Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.080453

ARTICLE

Weighted k-NNC: An Efficient Computation Reduction Method for Metaheuristic-Based Structural Optimization

Anh-Vu Nguyen[1], Tien-Chuong Vu[1], Ba-Duan Nguyen[1], Hoang-Anh Pham[1,*] and Ravipudi Venkata Rao[2]

[1]Hanoi University of Civil Engineering, 55 Giai Phong road, Hanoi, Vietnam
[2]Sardar Vallabhbhai National Institute of Technology, Surat, India

*Corresponding Author: Hoang-Anh Pham. Email: anhph2@huce.edu.vn

Received: 10 February 2026; Accepted: 16 March 2026; Published: 27 April 2026

ABSTRACT: Structural optimization is essential for finding optimal designs in practical engineering tasks. Metaheuristic algorithms have been widely applied in structural optimization problems in recent years, especially when dealing with discrete design variables, the nonlinearity of the objective function and constraints. Unlike gradient-based algorithms, which rely on the slope variation of a function, metaheuristic algorithms do not require derivative calculations and thus avoid being trapped in local optimum. However, metaheuristic algorithms often require numerous function evaluations, involving costly structural analyses, thus increasing computational load considerably. This paper investigates a method to reduce computational load, specifically by reducing the number of function evaluations for metaheuristic-based structural optimization problems. The proposed strategy is based on eliminating unpromising designs during the optimization process. For each newly generated solution, an early assessment through its k nearest neighbors, named k-nearest neighbor comparison (k-NNC), is applied, acting as a filter. If a solution is deemed less promising, it is eliminated without going through the function evaluation step. Conversely, if a solution is deemed good, it is retained for the next comparison and selection step. This paper presents the implementation sequence of k-NNC, highlighting its disadvantages in terms of efficiency and accuracy. From this, a new method, the distance-weighted k-nearest neighbor comparison (wkNNC), has been developed. In wkNNC, the distance from the k neighbors to the solution under consideration is used as the weight for comparison. Furthermore, an archive of infeasible solutions and the potential solution refinement are introduced for enhancing the accuracy and efficiency of wkNNC. The superiority of wkNNC is demonstrated in the sizing optimization of some benchmark discrete cross-section truss structures. The wkNNC method, combined with the Best-Worst-Random (BWR) algorithm, achieves a computational load reduction of over 80 percent.

KEYWORDS: Structural optimization; metaheuristic algorithm; computation reduction; k-NN; distance-weighted k-NNC; BWR algorithm

1 Introduction

Structural optimization has become an indispensable tool in modern engineering design, enabling engineers to develop efficient, lightweight, and cost-effective structures while satisfying multiple constraints including strength, stiffness, stability, and manufacturability [1,2]. Over the past three decades, metaheuristic algorithms have emerged as powerful optimization techniques for structural optimization [3–7]. The superior of metaheuristics is their capability of handling complex, non-convex, and multi-modal structural optimization problems where traditional gradient-based methods often struggle [8–10]. These metaheuristic

algorithms, including Genetic Algorithms (GA) [11], Simulated Annealing (SA) [12], Particle Swarm Optimization (PSO) [13], Differential Evolution (DE) [14], Teaching Learning Based Optimization (TLBO) [15], and more recent variants such as Grey Wolf Optimizer (GWO) [16], Whale Optimization Algorithm (WOA) [17], Jaya algorithm [18], Rao algorithms [19], and so on [20], have demonstrated remarkable robustness in finding near-optimal solutions without requiring gradient information or convexity assumptions.

However, the primary limitation of metaheuristic algorithms in structural optimization applications is their substantial computational cost, which stems from the requirement to evaluate thousands or even millions of candidate designs during the search process. Each function evaluation typically involves finite element analysis (FEA) [21,22], which can be extremely time-consuming, particularly for large-scale structures, nonlinear problems, or dynamic analyses [23]. For instance, a single FEA of a complex three-dimensional structure may require several minutes to hours, and when multiplied by the population size and number of generations in a metaheuristic algorithm, the total computational time can become prohibitive for practical engineering applications. This computational burden has been identified as the main drawback preventing the widespread adoption of metaheuristic-based structural optimization in the real-world.

Recognizing this critical challenge, researchers have developed numerous strategies to reduce computational costs while maintaining solution quality. These approaches can be broadly categorized into several methodologies, such as surrogate-based optimization techniques [24–27], dimensionality reduction methods [28], efficient reanalysis procedures [29,30], parallel and distributed computing strategies [31,32], and hybrid frameworks combining multiple approaches [33–35]. Among various computational acceleration strategies, k-Nearest Neighbor (k-NN) based approaches have emerged as promising alternatives to other surrogate modeling techniques due to their simplicity, non-parametric nature, and ability to handle high-dimensional problems effectively. Therefore, this contribution focuses on computation reduction by the k-NN based approach.

The k-NN method, originally developed for pattern recognition and classification [36,37], operates on the principle that similar inputs produce similar outputs. In the context of structural optimization, this translates to the idea that structurally similar designs will exhibit similar performance characteristics. Unlike global surrogate models such as Kriging or polynomial response surfaces that attempt to approximate the entire design space, k-NN provides local approximations based only on the most relevant historical data points, making it particularly suitable for problems with complex, non-stationary response landscapes [38].

The integration of k-NN with metaheuristic algorithms typically follows a database-driven framework where all evaluated designs and their responses are stored in a historical database. The process comprises three main stages. First, an initial database (or population) is constructed by generating a number of designs using sampling techniques, and then evaluating them using exact function evaluation (e.g., finite element analysis, FEA). In the second stage, within each iteration of the metaheuristic, new candidate designs are generated by metaheuristic's operators (crossover, mutation, velocity update, etc.), and then k nearest neighbors are used to approximate fitness by k nearest neighbors from the database; only the most promising candidates are selected for actual FEA evaluation (pre-screening), and the database is continuously updated. The final stage continues until convergence. This process effectively leverages the computational efficiency of k-NN (prediction cost is dominated by neighbor search) while continuously improving approximation accuracy as the database grows. The key insight is that k-NN does not need to be globally accurate across the entire design space—it only needs to reliably distinguish promising from unpromising candidates in the current population.

In structural optimization, k-NN has been successfully combined with metaheuristic algorithms for reducing the computational cost. Singh and Kapania [39] used k-NN as active learner for accelerating the GA optimization of complex structures and showed that k-NN assisted GA could reduce total FEA evaluations

by 50% while converging to solutions within 2%–5% of those found by standard GA. Krempser et al. [40] developed a k-NN based surrogate model in order to improve the performance of DE in computationally expensive truss optimization problems. Pham [41] introduced a new approach for computation reduction, the nearest-neighbor-comparison (NNC), pre-screening candidate (trial) designs based on comparing their nearest neighbor (k = 1) in the population with the target solution. The NNC strategy was integrated with an improved DE algorithm for continuous sizing of truss structures under frequency constraints and provided about 50% reduction of FEA evaluations (see Ref. [42]). Recently, the NNC concept has been extended to solve discrete truss sizing problems (see Ref. [43]), where k (k > 1) nearest neighbors of a candidate solution were used in the NNC framework, forming the k-Nearest Neighbor Comparison (k-NNC) method. The k-NNC has been successfully combined with Rao algorithms [19] and reduced 40%–60% total FEA evaluations.

The key difference between k-NNC introduced by Pham et al. [43] and other k-NN models is that k-NNC functions as a specialized prescreening filter. In k-NNC, k nearest neighbors are not used to approximate the fitness function value (regression), nor are they used for classification. As presented in Section 2, k-NNC compares the k nearest neighbors of a potential design with the target solution at the selection step of the metaheuristic algorithm; and only when the majority of k nearest neighbors are better is the potential design further evaluated using exact FEA computation. In this way, the drawbacks of the k-NN method, such as accuracy (when working with a sparse database) or computational cost (when working with a large database), are overcome. Besides, since the k nearest neighbors are derived from the current population, k-NNC does not require additional memory to store training data like traditional k-NN models [43].

While the k-NNC framework demonstrated significant potential by reducing FEA evaluations by up to 60%, it possesses two critical theoretical limitations that this study aims to overcome. First, the original k-NNC exhibits a 'population convergence bias'. As the metaheuristic process progresses, the population naturally migrates toward the feasible region. Consequently, the k-nearest neighbors found within the current population become predominantly feasible, causing the filter to lose its 'spatial memory' of the infeasible boundaries. This leads to a significant decrement in filtering accuracy in later iterations, where the algorithm may misjudge and retain more than 50% of trial designs that are actually infeasible. Second, the k-NNC framework assumes a uniform influence for all k-neighbors. In k-NNC, every neighbor is assigned an equal vote regardless of its actual distance from the trial solution, which contradicts the fundamental principle that closer neighbors should provide more reliable information about the local landscape.

To address these deficiencies, this paper introduces a conceptual leap through the weighted k-NNC (wkNNC) framework. This new approach moves beyond simple voting by implementing a distance-weighted dominance mechanism, utilizing the inverse distance to prioritize the influence of closer neighbors. More importantly, we introduce an active infeasible archive that serves as a permanent 'memory map' of the design space's infeasible regions. By seeking neighbors from both the current population and this specialized archive, the proposed wkNNC maintains a high resolution of the feasible/infeasible boundary throughout the entire optimization process. Additionally, a potential solution refinement procedure is incorporated as a safety mechanism to prevent the accidental rejection of high-quality designs located near boundary regions. In this way, wkNNC restructures the comparison process in k-NNC to create an "adaptive filter" based on local search history. These enhancements transform the filter from a simple proximity checker into an adaptive, memory-driven decision system.

The remaining parts of the paper are presented as follow. The k-NNC frameworks and its limitations are described and discussed in Section 2. Based on the identified disadvantages, the distance-weighted k-nearest neighbor comparison (wkNNC) is proposed in Section 3. The superiority of wkNNC is demonstrated

in Section 4 through discrete sizing of benchmark structural problems of different scales, including the 10-bar, 25-bar, 52-bar, 72-bar, 160-bar, and 200-bar trusses. The findings are concluded in Section 5.

2 The k-NNC Assisted Metaheuristic Optimization

2.1 Brief of Structural Optimization Problem

In the field of structural engineering, structural optimization is the process of finding a set of design variables to minimize (or maximize) one or more design objectives, while still satisfying all constraints regarding load-bearing capacity, serviceability, and structural requirements. Within the scope of this study, the single-objective structural optimization problem will be considered. For example, the mathematical representation of a structural weight optimization problem is written as follows:

$$\begin{aligned} &\text{Minimize } w = f(\boldsymbol{x}) \\ &g_j(\boldsymbol{x}) \le 0 \quad j = 1, \ldots, N \\ &x_i^{low} \le x_i \le x_i^{up} \quad i = 1, \ldots, D \end{aligned} \tag{1}$$

where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_D\}$is a vector of D design variables; w is the weight of the structure; $f(\boldsymbol{x})$ and $g_j(\boldsymbol{x})$ are the objective and constraint functions, respectively; and x_i^{low} and x_i^{up} are the lower and upper limits of the i-th design variable. Design variables are quantities that characterize the structure, such as: geometric parameters of the cross-section (cross-section dimensions, cross-sectional area, moment of inertia of the cross-section, or thickness, etc.); mechanical properties of the material (elastic modulus, material density, etc.); and nodal coordinates. The constraints reflect the technical requirements such as stress, displacement, and vibration frequency that the structure must satisfy. These parameters often have complex, nonlinear relationships with the design variables, leading to challenges in solving structural optimization problems. Furthermore, constraint functions in structural optimization problems are often black-box, meaning they are not described in explicit mathematical forms, making their derivatives difficult or impossible to calculate. Moreover, design variables are often discrete. These characteristics of structural optimization problems render gradient-based algorithms, which are powerful due to their convergence speed, ineffective or unusable.

Metaheuristic algorithms, on the other hand, do not require complex derivative calculations, are well-adapted to discrete problems, and are therefore a suitable choice for solving practical problems. Metaheuristic algorithms have been successfully applied in many structural optimization problems due to their ease of integration with existing structural analysis tools (where the evaluation of the objective function and constraints is performed using tools independent of the algorithm). A brief introduction to the workflow of a metaheuristic algorithm and its performance in a 3-bar truss problem is presented below.

2.2 Brief of Population-Based Metaheuristic

The general workflow of a population-based metaheuristic algorithm includes the following main steps:

1. Initializing population: A set $\boldsymbol{P}$ of NP solutions $\boldsymbol{x}^{(p)}$ $(p = 1, \ldots, NP)$ is generated in the design domain (usually randomly generated with a uniform distribution);
2. Creating trial solutions: From this population, new designs $\boldsymbol{x}'^{(p)}$ are created through algorithm operators. This step may include crossover process, mutation (e.g., GA, DE), or directional move (e.g., PSO);
3. Evaluating the trial solutions: The objective function value and constraint values corresponding to $\boldsymbol{x}'^{(p)}$ are computed;

4. Updating the population: Comparison of trial solutions $\boldsymbol{x}'^{(p)}$ with the current designs $\boldsymbol{x}^{(p)}$ is performed using the objective function and constraint values. If $\boldsymbol{x}'^{(p)}$ is better than $\boldsymbol{x}^{(p)}$, it will replace $\boldsymbol{x}^{(p)}$ in the population;
5. Checking for convergence or termination condition: when the condition is met, the algorithm will stop and report the best optimal result; otherwise, it will return to step 2.

Typically, to achieve the required level of convergence, metaheuristic algorithms must perform many iterations. This is because the process of generating new designs often involves randomness. Therefore, metaheuristics require numerous function calls to evaluate the objective function and constraints for the generated designs. The computational time becomes enormous once the function evaluations are expensive, as often seen in structural design problems involving FEA evaluations.

To illustrate, consider a structural weight optimization problem for a simple truss consisting of 3 members as shown in Fig. 1. The design variables are the cross-section areas of the truss members A_1 and A_2 (since $A_3 = A_1$). Assuming all members are made of the same material, the objective function is the total volume of the truss, $V = 2\sqrt{2}LA_1 + LA_2$, which is a linear function of A_1 and A_2. The constraint is that the stress in the truss members does not exceed 2 kN/cm^2. Fig. 2 shows the design domain with stress curves having a value of 2 kN/cm^2. The feasible boundary can be seen as a non-linear curve. The optimal volume of the truss is 263.89584 cm^3, corresponding to bar areas $A_1 = 0.788675$ cm^2 and $A_2 = 0.40825$ cm^2, which is a point on the feasibility boundary ($\sigma_3 = 2$ kN/cm^2) shown as '*' in Fig. 2. To achieve a result close to 263.896 cm^3, the "DE/rand/1/bin" algorithm (see Ref. [14]) with a population $NP = 20$ requires in average 50 iterations, corresponding to approximately 1000 evaluations of the objective function and constraints. Fig. 3 illustrates the population convergence at iterations 1, 5, 20, and 50.

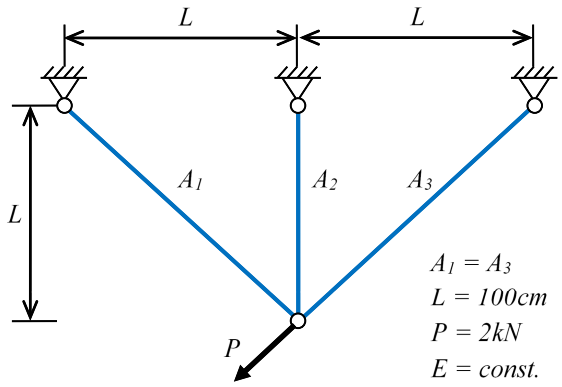


Figure 1: The 3-bar truss model.

2.3 K-Nearest Neighbor Comparison (k-NNC)

(a) Basics of k-NNC

The major drawback of metaheuristic algorithms is the large number of function calls required. As the specific nature of structural optimization, the "cost" of function evaluation is heavily skewed toward finite element analysis (FEA). In our case, the computational burden is not as the simple calculation of structural weight (the objective function), but as the structural analysis (FEA) required to check constraints like stress and displacement. Calculating structural weight is typically a "cheap" algebraic operation involving bar lengths and areas. However, the cost of a single FEA can take minutes to hours, particularly for large-scale structures.

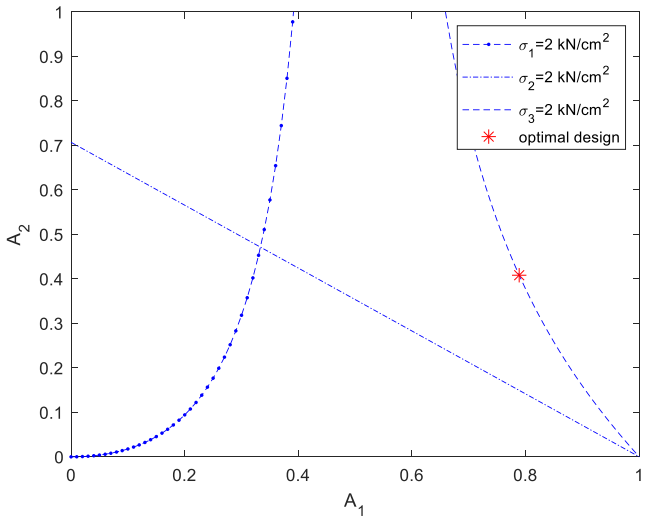


Figure 2: The design domain of the 3-bar truss problem and stress boundaries.

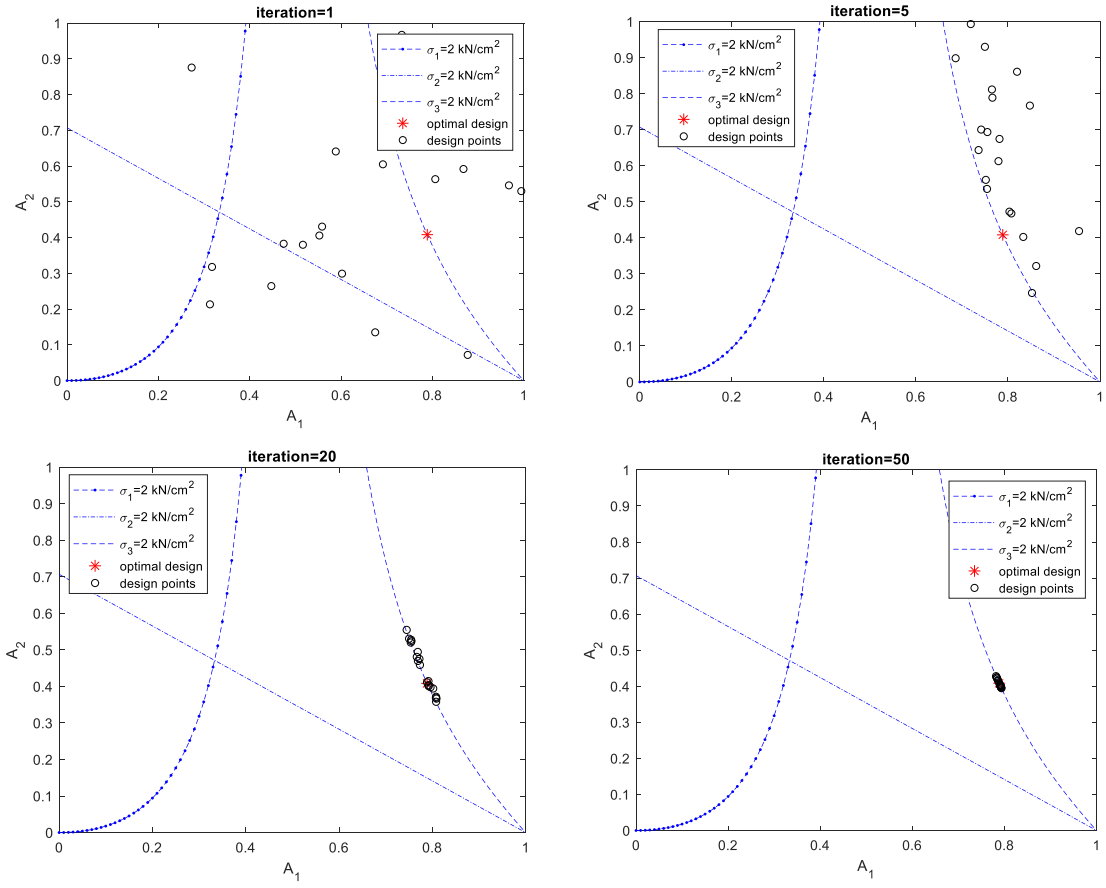


Figure 3: Population convergence in 3-bar truss optimization by DE/rand/1/bin.

Pham et al. [43] proposed a novel method called k-Nearest Neighbor Comparison (k-NNC) to reduce unnecessary and costly function evaluations. The operating principle of k-NNC is based on using nearest

neighbors to predict the potential of a new solution before performing costly function evaluations. The following are the specific steps for implementing k-NNC:

1. Identifying nearest neighbors: For each newly created candidate solution $\boldsymbol{x}'^{(p)}$, find k nearest neighbors (k-NNs) in the current population $\boldsymbol{P}$. The measure commonly used is the normalized Euclidean distance to determine the similarity between solutions.
2. Evaluating the potential of $\boldsymbol{x}'^{(p)}$: Compare the known fitness value of each of its neighbors with the fitness value of the current design solution $\boldsymbol{x}^{(p)}$,
   - A new solution $\boldsymbol{x}'^{(p)}$ is considered a potentially useless solution (PUS) if the majority of its k-NNs are not better than $\boldsymbol{x}^{(p)}$.
   - Conversely, if the majority of its k-NNs are of better than $\boldsymbol{x}^{(p)}$, the new solution $\boldsymbol{x}'^{(p)}$ is considered potentially useful.
3. Removal or evaluation:
   - If deemed potentially useless (PUS), $\boldsymbol{x}'^{(p)}$ will be removed immediately without performing function evaluation and comparison with $\boldsymbol{x}^{(p)}$.
   - If deemed potentially useful, $\boldsymbol{x}'^{(p)}$ will be evaluated and compared with $\boldsymbol{x}^{(p)}$ as normal.

The flowchart of the k-NNC assisted metaheuristic algorithm is given in Fig. 4.

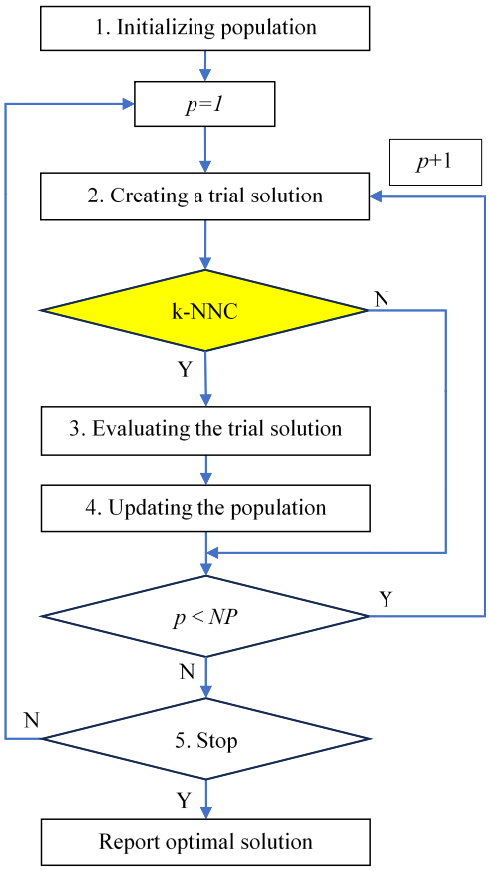


Figure 4: Flowchart of k-NNC assisted metaheuristic algorithm.

The k-NNC is an adaptive classification strategy designed as a pre-screening to reduce computational costs in optimization using metaheuristic algorithms. As shown in Pham et al. [43], the application of k-NNC in discrete sizing of truss structures can reduce structural analyses for Rao-1 and Rao-2 algorithms by up to 60%.

It is noteworthy that k-NNC incurs computational costs. The "prediction cost" of the k-NNC filter is dominated by the neighbor search. While searching for neighbors does add a computational step, it involves calculating normalized Euclidean distances, which is fundamentally a mathematical calculation far simpler than solving the large systems of equations required for FEA. The distance calculations become inexpensive due to a small database size. To prevent the computational cost from escalating (a known disadvantage of k-NN when working with large databases), k-NNC commonly works on a limit search space. Specifically, it searches for neighbors only within the current population. This ensures that the distance calculation remains a low-dimensional search regardless of how many iterations the optimization performs. The trade-off is highly favorable since the algorithm might skip up to 60% of structural analyses. Given that FEA is identified as the "main drawback" preventing metaheuristic adoption in real-world engineering, the marginal cost of performing distance calculations for a small population is viewed as negligible compared to the thousands of seconds saved by avoiding unnecessary FEA evaluations.

(b) Limitation of k-NNC

It can be seen that the efficiency and accuracy of the k-NNC filter depend on the degree of similarity of $\boldsymbol{x}'^{(p)}$ with its k-NNs. The k-NNC strategy relies on the existing population $\boldsymbol{P}$ to determine the worthiness of trial solutions $\boldsymbol{x}'^{(p)}$. As stated in Pham et al. [43], the accuracy of the k-NNC increases with increasing population size. However, this also increases the computational cost to find k-NNs. More importantly, the k-NNC exhibits a bias towards regions of the search space that the current population covers. As the optimization process continues, the population will gradually converge towards the feasible solution region. This means that the k-NNs of $\boldsymbol{x}'^{(p)}$ (if only taken from the current population) are feasible designs. If $\boldsymbol{x}'^{(p)}$ is created in the unfeasible region, k-NNC may still consider it a potential design and not reject it. This is illustrated in Fig. 5 for the 3-bar truss problem optimized by the "DE/rand/1/bin" algorithm combined with k-NNC. In Fig. 5, the black circles represent designs in the current population; the red filled circles represent trial designs that were judged as potentially designs and retained for the selection step; and the blue filled squares represent trial designs that were considered as unpromising and rejected. It can be seen that in the first iteration (left figure), the population are spread across the entire design domain, and k-NNC judges the trial solutions well (most of the retained $\boldsymbol{x}'^{(p)}$ are in better positions than $\boldsymbol{x}^{(p)}$). However, in the 5-th iteration, when most of the population are in the feasible region, k-NNC misjudges and retains 6 trial designs in the unfeasible region, accounting for over 50% of the retained trial designs.

Another important point is that in k-NNC, all k-NNs have equal importance when compared to $\boldsymbol{x}^{(p)}$. This means that every vote will have a value of 1 if a k-NN is better than $\boldsymbol{x}^{(p)}$ and −1 if the k-NN is worse than $\boldsymbol{x}^{(p)}$. It is well known that distance plays an important role in k-NN approach and it should be assigned higher influence to closer neighbors and lower influence to farther ones.

Based on the identified disadvantages of k-NNC, a novel approach—distance-weighted k-nearest neighbor comparison (wkNNC)—is proposed. The method is described in the next section.

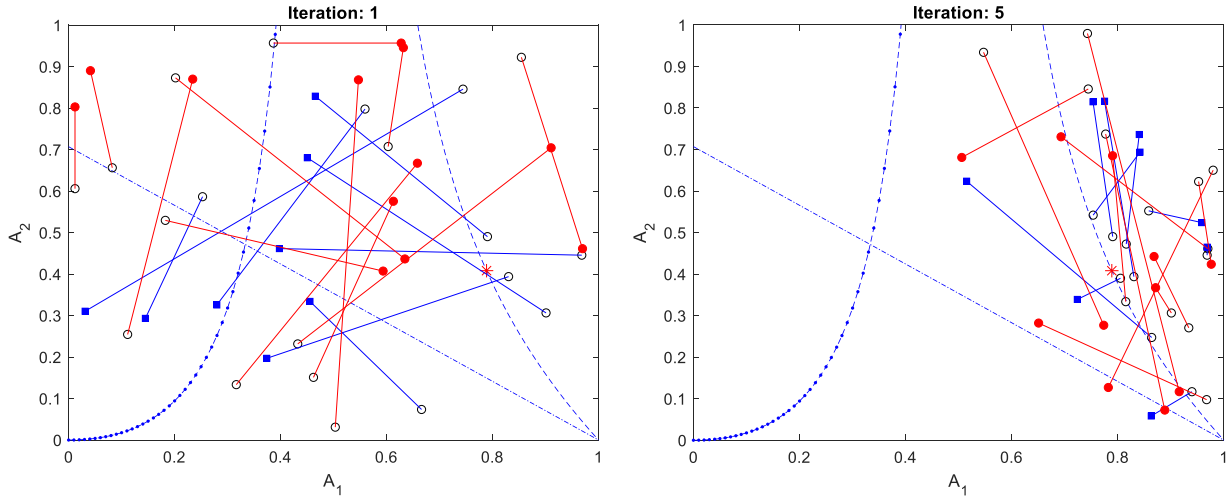


Figure 5: Effectiveness of k-NNC for 3-bar truss optimization.

3 Proposed Weighted k-Nearest Neighbor Comparison (wkNNC)

3.1 Concept of Distance-Weighted Dominance

Let $\boldsymbol{x}_{n1}, \boldsymbol{x}_{n2}, \ldots, \boldsymbol{x}_{nk}$ be the k-NNs of $\boldsymbol{x}'^{(p)}$ and d_{ni} is the distance between $\boldsymbol{x}_{ni}$ and $\boldsymbol{x}'^{(p)}$. If $\boldsymbol{x}_{ni}$ better than $\boldsymbol{x}^{(p)}$, we say that $\boldsymbol{x}_{ni}$ dominates $\boldsymbol{x}^{(p)}$, and a distance-weighted dominance value is defined as:

$$I_{ni}^{+} = \frac{1}{d_{ni}} \Big/ \sum_{nj=1}^{k} \frac{1}{d_{nj}} \tag{2}$$

where d_i and d_j are respectively the distance from $\boldsymbol{x}_{ni}$ and $\boldsymbol{x}_{nj}$ to $\boldsymbol{x}'^{(p)}$. Now, the distance-weighted dominance value of a $\boldsymbol{x}_{ni}$ that is not better than $\boldsymbol{x}^{(p)}$ is:

$$I_{ni}^{-} = \frac{-1}{d_{ni}} \Big/ \sum_{nj=1}^{k} \frac{1}{d_{nj}} \tag{3}$$

It is easily proved that $\sum I_{ni}^{+} + \sum |I_{ni}^{-}| = 1$. Eqs. (2) and (3) indicate that the bigger distance-weighted dominance value is assigned to the closer neighbor.

Using the distance-weighted dominance values of k-NNs, the judgement for a trial design $\boldsymbol{x}'^{(p)}$ is as:

- If $\sum I_{ni}^{+} + \sum I_{ni}^{-} > 0$, $\boldsymbol{x}'^{(p)}$ is a potential design and it is kept for further evaluation.
- If $\sum I_{ni}^{+} + \sum I_{ni}^{-} \leq 0$, $\boldsymbol{x}'^{(p)}$ is a PUS and it is eliminated without further evaluation.

To deal with the zero distance when neighbors are duplicate solutions of the trial solution, which is often possible in problems with discrete variables, a small amount (10^{-16}) is added to each calculated distance value.

3.2 Active Infeasible Archive

As shown, using only neighbors taken from $\boldsymbol{P}$ can lead to incorrect judgments. To increase the accuracy of k-NNC, both feasible and infeasible regions should be explored. Thus, we store the infeasible designs that are eliminated during the selection process in an archive $\boldsymbol{P}^{-}$. The k-NNs of a trial design $\boldsymbol{x}'^{(p)}$ are sought in both the current population $\boldsymbol{P}$ and the archive $\boldsymbol{P}^{-}$. This will ensure the high similarity degree among k-NNs and $\boldsymbol{x}'^{(p)}$ throughout the optimization process, especially the case $\boldsymbol{x}'^{(p)}$ is in the infeasible region. To save

computational costs, the size of $\boldsymbol{P}^{-}$ is maintained at NP (equal to the population size). When the size of $\boldsymbol{P}^{-}$ exceeds NP, we randomly remove one member from $\boldsymbol{P}^{-}$.

The benefit of using $\boldsymbol{P}^{-}$ is that it can improve the judgment of k-NNC. Including neighbors from the infeasible region helps the algorithm assess the potential of a new candidate more accurately, especially when that candidate is near or within the infeasible region. In short, the infeasible archive $\boldsymbol{P}^{-}$ acts as a "memory map" of infeasible regions, giving k-NNC a more comprehensive view of the design space, thereby allowing more unpromising trial designs to be detected and eliminated, significantly saving costly function evaluations while ensuring optimal solution quality. Notably, the archive $\boldsymbol{P}^{-}$ is also gradually updated with new trial designs, ensuring the accuracy of k-NNC throughout the optimization process. The use of the infeasible archive is a solution to the "bias" problem in the current population-based k-NNC model.

The effectiveness of the infeasible archive is illustrated in Fig. 6. In Fig. 6, the black "x" denotes the members of $\boldsymbol{P}^{-}$ at the 10-th and 15-th iterations. As seen from Fig. 6, most of trial designs located in the infeasible region are skipped.

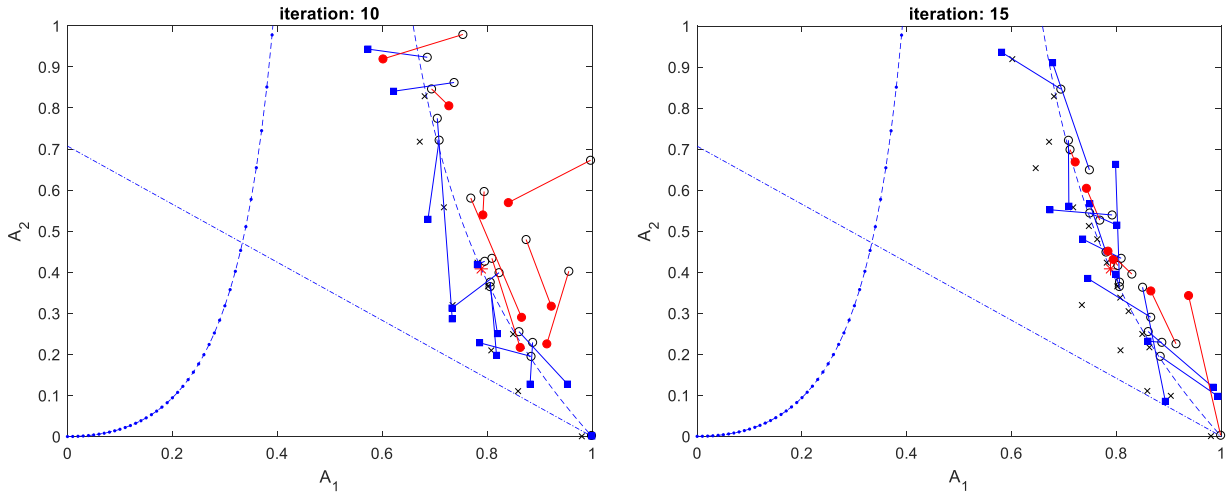


Figure 6: The effectiveness of infeasible archive in 3-bar truss optimization.

3.3 Potential Solution Refinement

The infeasible archive $\boldsymbol{P}^{-}$ brings higher accuracy and more trial solutions are skipped. However, it may also increase the possibility of rejection of good trials if they locate closed to the members of $\boldsymbol{P}^{-}$. To address this issue, a refinement is carried out for all trials judged as a PUS. The refinement procedure is conducted as follow.

1. Approximate the objective function value and constraint values of $\boldsymbol{x}'^{(p)}$ by weighted average values from k-NNs as:

$$\begin{aligned} \tilde{f}\left(\boldsymbol{x}'^{(p)}\right) &= \sum_{ni=1}^{k}\left(\frac{f(\boldsymbol{x}_{ni})}{d_{ni}} \Big/ \sum_{nj=1}^{k}\frac{1}{d_{nj}}\right) \\ \tilde{g}\left(\boldsymbol{x}'^{(p)}\right) &= \sum_{ni=1}^{k}\left(\frac{g(\boldsymbol{x}_{ni})}{d_{ni}} \Big/ \sum_{nj=1}^{k}\frac{1}{d_{nj}}\right) \end{aligned} \tag{4}$$

   where $\tilde{f}\left(\boldsymbol{x}'^{(p)}\right)$ is the approximated values of the objective function of $\boldsymbol{x}'^{(p)}$; $\tilde{g}\left(\boldsymbol{x}'^{(p)}\right)$ are the approximated values of the constraint functions.
2. Compare $\boldsymbol{x}'^{(p)}$ with $\boldsymbol{x}^{(p)}$ by using these approximated function values. If $\tilde{f}\left(\boldsymbol{x}'^{(p)}\right) < f\left(\boldsymbol{x}^{(p)}\right)$ and $\tilde{g}\left(\boldsymbol{x}'^{(p)}\right) < 0$ then retain $\boldsymbol{x}'^{(p)}$.

The refinement procedure helps to avoid potential solution from being omitted by k-NNC. This is a novel contribution to balancing computational speed and algorithmic security, helping to maintain population diversity even when the filter is highly active.

Combining the proposed improvements described in Sections 3.1–3.3 into the k-NNC framework in Section 2, a new method named weighted k-nearest neighbor comparison (wkNNC) is formed.

3.4 The BWR-wkNNC Framework

In this study, the proposed wkNNC method is combined with the Best-Worst-Random (BWR) algorithm [44]. The reason for choosing BWR is that it is a parameter-free, efficient, and robust algorithm. BWR inherits the simple philosophy from the Rao algorithms (Rao-1, Rao-2, Rao-3) [19] but is improved in the search mechanism to enhance explorative capability, suitable to handle complex engineering problems, such as discrete and multimodal ones [45–48]. Combining wkNNC with a powerful algorithm like BWR helps compensate for the use of a small population, ensuring that global search capabilities are preserved even in large-scale problems.

(a) BWR algorithm

The basic steps of BWR are similar to those described in the general procedure for population-based metaheuristic algorithms presented in Section 2.2. The difference in BWR lies in the mechanism for generating new trial designs in step 2. Corresponding to each $\boldsymbol{x}^{(p)}$, BWR creates a trial design $\boldsymbol{x}'^{(p)}$ by the following operator:

- Generate a random number r in range [0,1]
- If $r > 0.5$:

$$\boldsymbol{x}'^{(p)} = \boldsymbol{x}^{(p)} + \boldsymbol{r}_1 \left(\boldsymbol{x}^{best} - T \times \boldsymbol{x}^{rand}\right) - \boldsymbol{r}_2 \left(\boldsymbol{x}^{worst} - \boldsymbol{x}^{rand}\right) \tag{5}$$

- If $r \leq 0.5$:

$$\boldsymbol{x}'^{(p)} = \boldsymbol{x}^{up} - \boldsymbol{r}_3 \left(\boldsymbol{x}^{up} - \boldsymbol{x}^{low}\right) \tag{6}$$

where $\boldsymbol{x}^{best}$, $\boldsymbol{x}^{worst}$, and $\boldsymbol{x}^{rand}$ are, respectively, the best design, the worst design, and a random design in the current population; T has value of 1 or 2 randomly; $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, and $\boldsymbol{r}_3$ are three vectors of D uniformly random numbers in range [0, 1].

It is seen that Eq. (5) generates the new trial design towards the best design and avoids the worst design through interaction with random designs. On the other hand, Eq. (6) generates design variable values within the design domain. Eq. (6) acts like a "mutation with amnesia" and it resets a variable to a random legal value to keep the search global. Eq. (6) is pure exploration with no memory, no direction, and no bias toward the current population. This helps to escape local minima, maintain population diversity, and prevent premature convergence. Therefore, BWR is able to maintain high population diversity and effectively escape local extrema. Nevertheless, due to its great explorative capability, BWRs often have slow convergence rates and require many function evaluations. Thus, combining BWR with wkNNC ensures a balance between robustness and efficiency.

(b) Implementation of BWR-wkNNC

The workflow of the BWR algorithm combined with wkNNC (BWR-wkNNC) are as follows.

Step 1—Initializing

- Define the population size NP, the termination criteria (e.g., the maximum number of iterations or the maximum number of function evaluations), and the number of neighbors, k.
- Create an initial population $\boldsymbol{P}$ of NP solutions $\boldsymbol{x}^{(p)} = \{x_1^{(p)}, x_2^{(p)}, \ldots, x_D^{(p)}\}$ in the design domain, where each design variable $x_i^{(p)}$ is randomly generated according to

$$x_i^{(p)} = x_i^{low} + rand\,[0,1]\left(x_i^{up} - x_i^{low}\right),\ p = 1, \ldots, NP,\ i = 1, \ldots, D \tag{7}$$

- Compute the objective function value and constraint function values for each $\boldsymbol{x}^{(p)}$, i.e., $f\left(\boldsymbol{x}^{(p)}\right)$ and $g_j\left(\boldsymbol{x}^{(p)}\right)$. Find the best design $\boldsymbol{x}^{best}$ and the worst design $\boldsymbol{x}^{worst}$ in the population.
- Reserve an empty archive $\boldsymbol{P}^{-}$.

Step 2—Generating

- Corresponding to each $\boldsymbol{x}^{(p)}$, generate a random number $r \in [0,1]$; and select a random design $\boldsymbol{x}^{rand}$ from the current population.
- If $r < 0.5$, create a trial design $\boldsymbol{x}'^{(p)}$ by Eq. (5), otherwise, by Eq. (6).

Step 3—Pre-scanning and evaluating

Pre-scanning: Apply wkNNC to judge $\boldsymbol{x}'^{(p)}$:

- Find in $\boldsymbol{P}$ and $\boldsymbol{P}^{-}$ k designs $(\boldsymbol{x}_{n1}, \boldsymbol{x}_{n1}, \ldots, \boldsymbol{x}_{nk})$ closest to $\boldsymbol{x}'^{(p)}$ by using normalized Euclidean distance.
- Compare each nearest neighbor $\boldsymbol{x}_{ni}$ with $\boldsymbol{x}^{(p)}$, then determine the distance-weighted dominance value I_{ni}^{+} (if $\boldsymbol{x}_{ni}$ is better than $\boldsymbol{x}^{(p)}$) or I_{ni}^{-} (if $\boldsymbol{x}_{ni}$ is not better than $\boldsymbol{x}^{(p)}$).
- Calculate $\sum I_{ni}^{+}$ and $\sum I_{ni}^{-}$ and the evaluation flag, $eval^{(p)} = \sum I_{ni}^{+} + \sum I_{ni}^{-}$.
- If $eval^{(p)} < 0$, calculate the approximate values $\tilde{f}\left(\boldsymbol{x}'^{(p)}\right)$ and $\tilde{g}\left(\boldsymbol{x}'^{(p)}\right)$ according to Eq. (4); set $eval^{(p)} = 1$ if $\tilde{f}\left(\boldsymbol{x}'^{(p)}\right) < f\left(\boldsymbol{x}^{(p)}\right)$ and $\tilde{g}\left(\boldsymbol{x}'^{(p)}\right) < 0$.

Evaluating: For a trial design $\boldsymbol{x}'^{(p)}$ with $eval^{(p)} > 0$, compute $f\left(\boldsymbol{x}'^{(p)}\right)$ and $g_j\left(\boldsymbol{x}'^{(p)}\right)$ and continue Step 4, otherwise, go back to Step 2.

Step 4—Updating

- Replace $\boldsymbol{x}^{(p)}$ in the population by $\boldsymbol{x}'^{(p)}$ if $\boldsymbol{x}'^{(p)}$ is better than $\boldsymbol{x}^{(p)}$.
- If $\boldsymbol{x}'^{(p)}$ is not better than $\boldsymbol{x}^{(p)}$ and infeasible (one of $g_j\left(\boldsymbol{x}'^{(p)}\right) > 0$), put $\boldsymbol{x}'^{(p)}$ into $\boldsymbol{P}^{-}$. Truncate $\boldsymbol{P}^{-}$ if its size exceeds NP by removing from $\boldsymbol{P}^{-}$ a random member.
- Update $\boldsymbol{x}^{best}$ and $\boldsymbol{x}^{worst}$.

Step 5—Terminating

- Stop the optimization process if the termination criteria are satisfied, otherwise continue from Step 2.

4 Numerical Investigation

The effectiveness of the proposed wkNNC method is investigated in discrete truss sizing optimization problems. The tested structures, including 10-bar, 25-bar, 52-bar, 72-bar, 160-bar, and 200-bar trusses, are benchmark structural optimization problems used in many studies to examine the effectiveness of metaheuristic algorithms [49,50]. The objective is to minimize the total weight of the structures while ensuring the member stress and nodal displacement within the allowable limits.

The optimization problem for a truss having N_m bars is defined as:

$$\begin{aligned} &\text{Minimize } W = \rho \sum_{m=1}^{N_m} A_m L_m \\ &\sigma_m^{min} \le \sigma_m \le \sigma_m^{max}, m = 1..N_m \\ &\left|d_j\right| \le \Delta_{max}, j = 1..N_d \end{aligned} \tag{8}$$

where A_m and L_m are respectively the cross-sectional area and length of m-th member; σ_m is the member stress; d_n is the nodal displacement; ρ is the material density; σ_m^{min} and σ_m^{min} are stress limits; and Δ_{max} is the allowable displacement. Stress and displacement are obtained by finite element analysis using self-developed solver in MATLAB.

4.1 Description of Tested Structures

(a) The 10-bar planar truss:

The 10-bar truss structure is shown in Fig. 7. Two downward concentrated loads, each with a magnitude of 100 kips, are applied at nodes 2 and 4. The material has a density ρ = 0.1 lb/in^3 and an elastic modulus E = 10^4 ksi. The cross-sectional area of the bars is selected from the following set of 42 discrete values (in^2): 1.62, 1.80, 1.99, 2.13, 2.38, 2.62, 2.63, 2.88, 2.93, 3.09, 3.13, 3.38, 3.47, 3.55, 3.63, 3.84, 3.87, 3.88, 4.18, 4.22, 4.49, 4.59, 4.80, 4.97, 5.12, 5.74, 7.22, 7.97, 11.5, 13.5, 13.9, 14.2, 15.5, 16.0, 16.9, 18.8 19.9, 22.0, 22.9, 26.5, 30.0, and 33.5. The problem considers two types of constraints: stress and displacement constraints. The allowable stress in the truss members is ±25 ksi, and the maximum allowable displacement at the nodes is 2 inches.

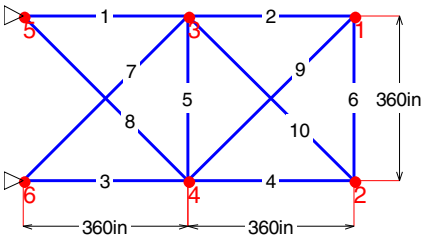


Figure 7: The 10-bar truss layout.

(b) The 25-bar spatial truss:

The truss layout is shown in Fig. 8. The bars are divided into 8 groups with different properties detailed in Table 1. The elastic modulus is 10^4 ksi, the density is 0.1 lb/in^3 and is the same for all bars. The cross-sectional area of the bars is selected according to the values (in^2): 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.8, 3.0, 3.2, 3.4. The stress limits are given in Table 1. The allowable nodal displacement in the x, y, z directions is 0.35 inches. Two load cases are considered as follows (unit: kip):

Case 1:

- Node 1: $[F_x, F_y, F_z] = [0, 20, -5]$
- Node 2: $[F_x, F_y, F_z] = [0, -20, -5]$

Case 2:

- Node 1: $[F_x, F_y, F_z] = [1, 10, -5]$
- Node 2: $[F_x, F_y, F_z] = [0, 10, -5]$
- Node 3: $[F_x, F_y, F_z] = [0.5, 0, 0]$
- Node 6: $[F_x, F_y, F_z] = [0.5, 0, 0]$

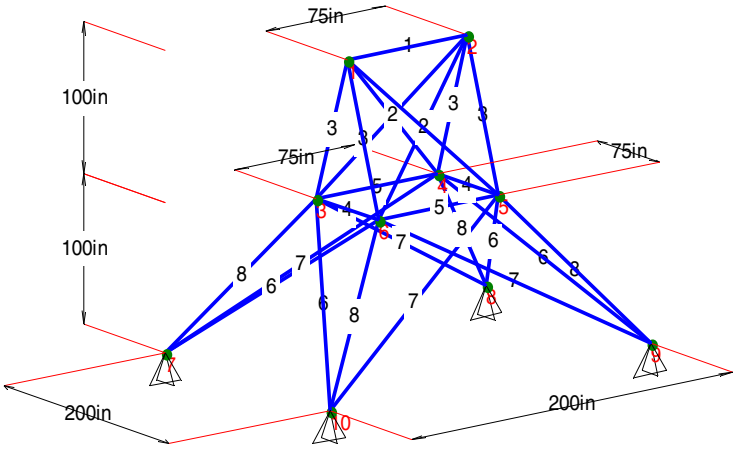


Figure 8: The 25-bar truss layout.

Table 1: Allowable tensile and compressive stresses correspond to each group of bars.

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

(c) The 52-bar planar truss:

The truss layout with the members divided into 12 groups is detailed in Fig. 9. The elastic modulus is 207 GPa, the density is 7860 kg/m^3 and is the same for all members. The cross-sectional area of the members is selected according to the AISC standard, with detailed values given in Table 2. In this problem, only the stress constraint with a stress limit of ±180 MPa is considered. The truss has the following loads: Vertical load of 200 kN and horizontal load of 100 kN applied at Nodes 17, 18, 19 and 20.

(d) The 72-bar spatial truss:

The layout of the truss with the members divided into 16 groups is detailed in Fig. 10. The elastic modulus is 10^4 ksi, the density is 0.1 lb/in^3 and is the same for all members. The cross-sectional area of the members is selected from the values given in Table 3. The stress limit is ±25 ksi and the allowable nodal displacement in the x, y, z directions is 0.25 inches. Consider the following two load cases (units: kip):

- Case 1: Node 1: $[F_x, F_y, F_z] = [5, 5, 5]$
- Case 2: Nodes 1, 2, 3 and 4: $[F_x, F_y, F_z] = [5, 5, 5]$

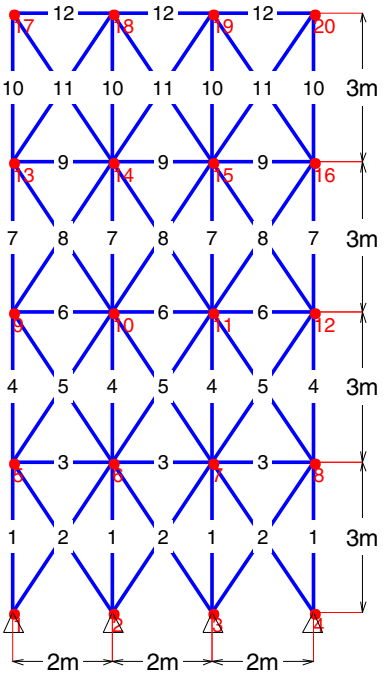


Figure 9: The 52-bar truss layout.

Table 2: List of cross-sectional area for 52-bar truss structure.

No.	Area (mm^2)	No.	Area (mm^2)	No.	Area (mm^2)	No.	Area (mm^2)
1	71.613	17	1008.385	33	2477.414	49	7419.340
2	90.968	18	1045.159	34	2496.769	50	8709.660
3	126.451	19	1161.288	35	2503.221	51	8967.724
4	161.29	20	1283.868	36	2696.769	52	9161.272
5	198.064	21	1374.191	37	2722.575	53	9999.980
6	252.258	22	1535.481	38	2896.768	54	10,322.560
7	285.161	23	1690.319	39	2961.284	55	10,903.204
8	363.225	24	1696.771	40	3096.768	56	12,129.008
9	388.386	25	1858.061	41	3206.445	57	12,838.684
10	494.193	26	1890.319	42	3303.219	58	14,193.520
11	506.451	27	1993.544	43	3703.218	59	14,774.164
12	641.289	28	729.031	44	4658.055	60	15,806.420
13	645.16	29	2180.641	45	5141.925	61	17,096.740
14	792.256	30	2238.705	46	5503.215	62	18,064.480
15	816.773	31	2290.318	47	5999.988	63	19,354.800
16	939.998	32	2341.931	48	6999.986	64	21,612.860

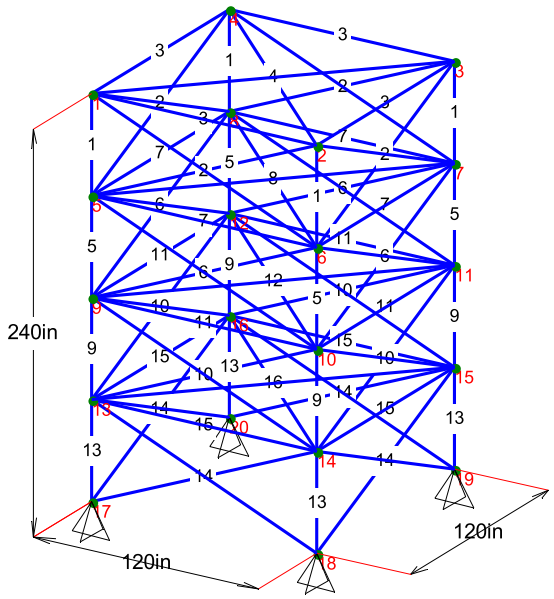


Figure 10: The 72-bar truss layout.

Table 3: List of cross-sectional area for 72-bar truss structure.

No.	Area (in^2)	No.	Area (in^2)	No.	Area (in^2)	No.	Area (in^2)
1	0.111	17	1.563	33	3.840	49	11.500
2	0.141	18	1.620	34	3.870	50	13.500
3	0.196	19	1.80	35	3.880	51	13.900
4	0.25	20	1.990	36	4.180	52	14.200
5	0.307	21	2.130	37	4.220	53	15.500
6	0.391	22	2.380	38	4.490	54	16.000
7	0.442	23	2.620	39	4.590	55	16.900
8	0.563	24	2.630	40	4.800	56	18.800
9	0.602	25	2.880	41	4.970	57	19.900
10	0.766	26	2.930	42	5.120	58	22.000
11	0.785	27	3.090	43	5.740	59	22.900
12	0.994	28	3.130	44	7.220	60	24.500
13	1.000	29	3.380	45	7.970	61	26.500
14	1.228	30	3.470	46	8.530	62	28.000
15	1.266	31	3.550	47	9.300	63	30.000
16	1.457	32	3.630	48	10.850	64	33.500

(e) The 160-bar spatial truss:

The layout of the truss is shown in Fig. 11. The members are divided into 38 groups. The elastic modulus is 2.047×10^6 kgf/cm^2, the density is 0.00785 kg/cm^3 and is the same for all members. The cross-sectional area of the members is selected from values given in Table 4. The allowable stress is ±1500 kgf/cm^2 and the members under compression are checked for stability according to Eq. (9).

$$\sigma^{st} = \begin{cases} 1300 - \dfrac{(L/r)^2}{24} & \text{if } \dfrac{L}{r} \le 120 \\ \dfrac{10^7}{(L/r)^2} & \text{if } \dfrac{L}{r} > 120 \end{cases} \tag{9}$$

where σ^{st} is the compressive stress limit; L is the length of the bar; r is the radius of gyration of the cross-section detailed in Table 4. The truss has 8 loading cases as detailed in Table 5.

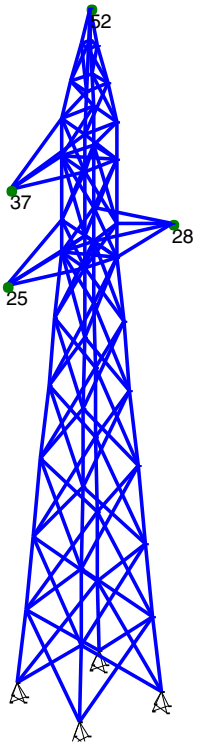


Figure 11: The 160-bar truss layout.

Table 4: List of cross-sectional properties for 160-bar truss structure.

No.	Area (cm^2)	r (cm)	No.	Area (cm^2)	r (cm)	No.	Area (cm^2)	r (cm)
1	1.84	0.47	15	9.4	1.35	29	33.9	2.33
2	2.26	0.57	16	10.47	1.36	30	34.77	2.97
3	2.66	0.67	17	11.38	1.45	31	39.16	2.54
4	3.07	0.77	18	12.21	1.55	32	43	2.93
5	3.47	0.87	19	13.79	1.75	33	45.65	2.94
6	3.88	0.97	20	15.39	1.95	34	46.94	2.94
7	4.79	0.97	21	17.03	1.74	35	51	2.92
8	5.27	1.06	22	19.03	1.94	36	52.1	3.54
9	5.75	1.16	23	21.12	2.16	37	61.82	3.96
10	6.25	1.26	24	23.2	2.36	38	61.9	3.52
11	6.84	1.15	25	25.12	2.57	39	68.3	3.51

(Continued)

Table 4 (continued)

No.	Area (cm^2)	r (cm)	No.	Area (cm^2)	r (cm)	No.	Area (cm^2)	r (cm)
12	7.44	1.26	26	27.5	2.35	40	76.38	3.93
13	8.06	1.36	27	29.88	2.56	41	90.6	3.92
14	8.66	1.46	28	32.76	2.14	42	94.13	3.92

Table 5: Loading cases for the 160-bar truss structure.

Case	Node	F_x (N)	F_y (N)	F_z (N)	Case	Node	F_x (N)	F_y (N)	F_z (N)
1	52	868	0	491	5	52	917	0	491
	37	996	0	546		37	951	0	546
	25	1091	0	546		25	1015	0	546
	28	1091	0	546		28	636	1259	428
2	52	493	1245	363	6	52	917	0	491
	37	996	0	546		37	572	1303	428
	25	1091	0	546		25	1015	0	546
	28	1091	0	546		28	1015	0	546
3	52	917	0	491	7	52	917	0	491
	37	951	0	546		37	951	0	546
	25	1015	0	546		25	1015	0	546
	28	1015	0	546		28	636	1303	428
4	52	917	0	546	8	52	498	1460	363
	37	572	1259	428		37	951	0	546
	25	1015	0	546		25	1015	0	546
	28	1015	0	546		28	1015	0	546

(f) The 200-bar planar truss:

The truss structure is shown in Fig. 12. The material has a density $\rho = 0.283$ lb/in^3 and an elastic modulus E = 30,000 ksi. The problem only considers the stress constraint with an allowable limit of ±10 ksi. The truss members are divided into 29 cross-sectional area groups, with values selected from the following set of 30 discrete values (in^2): 0.1, 0.347, 0.44, 0.539, 0.954, 1.081, 1.174, 1.333, 1.488, 1.764, 2.142, 2.697, 2.8, 3.131, 3.565, 3.813, 4.805, 5.952, 6.572, 7.192, 8.525, 9.3, 10.85, 13.33, 14.29, 17.17, 19.18, 23.68, 28.08, and 33.7. Three load cases are considered:

- Case 1: Horizontal load of 1.0 kip at nodes 1, 6, 15, 20, 29, 34, 43, 48, 57, 62, and 71.
- Case 2: Downward vertical load of 10.0 kips at nodes: 1–5, 6, 8, 10, 12, 14, 15–19, 20, 22, 24, 26, 28, 29–33, 34, 36, 38, 40, 42, 43–47, 48, 50, 52, 54, 56, 57–61, 62, 64, 66, 68, 70, and 71–75.
- Case 3: A combination of Case 1 and Case 2.

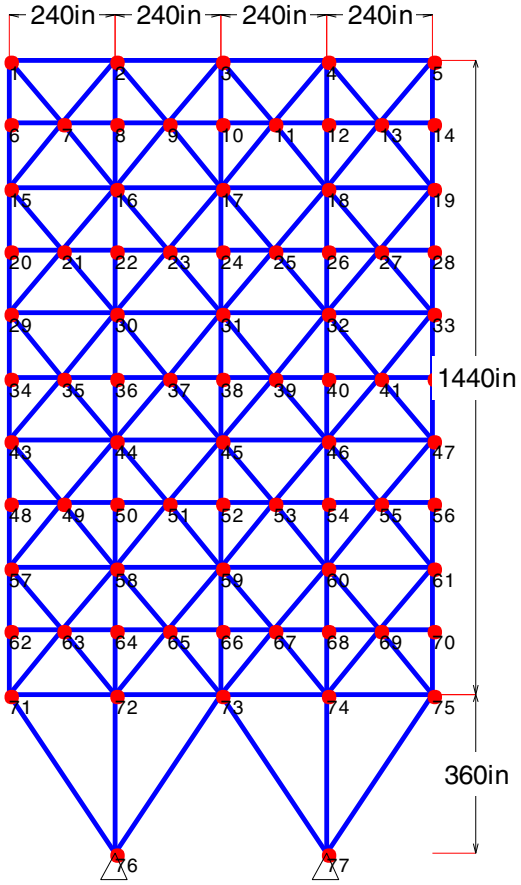


Figure 12: The 200-bar truss layout.

4.2 Test Conditions

(a) Constraint handling:

The constraint function of a design $\boldsymbol{x}$ are defined as:

$$g(x) = \max\left(c_m^{\sigma}, c_j^{\Delta}\right), m = 1..N_m, j = 1..N_d \tag{10}$$

where c_m^{σ} is the stress constraint violation, computed as

$$c_m^{\sigma} = \begin{cases} \dfrac{\sigma_m}{\sigma_m^{max}} - 1, & if\ \sigma_m \geq 0 \\ \dfrac{|\sigma_m|}{|\sigma_m^{min}|} - 1, & if\ \sigma_m < 0 \end{cases} \tag{11}$$

and c_j^{Δ} is the displacement constraint violation, computed as

$$c_j^{\Delta} = \frac{|d_j|}{\Delta_{max}} - 1 \tag{12}$$

(b) Solution comparison

For solution comparison in this study, the Deb's rules are applied. A design $\boldsymbol{x}'$ is better than a design $\boldsymbol{x}$ when:

$$\begin{aligned} &g(\boldsymbol{x}') < g(\boldsymbol{x}), g(\boldsymbol{x}) > 0 \\ &g(\boldsymbol{x}'), g(\boldsymbol{x}) \leq 0, \text{ and } W(\boldsymbol{x}') < W(\boldsymbol{x}) \end{aligned} \tag{13}$$

Notably, when using the condition Eq. (13), we can save the constraint computations $g(\boldsymbol{x}')$ if $W(\boldsymbol{x})$, $g(\boldsymbol{x})$ and $W(\boldsymbol{x}')$ are known. Specifically, in the case where $g(\boldsymbol{x}) \leq 0$ and $W(\boldsymbol{x}') > W(\boldsymbol{x})$, it can be immediately concluded that $\boldsymbol{x}'$ is inferior to $\boldsymbol{x}$ without needing to calculate $g(\boldsymbol{x}')$. Thus, the number of structural analysis (NSA) will be reduced and will always be lower than the number of objective function evaluations (NFE). This is significant for the structural weight optimization problem, because usually the cost of one constraint calculation, related to structural model analysis, is much more expensive than one structural weight calculation.

(c) Algorithm setting

To evaluate the effectiveness of the proposed improvements, we consider four algorithms: (1) original BWR; (2) BWR combined with k-NNC (BWR-kNNC); (3) BWR combined with k-NNC and infeasible archive (BWR-ikNNC); and (4) BWR combined with wkNNC (BWR-wkNNC). For each problem, the comparison of algorithms is performed statistically across multiple independent runs (25 in this study) with a completely randomly generated initial population. This is a common approach in structural optimization, as the randomness of metaheuristic algorithms means that multiple runs may be required to achieve a reliable design. Obtained statistical results include the best optimal weight, the average optimal weight, the worst optimal weight, and the standard deviation are analyzed and compared. In each run, the optimization process ends when the relative error value of the weights in the population, $e = \left|\overline{W}/W^{best} - 1\right|$, is less than a predetermined value, taken as 10^{-6}, where $\overline{W}$ and W^{best} are the mean and best weight values, respectively. The number of neighbors is chosen as 3 based on a sensitivity analysis. The size of population and infeasible archive is 25 for all tests. Keeping $NP = 25$ is a deliberate strategy to ensure that the computational cost of neighborhood search does not explode as the number of iterations increases. If NP is too large, the cost of finding neighbors can negate the advantage of reducing the number of FEA analyses.

4.3 Results and Discussion

First, the effects of the number of neighbors on the performance of wkNNC is considered. Then, the effectiveness of k-NNC, ikNNC, and wkNNC is evaluated. Finally, the results of BWR-wkNNC are compared with those of other published metaheuristic algorithms.

(a) Effects of number of neighbors

The investigation was conducted for the 10-bar truss structure. Table 6 presents the results comparing the performance of BWR-wkNNC when varying different k values (from 2 to 5) and compared with the case without using wkNNC.

Table 6: Optimal weight for 10-bar truss by BWR-wkNNC with different k-values.

***k*-value**	**Best**	**Mean**	**Worst**	**SD**	**NFE/NSA**
without wkNNC	5490.7379	5490.7379	5490.7379	1.87E−12	**18,394/15,435**
2	5490.7379	5495.6925	5531.0356	10.8215	2606/1657
3	*5490.7379*	*5494.2888*	*5527.8362*	*9.4059*	*2866/1699*

(Continued)

Table 6 (continued)

k-value	Best	Mean	Worst	SD	NFE/NSA
4	5490.7379	5499.0333	5536.5584	15.1806	2554/1531
5	5490.7379	5494.6608	5536.5584	10.0785	2796/1679

Some observations are as follows:

1. Accuracy (Best weight)
   - All k values, as well as the case without using wkNNC, achieve the best optimal weight of 5490.7379 lb. This shows that integrating wkNNC and changing the k parameter does not affect the ability of the BWR algorithm to find the global optimal solution.
2. Stability (Mean and SD):
   - With k = 3, the algorithm yields the most stable results with the lowest mean (5494.2888 lb) and smallest SD (9.4059 lb) among the cases using wkNNC.
   - k = 4 has the least stability with the highest mean (5499.0333 lb) and highest SD (15.1806 lb); k = 2 and k = 5 have average stability, with SDs of 10.8215 lb and 10.0785 lb, respectively.
   - The case without wkNNC has an extremely low SD (1.87E−12 lb), since all 25 runs yielded the same best result.
3. Computational efficiency (NFE/NSA):
   - Without using wkNNC, the number of objective function evaluations (NFE) is up to 18,394. When applying wkNNC, the NFE decreases to only about 2554 to 2866, i.e., a reduction of up to 86% in the number of function evaluations.
   - Computational efficiency does not differ significantly with different k values.

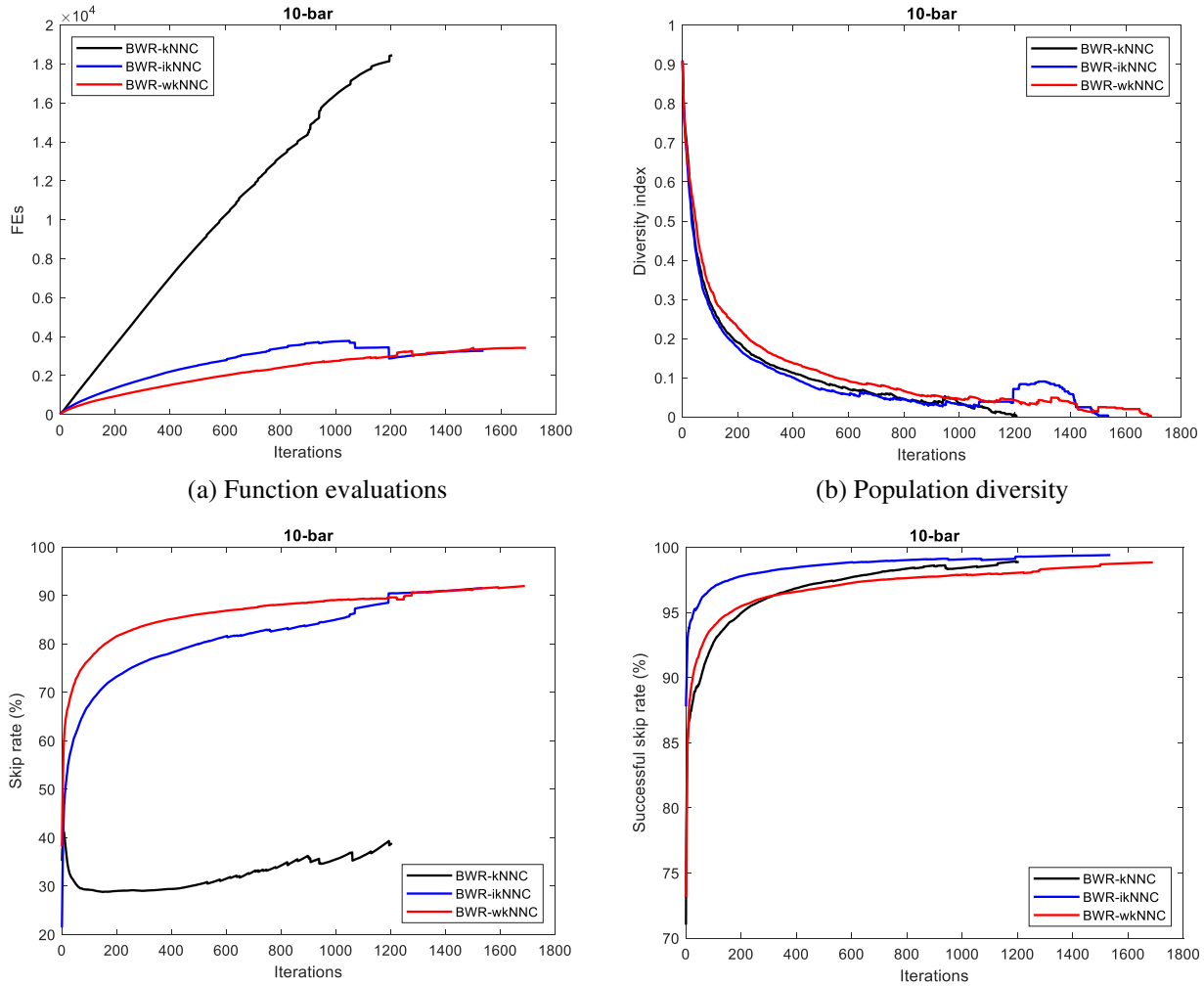
Based on the above analyses, the value k = 3 is chosen for further investigation because it provides the best balance between computational efficiency and solution stability. Although this analysis is performed on the 10-bar problem, the goal is to find a configuration that is "good enough" to serve as a benchmark for comparison between algorithms in more complex problems. With a relatively small population size (*NP* = 25), using a large k value would not guarantee accuracy because the designs are far apart. Conversely, using a small k value (1 or 2) increases the likelihood of bias. As experimental results in Section 4.3 (b) show, with k = 3, the wkNNC method still significantly reduces the number of function evaluations (NFE) and structural analysis (NSA) on all problems.

(b) Effectiveness of k-NNC, ikNNC, and wkNNC

Table 7 and Fig. 13 present the analysis results regarding the computational reduction, accuracy, and convergence of k-NNC, ikNNC, and wkNNC in the 10-bar truss problem. The data from Table 7 and the graphs in Fig. 13 shows a significant improvement from kNNC to ikNNC and wkNNC in the ability to reduce computational costs while maintaining high accuracy.

Table 7: Efficiency of kNNC, ikNNC, and wkNNC for 10-bar truss with *NP* = 25 and *k* = 3.

	BWR-kNNC	BWR-ikNNC	BWR-wkNNC
Skip rate (%)	38.80	84.24	89.26
Successful skip rate (%)	98.51	99.06	98.11



(a) Function evaluations

(b) Population diversity

(c) Function evaluation skip rate

(d) Successful skip rate

Figure 13: Performance of kNNC, ikNNC, and wkNNC for the 10-bar truss with $NP = 25$ and $k = 3$.

1. Performance in reducing computation (Skip rate—rate of skipping objective function evaluations):
   - ikNNC shows the biggest leap in performance compared to k-NNC, with the skip rate increasing from 38.80% to 84.24%. This indicates that the inclusion of the infeasible archive helped detect and skip redundant computations much more effectively.
   - With wkNNC, the skip rate continues to improve, reaching 89.26%. Fig. 13a,c visually demonstrates that BWR-wkNNC maintained a very high skip rate throughout the optimization process.
2. Prediction reliability (Successful skip rate): The successful skip rate indicates the accuracy of the skips:
   - ikNNC achieves the highest accuracy, with a rate of 99.06%, followed by k-NNC (98.51%). For wkNNC, the success rate decreased slightly to 98.11%. However, this number is still very high, ensuring the stability of the optimal solution. Fig. 13d shows that the success rates of all three algorithms remained quite stable across iterations.
3. Population convergence and diversity: The population diversity is measured by the diversity index DI calculated by Eq. (14). Despite omitting nearly 90% of the function evaluations, the population diversity plot for the NNC variants gradually decreased with each iteration (Fig. 2b), indicating that the improvements did not cause premature convergence or dispersion in the search space.

$$DI = \frac{1}{NP}\sum_{p=1}^{NP}\left[\sum_{i=1}^{D}\frac{(x_i^{(p)} - \overline{x}_i)}{x_i^{up} - x_i^{low}}\right], \overline{x}_i = \frac{1}{NP}\sum_{p=1}^{NP} x_i^{(p)} \tag{14}$$

Statistical results for all six problems are given in Table 8 and Figs. 14–19. Below is a detailed discussion and evaluation for each truss problem, focusing on a comparison between BWR and BWR-wkNNC through two aspects: computational efficiency and convergence characteristics.

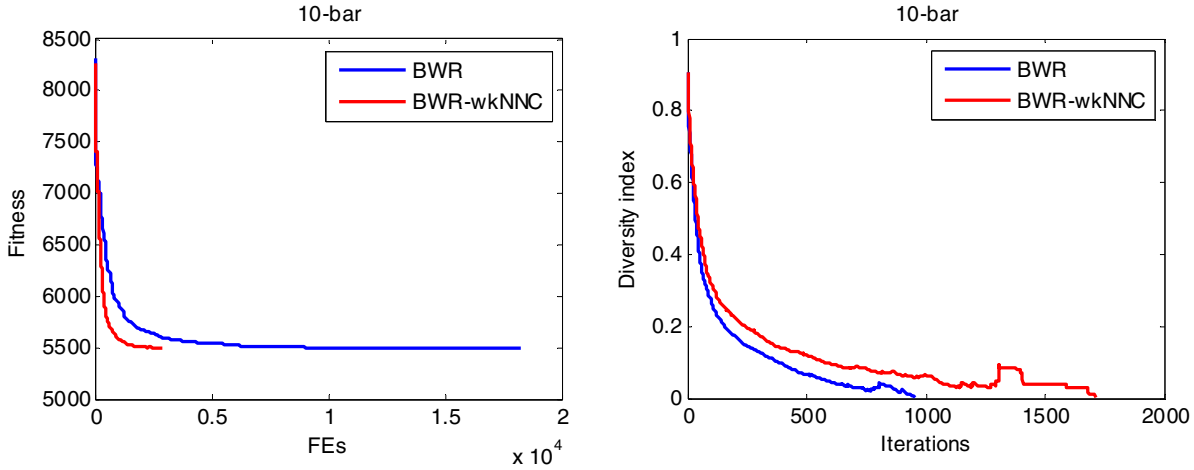
Table 8: Results of BWR, BWR-kNNC, BWR-ikNNC, BWR-wkNNC with $NP = 25$ and $k = 3$.

Problem	Method	Best	Mean	Worst	SD	NFE/NSA
10-bar	Ref.	5490.7379				
	BWR	5490.7379	5490.7379	5490.7379	1.87E−12	18,394/15,435
	BWR-kNNC	5490.7379	5491.0434	5498.3746	1.5274	12,929/11,056
	BWR-ikNNC	5490.7379	5492.2714	5521.4384	6.2652	3071/2142
	BWR-wkNNC	5490.7379	5494.2888	5527.8362	9.4059	2866/1699
25-bar	Ref.	551.0372				
	BWR	551.0372	551.0883	552.3355	0.23225	20,091/16,532
	BWR-kNNC	551.0372	551.2444	552.443	0.43629	9471/8207
	BWR-ikNNC	551.0372	551.1554	552.9335	0.40194	2376/1738
	BWR-wkNNC	551.0372	551.4317	553.9892	0.85014	2329/1396
52-bar	Ref.	1902.6055				
	BWR	1902.6055	1902.7643	1905.0147	0.40243	33,627/29,523
	BWR-kNNC	1902.6055	1902.7589	1903.3661	0.28339	22,607/20,428
	BWR-ikNNC	1902.6055	1904.6683	1914.8367	3.3487	6620/5790
	BWR-wkNNC	1902.6055	1904.8343	1914.6236	3.9513	4799/4011
72-bar	Ref.	389.3342				
	BWR	389.3342	389.5907	391.837	0.52174	42,574/33,231
	BWR-kNNC	389.3342	389.4481	389.8721	0.16777	28,262/22,777
	BWR-ikNNC	389.3342	390.4351	398.5913	1.5893	8711/5876
	BWR-wkNNC	389.3342	390.6462	393.8492	1.4931	6973/4549

(Continued)

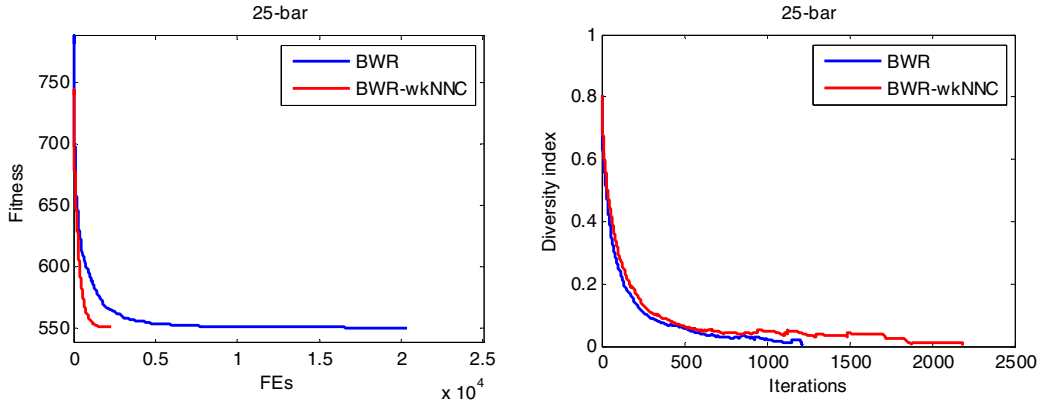
Table 8 (continued)

Problem	Method	Best	Mean	Worst	SD	NFE/NSA
160-bar	Ref.	1336.0308				
	BWR	1336.0308	1339.5999	1347.1778	3.5500	63,389/55,096
	BWR-kNNC	1336.0308	1338.7645	1345.3272	2.4982	52,344/45,717
	BWR-ikNNC	1336.0308	1340.8234	1377.0411	8.1347	15,579/12,543
	BWR-wkNNC	1336.0308	1339.7084	1346.9053	3.3108	12,311/9625
200-bar	Ref.	–				
	BWR	27,298.940	27,722.5699	28,563.0147	336.7294	84,355/72,932
	BWR-kNNC	27,407.962	27,894.0553	29,011.3267	384.5525	59,950/52,082
	BWR-ikNNC	27,384.850	28,361.6386	35,098.4095	1441.0224	15,390/12,772
	BWR-wkNNC	27,430.305	28,166.6289	28,894.5516	387.9270	10,511/8571



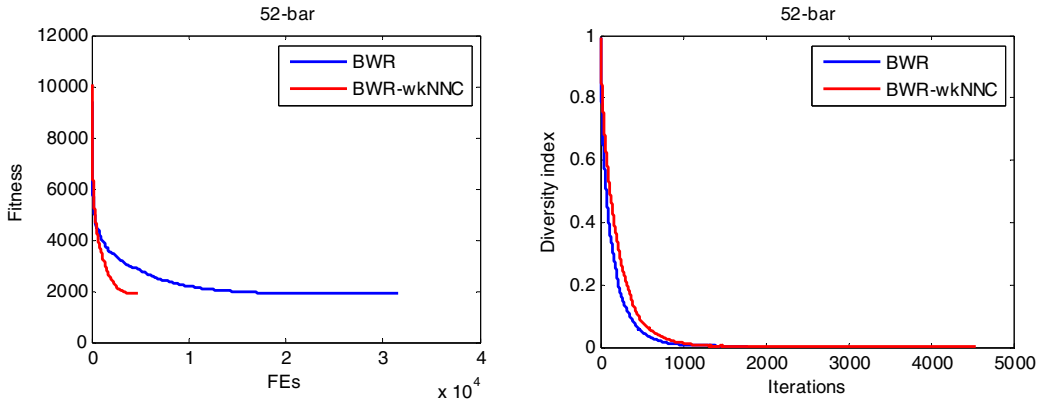
(a) Structural weight (b) Population diversity

Figure 14: Optimization of 10-bar truss: BWR vs. BWR-wkNNC.



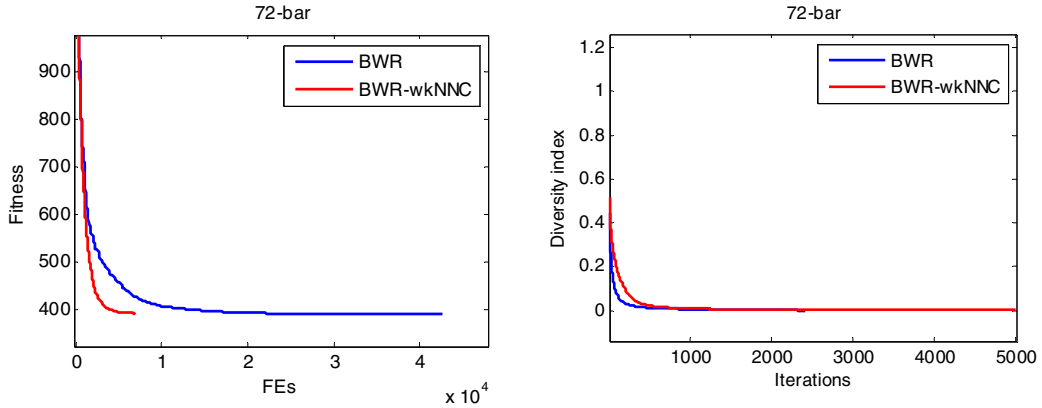
(a) Structural weight (b) Population diversity

Figure 15: Optimization of 25-bar truss: BWR vs. BWR-wkNNC.



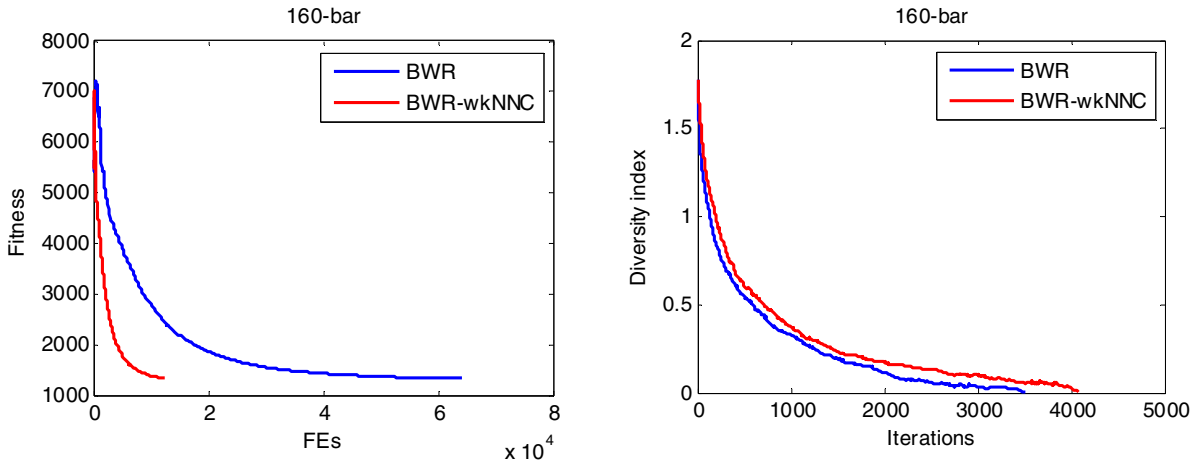
(a) Structural weight (b) Population diversity

Figure 16: Optimization of 52-bar truss: BWR vs. BWR-wkNNC.



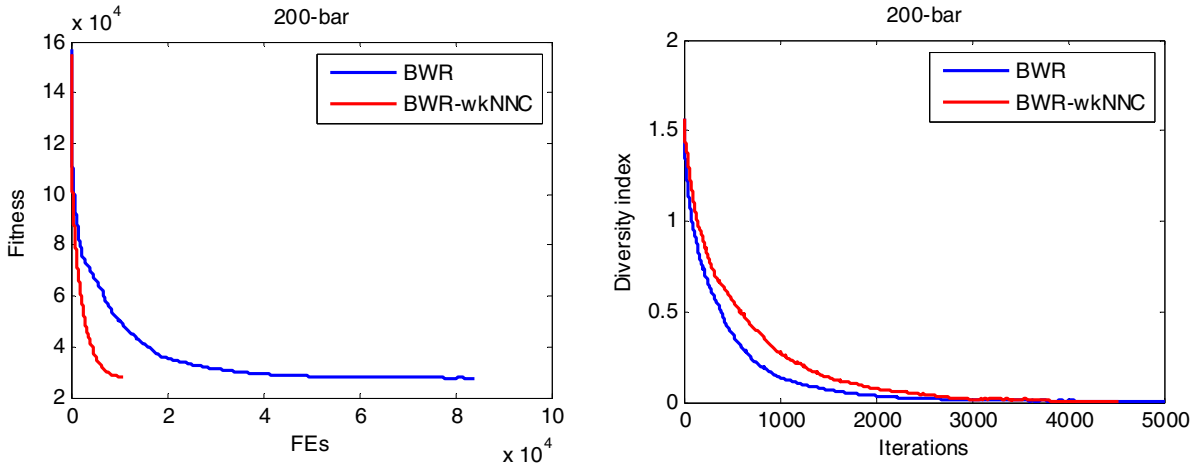
(a) Structural weight (b) Population diversity

Figure 17: Optimization of 72-bar truss: BWR vs. BWR-wkNNC.



(a) Structural weight (b) Population diversity

Figure 18: Optimization of 160-bar truss: BWR vs. BWR-wkNNC.



(a) Structural weight (b) Population diversity

Figure 19: Optimization of 200-bar truss: BWR vs. BWR-wkNNC.

(i) The 10-bar truss:

- All algorithms found the best optimal value to be 5490.7379 lb. However, BWR-wkNNC was the most cost-effective with only 2866 objective function evaluations (NFEs), a reduction of more than 6 times compared to the original BWR (18,394).
- The convergence curve of structural weights (Fig. 14a) shows that BWR-wkNNC achieves a similar optimal result with a much lower computational cost, compared with BWR.
- Regarding population diversity (Fig. 14b), the two population diversity curves are quite similar at the beginning, demonstrating that the use of wkNNC does not change the search nature of the BWR algorithm. In later iterations, BWR-wkNNC maintains a more diverse population.

(ii) The 25-bar truss:

- All methods achieved a value of 551.0372. BWR-wkNNC continues to lead in computation efficiency with 2329 FEs compared to BWR's 20,091. The resource efficiency in this problem was nearly 9 times.
- In Fig. 15a, compared to BWR, BWR-wkNNC demonstrated a superior speed rate.
- Fig. 15b shows the similarity between the two population diversity curves, ensuring that the ability to explore the design space of BWR-wkNNC is preserved even though the number of objective function evaluations is significantly reduced. Similar to the 10-bar truss, BWR-wkNNC maintains population diversity for longer than BWR.

(iii) The 52-bar truss:

- The optimal result of 1902.6055 was maintained across all algorithms. BWR-wkNNC achieved the highest computational performance with 4799 NFEs.
- The structural weight convergence curve in Fig. 16a shows that BWR-wkNNC completed the optimization process extremely efficiently compared to BWR.
- Fig. 16b shows that the population diversity of BWR-wkNNC and RWR decreased along almost the same path through the iterations. This indicates that the wkNNC prediction mechanism is very reliable.

(iv) The 72-bar truss:

- All four algorithms achieved a best value of 389.3342. BWR-wkNNC consumed 6973 NFEs, while BWR required 42,574.
- Fig. 17a shows that BWR-wkNNC continues to maintain its computational advantage over BWR.
- In Fig. 17b, population diversity is maintained at a higher level by BWR-wkNNC, demonstrating that wkNNC effectively replaces exact computations without losing important population information.

(v) The 160-bar truss:

- This is a large-scale and complex truss problem; yet all BWR variants still find an optimal weight of 1336.0308. BWR-wkNNC (SD = 3.3108) shows significantly higher stability than BWR-ikNNC (SD = 8.1347) and only requires 12,311 NFEs.
- The computational cost was significantly reduced during the optimization with the presence of wkNNC as seen in Fig. 18a.
- Fig. 18b clearly illustrates that BWR-wkNNC maintains population diversity like RWR (Fig. 18b), showing high stability even in large variable spaces.

(vi) The 200-bar truss:

- In this complex problem, the original BWR achieved the best result (27,298.9403). Other algorithms closely matched with very small deficiencies. It is observed that while BWR-wkNNC is significantly more efficient, requiring only 10,511 NFEs compared to 84,355 for BWR, there is a minor degradation in the best weight (27,430.3049) compared to the original BWR (27,298.9403). This loss of global optimality, approximately 0.48%, can be attributed to the sparsity of the neighbor database in the high-dimensional search space (D = 29). In such large-scale problems, the wkNNC filter may occasionally misidentify high-quality candidates near the feasibility boundary as unpromising due to the limited local information provided by only 50 data points (current population and archive). This represents a standard accuracy-efficiency trade-off; however, considering that BWR-wkNNC achieved this result with nearly 8 times fewer function evaluations, the method remains highly superior for practical engineering applications where computational time is the primary obstacle.
- Fig. 19a further illustrates the computational cost reduction capabilities of BWR-wkNNC.

– Regarding population diversity, the population diversity convergence curve shows a similarity between BWR-wkNNC and BWR, with BWR-wkNNC maintaining a slightly higher value (see Fig. 19b). This confirms that BWR-wkNNC retains the robust search capabilities of BWR in large problems.

(c) Comparison with other state-of-the-art metaheuristics

Comparisons are made with other metaheuristic algorithms performed on the same problems, as summarized in Table 9. The algorithms selected for comparison, including aeDE [51], EFA [52], CETDE [53], and BO [54], are those that yielded highly competitive results. Comparison criteria include accuracy, stability, and efficiency. A comparison for the 25-bar problem is not conducted because the results for this problem are not included in the published comparison algorithms.

Table 9: Comparison between BWR-wkNNC and other metaheuristic algorithms.

Problem	Method	Best	Mean	Worst	SD	NSA
10-bar	aeDE [51]	5490.7379	5502.623	5549.204	20.780	2550
	EFA [52]	5490.738	5528.227	5551.985	18.367	2070
	CETDE [53]	5490.738	5509.197	5547.064	20.218	20.218
	BO [54]	5490.7379	5504.2088	5514.4028	8.901	1260
	BWR-wkNNC	5490.7379	5493.6019	5522.2205	7.3179	1705
52-bar	aeDE [51]	1902.605	1906.735	1925.714	6.679	3402
	EFA [52]	1902.605	1904.775	1910.942	3.045	2894
	CETDE [53]	1899.654*	1903.695	1924.582	5.902	3223
	BO [54]	1902.6055	1904.4047	1907.9661	1.891	5760
	BWR-wkNNC	1902.6055	1904.8343	1914.6236	3.9513	4011
72-bar	aeDE [51]	389.334	390.913	393.325	1.161	4101
	EFA [52]	389.334	391.376	393.826	1.376	3123
	CETDE [53]	389.070*	390.072	393.11	1.157	2868
	BO [54]	389.3342	390.2501	391.9601	0.982	7020
	BWR-wkNNC	389.3342	390.6462	393.8492	1.4931	4549
160-bar	aeDE [51]	1336.634	1355.875	1410.611	18.805	21,265
	EFA [52]	1336.704	1372.551	1429.253	34.706	15,183
	CETDE [53]	1332.9559*	1342.3202	1395.5229	15.9427	7336
	BO [54]	1336.3502	1338.3399	1345.4180	2.7087	60,800
	BWR-wkNNC	1336.0308	1339.7084	1346.9053	3.3108	9625
200-bar	aeDE [51]	27,858.500	28,425.871	29,415.000	481.590	11,644
	EFA [52]	27,421.944	28,434.603	30,180.343	749.0776	5023
	CETDE [53]	27,449.18	28,920.88	31,261.482	1086.581	6548
	BO [54]	–	–	–	–	–
	BWR-wkNNC	27,430.3049	28,166.6289	28,894.5516	387.9270	8571

Note: *These optimal results violate the constraints as indicated in Table 7 in Pham et al. [43].

(i) The 10-bar truss:

– Accuracy: BWR-wkNNC, along with other algorithms, found the best structural weight (Best) of 5490.7379.

- Stability: BWR-wkNNC excelled with the lowest Mean (5493.6019) and SD (7.3179) values. This demonstrates the algorithm's high reliability and minimal variability between runs.
- Efficiency: Although the BO algorithm had the lowest number of structural analysis (NSA), BWR-wkNNC maintained a competitive NSAs (1705) while achieving better stability.

(ii) The 52-bar truss and 72-bar truss:

- Accuracy: The best optimal weight achieved by BWR-wkNNC in these two examples is comparable to most competitors (1902.6055 in the 52-bar problem, 389.3342 in the 72-bar problem). The CETDE algorithm yielded lower results (e.g., 1899.654 in the 52-bar problem or 389.070 in the 72-bar problem), but these results violated design constraints (constraint violation value, CV = 0.0002 for 52-bar and CV = 0.00004 for 72-bar as reported in Pham et al. [43]).
- Stability and efficiency: BWR-wkNNC is not competitive with other algorithms in these two examples.

(iii) The 160-bar truss:

- Accuracy: Compared to other algorithms except CETDE, BWR-wkNNC found the smallest optimal weight value (1336.0308). Although CETDE yielded a lower weight, it violated constraints with CV = 0.0007 (see Ref. [43]).
- Efficiency: The most notable feature is that the NSAs (9625) of BWR-wkNNC is much lower than that of aeDE (21,265), EFA (15,183), and BO (60,800). This confirms that the wkNNC mechanism helps the algorithm solve large problems with very economical computational costs.
- Stability: BWR-wkNNC achieved an SD of 3.3108, second only to BO (using much higher computaitonal resource), demonstrating the algorithm's good stability.

(iv) The 200-bar truss:

- Accuracy: BWR-wkNNC achieves the second-best optimal weight value after EFA. However, other statistical results such as Mean (28,166.6289) and Worst (28,894.5516) are lower than other algorithms.
- Stability: The algorithm achieves the lowest SD (387.9270). In large-scale problems like the 200-bar problem, maintaining a low SD is very difficult, and this result confirms the greater stability of BWR-wkNNC compared to other algorithms.
- Computational efficiency: BWR-wkNNC performed better than aeDE and slightly worse than EFA and CETDE.

5 Conclusion

This paper introduces a new k-NNC framework, the weighted k-NNC, for reducing expensive structural analyses when performing structural optimization using metaheuristic algorithms. Additional improvements include: integrating the set of infeasible solutions into the database for the k-NNC model, adding distance weights to the k nearest neighbors, and including the refinement for potential designs. The effectiveness of wkNNC is demonstrated through its integration with the BWR algorithm and its application to discrete sizing optimization problems for truss structures. Some key results achieved are as follows:

- The wkNNC method (with the parameter value k = 3) significantly improves the ability to eliminate unnecessary structural analyses (skip rate up to nearly 90% with accuracy up to 98%). For large-scale problems (e.g., 160-bar and 200-bar trusses), the advantage of BWR-wkNNC becomes even more apparent (reducing NFE by almost 8 times in the 200-bar problem).
- Integrating wkNNC does not significantly change the convergence capability of the original algorithm or lose the diversity of the population.

- The wkNNC, when combined with BWR algorithm, produces a powerful and reliable optimization tool (BWR-wkNNC). Its performance, compared to some state-of-the-art algorithms, is summarized in three main points:
- High accuracy: BWR-wkNNC consistently ranks among the top in finding the best solutions.
- Leading stability: BWR-wkNNC surpasses algorithms such as aeDE, EFA, and BO in terms of result consistency, especially in problems with a large number of variables.
- Balanced efficiency: BWR-wkNNC achieves a balance between reducing the number of structural analysis and maintaining solution quality, especially in large-scale truss optimization problems.

In summary, the wkNNC method is a robust optimization tool that successfully resolves the computational bottleneck of metaheuristic structural optimization. By combining distance-weighted dominance, an active infeasible archive, and a potential solution refinement safeguard, it reduces the need for expensive FEA by approximately 80%–90% without compromising accuracy. This framework is exceptionally suited for the design of complex, large-scale engineering structures where the high cost of structural analysis previously prohibited the use of advanced metaheuristic algorithms.

Nevertheless, the wkNNC performance when combined with other metaheuristic algorithms, e.g., algorithms with different convergence dynamics, remains uncovered. Besides, the application of wkNNC to mix variables problems has been unexplored. These limitations need a further study.

Acknowledgement: None.

Funding Statement: This research is funded by Hanoi University of Civil Engineering (HUCE) under grant number 39–2025/KHXD-TĐ.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Hoang-Anh Pham; methodology, Hoang-Anh Pham, Anh-Vu Nguyen, Ravipudi Venkata Rao; software, Hoang-Anh Pham, Anh-Vu Nguyen; validation, Hoang-Anh Pham, Anh-Vu Nguyen; formal analysis, Anh-Vu Nguyen, Ba-Duan Nguyen, Tien-Chuong Vu; investigation, Anh-Vu Nguyen, Ba-Duan Nguyen, Tien-Chuong Vu; resources, Anh-Vu Nguyen, Tien-Chuong Vu, Ba-Duan Nguyen; data curation, Anh-Vu Nguyen; writing—original draft preparation, Hoang-Anh Pham, Anh-Vu Nguyen, Tien-Chuong Vu; writing—review and editing, Hoang-Anh Pham, Ravipudi Venkata Rao; visualization, Anh-Vu Nguyen; supervision, Hoang-Anh Pham; project administration, Hoang-Anh Pham; funding acquisition, Anh-Vu Nguyen. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, Hoang-Anh Pham, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Arora JS. Introduction to optimum design. 5th ed. Amsterdam, The Netherlands: Elsevier; 2024. doi:10.1016/C2018-0-03380-0.
2. Yang XS, Koziel S, Leifsson L. Computational optimization, modelling and simulation: recent trends and challenges. Procedia Comput Sci. 2013;18(1):855–60. doi:10.1016/j.procs.2013.05.250.
3. Lamberti L, Pappalettere C. Metaheuristic design optimization of skeletal structures: a review. Comp Tech Rev. 2010;4:1–32. doi:10.4203/ctr.4.1.
4. Saka MP, Hasançebi O, Geem ZW. Metaheuristics in structural optimization and discussions on harmony search algorithm. Swarm Evol Comput. 2016;28(6):88–97. doi:10.1016/j.swevo.2016.01.005.

5. Kaveh A. Advances in metaheuristic algorithms for optimal design of structures, Vol. 16. Cham, Switzerland: Springer International Publishing; 2014. 631 p. doi:10.1007/978-3-319-46173-1.
6. Kaveh A. Applications of metaheuristic optimization algorithms in civil engineering. Cham, Switzerland: Springer International Publishing; 2017. doi:10.1007/978-3-319-48012-1.
7. Kaveh A, Ilchi Ghazaan M. Meta-heuristic algorithms for optimal design of real-size structures. Cham, Switzerland: Springer International Publishing; 2018. doi:10.1007/978-3-319-78780-0.
8. Ficarella E, Lamberti L, Degertekin SO. Comparison of three novel hybrid metaheuristic algorithms for structural optimization problems. Comput Struct. 2021;244(1):106395. doi:10.1016/j.compstruc.2020.106395.
9. Lagaros ND, Plevris V, Kallioras NA. The mosaic of metaheuristic algorithms in structural optimization. Arch Comput Methods Eng. 2022;29(7):5457–92. doi:10.1007/s11831-022-09773-0.
10. Ghaemifard S, Ghannadiasl A. A comparison of metaheuristic algorithms for structural optimization: performance and efficiency analysis. Adv Civ Eng. 2024;2024(1):2054173. doi:10.1155/2024/2054173.
11. Holland JH. Genetic algorithms. Sci Am. 1992;267(1):66–73. doi:10.1038/scientificamerican0792-66.
12. Kirkpatrick S, Gelatt CD Jr, Vecchi MP. Optimization by simulated annealing. Science. 1983;220(4598):671–80. doi:10.1126/science.220.4598.671.
13. Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of the ICNN'95—International Conference on Neural Networks; 1995 Nov 27–Dec 1; Perth, WA, Australia. p. 1942–8. doi:10.1109/ICNN.1995.488968.
14. Storn R, Price K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim. 1997;11(4):341–59. doi:10.1023/A:1008202821328.
15. Rao RV, Savsani VJ, Vakharia DP. Teaching-learning–based optimization: a novel method for constrained mechanical design optimization problems. Comput Aided Des. 2011;43(3):303–15. doi:10.1016/j.cad.2010.12.015.
16. Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. Adv Eng Softw. 2014;69:46–61. doi:10.1016/j.advengsoft.2013.12.007.
17. Mirjalili S, Lewis A. The whale optimization algorithm. Adv Eng Softw. 2016;95(12):51–67. doi:10.1016/j.advengsoft.2016.01.008.
18. Rao RV. Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int J Ind Eng Comput. 2016;7:19–34. doi:10.5267/j.ijiec.2015.8.004.
19. Rao RV. Rao algorithms: three metaphor-less simple algorithms for solving optimization problems. Int J Ind Eng Comput. 2020;11(1):107–30. doi:10.5267/j.ijiec.2019.6.002.
20. Sapkota SC, Cavaleri L, Khatri A, Pandey S, Paudel S, Asteris PG. Optimization of truss structures using nature-inspired algorithms with frequency and stress constraints. Comput Model Eng Sci. 2026;146(1):1–10. doi:10.32604/cmes.2025.069691.
21. Zienkiewicz OC, Taylor RL. The finite element method. 5th ed. Oxford, UK: Butterworth-Heinemann; 2000.
22. Bathe KJ. Finite element procedures. 2nd ed. Watertown, MA, USA: Klaus-Jurgen Bathe; 2014.
23. Forrester A, Sobester A, Keane A. Engineering design via surrogate modelling: a practical guide. Hoboken, NJ, USA: John Wiley & Sons, Inc.; 2008.
24. Forrester AIJ, Keane AJ. Recent advances in surrogate-based optimization. Prog Aerosp Sci. 2009;45(1–3):50–79. doi:10.1016/j.paerosci.2008.11.001.
25. Koziel S, Leifsson L. Surrogate-based modeling and optimization: applications in engineering. New York, NY, USA: Springer; 2013. doi:10.1007/978-1-4614-7551-4.
26. Nguyen TH, Vu AT. An efficient differential evolution for truss sizing optimization using AdaBoost classifier. Comput Model Eng Sci. 2023;134(1):429–58. doi:10.32604/cmes.2022.020819.
27. Wu C, Kong W. Interior point method-assisted differential evolution for expensive optimization of secondary source deployment. Comput Model Eng Sci. 2026;146(2):1–10. doi:10.32604/cmes.2026.077613.
28. Serani A, Diez M. A survey on design-space dimensionality reduction methods for shape optimization. Arch Comput Methods Eng. 2025. doi:10.1007/s11831-025-10349-x.
29. Kirsch U, Bogomolni M, Sheinman I. Efficient structural optimization using reanalysis and sensitivity reanalysis. Eng Comput. 2007;23(3):229–39. doi:10.1007/s00366-007-0062-1.

30. Cao H, Li H, Wang M, Huang B, Sun Y. A structural reanalysis assisted harmony search for the optimal design of structures. Comput Struct. 2022;270:106844. doi:10.1016/j.compstruc.2022.106844.
31. Papadrakakis M, Lagaros ND, Fragakis Y. Parallel computational strategies for structural optimization. Int J Numer Methods Eng. 2003;58(9):1347–80. doi:10.1002/nme.821.
32. Mukherjee S, Lu D, Raghavan B, Breitkopf P, Dutta S, Xiao M, et al. Accelerating large-scale topology optimization: state-of-the-art and challenges. Arch Comput Methods Eng. 2021;28(7):4549–71. doi:10.1007/s11831-021-09544-3.
33. Goel T, Haftka RT, Shyy W, Queipo NV. Ensemble of surrogates. Struct Multidiscip Optim. 2007;33(3):199–216. doi:10.1007/s00158-006-0051-9.
34. Shan S, Wang GG. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. Struct Multidiscip Optim. 2010;41(2):219–41. doi:10.1007/s00158-009-0420-2.
35. Zhang Y, Wang H, Liu J, Liu F, Lv X. Intelligent structural design of composite concrete-encased steel columns based on hybrid machine learning and multiobjective optimization. Struct Concr. 2026;27(1):49–72. doi:10.1002/suco.70292.
36. Fix E, Hodges JL. Discriminatory analysis, nonparametric discrimination, consistency properties. Randolph Field, TX, USA: USAF School of Aviation Medicine; 1951. p. 1–21. Project 21-49-004. Report 4.
37. Syriopoulos PK, Kalampalikis NG, Kotsiantis SB, Vrahatis MN. kNN classification: a review. Ann Math Artif Intell. 2025;93(1):43–75. doi:10.1007/s10472-023-09882-x.
38. Hastie T, Tibshirani R, Friedman J. The elements of statistical learning. New York, NY, USA: Springer; 2009. doi:10.1007/978-0-387-84858-7.
39. Singh K, Kapania RK. ALGA: active learning-based genetic algorithm for accelerating structural optimization. AIAA J. 2021;59(1):330–44. doi:10.2514/1.j059240.
40. Krempser E, Bernardino HS, Barbosa HJ, Lemonge AC. Differential evolution assisted by surrogate models for structural optimization problems. In: Proceedings of the International Conference on Computational Structures Technology (CST); 2012 Sep 4–7; Dubrovnik, Croatia.
41. Pham HA. Reduction of function evaluation in differential evolution using nearest neighbor comparison. Vietnam J Comput Sci. 2015;2(2):121–31. doi:10.1007/s40595-014-0037-2.
42. Pham HA. Truss optimization with frequency constraints using enhanced differential evolution based on adaptive directional mutation and nearest neighbor comparison. Adv Eng Softw. 2016;102(12):142–54. doi:10.1016/j.advengsoft.2016.10.004.
43. Pham HA, Dang VH, Vu TC, Nguyen BD. An efficient k-NN-based *Rao* optimization method for optimal discrete sizing of truss structures. Appl Soft Comput. 2024;154(2):111373. doi:10.1016/j.asoc.2024.111373.
44. Rao RV, Shah R. BMR and BWR: two simple metaphor-free optimization algorithms for solving real-life non-convex constrained and unconstrained problems. arXiv:2407.11149. 2024.
45. Kınalı NS, Yakak B, Dede T, Atmaca B. Optimum design of steel space truss roofs with BMR and BWR algorithms. In: Advanced engineering optimization through intelligent techniques. Singapore: Springer Nature; 2025. p. 75–85. doi:10.1007/978-981-96-8070-2_7.
46. Rao RV, Davim JP. Optimization of different metal casting processes using three simple and efficient advanced algorithms. Metals. 2025;15(9):1057. doi:10.3390/met15091057.
47. Rao RV, Davim JP. Single, multi-, and many-objective optimization of manufacturing processes using two novel and efficient algorithms with integrated decision-making. J Manuf Mater Process. 2025;9(8):249. doi:10.3390/jmmp9080249.
48. Rao RV, Kishorbhai JN. Fused deposition modeling process optimization using two simple advanced methods. In: Advanced engineering optimization through intelligent techniques. Singapore: Springer Nature; 2025. p. 43–51. doi:10.1007/978-981-96-8070-2_4.
49. Gandomi AH, Yang XS. Benchmark problems in structural optimization. In: Computational optimization, methods and algorithms. Berlin/Heidelberg, Germany: Springer; 2011. p. 259–81. doi:10.1007/978-3-642-20859-1_12.

50. Öztürk HT, Kahraman HT. Meta-heuristic search algorithms in truss optimization: research on stability and complexity analyses. Appl Soft Comput. 2023;145(1):110573. doi:10.1016/j.asoc.2023.110573.
51. 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. doi:10.1016/j.compstruc.2015.11.014.
52. Le DT, Bui DK, Ngo TD, Nguyen QH, Nguyen-Xuan H. A novel hybrid method combining electromagnetism-like mechanism and firefly algorithms for constrained design optimization of discrete truss structures. Comput Struct. 2019;212:20–42. doi:10.1016/j.compstruc.2018.10.017.
53. Tang H, Lee J. Chaotic enhanced teaching-based differential evolution algorithm applied to discrete truss optimization. Structures. 2023;49(5):730–47. doi:10.1016/j.istruc.2023.01.153.
54. Goodarzimehr V, Topal U, Das AK, Vo-Duy T. Bonobo optimizer algorithm for optimum design of truss structures with static constraints. Structures. 2023;50(6):400–17. doi:10.1016/j.istruc.2023.02.023.