section

The limitations of the slenderness ratio (KL/r) are 200 for compression members while 300 for tension members. Finally, the allowable vertical displacement of this structure equals 7.5 cm.

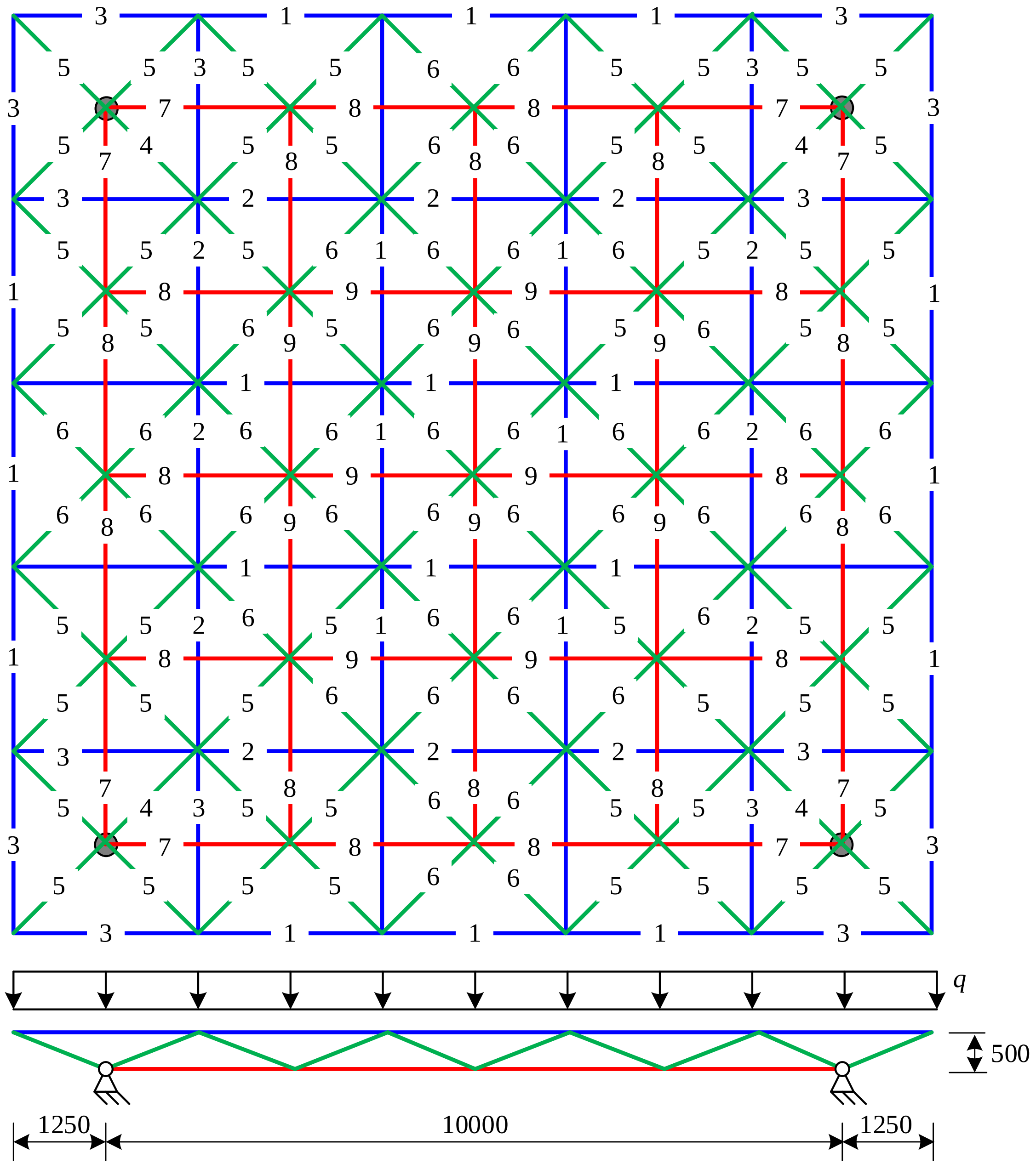

Fig. 8. Configuration of the 10x10m double layer grid structure.

Table 1
List of available profiles.

No.	Profile	A (cm^2)	r (cm)	No.	Profile	A (cm^2)	r (cm)
1	ϕ33.70×2.6	2.54	1.1	14	ϕ159.0×4.0	19.48	5.4814
2	ϕ48.30×2.6	3.73	1.62	15	ϕ168.3×4.0	20.65	5.8102
3	ϕ60.30×3.2	5.74	2.02	16	ϕ193.7×4.5	26.75	6.6922
4	ϕ76.10×3.2	7.329	2.5799	17	ϕ219.1×5.0	33.63	7.5716
5	ϕ82.50×3.2	7.972	2.806	18	ϕ244.5×5.4	40.56	8.4557
6	ϕ88.90×3.2	8.616	3.0321	19	ϕ273.0×5.6	47.04	9.457
7	ϕ101.6×3.6	11.08	3.4672	20	ϕ298.5×5.9	54.23	10.3471
8	ϕ108.0×3.6	11.81	3.6934	21	ϕ323.9×5.9	58.94	11.245
9	ϕ114.3×3.6	12.52	3.9161	22	ϕ355.6×6.3	69.13	12.3536
10	ϕ127.0×4.0	15.45	4.3504	23	ϕ368.0×6.3	71.59	12.7895
11	ϕ133.0×4.0	16.21	4.5629	24	ϕ406.4×6.3	79.19	14.1475
12	ϕ139.7×4.0	17.05	4.8004	25	ϕ419.0×7.1	91.88	14.5645
13	ϕ152.4×4.0	18.65	5.2483	26	ϕ457.2×7.1	100.4	15.915

4.2. Experimental setup

This problem is categorized as a discrete optimization problem in which design variables are selected among a limited number of values. In this work, a simple technique is employed to handle discrete variables. The design variable $A_i \in \mathbf{S}$ in Eq. (5) are replaced by an integer $I_i \in \{1,2,\ldots,26\}$ representing the sequence number of the considered profile in the list **S**. During the optimization process, a continuous value that is newly generated by the mutation and crossover operators is rounded to the nearest integer value. The cross-sectional area A_i and the radius of gyration r_i corresponding to the position I_i are looked up from Table 1. Next, their values are used to evaluate the feasibility of the solution as well as save as the inputs of the training data. According to the evaluation result, this solution is labeled and the value is saved as the output of the training data. The data processing flow is illustrated in Fig. 9.

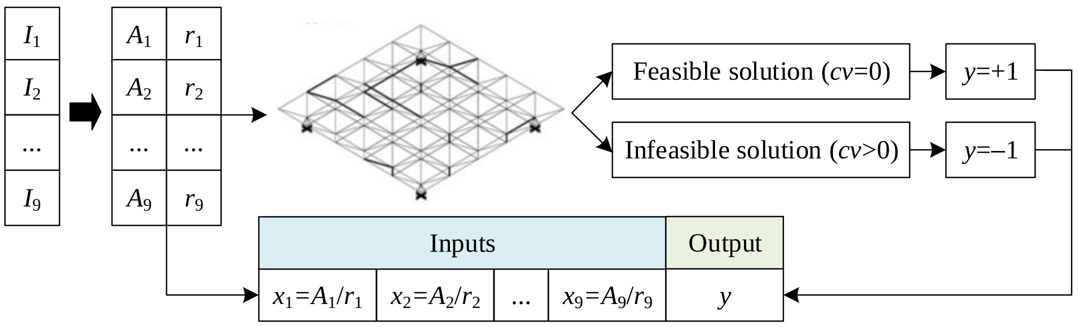

Fig. 9. Data processing flow.

Both the DE and the CADE use “DE/current-to-best/1” which is recognized as one of the most effective mutation strategies. For a fair comparison, two algorithms are set with the same parameters as follows: the number of individuals NP=25, the scale factor F=0.8, the crossover rate Cr=0.9, and the maximum iterations g_{max}=100. In particular, the number of the first stage of the CADE is set to g_1=20 which means the training dataset consists of 500 samples. Both the DE and the CADE algorithms are written in Python language. The finite element analysis code is developed based on the direct stiffness method. All ML models are constructed using the open-source library scikit-learn [22]. Optimal parameters of these models that were found after the hyperparameter tuning process are presented in Table 2.

Table 2
Summary of ML model parameters.

	Parameters
ANN	activation='relu', solver='adam', batch_size=10, max_iter=1000, hidden_layer_sizes=(200,200,200,), shuffle=True, early_stopping=True
SVM	kernel='rbf', C=1000, gamma=0.1
kNN	n_neighbors=50
DT	criterion='gini', max_depth=None
RF	n_estimators=50, max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=1e-07, bootstrap=True
AdaBoost	DecisionTreeClassifier(max_depth=1), n_estimators=50

The performance of machine learning models is evaluated through the accuracy metric. Due to the limited number of data samples, a widely-used technique called k-fold cross-validation is employed. In more detail, the training dataset is split into k subsets, in which the model is trained on (k-1) parts and then the trained model is validated on the remaining one. The process is repeadly carried out for k times. In this study, the parameter k is set to 5 and the mean values of the obtained accuracies are reported in Table 3. Next, the model is re-trained with the full training dataset and it will be embedded into the DE optimization. Particularly for the ANN model, the early stopping technique is used for preventing the overfitting phenomena. This feature is available in the library scikit-learn and it is enabled by the setting “early_stopping=True” when defining the model.

Table 3
Accuracies of ML models.

	ANN	**SVM**	**kNN**	**DT**	**RF**	**AdaBoost**
Accuracy	0.775	0.757	0.651	0.884	0.903	0.905

It is clearly seen that two ensemble models including the RF and the AdaBoost reach high accuracy (over 90%), followed by the DT (88.4%). The ANN and the SVM achieve an accuracy of approximately 75% while the kNN has the worst performance with an accuracy of 65.1%. The observation is consistent with the findings of Ref. [19].

4.3. Results

These are a total of seven cases that are carried for comparison. In general, the name of each case has the same form "A(B)" in which A represents the optimization algorithm and B denotes the ML classifier. For example, the case "CADE(AdaBoost)" means that the CADE is used to optimize the weight of the structure while the AdaBoost classifier is employed to reduce fitness evaluation. In particular, the name "DE" particularly denotes that the original DE is employed in this case. Each case is carried out 30 times. The best results in 30 independent runs are presented in Table 4. It is noted that all six cases of the CADE are performed, however, the best designs found by them are the same. Therefore, Table 4 presents one case of CADE. Additionally, the design found by Gholizadeh [23] using the commercial software SAP2000 is also reported to demonstrate the effectiveness of the proposed method. Fig. 10 shows the convergence curves of seven cases. The values indicated in Fig. 10 are averages of 30 runs. To compare the efficiency of six classifiers in reducing fitness evaluations, five metrics are used including the minimum weight (best), the average weight (mean), the standard deviation (SD), the average number of fitness evaluations (nFE), and the average computing time (time) of 30 independent runs. The obtained values of five metrics for six cases are reported in Table 5.

Table 4
Best results for the double-layer grid structure.

	Gholizadeh [23]	**DE**	**CADE**
A_1	ϕ76.10×3.2	ϕ76.10×3.2	ϕ76.10×3.2
A_2	ϕ101.6×3.6	ϕ101.6×3.6	ϕ101.6×3.6
A_3	ϕ33.70×2.6	ϕ33.70×2.6	ϕ33.70×2.6
A_4	ϕ101.6×3.6	ϕ101.6×3.6	ϕ101.6×3.6
A_5	ϕ60.30×3.2	ϕ60.30×3.2	ϕ60.30×3.2
A_5	ϕ33.70×2.6	ϕ33.70×2.6	ϕ33.70×2.6
A_7	ϕ76.10×3.2	ϕ76.10×3.2	ϕ76.10×3.2
A_8	ϕ48.30×2.6	ϕ33.70×2.6	ϕ33.70×2.6
A_9	ϕ33.70×2.6	ϕ48.30×2.6	ϕ48.30×2.6
Weight (kg)	1702.73	1683.904	1683.904

Table 5
Evaluation metrics of six classifiers.

	best (kg)	**mean (kg)**	**SD (kg)**	**nFE**	**time (s)**
CADE(ANN)	1683.904	1686.596	6.151	2095	80.6
CADE(SVM)	1683.904	1719.392	155.850	1820	66.7
CADE(kNN)	1683.904	1687.766	7.193	1818	45.4
CADE(DT)	1683.904	1691.031	12.298	2030	43.7
CADE(RF)	1683.904	1691.710	11.002	1819	54.5
CADE(AdaBoost)	1683.904	1687.218	6.762	1743	48.9

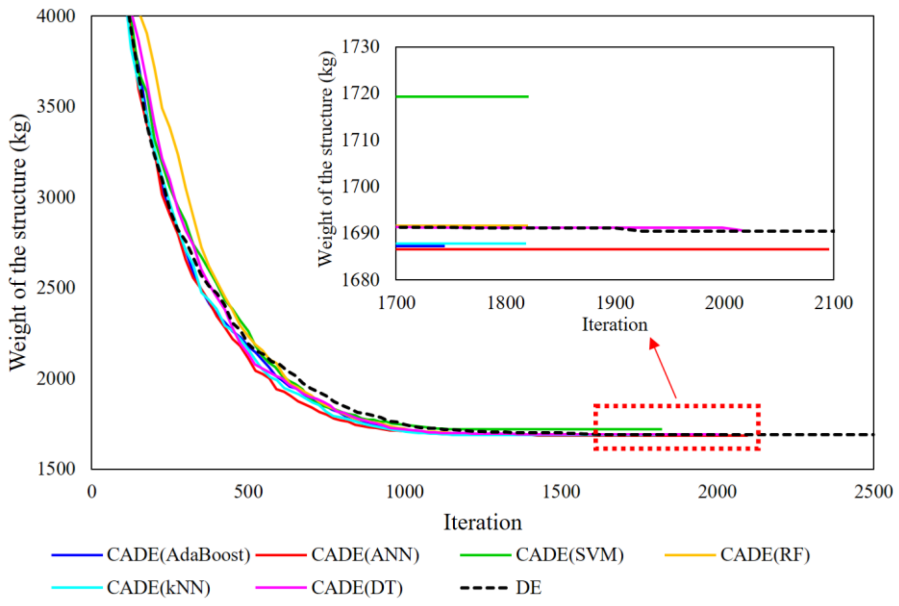

Fig. 10. Convergence curves of seven cases.

4.4. Comparison and discussion

Based on the results described in Table 5, six CADE cases are ranked according to each metric. Next, their ranks are displayed on a radar chart as shown in Fig. 11. Some observations are as follows.

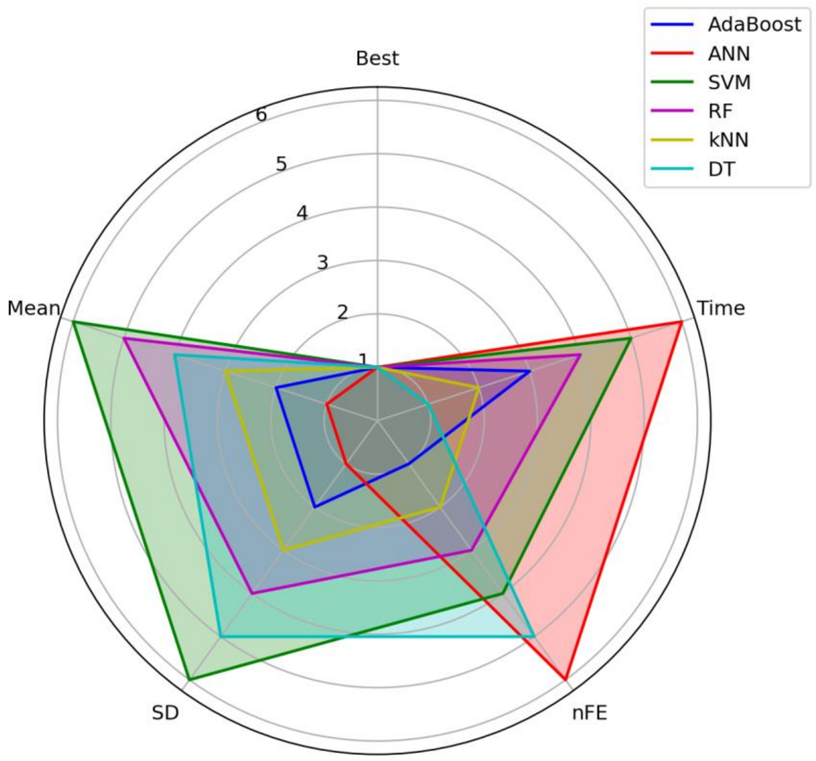

Fig. 11. Comparison of six ML classifiers in the CADE.

First of all, it can be seen that the weights of the grid structure found by the DE and the CADE (1683.904 kg) are smaller than that of Gholizadeh (1702.73 kg). It demonstrates the effectiveness of applying metaheuristic algorithms in structural optimization.

Secondly, Fig. 10 indicates that the convergence speeds of all CADE cases are faster than the original DE. Specifically, the DE finishes after 2500 iterations while six CADE cases converge after about 1700 to 2100 iterations. Among six cases, the CADE(AdaBoost) is fastest with the average number of fitness evaluations of 1743, followed by the CADE(kNN) with 1818 evaluations, the CADE(RF) with 1819 evaluations, the CADE(SVM) with 1820 evaluation, the CADE(DT) with 2030 evaluations, and the CADE(ANN) with 2095 evaluations. In comparison with the original DE, the reduction rates of six cases are 30.3%, 27.3%, 27.2%, 27.2%, and 16.2%, respectively.

However, there is an inconsistency when comparing the number of fitness evaluations and the computing time. Although the CADE(DT) requires an average of 2030 evaluations, the total time of the CADE(DT) is 43.7s, ranking first in 6 cases. Next is the CADE(kNN), the CADE(AdaBoost), the CADE(RF), the CADE(SVM), and the CADE(ANN). The reason is that the training speed of the DT model is very fast. It should be noted that for large-scale structures where the time for each finite element analysis is much larger than the training model time, the above ranking can be changed.

Comparing in terms of the optimal weight, the CADE(ANN) is superior to others. In more detail, the mean value and the SD value of the CADE(ANN) are 1686.596 kg and 6.151 kg, respectively. The following cases are: the CADE(AdaBoost) with mean=1687.218 kg and SD=6.762 kg; the CADE(kNN) with mean=1687.766 kg and SD=7.193 kg, the CADE(DT) with mean=1691.031 kg and SD=12.298 kg, the CADE(RF) with mean=1691.71 kg and SD=11.002 kg, and the CADE(SVM) with mean=1719.392 kg and SD=155.85 kg.

Overall, it is apparent that the application of ML classifiers into the optimization process significantly reduces the number of fitness evaluations. The reduction rate ranges from 16% to 30%. The CADE(AdaBoost) achieves the best performance when balancing five evaluation metrics.

5. Conclusions

This paper presents a hybrid method called CADE with the aim of reducing the number of fitness evaluations of metaheuristic algorithms. In this method, an ML classifier is constructed based on historical fitness evaluations in order to predict the feasibility of newly produced individuals. In later generations, the ML classifier is employed as a filter to eliminate worse individuals, thereby, omitting many useless fitness evaluations.

For comparing the efficiencies of ML classifiers, a 10x10m double-layer grid structure is optimized using six cases of the CADE with six different ML classifiers. The results show that applying ML classifier rejects approximately 16 to 30% of fitness evaluations. Among six compared ML classifiers, the AdaBoost gives the best performance. With lower computationally cost, the AdaBoost classifier-assisted Differential Evolution is very effective when optimizing large-scale structures.

Acknowledgments

The author T.-H. Nguyen was funded by Vingroup Joint Stock Company and supported by the Domestic Ph.D. Scholarship Programme of Vingroup Innovation Foundation (VINIF), Vingroup Big Data Institute (VINBIGDATA) code VINIF.2020.TS.134.

Funding

This research received funding from Vingroup Innovation Foundation VINIF under grant number VINIF.2020.TS.134.

Conflicts of Interest

The authors declare no conflict of interest.

Authors Contribution Statement

T.-H. Nguyen: Conceptualization; Methodology; Data curation; Formal analysis; Investigation; Writing – original draft. A.-T. Vu: Conceptualization; Methodology; Investigation; Supervision; Writing – review & editing.

References

[1] Maxwell JC. I.— On Reciprocal Figures, Frames, and Diagrams of Forces. Trans R Soc Edinburgh 1870;26:1–40. https://doi.org/10.1017/S0080456800026351.

[2] Michelle AGM. The Limits of Economy of Material in Frame-structures, Philos Mag 1906.

[3] Schmit LA. Structural design by systematic synthesis. Proc. Second Natl. Conf. Electron. Comput. ASCE, Sept., 1960, 1960.

[4] Goldberg DE, Samtani MP. Engineering optimization via genetic algorithm. Electron. Comput., ASCE; 1986, p. 471–82.

[5] Papadrakakis M, Lagaros ND, Thierauf G, Cai J. Advanced solution methods in structural optimization based on evolution strategies. Eng Comput 1998;15:12–34. https://doi.org/10.1108/02644409810200668.

[6] Wang Z, Tang H, Li P. Optimum Design of Truss Structures Based on Differential Evolution Strategy. 2009 Int. Conf. Inf. Eng. Comput. Sci., IEEE; 2009, p. 1–5. https://doi.org/10.1109/ICIECS.2009.5365996.

[7] Perez RE, Behdinan K. Particle swarm approach for structural design optimization. Comput Struct 2007;85:1579–88. https://doi.org/10.1016/j.compstruc.2006.10.013.

[8] Kaveh A, Azar BF, Talatahari S. Ant Colony Optimization for Design of Space Trusses. Int J Sp Struct 2008;23:167–81. https://doi.org/10.1260/026635108786260956.

[9] Papadrakakis M, Lagaros ND, Tsompanakis Y. Optimization of Large-Scale 3-D Trusses Using Evolution Strategies and Neural Networks. Int J Sp Struct 1999;14:211–23. https://doi.org/10.1260/0266351991494830.

[10] Salajegheh E, Gholizadeh S. Optimum design of structures by an improved genetic algorithm using neural networks. Adv Eng Softw 2005;36:757–67. https://doi.org/10.1016/j.advengsoft.2005.03.022.

[11] Kaveh A, Gholipour Y, Rahami H. Optimal Design of Transmission Towers Using Genetic Algorithm and Neural Networks. Int J Sp Struct 2008;23:1–19. https://doi.org/10.1260/026635108785342073.

[12] Salajegheh E, Salajegheh J, SEYEDPOUR SM, Khatibinia M. Optimal design of geometrically nonlinear space trusses using an adaptive neuro-fuzzy inference system 2009.

[13] Chen T-Y, Cheng Y-L. Data-mining assisted structural optimization using the evolutionary algorithm and neural network. Eng Optim 2010;42:205–22. https://doi.org/10.1080/03052150903110942.

[14] Nguyen T-H, Vu A-T. Using Neural Networks as Surrogate Models in Differential Evolution Optimization of Truss Structures, 2020, p. 152–63. https://doi.org/10.1007/978-3-030-63007-2_12.

[15] Rosso MM, Cucuzza R, Di Trapani F, Marano GC. Nonpenalty Machine Learning Constraint Handling Using PSO-SVM for Structural Optimization. Adv Civ Eng 2021;2021:1–17. https://doi.org/10.1155/2021/6617750.

[16] Nguyen T-H, Vu A-T. Application of Artificial Intelligence for Structural Optimization, 2022, p. 1052–64. https://doi.org/10.1007/978-981-16-3239-6_82.

[17] Kim S-E, Vu Q-V, Papazafeiropoulos G, Kong Z, Truong V-H. Comparison of machine learning algorithms for regression and classification of ultimate load-carrying capacity of steel frames. Steel Compos Struct 2020;37:193–209.

[18] Nguyen T-H, Vu A-T. A Comparative Study of Machine Learning Algorithms in Predicting the Behavior of Truss Structures, 2021, p. 279–89. https://doi.org/10.1007/978-981-15-7527-3_27.

[19] Nguyen T-H, Vu A-T. Evaluating structural safety of trusses using Machine Learning. Frat Ed Integrità Strutt 2021;15:308–18. https://doi.org/10.3221/IGF-ESIS.58.23.

[20] Ho-Huu V, Nguyen-Thoi T, Vo-Duy T, Nguyen-Trang T. An adaptive elitist differential evolution for optimization of truss structures with discrete design variables. Comput Struct 2016;165:59–75. https://doi.org/10.1016/j.compstruc.2015.11.014.

[21] Freund Y, Schapire RE. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. J Comput Syst Sci 1997;55:119–39. https://doi.org/10.1006/jcss.1997.1504.

[22] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.

[23] Gholizadeh S. Optimal design of double layer grids considering nonlinear behaviour by sequential grey wolf algorithm. Iran Univ Sci Technol 2015;5:511–23.