



ARTICLE

An Efficient Differential Evolution for Truss Sizing Optimization Using AdaBoost Classifier

Tran-Hieu Nguyen* and Anh-Tuan Vu

Hanoi University of Civil Engineering, Hanoi, 100000, Vietnam

*Corresponding Author: Tran-Hieu Nguyen. Email: hieunt2@huce.edu.vn

Received: 14 December 2021 Accepted: 25 February 2022

ABSTRACT

Design constraints verification is the most computationally expensive task in evolutionary structural optimization due to a large number of structural analyses that must be conducted. Building a surrogate model to approximate the behavior of structures instead of the exact structural analyses is a possible solution to tackle this problem. However, most existing surrogate models have been designed based on regression techniques. This paper proposes a novel method, called CaDE, which adopts a machine learning classification technique for enhancing the performance of the Differential Evolution (DE) optimization. The proposed method is separated into two stages. During the first optimization stage, the original DE is implemented as usual, but all individuals produced in this phase are stored as inputs of the training data. Based on design constraints verification, these individuals are labeled as “safe” or “unsafe” and their labels are saved as outputs of the training data. When collecting enough data, an AdaBoost model is trained to evaluate the safety state of structures. This model is then used in the second stage to preliminarily assess new individuals, and unpromising ones are rejected without checking design constraints. This method reduces unnecessary structural analyses, thereby shortens the optimization process. Five benchmark truss sizing optimization problems are solved using the proposed method to demonstrate its effectiveness. The obtained results show that the CaDE finds good optimal designs with less structural analyses in comparison with the original DE and four other DE variants. The reduction rate of five examples ranges from 18 to over 50%. Moreover, the proposed method is applied to a real-size transmission tower design problem to exhibit its applicability in practice.

KEYWORDS

Structural optimization; machine learning; surrogate model; differential evolution; AdaBoost classifier

1 Introduction

Nowadays, global competition has posed a big challenge for industrial companies to offer quality products at lower prices. The situation is similar for the field of construction where the contractors are frequently pressured to reduce costs while maintaining high standards. In this case, optimizing the weight of structures is seen as an effective strategy to gain a competitive advantage. Traditionally, the structure is often designed manually following the “trial-and-error” method. The final structural solution received from this way is deeply influenced by the designers’ experience and intuition.



Therefore, in order to help the designers find the best solution, the optimization algorithms were applied to the design process and have been constantly developed.

Generally, optimization algorithms are categorized into two groups: gradient-based group and metaheuristic group. The gradient-based optimization finds the minimum by going in the direction opposite sign of the derivative of the function. Hence, the speed of convergence is seriously influenced by the starting point, and the gradient-based optimization may fall to the local minimum. Additionally, the function is sometimes difficult or impossible to calculate the derivative. Contrary to gradient-based methods, metaheuristics do not require the calculation of derivative value. Instead, these algorithms use the randomization principle to find the optimum. By searching for a large set of solutions spreading over the design space, metaheuristics can easily escape a local optimum and eventually reach a global optimum. Because of their advantages like easy implementation as well as robustness, metaheuristics are increasingly applied to civil engineering problems. Some outstanding algorithms that are widely used in structural optimization can be listed here such as Genetic Algorithm (GA) [1,2], Evolution Strategies (ES) [3,4], Differential Evolution (DE) [5,6], Particle Swarm Optimization (PSO) [7,8]. Some recently proposed algorithms have shown competitive performance, for example, Artificial Bee Colony (ABC) [9], Harmony Search (HS) [10,11], Teaching-Learning-Based Optimization (TLBO) [12,13], Colliding Bodies Optimization (CBO) [14,15].

Despite many advantages, metaheuristics have one common drawback of requiring a large number of fitness evaluations. It should be highlighted that for structural optimization problems, the objective function can be quickly determined but the computation of constraints is very costly due to involving the structural analysis. Conducting a high number of structural analyses is the most time-consuming task when optimizing structures. Therefore, reducing the number of structural analyses becomes an important goal. To overcome this challenge, many efforts have been made to improve the performance of existing algorithms [16–19]. Besides, using surrogate models instead of computationally expensive structural analyses is a potential solution [20–28]. In recent decades, due to the fast growth of Artificial Intelligence (AI), especially Machine Learning (ML), the topic regarding ML-based surrogate-assisted metaheuristic algorithms has attracted interest from researchers.

Based on the literature, there are several strategies for managing surrogate models. The most straightforward way is to build surrogate models based on the data obtained from the original fitness evaluations and the whole models will be used during the optimization process. Since 1997, Papadrakakis et al. [20] employed feed-forward neural networks (NNs) to predict the response of structures. These NNs were combined with the ES algorithm for optimizing shapes and sections of structures. To cope with the error of NNs, a correction technique was proposed, in which the allowable constraint values were adjusted proportionally to the maximum error for the testing dataset. The research was then extended to large-scale 3D roof trusses [21]. Similarly, structural analyses were replaced by NNs during the GA-based optimization process [22]. In more detail, three kinds of NN including radial basis function (RBF) network, generalized regression (GR) network, and counter-propagation (CP) network were used for optimizing 25-bar truss structure as well as 1300-bar dome structure. The obtained results showed that the RBF networks outperformed the other two networks. In 2019, Taheri et al. [23] combined the RBF neural networks and the ABC algorithm for solving the size, shape, and topology optimization problems of steel lattice towers. Next, the adaptive fuzzy neural inference system (ANFIS) network is used instead of structural analyses carried out by MSTOWER software when optimizing transmission towers [24]. Nguyen et al. [25] utilized two deep NNs separately to estimate the stresses and displacements when optimizing truss structures. Recently, Mai et al. [26] also used deep NNs to reduce the computation cost of the design optimization of geometrically

nonlinear trusses. This strategy has been proven to be time-effective but the designs were not the true optimal designs due to the inaccuracy of surrogate models.

To avoid falling a false optimum, both the exact functions and the approximate functions are used simultaneously. For example, in [27], the combination of NN and GA was adapted to optimize transmission towers based on this strategy. The optimization process employed in this study comprises two stages: in the first stage, the RBF networks were used as structural analyzers until the stopping criterion and in the second stage, the exact structural analyses were conducted. Chen et al. [28] proposed a hybrid approach, in which the near-optimal point was located by the ES algorithm with the help of NN-based surrogate models, and the sequential quadratic programming algorithm was then used to find the exact optimal point.

These existing studies have been used a similar approach, in which the surrogate models have been constructed by regression techniques with the aim of predicting the degree of constraint violation of candidates. While this approach is very intuitive and has been successfully applied in a lot of studies, this is not the only one. An alternative approach is to use the classification technique to build surrogate models. In the literature, the classification-based surrogate models have been successfully applied to solve optimization problems. For instance, Kramer et al. [29] used a classification model for predicting the feasibility of the solutions. However, a feasible solution can be classified as an infeasible solution due to the poor quality model, leading to the rejection of potential solutions. Therefore, the constraint boundary is shifted to the infeasible region by a small distance to limit this error. Another study [30] proposed a probability which is called the influence coefficient to prevent the misclassification of the surrogate model. In 2018, Wang et al. [31] introduced a technique in which the boundary is adjusted to decrease the possibility of classifying feasible solutions to be infeasible.

Although classification techniques have been used previously, the proposed method in the present work is different. Instead of using an additional parameter as in former works, this study employs a classification surrogate model in conjunction with the direct comparison of the objective function values. In detail, a classifier will be built to identify whether the structural solution satisfies the design constraints or not. When optimizing the weight of trusses, if a new solution is predicted to violate the design constraints and its weight is heavier than that of the parent solution, it will be discarded without conducting structural analysis. In this way, some unnecessary structural analyses will be reduced and consequently, the computational time will be shortened.

The classification technique used in this study is Adaptive Boosting (as known as AdaBoost) developed by Freund et al. [32]. The AdaBoost classifier is combined with the DE algorithm to form an efficient optimizer called CaDE. AdaBoost is selected because it has been proven as a powerful algorithm for the task of classifying the structural safety of trusses [33]. In addition, the choice of the DE algorithm is motivated by its simplicity and robustness. The proposed method CaDE is first examined for the weight optimization of truss structures.

The remainder of this paper is arranged as follows. A brief introduction of the DE algorithm and the AdaBoost classifier are given in Section 2. The truss sizing optimization is presented in Section 3.1. Next, the detail of the proposed method CaDE is described in Section 3.2. Computational experiments are conducted in Section 4, in which, six benchmark trusses are optimized according to the proposed method to show its efficiency. In addition, a real-size transmission tower is also designed using the proposed method to exhibit its applicability in practice. Finally, Section 5 concludes this paper.

2 Background Information

2.1 Differential Evolution Algorithm

Differential Evolution is a metaheuristic algorithm that was firstly introduced by Storn et al. [34] in 1995. Thanks to its simplicity, this algorithm is widely applied to many fields such as communication, transportation, aerospace, civil engineering, etc. Since the first introduction, DE has been taken a top rank in many competitions relating to evolutionary computation [35,36]. Particularly, some studies that compare the performance of metaheuristic algorithms in structural optimization have been conducted [37]. The results show that DE outperforms the remaining algorithms.

DE is designed based on the natural evolutionary mechanism. According to the Darwinian theory, the main driving forces of evolution are mutation and natural selection. Similarly, DE consists of four basic steps as shown in Fig. 1, in which the initialization step is conducted one time at the first of the optimization process while three remaining steps are repeated till a stopping criterion is satisfied.

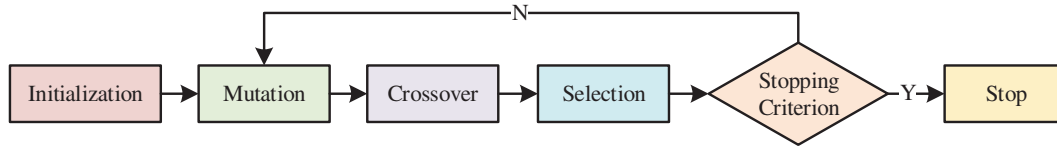


Figure 1: Main steps of differential evolution

The optimization process begins with an initialization of a population \mathbf{pop}_0 containing NP individuals. Each individual representing a candidate of the optimization problem is a D -dimensional vector:

$$\mathbf{x}_i^{(0)} = \{x_{i,1}^{(0)}, x_{i,2}^{(0)}, \dots, x_{i,D}^{(0)}\} \quad (1)$$

In order to find the global optimum, the initial population should be spread over the search space as much as possible. The most widely used method for generating the initial population is to use the uniform distribution. In detail, the j th component of the i th vector can be initialized based on this formula:

$$x_{i,j}^{(0)} = x_j^{\min} + \text{rand}_{i,j}[0, 1] (x_j^{\max} - x_j^{\min}) \quad (2)$$

where: x_j^{\min} and x_j^{\max} are the lower and upper bounds for the j th component of the i th vector; $\text{rand}[0, 1]$ is a uniformly distributed random number in the range $[0, 1]$.

Secondly, DE creates NP mutant vectors $\mathbf{v}_i^{(t)}$ corresponding to NP target vector $\mathbf{x}_i^{(t)}$ via the mutation operator. The classical mutation operator is to add a scaled difference vector between two random vectors to the third vector. It can be written in the mathematical form as follows:

$$\mathbf{v}_i^{(t)} = \mathbf{x}_{r1}^{(t)} + F \times (\mathbf{x}_{r2}^{(t)} - \mathbf{x}_{r3}^{(t)}) \quad (3)$$

where: $r1 \neq r2 \neq r3 \neq i$ are three integers that are randomly chosen from the range $[1, NP]$; t is the current generation number; F is the scaling factor.

The above mutation strategy is called “DE/rand/1” where the term “rand” means random and the number “1” denotes that only one difference vector is added. Besides “DE/rand/1”, four mutation strategies that were proposed by Storn et al. [34] are listed below:

$$\text{"DE/best/1"} \mathbf{v}_i^{(t)} = \mathbf{x}_{best}^{(t)} + F \times (\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}) \quad (4)$$

$$\text{"DE/target - to - best/1"} \mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} + F \times (\mathbf{x}_{best}^{(t)} - \mathbf{x}_i^{(t)}) + F \times (\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}) \quad (5)$$

$$\text{"DE/rand/2"} \mathbf{v}_i^{(t)} = \mathbf{x}_{r1}^{(t)} + F \times (\mathbf{x}_{r2}^{(t)} - \mathbf{x}_{r3}^{(t)}) + F \times (\mathbf{x}_{r4}^{(t)} - \mathbf{x}_{r5}^{(t)}) \quad (6)$$

$$\text{"DE/best/2"} \mathbf{v}_i^{(t)} = \mathbf{x}_{best}^{(t)} + F \times (\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}) + F \times (\mathbf{x}_{r3}^{(t)} - \mathbf{x}_{r4}^{(t)}) \quad (7)$$

in which: $\mathbf{x}_{best}^{(t)}$ is the best individual of the current population.

In the crossover step, a trial vector $\mathbf{u}_i^{(t)}$ is constructed by taking some components from $\mathbf{x}_i^{(t)}$ and the rest from $\mathbf{v}_i^{(t)}$. It can be expressed as:

$$u_{ij}^{(t)} = \begin{cases} v_{ij}^{(t)} & \text{if } j = K \text{ or } rand_{ij} [0, 1] \leq Cr \\ x_{ij}^{(t)} & \text{otherwise} \end{cases} \quad (8)$$

where: $u_{ij}^{(t)}$, $v_{ij}^{(t)}$ and $x_{ij}^{(t)}$ are the j th component of $\mathbf{u}_i^{(t)}$, $\mathbf{v}_i^{(t)}$, and $\mathbf{x}_i^{(t)}$, respectively; K is a random integer in $[1, D]$ to ensure that $\mathbf{u}_i^{(t)}$ differs $\mathbf{x}_i^{(t)}$ by getting at least one component from $\mathbf{v}_i^{(t)}$; and Cr is called the crossover rate.

Based on natural selection, only the fittest survive. For minimization problems, the fitness is measured through the value of the objective function. Only the one having the lower value of the objective function will be kept for the next generation. It can be described as follows:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{u}_i^{(t)} & \text{if } f(\mathbf{u}_i^{(t)}) \leq f(\mathbf{x}_i^{(t)}) \\ \mathbf{x}_i^{(t)} & \text{otherwise} \end{cases} \quad (9)$$

where: $f(\cdot)$ is the objective function that needs to be optimized.

2.2 AdaBoost Classifier

There exist many powerful machine learning classification models such as Support Vector Machine or Neural Networks. The abovementioned algorithms are categorized into the single learning group. However, there exists another group of algorithms, called ensemble methods, in which several classifiers are combined with the aim of improving the accuracy. Ensemble methods have been proven to be very effective for classification tasks. In this study, a boosting method, called AdaBoost, is employed. The idea of this algorithm is to train iteratively several weak learners using the weighted training data, and then incorporate them into the final classifier [32]. The weak learners can be any single learning algorithm, however, the most commonly used one is the Decision Tree. In the following section, the background theory of the AdaBoost classifier is presented.

Generally, AdaBoost produces several classifiers, called weak classifiers, each of them has a proper weight based on its accuracy. The final prediction is the combination of the output of weak classifiers using the weight majority voting technique. The more accurate classifier will contribute more impact on the final prediction by assigning a higher weight. In addition, AdaBoost keeps a set of weights over the training data. Initially, the weights of all samples are set equally. After an iteration, the weights of samples that were misclassified by the previous weak classifier are increased. It ensures that those samples will receive the attention of the weak classifier on the next iteration. The AdaBoost algorithm is illustrated in Fig. 2.

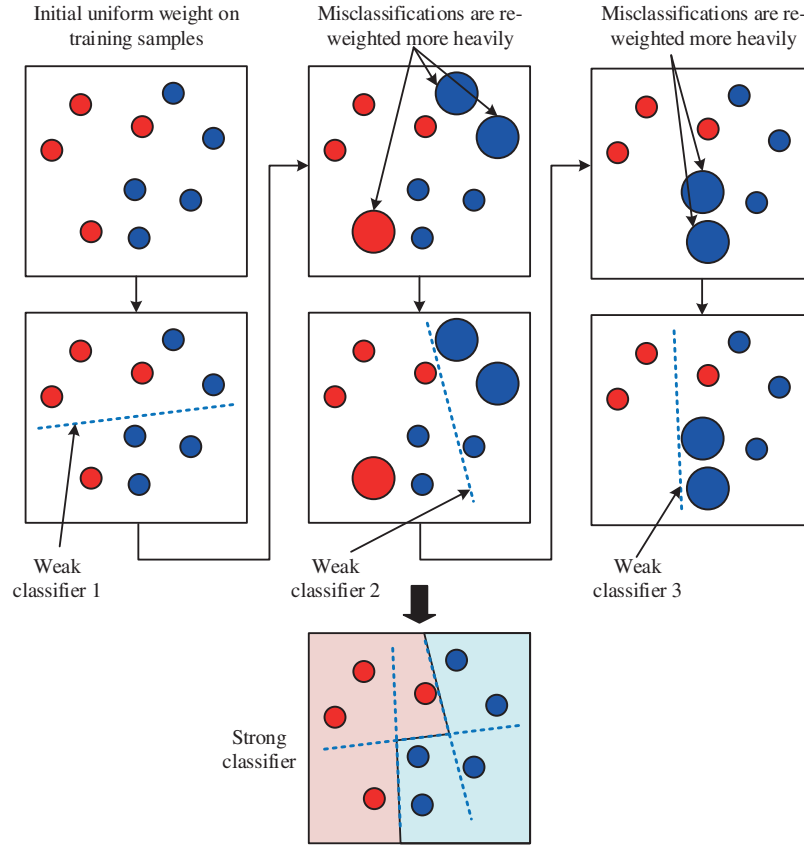


Figure 2: Illustration of the AdaBoost algorithm

Suppose the training dataset is $DB = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ in which $\mathbf{x}_i, i = 1, 2, \dots, n$ is a p -dimensional input vector containing p features; $y_i \in \{+1, -1\}$ is the output; and n is the number of samples in the training dataset.

AdaBoost begins by initializing sample weights uniformly as:

$$\mathbf{w}_1 = \{w_{1,1}, w_{1,2}, \dots, w_{1,n}\}; \quad w_{1,i} = \frac{1}{n}; \quad i = 1, 2, \dots, n \quad (10)$$

where: $w_{1,i}$ is the initial weight of the sample (\mathbf{x}_i, y_i) .

Then a process of four basic steps is iteratively conducted for T times. On iteration t , the weak classifier is h_t , and the weight vector is $\mathbf{w}_t = \{w_{t,1}, w_{t,2}, \dots, w_{t,n}\}$.

Step 1: the classifier h_t is fit to the weighted training data:

$$h_t(\mathbf{x}) = \text{DecisionTree}(DB, \mathbf{w}_t) \quad (11)$$

where: $\text{DecisionTree}(\cdot)$ indicates a single tree classifier.

Step 2: the error of the classifier on this iteration is calculated as follows:

$$e_t = \sum_{i=1}^n w_{t,i} \mathbf{1}(h_t(\mathbf{x}_i) \neq y_i) \quad (12)$$

where: $\mathbf{1}(h_t(\mathbf{x}_i) \neq y_i)$ is the function that returns 1 for correctly classified samples while 0 for misclassified samples.

Step 3: The weight of the weak classifier h_t can be determined based on its accuracy as follows:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - e_t}{e_t} \right) \quad (13)$$

Step 4: the weights of samples are updated according to the following rule:

$$w_{(t+1),i} = \frac{w_{t,i} \exp(-\alpha_t y_i h_t(\mathbf{x}))}{\sum_{i=1}^n w_{(t+1),i}} \quad (14)$$

After T such iterations, T weak classifiers h_t , $t = 1, 2, \dots, T$ are obtained. The final output is the weighted sum of the outputs of these weak classifiers:

$$H(\mathbf{x}) = \text{sign} \left\{ \sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right\} \quad (15)$$

where: $\text{sign}(x)$ is a function that returns -1 if $x < 0$, $+1$ if $x > 0$, 0 if $x = 0$.

3 The Proposed Method CaDE

3.1 Truss Sizing Optimization Using DE

Optimizing a truss structure is a specific constrained optimization problem where the objective function is frequently the weight of the truss structure and the constraints relate to the structural behavior like stress, displacement, or vibration conditions. Three categories of truss optimization problems include size, shape, and topology optimization, in which size optimization is widely used in practical design. For size optimization, the member cross-sectional areas are considered as design variables. The mathematical formulation of this type of optimization problem is expressed as follows:

Find $\mathbf{A} = \{ A_1, A_2, \dots, A_{ng} \}$

$$\text{to minimize } W(\mathbf{A}) = \rho \sum_{i=1}^{ng} A_i \sum_{j=1}^{nm(i)} L_j \quad (16)$$

subject to : $\begin{cases} g_k(\mathbf{A}) \leq 0, k = 1, 2, \dots, nc \\ A_{i,\min} \leq A_i \leq A_{i,\max}, i = 1, 2, \dots, ng \end{cases}$

where: \mathbf{A} is a vector containing ng design variables; $W(\mathbf{A})$ is the objective function which is the weight of the whole truss structure in this case; ρ is the density; A_i denotes a design variable which is the cross-sectional area of members belong to the i th group; L_j is the length of the j th member; ng is the number of groups; $nm(i)$ is the number of members belong to the i th group; $g_k(\mathbf{A})$ is the k th constraint; nc is the number of constraints; $A_{i,\min}$ and $A_{i,\max}$ are the lower and upper bounds of the design variable A_i , respectively.

To apply metaheuristic algorithms which are originally designed for unconstrained optimization, the most widely-used technique is to use the fitness function instead of the objective function:

$$Fit(\mathbf{A}) = \begin{cases} W(\mathbf{A}) + P(\mathbf{A}) & \text{if } cv(\mathbf{A}) > 0 \\ W(\mathbf{A}) & \text{otherwise} \end{cases} \quad (17)$$

in which: $Fit(\mathbf{A})$ is the fitness function; and $P(\mathbf{A})$ is the penalty function which is added whenever any constraint is violated; $cv(\mathbf{A})$ is the degree of constraint violation that is defined as the sum of all constraint violations:

$$cv(\mathbf{A}) = \sum_k \max(0, g_k(\mathbf{A})) \quad (18)$$

3.2 Accelerating DE with AdaBoost Classifier

The objective function of structural optimization problems, for example, the weight of structures, can be easily calculated by multiplying the volume of all members with their material density. However, the determination of the fitness function is very costly due to conducting a computationally expensive structural analysis. As result, the computation time of metaheuristic optimization tasks that require thousands or even millions of fitness function evaluations becomes prohibitively high.

Surrogate models have been frequently constructed to simulate the response of structures, and they have been used to quickly evaluate the fitness of individuals instead of exact structural analyses. Although the accuracy of surrogate modeling techniques has been greatly improved in recent years, there is always an error between the predictions and the ground truth due to the complexity of the structural response as well as the limitation of training data. Therefore, the direct replacement of exact structural analysis with surrogate models may lead to a false optimum [20–25].

The proposed method employs surrogate models in a different manner. The idea behind the proposed method is based on an observation that a trial vector that violates any constraint and has a larger objective function value is largely worse than its target vector and it cannot survive the selection. Detecting and eliminating these worse trial vectors during the optimizing process can save many useless structural analyses. The objective function of structural optimization problems can be quickly determined but the constraint verification is very costly due to performing a structural analysis. It leads to the idea of constructing a surrogate model that can quickly evaluate the safety state of the structure. Using regression models to estimate the degree of constraint violation for this task is not necessary. Instead, a classification model that can distinguish whether or not the individual satisfies constraints is sufficient.

The concept of the proposed method is the application of a machine learning classifier to identify trial vectors that are most likely worse than these target vectors during the optimization process. In this work, the AdaBoost algorithm is employed for that purpose and the proposed method is called CaDE. The pseudo-code of the CaDE is described in Algorithm 1.

Algorithm 1. Pseudo-code of the CaDE

```

1:   Set  $g = 0$ ,  $DB = \text{null}$ ,  $NP$ ,  $F$ ,  $Cr$ ,  $n\_iter1$ ,  $max\_iter$ 
#Phase 1: Building model
2:   Initialize a population  $\mathbf{pop}_0 = \{\mathbf{x}_i^{(0)} \mid i = 1, 2, \dots, NP\}$  using LHS technique
3:   for each  $\mathbf{x}_i^{(0)}$  in  $\mathbf{pop}_0$  do
4:     Evaluate  $\mathbf{x}_i^{(0)}$ 
5:     Assign  $y_i^{(0)}$  according to Eq. (19)
```

(Continued)

Algorithm 1. (Continued)

```

6:   Archive  $(\mathbf{x}_i^{(0)}, y_i^{(0)})$  into  $DB$ 
7:   end for
8:   while  $g < n\_iter1$  do
9:     for each  $\mathbf{x}_i^{(g)}$  in  $\mathbf{pop}_g$  do
10:       $\mathbf{u}_i^{(g)} = \text{Mutation-Crossover}(\mathbf{x}_i^{(g)})$  using two parameters  $F, Cr$ 
11:      Evaluate  $\mathbf{u}_i^{(g)}$ 
12:      Assign  $y_i^{(g)}$  according to Eq. (19)
13:      Archive  $(\mathbf{u}_i^{(g)}, y_i^{(g)})$  into  $DB$ 
14:       $\mathbf{x}_i^{(g+1)} = \text{Selection}(\mathbf{u}_i^{(g)}, \mathbf{x}_i^{(g)})$ 
15:       $\mathbf{pop}_{g+1} = \{\mathbf{x}_i^{(g+1)} \mid i = 1, 2, \dots, NP\}$ 
16:    end for
17:     $g = g + 1$ 
18:  end while
19:  Built an AdaBoost classification model  $C$  based on  $DB$ 
#Phase 2: Employing model
20:  while  $g < max\_iter$  do
21:    for each  $\mathbf{x}_i^{(g)}$  in  $\mathbf{pop}_g$  do
22:       $\mathbf{u}_i^{(g)} = \text{Mutation-Crossover}(\mathbf{x}_i^{(g)})$  using two parameters  $F, Cr$ 
23:       $y_{pred\_i}^{(g)} = C(\mathbf{u}_i^{(g)})$ 
24:      if  $y_{pred\_i}^{(g)} = -1$  then
25:        if  $W(\mathbf{u}_i^{(g)}) > W(\mathbf{x}_i^{(g)})$  then
26:           $\mathbf{x}_i^{(g+1)} = \mathbf{x}_i^{(g)}$  and Eliminate  $\mathbf{u}_i^{(g)}$ 
27:        else
28:          Evaluate  $\mathbf{u}_i^{(g)}$ 
29:           $\mathbf{x}_i^{(g+1)} = \text{Selection}(\mathbf{u}_i^{(g)}, \mathbf{x}_i^{(g)})$ 
30:        end if
31:      else
32:        Evaluate  $\mathbf{u}_i^{(g)}$ 
33:         $\mathbf{x}_i^{(g+1)} = \text{Selection}(\mathbf{u}_i^{(g)}, \mathbf{x}_i^{(g)})$ 
34:      end if
35:       $\mathbf{pop}_{g+1} = \{\mathbf{x}_i^{(g+1)} \mid i = 1, 2, \dots, NP\}$ 
36:    end for
37:     $g = g + 1$ 
38:  end while

```

Note: where: g is the current iteration number; DB is the database for training model; NP is the size of the population; F is the scaling factor; Cr is the crossover rate; n_iter1 is the numbers of iterations for the model building phase; max_iter is the number of iterations of the whole optimization process; *Mutation*, *Crossover*, and *Selection* are three basic operators of the DE as described in Section 2.1; $C(\cdot)$ is the AdaBoost classifier; $W(\cdot)$ is the objective function.

Generally, the CaDE has two phases: the model building phase and the model employing phase. The CaDE algorithm begins by initializing a population of NP vectors $\mathbf{pop}_0 = \{\mathbf{x}_i^{(0)} \mid i = 1, 2, \dots, NP\}$. It is noted that the initial population is generated using the Latin Hypercube Sampling (LHS)

technique because of its good space-filling capability. Each vector $\mathbf{x}_i^{(0)}$ is checked design constraints, and the label $y_i^{(0)}$ is assigned using the following formula:

$$y_i = \begin{cases} -1 & \text{if } cv(\mathbf{x}_i) > 0 \\ +1 & \text{otherwise} \end{cases} \quad (19)$$

As a result, the initial dataset DB consists of NP data samples $\{(\mathbf{x}_i^{(0)}, y_i^{(0)}) \mid i = 1, 2, \dots, NP\}$. Next, three operators (Mutation, Crossover, and Selection) are carried out sequentially as the original DE algorithm. All trial vectors $\mathbf{u}_i^{(g)}$ produced by the Mutation and Crossover operators are checked design constraints and assigned a label $y_i^{(g)}$ according to Eq. (19). All obtained results $(\mathbf{u}_i^{(g)}, y_i^{(g)})$ are also saved into the database DB . The loop is terminated after $(n_iter1 - 1)$ iterations. The final dataset comprises $n_iter1 \times NP$ data samples. At the end of this phase, an AdaBoost classification C model has been trained by the dataset DB as presented in Section 2.2.

In the model employing phase, each trial vector $\mathbf{u}_i^{(g)}$ is preliminarily evaluated using the classification model C . At this time, two conditions are applied in determining which trial vectors to go to the selection step or not. In more detail, a trial vector is rejected without exact evaluation when it has two conditions simultaneously: (i) it is predicted to violate design constraints ($y_{pred,i}^{(g)} = -1$), and (ii) its objective function value is larger than that of the corresponding target vector ($W(\mathbf{u}_i^{(g)}) > W(\mathbf{x}_i^{(g)})$). Otherwise, the constraint violation of the trial vector is evaluated using the structural analysis and the Selection operator is then conducted as usual. The model employing phase lasts until the pre-defined number of iteration max_iter reaches.

4 Numerical Experiments

Numerical experiments are performed on a personal computer with CPU Intel Core i5 2.5 GHz, 8 Gb RAM, and Windows 10 64 bit OS. The DE algorithm and the CaDE algorithm are developed by the authors using the Python programming language, meanwhile, the AdaBoost classifier is built using the open-source library scikit-learn [38]. The Python in-house codes for finite element analysis of truss structures are developed based on the direct stiffness method described in [39].

4.1 Illustrative Example

A three-bar truss example is carried out to illustrate the application of the CaDE algorithm to structural optimization. Although this structure is simple and the constraints can be directly evaluated, the stresses of the members are still determined through the finite element analyses in this experiment. This problem is used as an illustrative example because it has only two variables and it can be easily visualized.

The geometry, loading, and boundary conditions of the structure are presented in Fig. 3. The objective of this example is to find the cross-sectional areas A_1 and A_2 to minimize the weight of the truss while satisfying the condition that the stresses in all bars do not exceed the allowable stress. This problem is stated as follows:

$$\begin{aligned} &\text{Find} && \mathbf{x} = \mathbf{A} = \{0.1 \leq A_1, A_2 \leq 1\} \\ &\text{to minimize} && W(\mathbf{A}) = (2\sqrt{2}A_1 + A_2)L \\ &\text{subject to} && g_k(\mathbf{A}) = \sigma_k - [\sigma] \leq 0, k = 1, 2, 3 \end{aligned} \quad (20)$$

in which: the applied force $P = 2$ kN; $L = 100$ cm; σ_k is the stress in the k th member; and the allowable stress $[\sigma] = 2$ kN/cm².

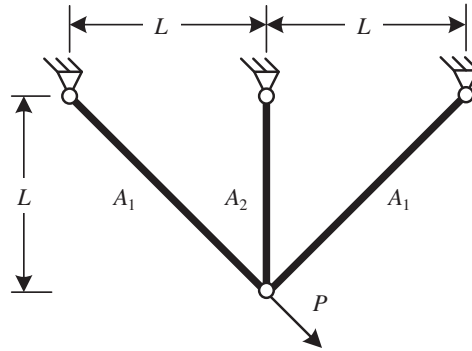


Figure 3: Three-bar truss structure

As mentioned above, the objective function $W(\mathbf{A})$ should be transformed into the fitness function. In this example, the fitness function is defined as $Fit(\mathbf{A}) = W(\mathbf{A})(1 + cv(\mathbf{A}))$ and it is plotted in Fig. 4.

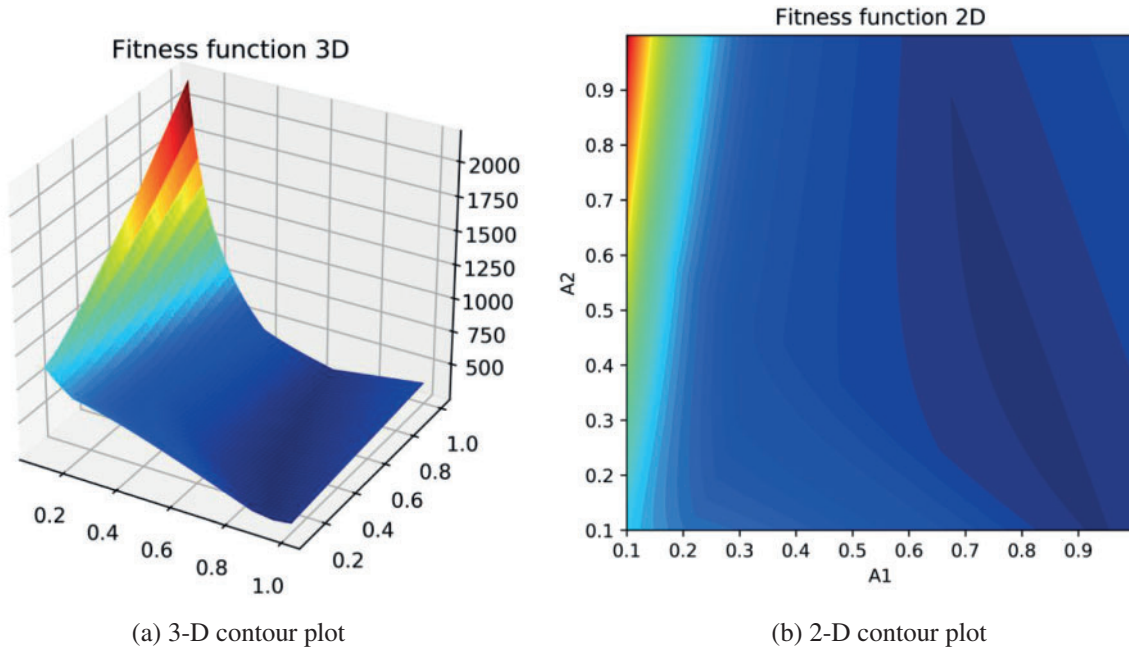


Figure 4: Visualization of the fitness function of the three-bar truss structure

The DE algorithm starts by initializing an initial population \mathbf{pop}_0 of NP individuals. In this example, the number of individuals of a population is set to $NP=20$. The initial individuals are represented together with the 2-D contour plot of the fitness function as shown in Fig. 5a. Then, three operators including mutation, crossover, and selection are sequentially performed to produce a population \mathbf{pop}_1 for the next iteration with two optimization parameters as follows: $F = 0.8$, $Cr = 0.9$. The manner in which the mutant vector $\mathbf{v}_i^{(0)}$ and the trial vector $\mathbf{u}_i^{(0)}$ are produced from the target vector $\mathbf{x}_i^{(0)}$ is shown in Fig. 5b. If the fitness function value of the trial vector $\mathbf{u}_i^{(0)}$ is smaller than that of the target vector $\mathbf{x}_i^{(0)}$, the trial vector $\mathbf{u}_i^{(0)}$ is selected to be the target vector $\mathbf{x}_i^{(1)}$ of the next population. Otherwise, the trial vector $\mathbf{u}_i^{(0)}$ is discarded and the target vector $\mathbf{x}_i^{(0)}$ is kept for the next iteration.

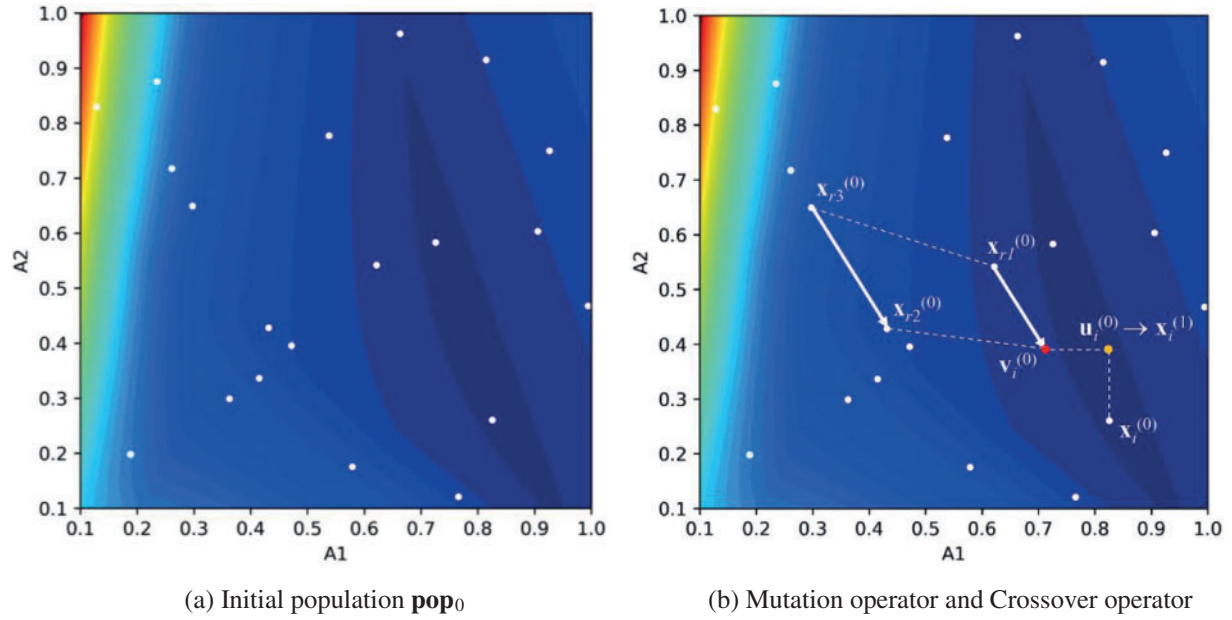


Figure 5: Visualization of three first operators of the DE algorithm

The above process is repeated until the stopping condition is reached. In this example, the stopping condition is set to be after 100 iterations. The populations at several iterations are shown in Fig. 6. It can be seen that all individuals of the population tend to move closer to the global optimal position during the optimization process. At the final iteration, almost all individuals are concentrated at the global optimal position. The optimal design found by the DE algorithm is as follows: $A_1 = 0.789 \text{ cm}^2$ and $A_2 = 0.408 \text{ cm}^2$. The number of structural analyses that must be conducted is $nSA = 2000$ times.

The optimization parameters F , Cr , max_iter of the CaDE are kept the same as in the DE. The number of iterations for the first phase is set to $n_iter1 = 5$, meaning the number of samples in the dataset DB is equal to $n_iter1 \times NP = 5 \times 20 = 100$. Fig. 7 presents all samples collected in the first phase, where vectors that are labeled $+1$ are represented as white dots while vectors that are labeled -1 are represented as red dots. Based on the obtained dataset, an AdaBoost classifier is built as the green dashed curve shown in Fig. 7.

In the second phase, the decision of performing exact structural analyses depends on the prediction of the AdaBoost classifier. Fig. 8 illustrates the optimization process of the proposed method at the 5th iteration, in which all white dots represent the target vectors $\mathbf{x}_i^{(5)}$ of the current population. After conducting two operators including mutation and crossover, the machine learning classifier is employed to predict the label of the trial vector $\mathbf{u}_i^{(5)}$. At this time, one of three possible cases can occur as follows:

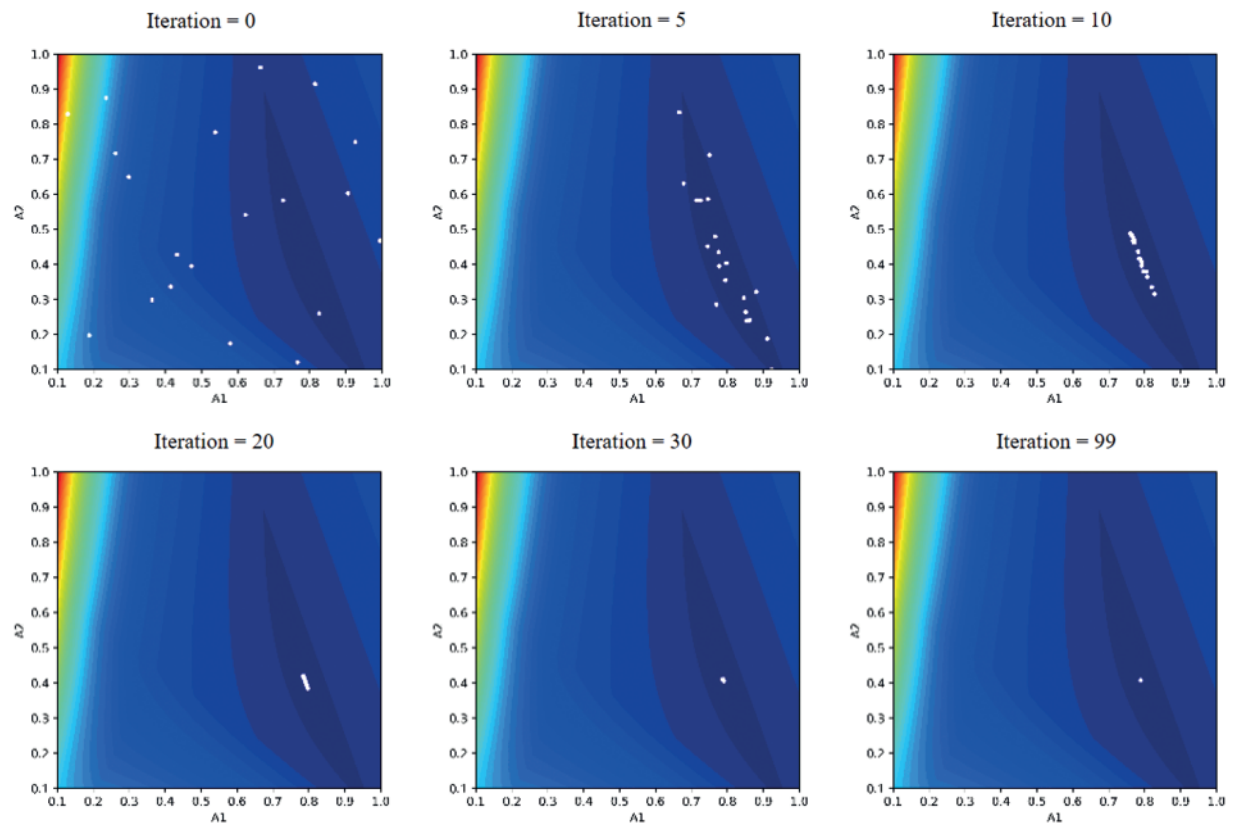


Figure 6: Improvement of the population during the DE optimization process

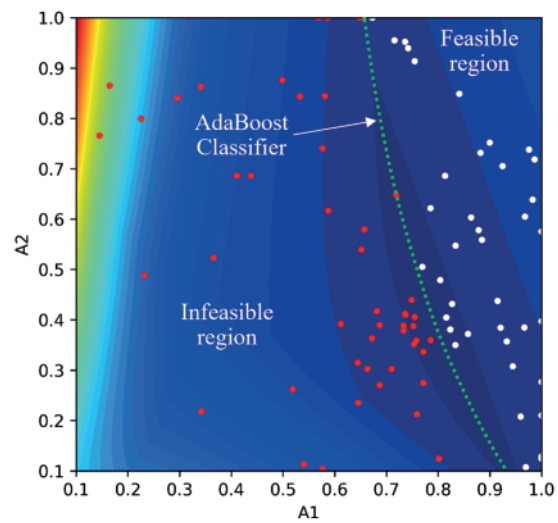


Figure 7: Building an AdaBoost classifier based on historical constraint evaluations

- Case 1:** If the predicted label $y_{pred,i}^{(5)} = +1$, the trial individual $\mathbf{u}_i^{(5)}$ is then checked design constraints using exact structural analysis. These individuals are represented as blue dots in Fig. 8;
- Case 2:** If the predicted label $y_{pred,i}^{(5)} = -1$ and the objective function of the trial vector $W(\mathbf{u}_i^{(5)})$ is smaller than or equal to that of the target vector $W(\mathbf{x}_i^{(5)})$, it is also checked design constraints using exact structural analysis. These individuals are represented as yellow dots in Fig. 8;
- Case 3:** If the predicted label $y_{pred,i}^{(5)} = -1$ and $W(\mathbf{u}_i^{(5)}) > W(\mathbf{x}_i^{(5)})$, it is eliminated without performing the exact structural analysis. These individuals are marked as magenta dots in Fig. 8.

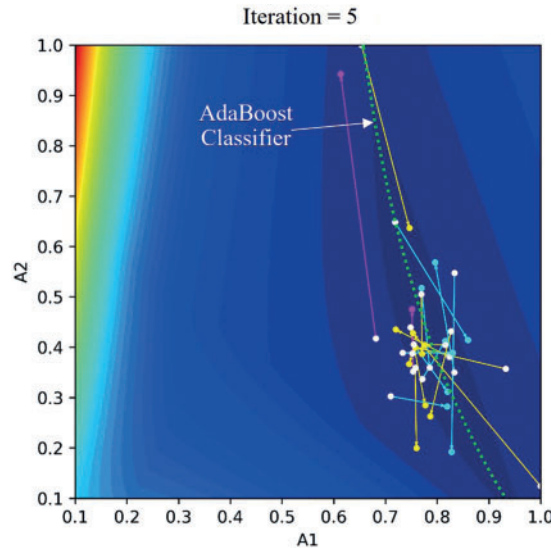


Figure 8: Illustration of the implementation of the proposed method CaDE

After reaching the stopping condition, the optimal cross-sectional areas of truss members found by the CaDE are $A_1 = 0.789 \text{ cm}^2$ and $A_2 = 0.408 \text{ cm}^2$ which are similar to the results found by the original DE. However, by applying the AdaBoost classifier, the number of exact structural analyses is greatly reduced. The optimization is independently performed 30 times and the statistical results show that the proposed method CaDE requires an average of 1770 structural analyses while the original DE algorithm needs 2000 structural analyses.

4.2 Five Benchmark Truss Problems

4.2.1 Description of the Problems

Five well-known truss sizing optimization problems collected from the literature including the 10-bar planar truss, the 17-bar planar truss, the 25-bar spatial truss, the 72-bar spatial truss, and the 200-bar planar truss are carried out in this section.

The first example considers a simple truss containing 10 members as illustrated in Fig. 9. This structure is made of the material having the modulus of elasticity $E = 10,000 \text{ ksi}$ and the density $\rho = 0.1 \text{ lb/in}^3$. This problem has 10 design variables $\mathbf{A} = \{A_i, i = 1, 2, \dots, 10\}$ where $0.1 \text{ in}^2 \leq A_i \leq 33.5 \text{ in}^2$ is the cross-sectional areas of i th member. Two vertical loads acting at node (2) and node (4) have the same magnitude $P = 100 \text{ kips}$. Stresses inside members must be lower than 25 ksi for both compression and tension. Displacements of nodes (1), (2), (3), (4) are limited within $\pm 2 \text{ in}$.

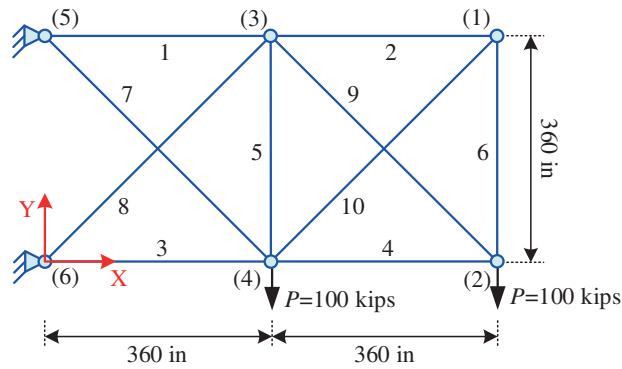


Figure 9: Layout of the 10-bar truss

The second example considers a 17-bar planar truss as presented in Fig. 10. This structure is fabricated from a material that has the modulus of elasticity $E = 30,000$ ksi and the material density $\rho = 0.268$ lb/in³. Each member in this structure has its own cross-sectional areas and there are 17 design variables in total $\mathbf{A} = \{0.1 \text{ in}^2 \leq A_i \leq 33.5 \text{ in}^2, i = 1, 2, \dots, 17\}$. A vertical force of 100 kips acts at node (9) of the truss. The allowable stresses in both tension and compression are $\sigma_{\text{allow}} = 50$ ksi and the allowable displacements along the X and Y directions are $\delta_{\text{allow}} = 2$ in.

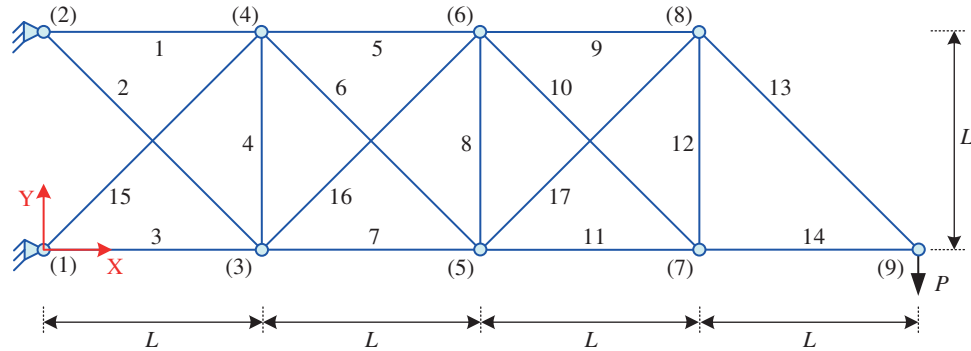


Figure 10: Layout of the 17-bar truss

In the third example, a spatial truss containing 25 members as presented in Fig. 11 is considered. This structure is made of the same material as the 10-bar truss ($E = 10,000$ ksi and $\rho = 0.1$ lb/in³). Members of this structure are classified into 8 groups where members belonging to the same group are assigned the same cross-sectional area that varies between 0.01 in² to 3.5 in². This structure is optimized according to both stress constraint and displacement constraint. The allowable stresses are presented in Table 1 and the permissible displacement of all nodes is ± 0.35 in. The loading data are shown in Table 2.

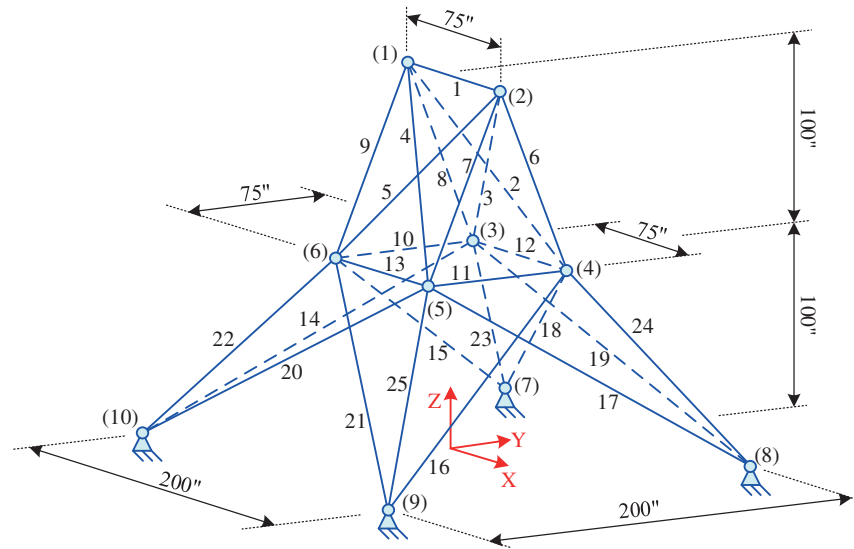


Figure 11: Layout of the 25-bar truss

Table 1: Member groups and the corresponding allowable stress values for the 25-bar truss

Group	Member	Allowable compressive stress (ksi)	Allowable tensile stress (ksi)
1	1	35.092	40.0
2	2, 3, 4, 5	11.590	40.0
3	6, 7, 8, 9	17.305	40.0
4	10, 11	35.092	40.0
5	12, 13	35.092	40.0
6	14, 15, 16, 17	6.759	40.0
7	18, 19, 20, 21	6.959	40.0
8	22, 23, 24, 25	11.082	40.0

Table 2: Loading data for the 25-bar truss (kips)

Node	Load Case 1			Load Case 2		
	F_X	F_Y	F_Z	F_X	F_Y	F_Z
1	—	20.0	−5.0	1.0	10.0	−5.0
2	—	−20.0	−5.0	—	10.0	−5.0
3	—	—	—	0.5	—	—
6	—	—	—	0.5	—	—

The layout of the 72-bar spatial truss investigated in the fourth example is displayed in [Fig. 12](#). Members are classified into 16 groups as described in [Table 3](#). The minimum and maximum areas of

all members are 0.01 in^2 and 4.0 in^2 . Two load cases acting on this structure are as follows: (LC1) two horizontal forces of 5 kips along the X-axis and Y-axis and a vertical force of 5 kips along the opposite direction of the Z-axis act at node (17); (LC2) four vertical forces of 5 kips along the opposite direction of the Z-axis act at four nodes (17), (18), (19), (20), respectively. The design constraints of this problem include stress constraint and displacement constraint. The allowable stresses of all members are $\pm 25 \text{ ksi}$ while the allowable displacements along the X and Y directions are $\pm 0.25 \text{ in}$.

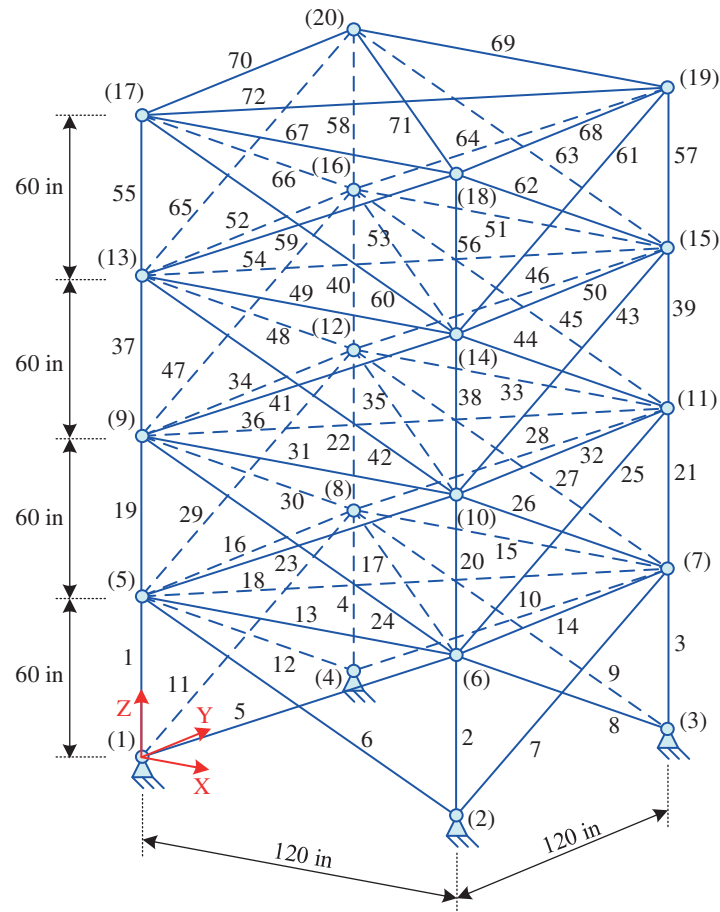


Figure 12: Layout of the 72-bar truss

The last problem is a 200-bar planar truss which is schematized in [Fig. 13](#). The modulus of elasticity E of the material used in this structure equals $30,000 \text{ ksi}$ while the material density ρ equals 0.283 lb/in^3 . There are three independent load cases as follows:

Table 3: Member groups for the 72-bar truss

Group	Member	Group	Member	Group	Member	Group	Member
1	1–4	5	19–22	9	37–40	13	55–58
2	5–12	6	23–30	10	41–48	14	59–66
3	13–16	7	31–34	11	49–52	15	67–70
4	17, 18	8	35, 36	12	53, 54	16	71, 72

(LC1): 1.0 kips acting along the positive x-direction at nodes (1), (6), (15), (20), (29), (34), (43), (48), (57), (62), and (71) (green arrows in Fig. 13);

(LC2): 10 kips acting along the negative y-direction at nodes (1)–(6), (8), (10), (12), (14), (16)–(20), (22), (24), (26), (28)–(34), (36), (38), (40), (42)–(48), (50), (52), (54), (56)–(62), (64), (66), (68), and (70)–(75) (orange arrows in Fig. 13);

(LC3): both (LC1) and (LC2) acting together.

There is no limitation of displacement in this problem. Stresses inside all members must be lower than ± 10 ksi. This structure has 29 groups of members as indicated in Table 4. It means this problem has 29 design variables $\mathbf{A} = \{A_i, i = 1, 2, \dots, 29\}$ in which A_i represents the cross-sectional area of members belonging the i th group. A_i is limited in the range of $[0.1, 15]$ in².

4.2.2 Parameter Studies

Obviously, the training dataset size and the number of weak classifiers strongly influence the accuracy of the AdaBoost model, thereby affecting the performance of the proposed method CaDE. In this section, a parametric study is carried out to investigate the influences of these parameters. The 25-bar truss structure described in Section 4.2.1 is taken as the optimization problem for the parametric study.

Firstly, five cases are carried out in which the amount of training samples is set to 100, 500, 1000, 1500, and 2000, respectively. In all cases, the number of weak classifiers used in the AdaBoost model is fixed as $T = 100$. In the second, four more cases are conducted in which the number of weak classifiers is set to 10, 25, 50, and 200, respectively while the number of training samples is fixed as 1000 data points. There are a total of 9 cases. Other parameters are the same for all cases as follows: the mutation strategy “DE/target-to-best/1”; $F = 0.8$; $Cr = 0.9$; $NP = 50$; and $max_iter = 500$. Each case is executed 30 times and the results are reported in Table 5.

It can be found that all cases achieve high convergence when the optimal weights found in 30 independent runs are almost the same. However, there is an influence of the parameters on the number of structural analyses that must be conducted during the optimization process. When the training dataset size is fixed to 1000, the case of 10 weak classifiers requires the largest number of structural analyses. When the number of weak classifiers varies from 10 to 100, the number of structural analyses decreases linearly but from 100 to 200, the value of nSA does not change much. When the number of weak classifiers is fixed to 100, the case of 1000 data samples has the fastest convergence speed.

Overall, it is recognized that the pair of parameters: the number of weak classifier $T = 100$ and the number of training samples = 1000 is a proper selection. This set of parameters is then applied to all further experiments.

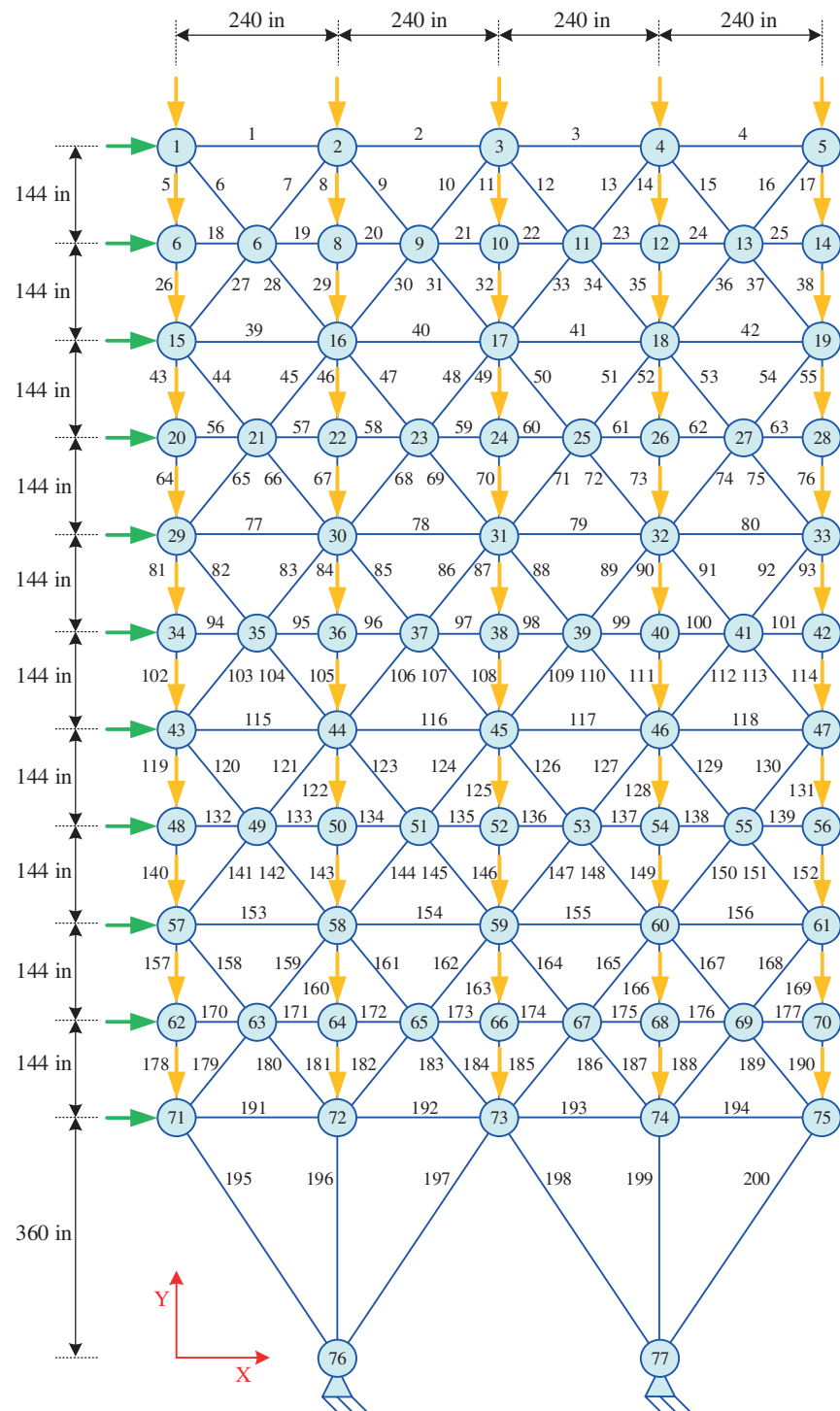


Figure 13: Layout of the 200-bar truss

Table 4: Member groups for the 200-bar truss

Group	Member	Group	Member
1	1–4	16	82, 83, 85, 86, 88, 89, 91, 92, 103, 104, 106, 107, 109, 110, 112, 113
2	5, 8, 11, 14, 17	17	115–118
3	19–24	18	119, 122, 125, 128, 131
4	18, 25, 56, 63, 94, 101, 132, 139, 170, 177	19	133–138
5	26, 29, 32, 35, 38	20	140, 143, 146, 149, 152
6	6, 7, 9, 10, 12, 13, 15, 16, 27, 28, 30, 31, 33, 34, 36, 37	21	120, 121, 123, 124, 126, 127, 129, 130, 141, 142, 144, 145, 147, 148, 150, 151
7	39–42	22	153–156
8	43, 46, 49, 52, 55	23	157, 160, 163, 166, 169
9	57–62	24	171–176
10	64, 67, 70, 73, 76	25	178, 181, 184, 187, 190
11	44, 45, 47, 48, 50, 51, 53, 54, 65, 66, 68, 69, 71, 72, 74, 75	26	158, 159, 161, 162, 164, 165, 167, 168, 179, 180, 182, 183, 185, 186, 188, 189
12	77–80	27	191–194
13	81, 84, 87, 90, 93	28	195, 197, 198, 200
14	95–100	29	196, 199
15	102, 105, 108, 111, 114		

Table 5: Influence of the parameters of the AdaBoost model on the performance of the CaDE

No. of training samples		100	500	1000	1500	2000	1000	1000	1000	1000
No. of weak classifiers		100	100	100	100	100	10	25	50	200
Weight (lb)	Mean	545.16	545.16	545.16	545.16	545.16	545.16	545.16	545.16	545.16
	SD	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
<i>nSA</i> (times)		19,033	18,994	17,599	17,683	18,436	21,888	20,481	18,291	17,474

4.2.3 Comparison with Regression-Based Surrogate Model

As described in the ‘Introduction’ section, many previous studies have used regression techniques for surrogate modeling. In this section, a comparison is conducted to exhibit the advantage of the classification-based surrogate models. The performances of three optimization methods are evaluated including the original DE method (called the DE), the AdaBoost classifier-assisted DE method (called the CaDE), and the AdaBoost regression-assisted DE method (called the RaDE).

The details of the DE and the CaDE are already introduced in [Section 2.1](#) and [Section 3.2](#). The RaDE has also two phases: the model building phase and the model employing phase. In the model

building phase, the original DE algorithm is employed as usual and all solutions are exactly evaluated through structural analyses and the degree of constraint violation of each solution is saved in the database. After n_iter1 iterations, an AdaBoost regression model is trained with the obtained dataset with the aim of predicting the constraint violation degree of a new solution. Then, the trained model is employed instead of the constraint verification in the second stage.

Five optimization problems mentioned in [Section 4.2.1](#) are solved using three methods DE, CaDE, RaDE, respectively. The setting parameters for these methods are summarized in [Table 6](#). Each problem is solved 30 independent times, and the statistical results including best, mean, worst, and standard deviation (SD) are reported in [Table 7](#). The average number of structural analyses conducted during the optimization process (nSA) as well as the degree of constraint violation (cv) are also presented.

Table 6: Parameters of the DE, CaDE and RaDE

		10-bar truss	17-bar truss	25-bar truss	72-bar truss	200-bar truss
DE	F	0.8	0.8	0.8	0.8	0.8
	Cr	0.9	0.9	0.9	0.9	0.9
	NP	50	50	50	50	50
	max_iter	500	500	500	500	1000
CaDE	F	0.8	0.8	0.8	0.8	0.8
	Cr	0.9	0.9	0.9	0.9	0.9
	NP	50	50	50	50	50
	max_iter	500	500	500	500	1000
	n_iter1	20	20	20	20	20
RaDE	F	0.8	0.8	0.8	0.8	0.8
	Cr	0.9	0.9	0.9	0.9	0.9
	NP	50	50	50	50	50
	max_iter	500	500	500	500	1000
	n_iter1	20	20	20	20	20

Based on the results in [Table 7](#), some observations can be pointed out as follows. First of all, it can be seen that the optimal designs found by the CaDE are very similar to the DE but the CaDE needs fewer structural analyses than the DE. For the four first problems, the DE requires 25000 structural analyses and for the last problem, the DE requires 50000 structural analyses. However, the required number of structural analyses of the CaDE for five problems are 17311 times, 16299 times, 17599 times, 18144 times, and 40683 times, respectively.

Secondly, the required numbers of structural analyses of the RaDE for all problems are very small compared to the DE and the CaDE. This method conducts only 1000 times of exact structural analyses to generate the training data. However, due to the error between the structural analyses and the surrogate modeling, the optimal designs found by the RaDE are not good. Although these designs do not violate design constraints, the weights of these designs are much greater than the optimal designs found by the DE and the CaDE.

Table 7: Comparison of optimal results of the DE, CaDE, and RaDE

Problem		DE	CaDE	RaDE
10-bar truss	best (lb)	5060.854	5060.854	5202.081
	mean (lb)	5061.908	5061.910	5468.830
	worst (lb)	5076.669	5076.669	6637.640
	SD (lb)	3.945	3.945	310.515
	<i>cv</i>	0.000	0.000	0.000
	<i>nSA</i>	25,000	17,311	1,000
17-bar truss	best (lb)	2581.911	2581.901	4166.500
	mean (lb)	2581.979	2582.015	5205.044
	worst (lb)	2582.115	2582.217	6304.125
	SD (lb)	0.051	0.087	529.577
	<i>cv</i>	0.000	0.000	0.000
	<i>nSA</i>	25,000	16,299	1,000
25-bar truss	best (lb)	545.163	545.163	551.821
	mean (lb)	545.163	545.163	568.117
	worst (lb)	545.163	545.163	595.473
	SD (lb)	0.000	0.000	10.091
	<i>cv</i>	0.000	0.000	0.000
	<i>nSA</i>	25,000	17,599	1,000
72-bar truss	best (lb)	363.826	363.826	543.706
	mean (lb)	363.838	363.844	694.867
	worst (lb)	363.871	363.920	865.005
	SD (lb)	0.012	0.020	75.948
	<i>cv</i>	0.000	0.000	0.000
	<i>nSA</i>	25,000	18,144	1,000
200-bar truss	best (lb)	25507.236	25490.653	58318.473
	mean (lb)	25671.221	25626.837	67122.934
	worst (lb)	26210.114	26018.725	79739.574
	SD (lb)	161.133	111.277	5780.261
	<i>cv</i>	0.000	0.000	0.000
	<i>nSA</i>	50,000	40,683	1,000

In general, using the regression-based surrogate modeling significantly reduces the number of structural analyses during the optimization process but the obtained results are not the global optimal. Otherwise, the classification-based surrogate modeling only reduces the number of structural analyses by about 25%, but the advantage of this method is that it retains the search performance of the original DE method.

4.2.4 Comparison with Other DE Variants

Furthermore, the results of these problems found by other DE variants are introduced for comparison. Four DE variants collected in the literature [40] include the composite differential evolution (CoDE), the self-adaptive control parameters differential evolution (JDE), the adaptive differential evolution with optimal external archive (JADE), and the self-adaptive differential evolution (SADE). The statistical results are introduced in Table 8.

Table 8: Comparison of optimal results of the CaDE and other DE variants [40]

Problem		CaDE	CoDE	JDE	JADE	SADE
10-bar truss	best (lb)	5060.854	5060.854	5060.858	5060.854	5060.887
	mean (lb)	5061.910	5060.854	5061.444	5060.886	5064.092
	worst (lb)	5076.669	5060.854	5076.674	5061.372	5079.279
	SD (lb)	3.945	0.000	2.877	0.102	6.151
	<i>nSA</i>	17,311	100,000	100,000	100,000	100,000
17-bar truss	best (lb)	2581.901	2581.890	2581.895	2581.890	2582.578
	mean (lb)	2582.015	2581.893	2581.925	2581.909	2605.089
	worst (lb)	2582.217	2581.898	2582.033	2582.048	2670.458
	SD (lb)	0.087	0.002	0.030	0.030	21.226
	<i>nSA</i>	16,299	100,000	100,000	100,000	100,000
25-bar truss	best (lb)	545.163	545.555	545.555	545.555	545.555
	mean (lb)	545.163	545.555	545.559	545.555	545.613
	worst (lb)	545.163	545.555	545.606	546.555	546.143
	SD (lb)	0.000	0.000	0.010	0.000	0.122
	<i>nSA</i>	17,599	100,000	100,000	100,000	100,000
72-bar truss	best (lb)	363.826	363.824	363.826	363.826	363.862
	mean (lb)	363.838	363.826	363.845	363.859	364.145
	worst (lb)	363.871	363.835	363.908	364.026	365.665
	SD (lb)	0.012	0.003	0.019	0.051	0.378
	<i>nSA</i>	25,000	100,000	100,000	100,000	100,000
200-bar truss	best (lb)	25490.653	25523.011	25579.000	25659.736	25734.695
	mean (lb)	25626.837	25716.826	25867.595	26175.928	26556.821
	worst (lb)	26018.725	25919.182	26541.229	26813.984	27636.608
	SD (lb)	111.277	120.334	191.068	301.066	518.116
	<i>nSA</i>	40,683	100,000	100,000	100,000	100,000

For the 10-bar truss, the optimal design of the CaDE is similar to the CoDE and JADE and better than the JDE and the SADE. For the 17-bar truss, the optimal weight found by the CaDE is very similar to those of the DE variants. For the third problem of the 25-bar truss, the minimum weight of the CaDE (545.163 lb) is lower than those of other DE variants (545.555 lb). For the fourth problem of

the 72-bar truss, the optimal weights found by the CaDE, the JDE, the JADE are the same (363.826 lb) and this value is slightly larger than the optimal weight obtained by the CoDE (363.824 lb). For the last problem, the weight of the 200-bar truss received from the CaDE is the smallest value (25490.653 lb), followed by the CoDE (25523.011 lb), the JDE (25579.000 lb), the JADE (25659.736 lb), and the SADE (25734.695 lb). Moreover, the smaller value of the standard deviation of the CaDE (111.277 lb) in comparison with the remaining algorithms (120.334 lb for the CoDE, 191.068 lb for the JDE, 301.116 lb for the JADE, and 518.116 lb for the SADE) indicates the reliability of the proposed method.

In terms of the convergence speed, the CaDE exhibits its superior. The CaDE requires from 16299 to 40683 structural analyses while the remaining methods conduct 100000 structural analyses for finding the optimal results. It is apparent that the CaDE is more speedy than the CoDE, the JDE, the JADE, and the SADE.

4.3 A Real-Size Transmission Tower

In this section, the CaDE is employed to optimize the weight of a real-size transmission tower for showing its applicability in the practical design. This tower consists of 244 members which are classified into 32 groups as presented in Fig. 14. Members belonging to the same group have the same cross-section which is chosen from 64 equal angle profiles from L35 × 3 to L200 × 25. It means this problem is a discrete optimization. The steel material used in this structure has mechanical properties as follows: the weight density $\rho = 7850 \text{ kg/m}^3$; the modulus of elasticity $E = 210,000 \text{ N/mm}^2$; the yield strength $F_y = 233.3 \text{ N/mm}^2$. This structure is designed according to the AISC-ASD specification in which the tensile stress constraint is as follows:

$$\sigma_t \leq [\sigma]_t = 0.6F_y \quad (21)$$

where: σ_t is the tensile stress of the member; $[\sigma]_t$ is the allowable tensile stress; F_y is the yield strength.

The buckling stress constraint is as follows:

$$\sigma_c \leq [\sigma]_b = \begin{cases} \left(F_y \left(1 - \frac{\lambda}{2C_c} \right) \right) / \left(\frac{5}{3} + \frac{3\lambda}{8C_c} - \frac{\lambda^3}{8C_c^3} \right) & \text{if } \lambda < C_c \\ \frac{12\pi^2 E}{23\lambda^2} & \text{if } \lambda \geq C_c \end{cases} \quad (22)$$

where: σ_c is the compressive stress of the member; $[\sigma]_b$ is the allowable buckling stress; $\lambda = KL/r$ is the maximum slenderness ratio of the member; K is the effective length factor which equals 1 in this case; L is the length of the member; and r is the radius of gyration of the member's cross-section; C_c is the slenderness ratio dividing the elastic and inelastic buckling regions which is determined using the following formula:

$$C_c = \sqrt{\frac{2\pi^2 E}{F_y}} \quad (23)$$

Additionally, the maximum slenderness ratio λ is limited to 200 for compression members while 300 for tension members. The tower is subjected to two independent load cases. Table 9 presents the force magnitudes of two applied load cases and the corresponding displacement limitation.

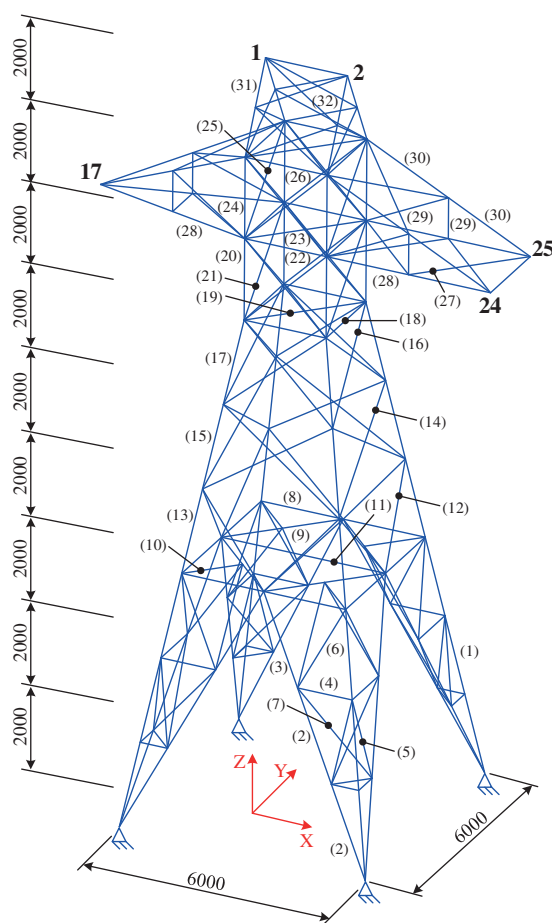


Figure 14: Layout of the transmission tower

Load condition	Joint number	Load (kN)		Displacement limitations (mm)	
		X	Z	X	Z
(LC1)	1	10	−30	45	15
	2	10	−30	45	15
	17	35	−90	30	15
	24	175	−45	30	15
	25	175	−45	30	15
(LC2)	1	—	−360	45	15
	2	—	−360	45	15
	17	—	−180	30	15
	24	—	−90	30	15
	25	—	−90	30	15

Load condition	Joint number	Load (kN)		Displacement limitations (mm)	
		X	Z	X	Z
(LC1)	1	10	−30	45	15
	2	10	−30	45	15
	17	35	−90	30	15
	24	175	−45	30	15
	25	175	−45	30	15
(LC2)	1	—	−360	45	15
	2	—	−360	45	15
	17	—	−180	30	15
	24	—	−90	30	15
	25	—	−90	30	15

This structure is optimized according to both the DE and the CaDE with the same parameters as follows: $NP = 50$; $F = 0.8$; $Cr = 0.9$; $max_iter = 1000$. The number of iterations for the first phase is set to $n_iter1 = 300$ which means 15000 samples have been collected for the training dataset. The statistical results obtained by the DE and the CaDE are presented in Table 10. The minimum weight of the tower found by the DE and the CaDE are the same (6420.652 kg) but other statistical results of the CaDE including mean, worst, and SD are smaller than those of the DE, indicating the stability of the proposed method. The number of structural analyses of the CaDE is only 24865 times, which is about 50% of the number of structural analyses performed by the DE. In other words, the CaDE omits about half of useless structural analyses. This is clearly shown in Fig. 15.

Table 10: Statistical results of the optimal designs found by the DE and the CaDE

		DE	CaDE
Weight (kg)	best	6420.652	6420.652
	mean	7558.171	7453.521
	worst	9804.104	8524.672
	SD	858.690	610.226
cv		0.000	0.000
nSA		50,000	24,865

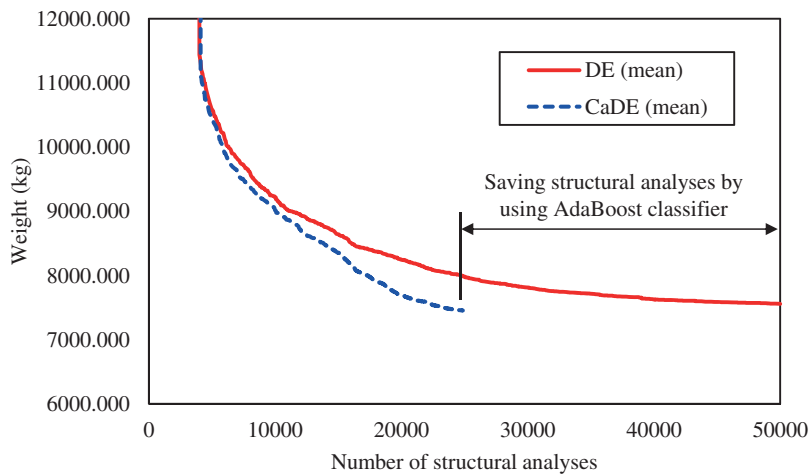


Figure 15: Comparison of the DE and the CaDE for the transmission tower problem

4.4 Evaluation of the Effectiveness of the Proposed Method CaDE

The average numbers of structural analyses (nSA) conducted in the six investigated examples as well as the reduction rates when using the AdaBoost classifier are listed in Table 11. Furthermore, the computing times of the DE and the CaDE are also reported.

It is clearly seen that the application of the AdaBoost classifier into the optimization process helps to reduce the number of structural analyses. Specifically, the CaDE has reduced the number of structural analyses by up to 50.3% for the transmission tower problem. The minimum reduction rate is 18.6% for the 200-bar truss problem.

Table 11: Average number of reduced structural analyses by using Adaboost classifier

Problems	Number of variables	<i>nSA</i> (times)		Reduction rate	Computing time (s)	
		DE	CaDE		DE	CaDE
10-bar truss	10	25,000	17,311	30.7%	21	250
17-bar truss	17	25,000	16,299	34.8%	40	271
25-bar truss	8	25,000	17,599	29.6%	68	269
72-bar truss	16	25,000	18,144	27.4%	220	411
200-bar truss	29	50,000	40,683	18.6%	966	1320
Transmission tower	32	50,000	24,865	50.3%	2306	1870

In contrast, the CaDE is slower than the DE in five benchmark truss problems when compared in terms of time. The CaDE is only faster than the DE in the transmission tower problem. This is because integrating the AdaBoost classifier into the DE increases the complexity of the algorithm. Moreover, the CaDE requires additional time for training and employing the AdaBoost model. However, for the transmission tower, this additional time is small in comparison with the computational time of fitness evaluation where involving structural analysis. Therefore the total computing time of the CaDE is lower than that of the DE in this case. In general, it can be concluded that the proposed method is effective with large-scale structures.

5 Conclusions

In this paper, the AdaBoost classifier-assisted Differential Evolution method, called CaDE, is proposed for optimizing the weight of truss structures. By integrating the AdaBoost classification technique into the DE algorithm, the unnecessary structural analyses are significantly reduced. More specifically, in the early generations, the original Differential Evolution algorithm is deployed but the data generated in this phase is collected to train an AdaBoost model with the aim of classifying the safety state of structures. In later generations, this model is used to detect and eliminate unpromising individuals.

Through five optimization problems of truss structures, the CaDE method is proved to be a robust and reliable algorithm. In most cases, the CaDE method accomplishes the optimal results as good as or better than those of other algorithms in the literature but the numbers of structural analyses performed by the CaDE method are lower than those performed by the original DE algorithm. Besides advantages, the proposed method has also some drawbacks. Firstly, it is more complex than the original DE algorithm. Moreover, the proposed method requires more computational time for training and employing the machine learning model. Therefore, it is suitable for large-scale structures in which the structural analysis task is very time-consuming.

In the future, the application of the CaDE method can be extended for other types of structures. Furthermore, using machine learning models to improve the performance of other optimization algorithms also forms an interesting research topic.

Acknowledgement: The authors gratefully acknowledge the financial support provided by HUCE. Besides, the first author T. H. N. would also like to thank VINIF for the support.

Funding Statement: This research is funded by Hanoi University of Civil Engineering (HUCE) in Project Code 35-2021/KHXD-TĐ.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Jenkins, W. M. (1991). Towards structural optimization via the genetic algorithm. *Computers & Structures*, 40(5), 1321–1327. DOI 10.1016/0045-7949(91)90402-8.
2. Rajeev, S., Krishnamoorthy, C. S. (1992). Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering*, 118(5), 1233–1250. DOI 10.1061/(ASCE)0733-9445(1992)118:5(1233).
3. Cai, J., Thierauf, G. (1996). Evolution strategies for solving discrete optimization problems. *Advances in Engineering Software*, 25(2–3), 177–183. DOI 10.1016/0965-9978(95)00104-2.
4. Papadrakakis, M., Lagaros, N. D., Thierauf, G., Cai, J. (1998). Advanced solution methods in structural optimization based on evolution strategies. *Engineering Computations*, 15(1), 12–34. DOI 10.1108/02644409810200668.
5. Wang, Z., Tang, H., Li, P. (2009). Optimum design of truss structures based on differential evolution strategy. *2009 International Conference on Information Engineering and Computer Science*, Wuhan, China, IEEE.
6. Wu, C. Y., Tseng, K. Y. (2010). Truss structure optimization using adaptive multi-population differential evolution. *Structural and Multidisciplinary Optimization*, 42(4), 575–590. DOI 10.1007/s00158-010-0507-9.
7. Camp, C. V., Meyer, B. J., Palazolo, P. J. (2004). Particle swarm optimization for the design of trusses. *Structures 2004: Building on the Past, Securing the Future, Structure Congress 2004*, Nashville, Tennessee, USA, ASCE.
8. Perez, R. L., Behdinan, K. (2007). Particle swarm approach for structural design optimization. *Computers & Structures*, 85(19–20), 1579–1588. DOI 10.1016/j.compstruc.2006.10.013.
9. Sonmez, M. (2011). Artificial Bee colony algorithm for optimization of truss structures. *Applied Soft Computing*, 11(2), 2406–2418. DOI 10.1016/j.asoc.2010.09.003.
10. Lee, K. S., Geem, Z. W. (2004). A new structural optimization method based on the harmony search algorithm. *Computers & Structures*, 82(9–10), 781–798. DOI 10.1016/j.compstruc.2004.01.002.
11. Lee, K. S., Han, S. W., Geem, Z. W. (2011). Discrete size and discrete-continuous configuration optimization methods for truss structures using the harmony search algorithm. *Iran University of Science & Technology*, 1(1), 107–126.
12. Degertekin, S. O., Hayalioglu, M. S. (2013). Sizing truss structures using teaching-learning-based optimization. *Computers & Structures*, 119, 177–188. DOI 10.1016/j.compstruc.2012.12.011.
13. Camp, C. V., Farshchin, M. (2014). Design of space trusses using modified teaching-learning based optimization. *Engineering Structures*, 62, 87–97. DOI 10.1016/j.engstruct.2014.01.020.
14. Kaveh, A., Mahdavi, V. R. (2014). Colliding bodies optimization method for optimum design of truss structures with continuous variables. *Advances in Engineering Software*, 70, 1–12. DOI 10.1016/j.advengsoft.2014.01.002.
15. Kaveh, A., Mahdavi, V. R. (2014). Colliding bodies optimization method for optimum discrete design of truss structures. *Computers & Structures*, 139, 43–53. DOI 10.1016/j.compstruc.2014.04.006.
16. Degertekin, S. O. (2012). Improved harmony search algorithms for sizing optimization of truss structures. *Computers & Structures*, 92, 229–241. DOI 10.1016/j.compstruc.2011.10.022.

17. Bureerat, S., Pholdee, N. (2016). Optimal truss sizing using an adaptive differential evolution algorithm. *Journal of Computing in Civil Engineering*, 30(2), 04015019. DOI 10.1061/(ASCE)CP.1943-5487.0000487.
18. Ho-Huu, V., Nguyen-Thoi, T., Vo-Duy, T., Nguyen-Trang, T. (2016). An adaptive elitist differential evolution for optimization of truss structures with discrete design variables. *Computers & Structures*, 165, 59–75. DOI 10.1016/j.compstruc.2015.11.014.
19. Pham, H. A. (2016). Truss optimization with frequency constraints using enhanced differential evolution based on adaptive directional mutation and nearest neighbor comparison. *Advances in Engineering Software*, 102, 142–154. DOI 10.1016/j.advengsoft.2016.10.004.
20. Papadrakakis, M., Lagaros, N. D., Tsompanakis, Y. (1998). Structural optimization using evolution strategies and neural networks. *Computer Methods in Applied Mechanics and Engineering*, 156(1–4), 309–333. DOI 10.1016/S0045-7825(97)00215-6.
21. Papadrakakis, M., Lagaros, N. D., Tsompanakis, Y. (1999). Optimization of large-scale 3-D trusses using evolution strategies and neural networks. *International Journal of Space Structures*, 14(3), 211–223. DOI 10.1260/0266351991494830.
22. Salajegheh, E., Gholizadeh, S. (2005). Optimum design of structures by an improved genetic algorithm using neural networks. *Advances in Engineering Software*, 36(11–12), 757–767. DOI 10.1016/j.advengsoft.2005.03.022.
23. Taheri, F., Ghasemi, M. R., Dizangian, B. (2020). Practical optimization of power transmission towers using the RBF-based ABC algorithm. *Structural Engineering and Mechanics*, 73(4), 463–479. DOI 10.12989/sem.2020.73.4.463.
24. Hosseini, N., Ghasemi, M. R., Dizangian, B. (2022). ANFIS-based optimum design of real power transmission towers with size, shape and panel variables using BBO algorithm. *IEEE Transactions on Power Delivery*, 37(1), 29–39. DOI 10.1109/TPWRD.2021.3052595.
25. Nguyen, T. H., Vu, A. T. (2020). Using neural networks as surrogate models in differential evolution optimization of truss structures. *International Conference on Computational Collective Intelligence*, Switzerland, Springer.
26. Mai, H. T., Kang, J., Lee, J. (2021). A machine learning-based surrogate model for optimization of truss structures with geometrically nonlinear behavior. *Finite Elements in Analysis and Design*, 196, 103572. DOI 10.1016/j.finel.2021.103572.
27. Kaveh, A., Gholipour, Y., Rahami, H. (2008). Optimal design of transmission towers using genetic algorithm and neural networks. *International Journal of Space Structures*, 23(1), 1–19. DOI 10.1260/026635108785342073.
28. Chen, T. Y., Cheng, Y. L. (2010). Data-mining assisted structural optimization using the evolutionary algorithm and neural network. *Engineering Optimization*, 42(3), 205–222. DOI 10.1080/03052150903110942.
29. Kramer, O., Barthelmes, A., Rudolph, G. (2009). Surrogate constraint functions for CMA evolution strategies. *Annual Conference on Artificial Intelligence*, Germany, Springer.
30. Poloczek, J., Kramer, O. (2013). Local SVM constraint surrogate models for self-adaptive evolution strategies. *Annual Conference on Artificial Intelligence*, Germany, Springer.
31. Wang, H., Jin, Y. (2018). A random forest-assisted evolutionary algorithm for data-driven constrained multiobjective combinatorial optimization of trauma systems. *IEEE Transactions on Cybernetics*, 50(2), 536–549. DOI 10.1109/TCYB.6221036.
32. Freund, Y., Schapire, R. E. (1997). A Decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119–139. DOI 10.1006/jcss.1997.1504.
33. Nguyen, T. H., Vu, A. T. (2021). Evaluating structural safety of trusses using machine learning. *Frattura ed Integrità Strutturale*, 15(58), 308–318. DOI 10.3221/IGF-ESIS.58.23.
34. Storn, R., Price, K. (1997). Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341–359. DOI 10.1023/A:1008202821328.

35. Das, S., Suganthan, P. N. (2010). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1), 4–31. DOI 10.1109/TEVC.2010.2059031.
36. Das, S., Mullick, S. S., Suganthan, P. N. (2016). Recent advances in differential evolution—An updated survey. *Swarm and Evolutionary Computation*, 27, 1–30. DOI 10.1016/j.swevo.2016.01.004.
37. Charalampakis, A. E., Tsiatas, G. C. (2019). Critical evaluation of metaheuristic algorithms for weight minimization of truss structures. *Frontiers in Built Environment*, 5. DOI 10.3389/fbuil.2019.00113.
38. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., et al. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12, 2825–2830.
39. Ferreira, A. J. (2009). *MATLAB codes for finite element analysis*. Netherlands: Springer.
40. Georgioudakis, M., Plevris, V. (2020). A comparative study of differential evolution variants in constrained structural optimization. *Frontiers in Built Environment*, 6. DOI 10.3389/fbuil.2020.00102.