INTERNATIONAL JOURNAL OF OPTIMIZATION IN CIVIL ENGINEERING
Int. J. Optim. Civil Eng., 2021; 11(2): 329-356

QUANTUM-BASED JELLYFISH SEARCH OPTIMIZER FOR STRUCTURAL OPTIMIZATION

A. Kaveh[*,†], K. Biabani Hamedani, M. Kamalinejad and A. Joudaki
[1]*School of Civil Engineering, Iran University of Science and Technology, Narmak, Tehran, Iran*

ABSTRACT

Jellyfish Search (JS) is a recently developed population-based metaheuristic inspired by the food-finding behavior of jellyfish in the ocean. The purpose of this paper is to propose a quantum-based Jellyfish Search algorithm, named Quantum JS (QJS), for solving structural optimization problems. Compared to the classical JS, three main improvements are made in the proposed QJS: (1) a quantum-based update rule is adopted to encourage the diversification in the search space, (2) a new boundary handling mechanism is used to avoid getting trapped in local optima, and (3) modifications of the time control mechanism are added to strike a better balance between global and local searches. The proposed QJS is applied to solve frequency-constrained large-scale cyclic symmetric dome optimization problems. To the best of our knowledge, this is the first time that JS is applied in frequency-constrained optimization problems. An efficient eigensolution method for free vibration analysis of rotationally repetitive structures is employed to perform structural analyses required in the optimization process. The efficient eigensolution method leads to a considerable saving in computational time as compared to the existing classical eigensolution method. Numerical results confirm that the proposed QJS considerably outperforms the classical JS and has superior or comparable performance to other state-of-the-art optimization algorithms. Moreover, it is shown that the present eigensolution method significantly reduces the required computational time of the optimization process compared to the classical eigensolution method.

Keywords: jellyfish search optimizer; quantum; structural optimization; dome structures; frequency constraints; optimal structural analysis.

Received: 20 March 2021; Accepted: 28 May 2021

[*]Corresponding author: School of Civil Engineering, Iran University of Science and Technology, Narmak, Tehran, Iran
[†]E-mail address: alikaveh@iust.ac.ir (A. Kaveh)

1. INTRODUCTION

The dynamic characteristics (i.e., natural vibration frequencies and mode shapes) are arguably the single most important property of a mechanical system affecting the dynamic behavior of the system [1]. For example, in a mechanical system with low-frequency vibrations, the dynamic response of the system is mainly a function of its fundamental frequency [2]. In such cases, the performance of the structure can be considerably improved by manipulating the selected frequency [3]. Structural optimization considering frequency constraints provides a systematic design approach for engineering designers to manipulate the dynamic characteristics of the structural systems in various ways. For example, in designing most space vehicles, to avoid the resonance phenomenon that causes the vibration failure, it is necessary to impose constraints on the natural frequency ranges of the designed vehicles. Since the early 1980s, some researchers have applied gradient-based optimization methods to the optimal design of structures with frequency constraints [4-10]. However, the structural optimization problems with frequency constraints are considered challenging optimization problems with highly nonlinear, non-convex, and multimodal search spaces [11]. Thus, gradient-based optimization methods may not be suitable for this type of optimization problem. On the other hand, metaheuristic optimization algorithms could be considered appropriate alternatives [12-19].

Metaheuristic optimization algorithms have been one of the most popular research areas in computer science for more than three decades. These approximate optimization techniques have been extensively applied to a wide variety of engineering optimization problems. This is because metaheuristics: (1) are easy to design and implement as compared to other optimization methods, (2) require no gradient information during the search, and (3) are not problem-specific [20]. Nature-inspired metaheuristics can be classified into four main categories based on the source of inspiration: evolution-based, physics-based, swarm-based, and human-based [20]. Recently, a novel swarm-based metaheuristic optimizer, named Jellyfish Search (JS), has been developed based on the food-finding behavior of jellyfish in the ocean [21]. Jellyfish move in the ocean in search of planktonic organisms such as fish eggs and larvae, phytoplankton, etc. The movement patterns of jellyfish in the ocean can be classified into two major types: (1) their following the ocean current and (2) their motions inside the jellyfish swarm. Therefore, two main phases are considered in JS. In the early stages of the search process, jellyfish tend to follow ocean currents in search of food, while as the search progresses, jellyfish tend to switch to passive and active motions inside the swarm. A time control mechanism is considered to govern the switching between these movement patterns. The first and second phases are designed to deal with the diversification and the intensification of the search, respectively. Our experimental results, which will be discussed in Section 6, indicate that the classical JS may easily get trapped in local optima due to the lack of diversification in the search space (exploration). Moreover, it seems that the trade-off between exploration and exploitation during the search should be balanced.

A large number of structural analyses are usually required to be performed in order to achieve optimal or near-optimal designs using metaheuristic algorithms. Thus, it may be very time-consuming or even impractical to solve large-scale structural optimization problems using metaheuristic algorithms. In structural optimization problems with frequency constraints, the structural analyses involve relatively large generalized eigenproblems to find the vibration characteristics of structures [22]. The dimensions of the involved matrices in the free-vibration eigenproblem of a structural system are proportional to the number of degrees of freedom. This indicates that the required computational time of the frequency-constrained optimization problems depends strongly on the size of the structure. Therefore, efficient eigensolution methods, which require less computational effort to solve a single free-vibration eigenproblem, could be very beneficial for frequency-constrained structural optimization problems, especially in the case of large-scale structures. Since the mid-2000s, the first author and his students have worked extensively on developing efficient methods for eigenvalue problems of symmetric, repetitive, and regular structures [23-29].

The main objective of this study is to propose a quantum-based Jellyfish Search algorithm, named Quantum JS (QJS), to solve structural optimization problems. Three main improvements made in the proposed QJS are (1) a quantum-based update rule to encourage the diversification in the search space, (2) a new boundary handling mechanism to avoid getting trapped in local optima, and (3) modifications of the time control mechanism to strike a better balance between global and local searches. The proposed QJS is applied to optimal design of cyclic symmetric dome structures with multiple frequency constraints. The required structural analyses are conducted by an efficient eigensolution method proposed by Kaveh et al. [30-32] for free vibration analysis of rotationally repetitive structures.

Section 2 reviews the classical Jellyfish Search (JS) algorithm. In Section 3, after pointing out the drawbacks of the classical JS, the proposed quantum-based version of JS (QJS) is outlined. In Section 4, the formulation of the truss optimization problem subject to frequency constraints is presented. In Section 5, the free vibration analysis of cyclic symmetric structures is presented. In Section 6, two large-scale cyclic symmetric dome structures are optimized to demonstrate the effectiveness and computational efficiency of the proposed method. Finally, the last section draws concluding remarks and provides possible extensions of this work.

2. JELLYFISH SEARCH (JS) OPTIMIZER[1]

Jellyfish Search (JS) optimizer is one of the most recent swarm-based metaheuristics developed by Chou and Truong [21]. The JS algorithm mimics the food-finding behavior of jellyfish in the ocean. Movement patterns of jellyfish in the ocean can be categorized into two main types: (1) following ocean currents to form jellyfish swarms, known as jellyfish blooms, and (2) motions inside swarms of jellyfish. Fig. 1 shows the movement patterns of jellyfish in the ocean. The JS optimizer takes into account both the diversification and the intensification of the search. Indeed, at the beginning of the search process, jellyfish follow ocean currents in search of food sources. As time goes by, jellyfish tend to switch to passive

1. The source code of the Jellyfish Search algorithm is available at https://www.researchgate.net/publication/343499745_Jellyfish_Search_Algorithm_Source_Code

and active motions inside the swarms of jellyfish for intensification. A time control mechanism is provided to govern the switching between these two types of motion. In the following subsection, the mathematical formulation of the classical JS algorithm is reviewed.

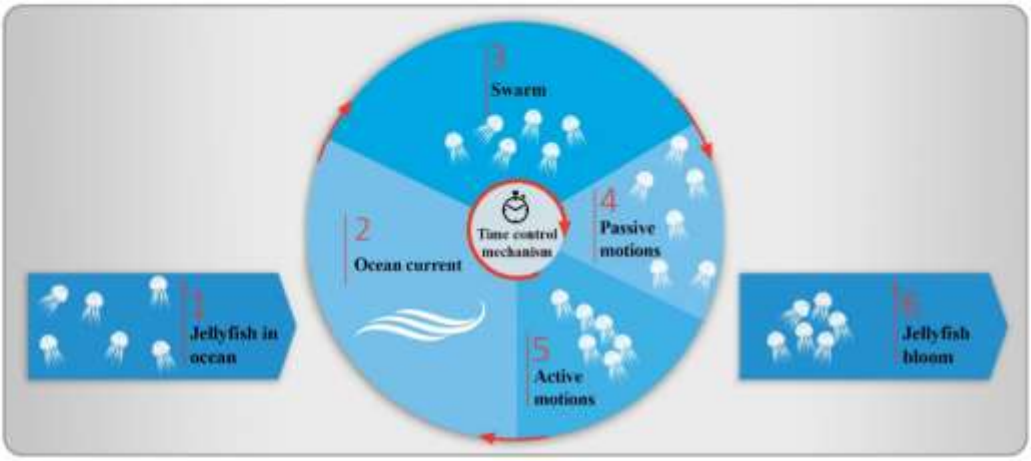

Figure 1. Movement patterns of jellyfish in the ocean

2.1 Mathematical model of the JS optimizer

The JS algorithm is designed based on the following three idealized rules [21]:

(1) Jellyfish either follow the ocean current or move inside the jellyfish swarm, and a time control mechanism is provided to govern the switching between these two types of motion.

(2) Jellyfish move around in the ocean to find food. Regions with greater availability of food sources are more likely to attract more significant numbers of jellyfish.

(3) Each solution is represented by a location, and its corresponding objective function is represented by the quantity of food found at the location.

2.1.1 Ocean current

Jellyfish can detect ocean currents and feed on smaller planktonic organisms such as fish eggs and larvae, phytoplankton, etc. The direction of the ocean current ($\overrightarrow{trend}$) is determined as follows:

$$\overrightarrow{trend} = \frac{1}{n_{Pop}} \sum \overrightarrow{trend_i} = \frac{1}{n_{Pop}} \sum (X^* - e_c X_i) = X^* - e_c \frac{\sum X_i}{n_{Pop}} = X^* - e_c \mu \tag{1}$$

where n_{Pop} is the number of jellyfish, $\overrightarrow{trend}$ is the direction of the ocean current, X^* is the location of the current best jellyfish of the swarm, μ is the mean location of all jellyfish, X_i is the location of the i-th jellyfish, and e_c is a factor governing the attraction.

Let $e_c\mu$ be denoted by df. Then, Eq. (1) can be rewritten as follows:

$$\overrightarrow{trend} = X^* - df \tag{2}$$

Assuming a normal distribution for the jellyfish's locations, a distance of $\pm\beta\sigma$ around the mean location has a higher likelihood of containing more jellyfish, where σ is the standard deviation of the normal distribution and β ($\beta > 0$) is a coefficient of distribution related to the length of $\overrightarrow{trend}$. If the standard deviation σ is assumed to be given by $\mu \times rand^{\sigma}(0,1)$, the following equation can be derived:

$$df = \beta \times \sigma \times rand^{f}(0,1) = \beta \times \mu \times rand^{\sigma}(0,1) \times rand^{f}(0,1) \tag{3}$$

To simplify the calculations, Eq. (3) is rewritten as follows:

$$df = \beta \times \mu \times rand(0,1) \tag{4}$$

By substituting Eq. (4) into Eq. (2), the following equation for $\overrightarrow{trend}$ is obtained:

$$\overrightarrow{trend} = X^* - \beta \times \mu \times rand(0,1) \tag{5}$$

The new location of each jellyfish can be obtained as follows:

$$X_i(t+1) = X_i(t) + rand(0,1) \times \overrightarrow{trend} \tag{6}$$

where $X_i(t+1)$ and $X_i(t)$ are the new location and the current location of the i-th jellyfish, respectively. After updating each jellyfish's location via Eq. (6), the better location (i.e., the location with greater availability of food sources) is taken as the jellyfish's current location.

2.1.2 Jellyfish bloom

Inside a jellyfish bloom, jellyfish exhibit two types of motion: passive motions (type A) and active motions (type B), between which the jellyfish switch. At the first stages of the search process (i.e., when the jellyfish bloom has just been formed), most jellyfish tend to exhibit type A motion, but type B motion is favored as time goes by. In the following two subsections, these two types of motion are modeled mathematically.

2.1.2.1 Passive motion (type A)

Type A motion is associated with the motion of jellyfish around their own current locations, with the aim of finding better locations. The new location of each jellyfish is determined as follows:

$$X_i(t+1) = X_i(t) + \gamma \times rand(0,1) \times (U_b - L_b) \tag{7}$$

where $X_i(t+1)$ and $X_i(t)$ are the new location and the current location of the i-th jellyfish, respectively, L_b and U_b are the lower and upper bounds of the search space, respectively, and γ ($\gamma > 0$) is a coefficient of motion related to the length of passive motion. After updating each jellyfish's location via Eq. (7), the better location (i.e., the location with greater availability of food sources) is taken as the jellyfish's current location.

Based on the result of sensitivity analysis carried out by Chou and Truong [21] to investigate the effectiveness of parameters β and γ, the JS optimizer can find the best optimal results when $\beta = 3$ and $\gamma = 0.1$.

2.1.2.2 Active motion (type B)

To simulate type B motion of each jellyfish i, a jellyfish j other than the one of interest is selected randomly (i.e., $i \neq j$). Then, the jellyfish i and j interact with each other to move toward locations with greater availability of food sources. To this end, the jellyfish i moves directly towards the jellyfish j if the quantity of food at the j-th jellyfish's location exceeds that of the i-th jellyfish. Otherwise, the jellyfish i moves directly away from the jellyfish j. So, each jellyfish moves toward a location with greater availability of food sources. Type B motion encourages diversification in the search space. Type B motion can be formulated as:

$$X_i(t+1) = X_i(t) + rand(0,1) \times \overrightarrow{Direction} \tag{8}$$

where

$$\overrightarrow{Direction} = \begin{cases} X_i(t) - X_j(t) & \text{if} \quad f(X_i) < f(X_j) \\ X_j(t) - X_i(t) & \text{if} \quad f(X_i) \geq f(X_j) \end{cases} \tag{9}$$

where $f(X_i)$ and $f(X_j)$ are the objective function values of the jellyfish i and j, respectively, $\overrightarrow{Direction}$ is the vector of the active motion. After updating each jellyfish's location via Eqs. (8) and (9), the better location (i.e., the location with greater availability of food sources) is taken as the jellyfish's current location.

2.1.3 Time control mechanism

The time control mechanism is provided to regulate the type of motion of jellyfish over time. It controls not only type A and type B motions of jellyfish inside the jellyfish bloom but also their movements toward ocean currents. To this end, the time control mechanism employs a threshold constant C_0 and a time control function $c(t)$. As can be seen from Eq. (10), the time control function is a random number that fluctuates between 0 and 1 but shows an overall decreasing trend over time, as shown in Fig. 2. If the value of the time control function exceeds C_0, the jellyfish follow the ocean current, whereas if it does not exceed C_0, the jellyfish move inside the jellyfish bloom. Chou and Truong recommended a value of 0.5 for C_0 [21]. The time control function is given by:

$$c(t) = \left| \left(1 - \frac{t}{MaxIter}\right) \times (2 \times rand(0,1) - 1) \right| \tag{10}$$

where t is the time index specified as the iteration number, and $MaxIter$ is the maximum iteration number.

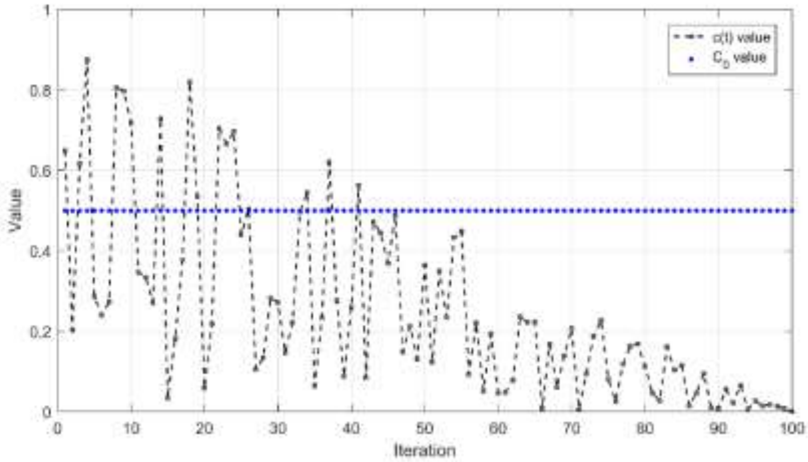

Fig. 2 Time control function with an overall decreasing trend

To regulate the jellyfish's movements inside a jellyfish bloom (type A and type B motions), the function $\left(1-c(t)\right)$ is used as follows: If $rand(0,1)$ exceeds $\left(1-c(t)\right)$, type A motion is favored. On the other hand, if $rand(0,1)$ is lower than $\left(1-c(t)\right)$, type B motion is favored. Since $\left(1-c(t)\right)$ shows an overall increasing trend from 0 to 1 overtime, the jellyfish tend to exhibit type A motion within the first stages of the search process, but as time passes, type B motion becomes more likely.

2.2 Population initialization

In the JS optimizer, instead of using a simple random initialization, the Logistic map is used to generate the initial population. The Logistic map is formulated as follows:

$$X_{i+1} = \eta X_i(1 - X_i), \qquad 0 \le X_0 \le 1 \tag{11}$$

where X_i is the chaotic logistic value for the i-th jellyfish's location and X_0 is a randomly generated location used for generating the initial population of jellyfish. Obviously, X_i is a number between 0 and 1 under the conditions that $X_0 \in [0,\ 1]$ and $X_0 \notin \{0.0, 0.25, 0.5, 0.75, 1.0\}$. The parameter η is set to 4 in all experiments.

2.3 Boundary handling mechanism

If a jellyfish exceeds the boundaries of the search space, it will be located within the boundaries through the following equation:

$$\begin{cases} X'_{i,d} = \left(X_{i,d} - U_{b,d}\right) + L_{b,d} & \text{if} \quad X_{i,d} > U_{b,d} \\ X'_{i,d} = \left(X_{i,d} - L_{b,d}\right) + U_{b,d} & \text{if} \quad X_{i,d} < L_{b,d} \end{cases} \tag{12}$$

where $X_{i,d}$ is the current location of the d-th dimension of the i-th jellyfish, $X'_{i,d}$ is the updated location of the d-th dimension of the i-th jellyfish after satisfying the boundary constraints of the search space, and $L_{b,d}$ and $U_{b,d}$ are the lower and upper bounds of the d-th dimension of the search space, respectively. Fig. 3 illustrates the boundary handling mechanism of Eq. (12). As the figure shows, if the lower bound of the d-th dimension of the search space is violated, the boundary handling mechanism will return its upper bound, and vice versa.

The pseudo-code of the classical JS algorithm is provided in Fig. 4.

Figure 3. Boundary handling mechanism

```
Pseudocode of Jellyfish Search algorithm
Initialization phase
Set the algorithm parameters: population size (n_Pop) and maximum iteration (MaxIter)
Generate the initial population X_i (i = 1,2,⋯,n_Pop) using Eq. (11)
Evaluate the initial population, f(X_i) (i = 1,2,⋯,n_Pop)
Determine the currently best jellyfish (X*)
Initialize time: t = 1
Cyclic body of the algorithm
While t ≤ MaxIter
   For i = 1: n_Pop
   Calculate the time control c(t) using Eq. (10)
      If c(t) ≥ 0.5
         Determine the direction of the ocean current by Eq. (5)
         Determine the new location of the i-th jellyfish by Eq. (6)
      Else
         If (1 − c(t)) < rand(0, 1)
         Determine the new location of the i-th jellyfish by Eq. (7)
         Else
         Determine the new location of the i-th jellyfish by Eqs. (8) and (9)
         End if
      End if
   Check boundary conditions using Eq. (12)
   Evaluate the new location of the i-th jellyfish
   Update the location of the i-th jellyfish (X_i)
   Update the location of the currently best jellyfish (X*)
   End for i
t = t + 1
End while
Output the best results
```

Figure 4. Pseudo-code of the classical JS algorithm

3. QUANTUM JELLYFISH SEARCH (QJS) ALGORITHM

During the early iterations of the classical JS, the dominant tendency of jellyfish is to follow ocean currents. As indicated before, the direction of the ocean current is determined based on the difference between the location of the current best jellyfish of the swarm and the mean location of all jellyfish of the swarm. Thus, in the early stages of the search process, the mean of the jellyfish swarm moves towards the location of the current best jellyfish of the swarm. However, it should be noted that in the early iterations, the current best jellyfish of the swarm is probably not a high-quality jellyfish. This may lead to undesirable premature convergence in the early stages of the search. The experimental results confirm that the classical JS suffers from premature convergence, especially when dealing with complex optimization problems. Therefore, in order to address this issue and improve the exploration capability of the classical JS, based on the work of Kaveh et al. [33], a quantum-based update rule is proposed for the exploration phase of the classical JS, as follows: the local attractor of each jellyfish of the swarm is determined by Eq. (13):

$$X_{i,d}^{t}(t) = X_{i,d}(t) \times rand1_{i,d}(0,1) + \left(1 - rand1_{i,d}(0,1)\right) \times X_{d}^{Best} \tag{13}$$

where X_i^t is the local attractor of the i-th jellyfish of the swarm, X_i is the current location of the i-th jellyfish of the swarm, X^{Best} is the best jellyfish found so far, and $rand1$ is a random number uniformly distributed on $(0,1)$. Eq. (13) indicates that X_i^t, the local attractor of the i-th jellyfish, lies on the line connecting X_i and X^{Best} so that it moves following X_i and X^{Best}.

The location of each jellyfish is updated according to the following equation:

$$\begin{aligned} X_{i,d}(t+1) = X_{i,d}^{t}(t) + (-1)^{randi([1,2])} \times \log\left(1/rand2_{i,d}(0,1)\right) \times \beta(t) \\ \times \left|\mu_d - X_{i,d}(t)\right| \end{aligned} \tag{14}$$

where

$$\beta(t) = 1 - \left(1 - rand3(0,1)\right) \times \left(\frac{t}{MaxIter}\right) \tag{15}$$

where $X_i(t+1)$ represents the new location and the current location of the i-th jellyfish, $rand2$ and $rand3$ are random numbers uniformly distributed on $(0,1)$, μ is the mean location of all jellyfish of the swarm, and $\beta(t)$ controls the convergence of the jellyfish toward the best jellyfish found so far.

In the classical JS, if a solution exceeds the boundary of the search space, the boundary handling mechanism of Eq. (12) brings it back to the opposite bound, as seen from Fig. 3. Such a boundary handling mechanism may cause difficulties in the convergence process. Indeed, it has been recognized that the optimal solutions often lie close to (or even on) the boundary of the search space [34]. Thus, during the optimization process, especially in the final stages, many solutions, which are probably close to optimal, are likely to move out of the boundaries of the search space. However, the boundary handling mechanism of Eq. (12) significantly alters the values of the design variables exceeding their corresponding bounds. As a consequence of such a boundary handling mechanism, some potentially good solutions, which slightly exceed the boundary of the search space, may be lost during the search process. in the proposed QJS, in order to address this issue, we utilize a simple boundary handling mechanism, as follows:

$$\begin{cases} X'_{i,d} = U_{b,d} & \text{if} \quad X_{i,d} > U_{b,d} \\ X'_{i,d} = L_{b,d} & \text{if} \quad X_{i,d} < L_{b,d} \end{cases} \tag{16}$$

If a solution exceeds the boundary of the search space, the boundary handling mechanism of Eq. (16) brings it back to the violated bound.

Fig. 2 shows a typical trend of the time control function of Eq. (10). As can be seen from the figure, in only 16 out of 100 iterations (16%), the time control function value is greater than the threshold constant $C_0 = 0.5$. Hence, in only 16 iterations, mainly from the first ones (i.e., before the 42 th iteration), jellyfish follow the ocean current, while, in the other 84 iterations, jellyfish exhibit passive and active motions inside the swarm. As mentioned before, following ocean currents encourages diversification in the search space (i.e., global exploration), while exploitation of the search takes place through passive and active motions inside jellyfish swarms. As a result, classical JS seems to suffer from the lack of global exploration of the search space and focuses mainly on local exploitation of the best solutions found. In the proposed QJS, in order to address this issue and achieve a better balance between diversification and intensification of the search process, a simple linear time control mechanism is proposed as follows:

$$c(t) = \left|\left(1 - \frac{t}{MaxIter}\right)\right| \tag{17}$$

Such a deterministic definition of time control mechanism leads to a simple trade-off between intensification and diversification during the search. The first half of iterations (i.e., when $c(t) \geq 0.5$) are dedicated to the global exploration of the search space, while the second half (i.e., when $c(t) < 0.5$) deals with local exploitation of the best solutions found.

In the classical JS, passive (type A) motion is associated with the motion of jellyfish around their current locations, with the aim to find better locations. Each jellyfish, whether good or bad, exploits its own neighborhood, which may cause difficulties such as slow convergence rate and easily getting trapped in local optima. In the proposed QJS, in order to enhance the exploitation capability of the classical JS and speed up its convergence rate without loss of diversity, passive (type A) motion is associated with the motion of jellyfish around the location of the best jellyfish found so far, and is defined as follows:

$$X_i(t+1) = X^{Best} + (-1)^{randi([1,2])} \times rand(0,1) \times (U_b - L_b) \tag{18}$$

where X^{Best} is the location of the best jellyfish found so far and $randi([1,2])$ returns a pseudorandom scalar integer between 1 and 2. The term $randi([1,2])$ allows to explore the whole neighborhood of the best jellyfish found so far.

Fig. 5 shows the pseudo-code of the Quantum JS (QJS) algorithm.

Pseudo-code of Quantum Jellyfish Search algorithm

Initialization phase

Set the algorithm parameters: population size (n_{Pop}) and maximum iteration ($MaxIter$)
Generate the initial population X_i ($i = 1,2,\cdots,n_{Pop}$) using Eq. (11)
Evaluate the initial population, $f(X_i)$ ($i = 1,2,\cdots,n_{Pop}$)
Determine the currently best jellyfish (X^*)
Initialize time: $t = 1$

Cyclic body of the algorithm

While $t \le MaxIter$
Calculate the time control $c(t)$ using Eq. (17)
If $c(t) \ge 0.5$
Determine the value of $\beta(t)$ using Eq. (15)
For $i = 1{:}\, n_{Pop}$
Determine the local attractor of of the i-th jellyfish by Eq. (13)
Check boundary conditions using Eq. (16)
Determine the new location of the i-th jellyfish by Eq. (14)
End for i
Else
If $(1 - c(t)) < rand(0,1)$
For $i = 1{:}\, n_{Pop}$
Determine the new location of the i-th jellyfish by Eq. (18)
End for i
Else
For $i = 1{:}\, n_{Pop}$
Determine the new location of the i-th jellyfish by Eqs. (8) and (9)
End for i
End if
End if
For $i = 1{:}\, n_{Pop}$
Check boundary conditions using Eq. (16)
Evaluate the new location of the i-th jellyfish
Update the location of the i-th jellyfish (X_i)
Update the location of the currently best jellyfish (X^*)
End for i
$t = t + 1$
End while
Output the best results

Figure 5. Pseudo-code of the QJS algorithm

4. MATHEMATICAL FORMULATION OF THE OPTIMIZATION PROBLEM

In a truss sizing optimization problem with frequency constraints, the aim is to minimize the total weight of the structure while satisfying some constraints on natural vibration frequencies. The cross-sectional areas of structural members are considered as continuous design variables. The layout of the structure is pre-defined and kept unchanged during the optimization process. The mathematical formulation of the optimization problem is as follows [1]:

$$\text{Find } \{X\} = [x_1, x_2, \dots, x_{nDV}] \tag{19}$$

$$\text{to minimize } P(\{X\}) = f(\{X\}) \times f_{penalty}(\{X\}) \tag{20}$$

$$\text{subject to: } \begin{cases} \omega_j \geq \omega_j^* & \text{for some natural vibration frequencies } j \\ \omega_k \leq \omega_k^* & \text{for some natural vibration frequencies } k \\ L_{b,i} \leq x_i \leq U_{b,i} & i = 1,2,\dots,nDV \end{cases} \tag{21}$$

where $\{X\}$ denotes the vector of design variables, including sizing design variables, nDV is the number of design variables, which is selected considering the member-grouping configuration, x_i is the cross-sectional area of the structural members of the i-th member group, $f(\{X\})$ is the objective function of the optimization problem to be minimized, which represents the total weight of the structure in a weight minimization problem, $f_{penalty}(\{X\})$ is the penalty function which is used to handle the problem constraints, and $P(\{X\})$ is the penalized objective function. $L_{b,i}$ and $U_{b,i}$ are the lower and upper bounds of the cross-sectional area of the structural members of the i-th member group, respectively, ω_j and ω_k are the j-th and the k-th natural vibration frequencies of the structure, respectively, ω_j^* is the lower bound of the j-th natural vibration frequency of the structure, and ω_k^* is the upper bound of the k-th natural vibration frequency of the structure. The objective function is considered to be the total weight of the structure and can be defined as follows:

$$f(\{X\}) = W(\{X\}) = \sum_{i=1}^{nE} \rho_i \times A_i \times L_i \tag{22}$$

where ρ_i, A_i, and L_i are the material density, cross-sectional area, and length of the i-the structural member, respectively, nE is the number of structural members of the structure, and $W(\{X\})$ is the total weight of the structure.

Various strategies have been suggested to handle constraints in optimization problems, one of the most popular of which is penalizing strategies. The main idea of penalizing strategies is to transform a constrained optimization problem into an unconstrained one by penalizing the infeasible solution and extending an unconstrained objective function [35]. Here, a dynamic penalty function is used to tackle the violated constraints [36]:

$$f_{penalty}(X) = (1 + \varepsilon_1 \times \upsilon)^{\varepsilon_2}, \qquad \upsilon = \sum_{i=1}^{nC} \upsilon_i \tag{23}$$

where nC is the number of constraints of the problem, ε_1 and ε_2 are the penalty parameters that affect the severity of violated constraints, and υ denotes the sum of the constraint violations. The value of υ_i is set to zero if the i-th constraint is satisfied, while in the case of a violated constraint, it is selected considering the severity of the violation. The mathematical expression of υ_i is as follows:

$$\upsilon_i = \begin{cases} \left|1 - \dfrac{\omega_i}{\omega_i^*}\right| & \text{if the } i-\text{th frequency constraint is violated} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

Dynamic penalty functions take into account the progress of the optimization process so that penalty is imposed at a dynamic or increasing rate [35]. This means that a low degree of penalty is imposed at the beginning of the search process. However, as the search process progresses, the degree of the penalty also gradually increases [37]. Such a dynamic strategy encourages the diversification in the search space (i.e., more exploration) in the early iterations of the optimization process, but more emphasis on the intensification of the best solutions found (i.e., more exploitation) in the last iterations [22].

The parameters ε_1 and ε_2 control how much an infeasible solution is penalized. The severity of penalizing is very sensitive to these parameters. Hence, setting the parameters ε_1 and ε_2 is a challenging task and requires many preliminary trials [38]. If they are chosen too small, feasible regions of search space may not be explored effectively, and even the algorithm may never converge to a feasible solution. On the other hand, if they are too large, premature convergence may occur [39]. In this study, a constant value for the parameter ε_1 is chosen, whereas the parameter ε_2 increases monotonically with the number of iterations.

5. FREE VIBRATION ANALYSIS OF CYCLIC SYMMETRIC STRUCTURES

Free vibration means the motion of a structure without any externally applied vibration forcing [40]. Vibration characteristics (i.e., natural vibration frequencies and mode shapes) play an essential role in the dynamic analysis of structures [41]. Determining the vibration characteristics of an undamped structure requires the solution of the following algebraic equation, known as the matrix eigenvalue problem [42]:

$$K\phi_i = \gamma_i M\phi_i, \qquad i = 1,2,\cdots,N \tag{25}$$

where K is the elastic stiffness matrix of the structure (hereafter called stiffness matrix of the structure), M is the mass matrix of the structure, which is a linear combination of structural and non-structural mass matrices, ϕ_i is the i-th natural mode shape of vibration of the structure corresponding to the i-th eigenvalue (γ_i), and N is the number of degrees of freedom of the structure. The i-th natural frequency of vibration (ω_i) and its corresponding natural period of vibration (T_i) are related to the i-th eigenvalue as follows:

$$\gamma_i = \omega_i^2 = (2\pi/T_i)^2, \qquad i = 1,2,\cdots,N \tag{26}$$

There are many different methods to solve eigenvalue problems [43]. However, it should be noted that there is no single method that can always give a very efficient solution to every eigenvalue problem. Classical eigensolution methods do not take advantage of the potentially beneficial properties of the matrices involved in eigenvalue problems (i.e., the matrices K and M) and thus deal with large-dimension matrices. As a result, the required computational effort of these methods depends strongly on the size of the structure (i.e., the number of degrees of freedom) [22]. Hence, it is time-consuming and inefficient to solve the generalized eigenvalue problem given by Eq. (25) with classical eigensolution methods, especially for large-scale structures. However, in the case of general structures, it is inevitable to use classical eigensolution methods. On the other hand, the design optimization process of a structure with frequency constraints usually requires many free vibration analyses to be carried out. As mentioned before, the mathematical formulation of free vibration analysis of structures leads to the generalized eigenvalue problem given by Eq. (25). The most time-consuming part of frequency constraint optimization problems is usually the solution of eigenvalue problems [44]. As a result, design optimization of large-scale structures with frequency constraints could not be performed using the classical eigensolution methods in a reasonable time. Consequently, alternative efficient eigensolution methods, which take the maximum advantage of the properties of the matrices involved in eigenvalue problems in order to decrease the required computational time and memory, should be considered. The global stiffness matrix and mass matrix of a cyclic symmetric structure in the cylindrical coordinate system exhibit a unique pattern known as block circulant [45]. Circulant matrices can be expressed as the sum of Kronecker products in which the first components satisfy the commutative property of multiplication [30]. This property facilitates the block diagonalization of circulant matrices. Therefore, using this property of block circulant matrices, the initial generalized eigenvalue problem, derived from the free vibration analysis, is decomposed into highly smaller sub-eigenproblems [32]. This approach leads to not only the high accuracy of the free vibration analysis results but also a significant decrease in computational time as compared to classical eigenvalue solutions [45]. The details on the efficient eigensolution method can be found in [30-32].

6. CASE STUDIES

In this study, two cyclic symmetric dome optimization examples are considered to demonstrate the validity, efficiency, and accuracy of the present method for free vibration analysis of rotationally repetitive structures and the proposed QJS for solving sizing optimization of dome structures with multiple frequency constraints. These domes are taken from Kaveh et al. [45]. The first example is the sizing optimization of a 600-bar single-layer dome structure with 25 design variables. The second example is the sizing optimization of a 1410-bar double-layer dome structure with 47 design variables. Table 1 lists the material properties, cross-sectional area bounds, and frequency constraints of all examples. The results of QJS are compared with those of JS and other optimization methods reported in the literature. Moreover, the present efficient eigensolution method is compared in terms of accuracy and computational time with the existing classical eigensolution method. The optimization results are reported in terms of the best weight, average weight, worst weight, and standard deviation. The maximum number of objective function evaluations is taken as the termination criterion of the optimization process. In both examples, the population size n_{Pop} is chosen to be 20 for both the JS and the QJS, and the maximum number of iterations ($MaxIter$) is set to 1000. To consider the stochastic nature of the optimization process, ten independent runs are performed for each problem, and the optimal design results of the best run are reported. The finite element models and the optimization codes are implemented in the Matlab environment. It is noted that the optimizations are performed on a PC with Windows 10, Intel(R) Core (TM) i5-7200U CPU 2.50 GHz 2.71 GHz, and 8.00 GB RAM.

Table 1: Material properties, cross-sectional area bounds, and frequency constraints of investigated examples

Problem	Elasticity modulus E (N/m^2)	Material density ρ (kg/m^3)	Cross-sectional area bounds (m^2)	Frequency constraints (Hz)
600-bar dome-like truss	2×10^{11}	7850	$0.0001 \le A_i \le 0.01$	$\omega_1 \ge 5$, $\omega_3 \ge 7$
1410-bar dome-like truss	2×10^{11}	7850	$0.0001 \le A_i \le 0.01$	$\omega_1 \ge 7$, $\omega_3 \ge 9$

6.1 The 600-bar single-layer dome structure

The first design problem solved in this study is the 600-bar single-layer dome shown in Figs. 6 and 7. The entire structure is comprised of 216 nodes and 600 elements and could be generated by the cyclic repetition of a sub-structure with 9 nodes and 25 elements around the cyclic symmetry axis of the structure. Fig. 8 shows the details of a typical sub-structure, including nodal numbering. The angle of cyclic symmetry is equal to 15 degrees, which results in a total of 24 similar sub-structures. Table 2 summarizes the nodal coordinates of the first sub-structure in the Cartesian coordinate system. The connectivity information of the first sub-structure is also given in Table 3. The cross-sectional area of each element of the sub-structure is considered as an independent sizing design variable. However, the layout of the structure is kept unchanged during the optimization process. Therefore, this is a sizing optimization problem with 25 design variables. A non-structural mass of 100 kg is attached at all free nodes of the dome. As Table 1 shows, the frequency constraints are imposed on the first and third natural vibration frequencies. This problem was previously studied by different researchers using different metaheuristic optimization algorithms [15, 46-50].

Table 3 provides a comparison of the optimization results obtained by JS, QJS, and other referenced metaheuristics [15, 46-50]. It can be seen that the best and average optimized weights obtained by QJS over the ten independent runs are lighter than those corresponding to the best designs of all other considered optimization methods. Furthermore, it is clear that QJS significantly outperforms the classical JS in terms of optimized weight and standard deviation on optimized weight. QJS achieved a feasible optimized design corresponding to a weight of 6065.503 kg, 1.65% lighter than the optimum design found by JS (i.e., 6166.965 kg). Moreover, QJS found an average optimized weight of 6077.634 kg over ten independent runs, 19.50% lighter than the average weight of JS (i.e., 7549.676 kg). QJS required only 8820 analyses to find a feasible design corresponding to a structural weight of 6079.643 kg, which is lighter than the best weights found by all other referenced optimization methods [15, 46-50]. As can be seen from the table, there is a significant difference between the average weight and best weight values of the classical JS. This is because the classical JS easily gets trapped in local optima in some cases. In the present example, in 9 out of 10 optimization runs, the classical JS has been trapped in local optima far from the global optimum. Fig. 9 compares the average-weight convergence histories of the classical JS and QJS. As can be observed from the figure, the classical JS shows a very fast convergence rate in the early iterations of the search process, which may result in a premature loss of diversity in the population, and a premature convergence could occur. On the other hand, because of a better diversification of the search, especially in the early iterations, QJS has a high probability of escaping from local optima. Table 4 provides the first five natural frequencies of the optimized designs obtained by JS, QJS, and other referenced metaheuristics [15, 46-50]. It can be seen that the optimized designs of JS and QJS satisfy all frequency constraints.

Table 8 provides a comparison of computational efficiency between the classical and the proposed efficient eigensolution method. As can be seen from the table, through the efficient eigensolution method, instead of solving a direct eigenvalue problem of order 576, we need to find the eigenvalues for 24 matrices of order 24 (8 free nodes of sub-structure). This leads to a considerable saving in computational time. In fact, in the case of the 600-bar single-layer dome, the average computational time of the classical eigensolution method for free vibration analysis is calculated 0.0363 sec, which is more than six times the computational time required by the efficient eigensolution method (0.0054 sec). Furthermore, we have estimated that 7253.9406 sec (about 121 min) would be required to perform ten independent runs of the proposed QJS using the classical eigensolution method. However, using the efficient eigensolution method, the same algorithm requires only 1075.9423 sec (about 18 min) to perform these runs.

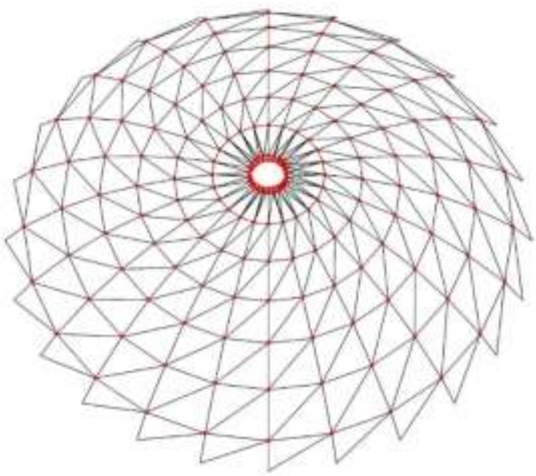

Figure 6. Schematic of the 600-bar single-layer dome

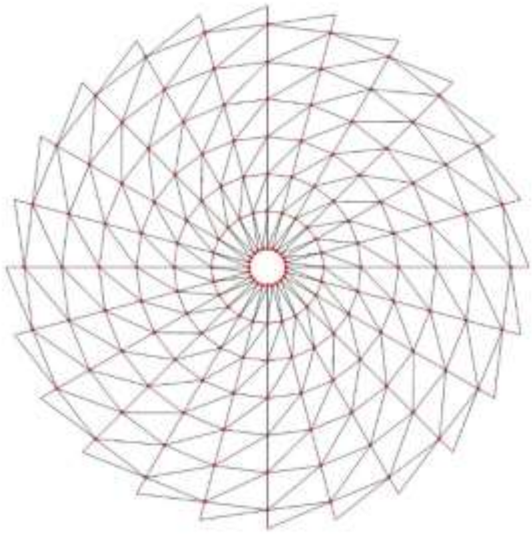

Figure 7. The 600-bar single-layer dome (top view)

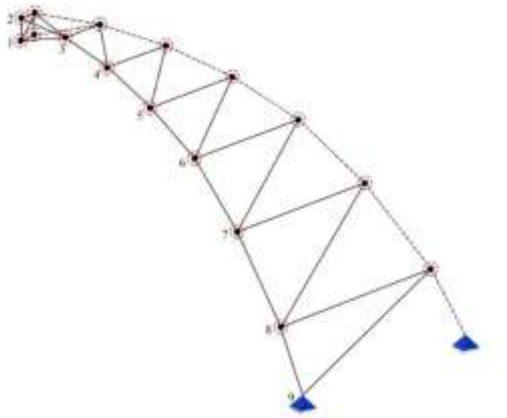

Figure 8. Details of a sub-structure of the 600-bar single-layer dome

Table 2: Nodal coordinates (m) of the sub-structure of the 600-bar single-layer dome

Node number	Coordinates (x, y, z)
1	(1.0, 0.0, 7.0)
2	(1.0, 0.0, 7.5)
3	(3.0, 0.0, 7.25)
4	(5.0, 0.0, 6.75)
5	(7.0, 0.0, 6.0)
6	(9.0, 0.0, 5.0)
7	(11.0, 0.0, 3.5)
8	(13.0, 0.0, 1.5)
9	(14.0, 0.0, 0.0)

Table 3: Comparison of optimal results of the 600-bar dome obtained by different algorithms (cm^2)

Element number (element nodes)	CBO [15]	DPSO [46]	VPS [47]	ECBO-Cascade [48]	MDVC-UPVS [49]	PFJA [50]	This study	
							JS	QJS
1 (1-2)	1.2404	1.365	1.3030	1.0299	1.2575	1.1867	**1.3667**	**1.2623**
2 (1-3)	1.3797	1.391	1.3998	1.3664	1.3466	1.2967	**1.3336**	**1.4105**
3 (1-10)	5.2597	5.686	5.1072	5.1095	4.9738	4.5771	**12.2413**	**5.1157**
4 (1-11)	1.2658	1.511	1.3882	1.3011	1.4025	1.3356	**5.1191**	**1.3939**
5 (2-3)	17.2255	17.711	16.9217	17.0572	17.3802	18.3157	**16.4818**	**17.5568**
6 (2-11)	38.2991	36.266	38.1432	34.0764	37.9742	38.5097	**33.5000**	**34.5863**
7 (3-4)	12.2234	13.263	11.8319	13.0985	13.0306	13.5917	**13.1109**	**13.0500**
8 (3-11)	15.4712	16.919	16.6149	15.5882	15.9209	16.8824	**15.8066**	**14.9897**
9 (3-12)	11.1577	13.333	11.3403	12.6889	11.9419	13.8766	**10.8633**	**11.3361**
10 (4-5)	9.4636	9.534	9.3865	10.3314	9.1643	9.5286	**10.5474**	**9.1993**
11 (4-12)	8.8250	9.884	8.7692	8.5313	8.4332	9.4218	**8.0920**	**8.3409**
12 (4-13)	9.1021	9.547	9.6682	9.8308	9.2375	9.7643	**9.7763**	**9.2362**
13 (5-6)	6.8417	7.866	6.9826	7.0101	7.2213	7.2431	**6.8615**	**7.5831**
14 (5-13)	5.2882	5.529	5.4445	5.2917	5.2142	5.3913	**5.3937**	**5.3152**
15 (5-14)	6.7702	7.007	6.3247	6.2750	6.7961	6.7468	**6.4819**	**6.5682**
16 (6-7)	5.1402	5.462	5.1349	5.4305	5.2078	5.1493	**4.8695**	**4.8128**
17 (6-14)	5.1827	3.853	3.3991	3.6414	3.4586	3.8342	**3.2424**	**3.5015**
18 (6-15)	7.4781	7.432	7.7911	7.2827	7.6407	8.0665	**7.4539**	**7.6773**
19 (7-8)	4.5646	4.261	4.4147	4.4912	4.3690	4.2800	**4.5403**	**4.2587**
20 (7-15)	1.8617	2.253	2.2755	1.9275	2.1237	2.2509	**2.3723**	**2.1748**
21 (7-16)	4.8797	4.337	4.9974	4.6958	4.5774	4.5372	**5.0763**	**4.7066**
22 (8-9)	3.5065	4.028	4.0145	3.3595	3.4564	3.5615	**3.9347**	**3.8047**
23 (8-16)	2.4546	1.954	1.8388	1.7067	1.7920	1.7744	**1.9255**	**1.9187**
24 (8-17)	4.9128	4.709	4.7965	4.8372	4.8264	4.6445	**4.5980**	**4.7502**
25 (9-17)	1.2324	1.410	1.5551	2.0253	1.7601	1.6141	**1.5421**	**1.5567**
Best weight (kg)	6182.01	6344.55	6133.02	6140.51	6115.10	6333.251	**6166.965**	**6065.503 [6079.643]**[2]
Average weight (kg)	6226.37	6674.71	6142.03	6175.33	6119.95	6380.31	**7549.676**	**6077.634**
Worst weight (kg)	-	-	-	-	-	-	**7798.256**	**6094.435**
Standard deviation (kg)	60.12	473.21	12.54	34.08	16.23	47.396	**463.445**	**9.356**
Maximum Number of FE analyses	20000	9000	30000	20000	18000	25000	**20000**	**20000**

Table 4: Natural frequencies (Hz) of the optimal designs for the 600-bar dome

Frequency number	CBO [15]	DPSO [46]	VPS [47]	ECBO-Cascade [48]	MDVC-UPVS [49]	PFJA [50]	This study	
							JS	QJS
1	5.000	5.000	5.0000	5.001	5.000	5.0011	**5.0097**	**5.0008**
2	5.000	5.000	5.0003	5.001	5.000	5.0011	**5.0097**	**5.0008**
3	7.000	7.000	7.0000	7.001	7.000	7.0000	**7.0017**	**7.0001**
4	7.000	7.000	7.0001	7.001	7.000	7.0000	**7.0017**	**7.0001**
5	7.001	7.000	7.0002	7.002	7.000	7.0000	**7.0017**	**7.0003**

2. QJS found a feasible optimum design corresponding to a structural weight of 6079.643 kg after 8820 analyses.

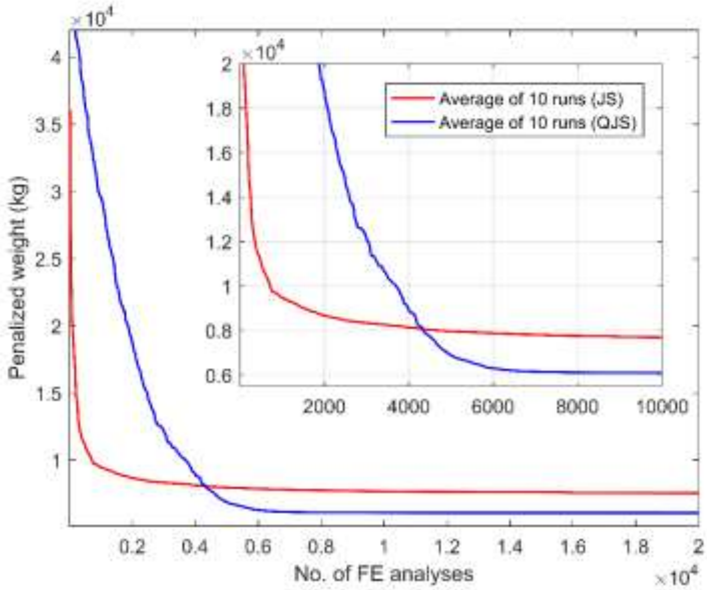

Figure 9. Average weight convergence histories for the 600-bar single-layer dome

6.2 The 1410-bar double-layer dome structure

The second design example considered in this study is the size optimization of the 1410-bar double-layer dome structure shown in Figs. 10 and 11. The entire structure is comprised of a total of 390 nodes and 1410 elements and could be generated by the cyclic repetition of the sub-structures shown in Fig. 12 around the cyclic symmetry axis of the structure. As can be seen from the figure, each sub-structure has 13 nodes and 47 elements. The angle of cyclic symmetry is equal to 12 degrees, which results in a total of 30 similar sub-structures. The Cartesian coordinates of the nodes of the first sub-structure are listed in Table 5. The connectivity information of the first sub-structure is given in Table 6. A non-structural mass of 100 kg is attached to all free nodes of the dome. The cross-sectional area of each element of the sub-structure represents a continuous sizing design variable of the problem. Therefore, this is a sizing optimization problem with 47 design variables. As Table 1 shows, the frequency constraints are imposed on the first and third natural vibration frequencies of the dome. This structure was previously optimized with different metaheuristic optimization algorithms [22, 46, 48-52].

Table 6 compares the optimization results obtained by JS, QJS, and other referenced optimization methods [22, 46, 48-52]. It can be seen that the best and average optimized weights obtained by QJS over the ten independent runs are lighter than those corresponding to the best designs of all other considered optimization methods. Furthermore, it is clear that QJS significantly outperforms the classical JS in terms of optimized weight and standard deviation on optimized weight. QJS achieved a feasible optimized design corresponding to a weight of 10278.571 kg, 3.61% lighter than the optimum design found by JS (i.e., 10663.092 kg). Moreover, QJS found an average optimized weight of 10369.689 kg over ten independent runs, 8.14% lighter than the average weight of JS (i.e., 11288.002 kg). QJS required only 12520 analyses to find a feasible design corresponding to a structural weight of 10319.228 kg, which is lighter than the best weights found by all other referenced optimization methods [22, 46, 48-52]. As can be seen from the table, there is a significant difference between the average weight and best weight values of the classical JS. This is because the classical JS easily gets trapped in local optima in some cases. Fig. 13 compares the average-weight convergence histories of the classical JS and QJS. As can be observed from the figure, the classical JS shows a very fast convergence rate in the early iterations of the search process, which may result in a premature loss of diversity in the population, and a premature convergence could occur. On the other hand, because of a better diversification of the search, especially in the early iterations, QJS has a high probability of escaping from local optima. Table 7 provides the first five natural frequencies of the optimized designs obtained by JS, QJS, and other referenced metaheuristics [22, 46, 48-52]. It can be seen that the optimized designs of JS and QJS satisfy all frequency constraints.

Table 8 provides a comparison of computational efficiency between the classical and the proposed efficient eigensolution method. As can be seen from the table, through the presented eigensolution method, instead of solving a direct eigenvalue problem of order 1080, we need to find the eigenvalues for 30 matrices of order 36 (12 free nodes of sub-structure). This leads to a considerable saving in computational time. In fact, in the case of the 1410-bar double-layer dome, the average computational time of the classical eigensolution method for free vibration analysis is calculated 0.1797 sec, which is more than 12 times the computational time required by the efficient eigensolution method (0.0140 sec). Furthermore, we have estimated that 35941.3224 sec (about 599 min) would be required to perform ten independent runs of the proposed QJS using the classical eigensolution method. However, using the efficient eigensolution method, the same algorithm requires only 2806.7635 sec (about 47 min) to perform these runs.

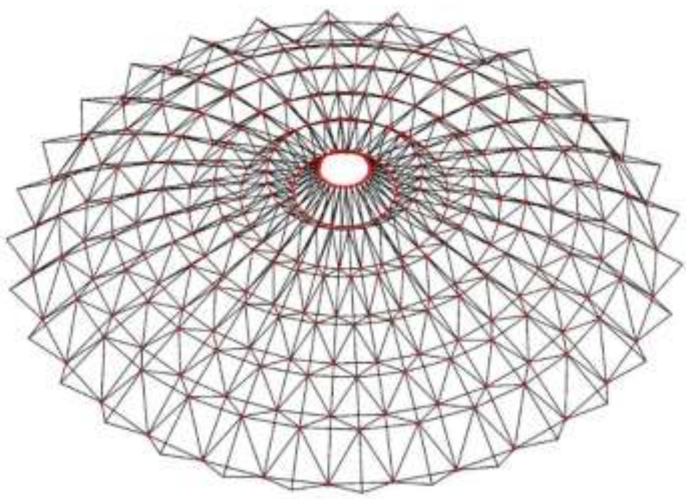

Figure 10. Schematic of the 1410-bar double-layer dome

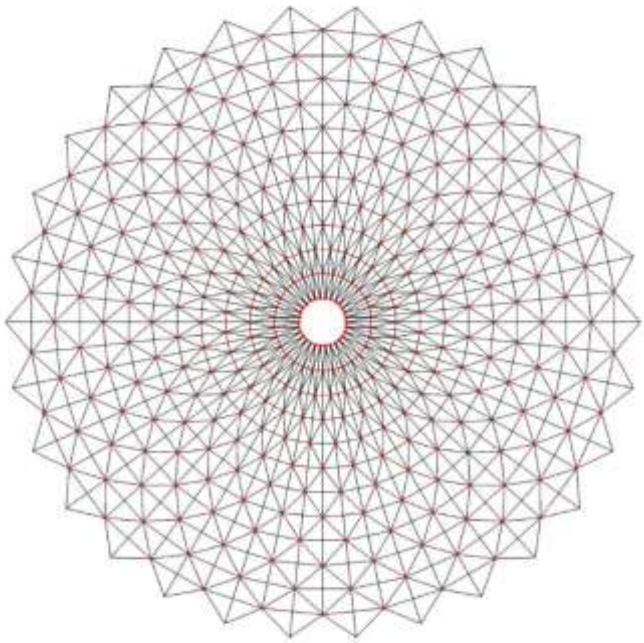

Figure 11. The 1410-bar double-layer dome (top view)

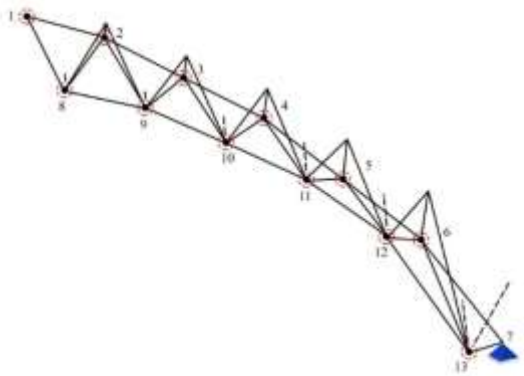

Fig. 12 Details of a sub-structure of the 1410-bar double-layer dome

Table 5: Coordinates (m) of the nodes of the 1410-bar double-layer dome

Node number	Coordinates (x, y, z)	Node number	Coordinates (x, y, z)
1	(1.0, 0.0, 4.0)	8	(1.989, 0.209, 3.0)
2	(3.0, 0.0, 3.75)	9	(3.978, 0.418, 2.75)
3	(5.0, 0.0, 3.25)	10	(5.967, 0.627, 2.25)
4	(7.0, 0.0, 2.75)	11	(7.956, 0.836, 1.75)
5	(9.0, 0.0, 2.0)	12	(9.945, 1.0453, 1.0)
6	(11.0, 0.0, 1.25)	13	(11.934, 1.2543, -0.5)
7	(13.0, 0.0, 0.0)		

Table 6: Comparison of optimal results of the 1410-bar dome obtained by different algorithms (cm^2)

Element number (element nodes)	CPA [22]	DPSO [46]	ECBO-Cascade [48]	MDVC-UPVS [49]	PFJA [50]	CRPSO [51]	MJA [52]	This study	
								JS	QJS
1 (1-2)	7.416	7.209	7.9969	5.8499	6.1902	2.5000	7.3465	**5.7165**	**7.1979**
2 (1-8)	4.768	5.006	6.1723	4.5115	4.4036	6.0000	4.2998	**4.9880**	**5.5897**
3 (1-14)	38.993	38.446	35.5011	19.4823	31.2253	18.0000	31.8485	**27.1197**	**34.8424**
4 (2-3)	8.966	9.438	10.2510	8.8480	8.4715	9.5000	8.8767	**7.8185**	**8.9950**
5 (2-8)	4.511	4.313	5.3727	5.0084	4.8590	6.0000	4.9778	**4.8252**	**5.2544**
6 (2-9)	1.544	1.494	1.3488	1.3568	1.5759	1.0000	1.7469	**4.9369**	**1.1174**
7 (2-15)	8.371	8.455	11.4427	17.4331	12.9451	29.5000	11.6099	**16.5200**	**12.2753**
8 (3-4)	9.276	9.488	9.7157	9.1098	9.3263	8.0000	9.2972	**7.8311**	**8.8234**
9 (3-9)	3.583	3.480	1.3005	2.8712	3.2716	2.0000	3.3406	**2.5401**	**2.6118**
10 (3-10)	3.476	3.495	2.5046	3.5473	3.2878	1.5000	3.2006	**3.8229**	**2.5921**
11 (3-16)	15.531	16.037	10.7849	12.3768	12.6719	1.0000	12.1131	**11.0850**	**9.3809**
12 (4-5)	10.285	9.796	10.1954	10.1099	10.0979	7.5000	9.7121	**11.0365**	**9.1070**
13 (4-10)	2.497	2.413	2.2300	2.5797	2.5803	1.0000	2.5294	**2.2098**	**2.2585**
14 (4-11)	5.397	5.681	5.1186	5.8381	5.3769	6.0000	5.8102	**6.9714**	**5.6850**
15 (4-17)	16.503	15.806	14.0053	13.6402	16.0581	14.5000	16.5566	**16.0879**	**14.6390**
16 (5-6)	8.193	8.078	8.9713	9.9096	8.6789	9.0000	8.3162	**8.3039**	**9.3749**
17 (5-11)	3.829	3.931	4.0756	3.6543	3.3199	1.0000	3.2415	**4.3745**	**3.4546**
18 (5-12)	6.151	6.099	5.9211	6.1529	6.4966	8.0000	6.4539	**5.7235**	**6.7717**
19 (5-18)	10.465	10.771	10.6915	11.2448	10.8804	19.5000	10.7040	**6.1606**	**11.9384**
20 (6-7)	13.925	13.775	10.6220	13.1071	14.0056	16.5000	13.8031	**12.1901**	**13.1518**
21 (6-12)	4.415	4.231	4.5064	5.2361	5.0843	5.0000	5.0161	**4.4068**	**5.4571**
22 (6-13)	6.863	6.995	8.4086	7.0691	6.9952	9.0000	7.6509	**6.5886**	**7.0355**
23 (6-19)	1.769	1.837	5.8405	2.0015	1.0270	1.0000	1.0762	**4.6185**	**1.0564**
24 (7-13)	4.339	4.397	5.0342	4.7178	4.3788	5.0000	4.3282	**4.3287**	**4.6344**
25 (8-9)	2.115	2.115	3.8932	2.6101	2.1951	6.5000	2.2062	**2.2753**	**2.6707**
26 (8-14)	4.951	4.923	6.1647	4.5434	4.2562	5.5000	4.8730	**4.4873**	**5.0560**
27 (8-15)	4.147	4.047	6.8990	4.6174	4.6605	7.0000	4.8202	**4.8310**	**6.3306**
28 (8-21)	6.044	5.906	11.6387	9.6758	8.8694	15.5000	9.0166	**8.5942**	**11.1431**
29 (9-10)	3.222	3.392	3.8343	3.6296	3.2333	4.5000	3.4591	**4.0457**	**3.9277**
30 (9-15)	1.970	1.902	1.4772	1.4891	1.7611	2.5000	1.9876	**3.1836**	**1.2864**
31 (9-16)	4.290	4.381	1.3075	3.4020	3.2831	2.5000	3.4317	**2.0009**	**2.2858**
32 (9-22)	8.020	8.442	4.4876	6.2153	7.1936	1.0000	7.7208	**6.4017**	**5.2431**
33 (10-11)	4.857	5.011	6.0196	5.9308	4.9840	6.0000	4.8261	**7.1641**	**4.9566**
34 (10-16)	3.689	3.577	2.6729	3.2334	3.6672	1.0000	2.9942	**3.8786**	**2.5732**
35 (10-17)	2.831	2.805	1.6342	2.7173	2.4062	1.0000	2.5166	**2.9547**	**2.3854**
36 (10-23)	1.985	2.024	1.8410	1.3932	2.1576	1.0000	1.8493	**4.0056**	**1.4009**
37 (11-12)	6.373	6.709	6.8841	6.5660	7.1043	10.0000	7.1007	**9.2614**	**7.5222**
38 (11-17)	4.865	5.054	4.1393	4.8170	5.2070	5.5000	5.1141	**4.0014**	**4.9231**
39 (11-18)	3.412	3.259	3.3264	3.2626	3.6853	3.5000	4.0067	**2.6725**	**3.8285**
40 (11-24)	1.027	1.063	1.0000	1.0165	1.0007	1.0000	1.0270	**1.0172**	**1.0086**
41 (12-13)	6.218	5.934	6.9373	7.2529	6.6302	7.5000	6.3676	**7.8945**	**6.6448**
42 (12-18)	7.342	7.057	4.4568	5.9226	6.6773	8.5000	4.3443	**5.6654**	**6.6450**
43 (12-19)	5.458	5.745	4.6758	5.3115	5.2167	7.5000	5.2791	**3.8422**	**5.4045**
44 (12-25)	1.140	1.185	1.0084	1.0010	1.0016	1.0000	1.0086	**1.0078**	**1.0024**
45 (13-19)	7.401	7.274	7.5103	7.7499	8.1289	7.5000	7.2667	**11.0757**	**7.9904**

46 (13-20)	4.578	4.798	5.2449	4.7836	4.5151	6.5000	4.3730	**4.5371**	**4.0605**
47 (13-26)	1.561	1.515	1.0550	1.0035	1.0010	1.0000	1.0761	**1.1070**	**1.0022**
Best weight (kg)	10435.47	10453.84	10504.20	10345.12	10326.296	11044.617	10334.852	**10663.092**	**10278.571**
									[10319.228][3]
Average weight (kg)	10658.48	11100.57	10590.67	10393.83	10399.828	13017.900	10420.668	**11288.002**	**10369.689**
Worst weight (kg)	-	-	-	-	-	-	-	**11742.704**	**10481.253**
Standard deviation (kg)	129.90	334.20	52.51	39.15	75.441	1454.813	79.966	**291.065**	**53.786**
Number of FE analyses	80000	50000	20000	20000	25000	20000	17500	**20000**	**20000**

Table 7: Natural frequencies (Hz) of the optimal designs for the 1410-bar dome

Frequency number	CPA [22]	DPSO [46]	ECBO-Cascade [48]	MDVC-UPVS [49]	PFJA [50]	CRPSO [51]	MJA [52]	This study	
								JS	QJS
1	7.000	7.001	7.0020	7.000	7.0009	7.0008	7.0003	**7.0002**	**7.0000**
2	7.000	7.001	7.0030	7.001	7.0009	-	7.0003	**7.0002**	**7.0000**
3	9.000	9.003	9.0010	9.000	9.0001	9.0068	9.0000	**9.0000**	**9.0002**
4	9.002	9.005	9.0010	9.000	9.0002	-	9.0002	**9.0066**	**9.0002**
5	9.002	9.005	9.0030	9.000	9.0002	-	9.0002	**9.0066**	**9.0006**

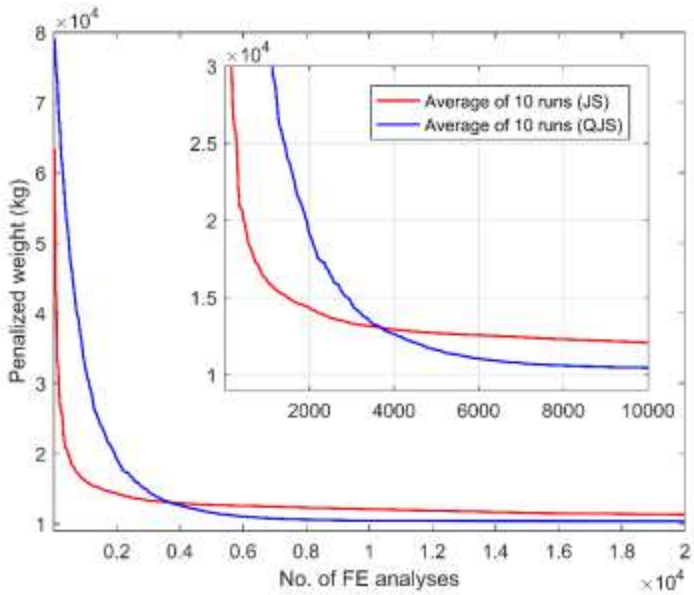

Figure 13. Average weight convergence histories for the 1410-bar double-layer dome

3. QJS found a feasible optimum design corresponding to a structural weight of 10319.228 kg after 12520 analyses.

Table 8: Comparison of the computational efficiency of the classical and efficient eigensolution methods

Problem	Classical eigensolution method		Efficient eigensolution method	
	Eigenvalue problem	CPU time (s)	Eigenvalue problem	CPU time (s)
600-bar dome-like truss	1 matrix of 576 × 576	0.0363	24 matrices of 24 × 24	0.0054
1410-bar dome-like truss	1 matrix of 1080 × 1080	0.1797	30 matrices of 36 × 36	0.0140

7. CONCLUSION

In this paper, a quantum-based Jellyfish Search algorithm, named Quantum JS (QJS), is proposed to solve structural optimization problems. Three main improvements are introduced in the proposed QJS: a quantum-based update rule to encourage the diversification in the search space, a new boundary control mechanism to avoid getting trapped in local optima, and modifications of the time control mechanism to strike a better balance between global and local searches. The purpose of the proposed QJS is to eliminate the drawback of the classical JS and to improve the balance between the diversification and the intensification tasks. The proposed QJS is applied to optimal design of cyclic symmetric dome structures with multiple frequency constraints. The structural analyses required in the optimization process are conducted by an efficient eigensolution method. The present eigensolution method takes advantage of the properties of the mass and stiffness matrices. It decomposes the initial free-vibration eigenproblem into some smaller sub-eigenproblems, which results not only in high accurate free vibration analysis results but also a substantial decrease in computational time as compared to the existing classical methods.

To illustrate the efficiency and accuracy of the present eigensolution method as well as the performance of the JS and the QJS, both of the algorithms are applied to solve the sizing optimization problem of two large-scale cyclic symmetric dome structures with multiple frequency constraints. To the best of our knowledge, this is the first time that JS is applied to solve frequency-constrained optimization problems. Optimization results confirm that the best weight and average weight found by the QJS are better than those reported in the literature. Furthermore, thanks to effective exploration of the search space (within the first stages of the search process) and the good trade-off between the global exploration and the local exploitation, the QJS outperforms the classical JS in both terms of effectiveness and robustness. In all design examples, the best weight, average weight, worst weight, and standard deviation on average weight obtained by the proposed QJS are much better than those of the JS. This shows that the proposed QJS can offer a robust and competitive optimization algorithm for the optimal design of truss structures with multiple frequency constraints. Moreover, the results show that the present eigensolution leads to a significant decrease in computational time.

Declaration of competing interest
The authors declare that there are no known conflicts of interest associated with this publication, and there has been no significant financial support for this work that could have influenced its outcome.

REFERENCES

1. Khatibinia M, Naseralavi SS. Truss optimization on shape and sizing with frequency constraints based on orthogonal multi-gravitational search algorithm, *J Sound Vib* 2014; **333:** 6349-69. https://doi.org/10.1016/j.jsv.2014.07.027.
2. Grandhi RV, Venkayya VB. Structural optimization with frequency constraints, *AIAA J* 1988; **26:** 858-66. https://doi.org/10.2514/3.9979.
3. Grandhi R. Structural optimization with frequency constraints-a review, *AIAA J* 1993; **31:** 2296-303. https://doi.org/10.2514/3.11928.
4. Rao SS, Reddy ES. Optimum design of stiffened conical shells with natural frequency constraints, *Comput Struct* 1981; **14:** 103-10. https://doi.org/10.1016/0045-7949(81)90089-4.
5. Bellagamba L, Yang TY. Minimum-mass truss structures with constraints on fundamental natural frequency, *AIAA J* 1981; **19:** 1452-8. https://doi.org/10.2514/3.7875
6. Khot NS. Optimization of structures with multiple frequency constraints, *Comput Struct* 1985; **20:** 869-76. https://doi.org/10.1016/0045-7949(85)90006-9.
7. Vanderplaats GN, Salajegheh E. An efficient approximation technique for frequency constraints in frame optimization, *Int J Numer Methods Eng* 1988; **26:** 1057-69. https://doi.org/10.1002/nme.1620260505.
8. Nakamura T, Ohsaki M. Sequential optimal truss generator for frequency ranges, *Comput Methods Appl Mech Eng* 1988; **67:** 189-209. https://doi.org/10.1016/0045-7825(88)90125-9.
9. Xie YM, Steven GP. A simple approach to structural frequency optimization, *Comput Struct* 1994; **53:** 1487-91. https://doi.org/10.1016/0045-7949(94)90414-6.
10. Tong WH, Liu GR. An optimization procedure for truss structures with discrete design variables and dynamic constraints, *Comput Struct* 2001; **79:** 155-62. https://doi.org/10.1016/S0045-7949(00)00124-3.
11. Kaveh A, Biabani Hamedani K, Kamalinejad M. Set theoretical variants of the teaching–learning-based optimization algorithm for optimal design of truss structures with multiple frequency constraints, *Acta Mech* 2020; **231:** 3645-72. https://doi.org/10.1007/s00707-020-02718-3.
12. Lingyun W, Mei Z, Guangming W, Guang M. Truss optimization on shape and sizing with frequency constraints based on genetic algorithm, *Comput Mech* 2005; **35:** 361-8. https://doi.org/10.1007/s00466-004-0623-8.
13. Gomes HM. Truss optimization with dynamic constraints using a particle swarm algorithm, *Expert Syst Appl* 2011; **38:** 957-68. https://doi.org/10.1016/j.eswa.2010.07.086.
14. Kaveh A, Zolghadr A. Truss optimization with natural frequency constraints using a hybridized CSS-BBBC algorithm with trap recognition capability, *Comput Struct* 2012; **102:** 14-27. https://doi.org/10.1016/j.compstruc.2012.03.016.
15. Kaveh A, Ghazaan MI. Optimal design of dome truss structures with dynamic frequency constraints, *Struct Multidiscip Optim* 2016; **53:** 605-21. https://doi.org/10.1007/s00158-015-1357-2.

16. Ho-Huu V, Nguyen-Thoi T, Truong-Khac T, Le-Anh L, Vo-Duy T. An improved differential evolution based on roulette wheel selection for shape and size optimization of truss structures with frequency constraints, *Neural Comput Appl* 2018; **29:** 167-85. https://doi.org/10.1007/s00521-016-2426-1.
17. Lieu QX, Do DT, Lee J. An adaptive hybrid evolutionary firefly algorithm for shape and size optimization of truss structures with frequency constraints, *Comput Struct* 2018; **195:** 99-112. https://doi.org/10.1016/j.compstruc.2017.06.016.
18. Kaveh A, Kamalinejad M, Hamedani KB. Enhanced versions of the shuffled shepherd optimization algorithm for the optimal design of skeletal structures, *Structures* 2021; **29:** 1463-95. https://doi.org/10.1016/j.istruc.2020.12.032.
19. Kaveh A, Biabani Hamedani K, Barzinpour F. Optimal size and geometry design of truss structures utilizing seven meta-heuristic algorithms: a comparative study, *IJOCE* 2020; **10:** 231-60.
20. Mirjalili S, Lewis A. The whale optimization algorithm, *Adv Eng Softw* 2016; **95:** 51-67. https://doi.org/10.1016/j.advengsoft.2016.01.008.
21. Chou JS, Truong DN. A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean, *Appl Math Comput* 2021; **389:** 125535. https://doi.org/10.1016/j.amc.2020.125535.
22. Kaveh A, Zolghadr A. Optimal design of cyclically symmetric trusses with frequency constraints using cyclical parthenogenesis algorithm, *Adv Struct Eng* 2018; **21:** 739-55. https://doi.org/10.1177%2F1369433217732492.
23. Kaveh A, Salimbahrami B. Eigensolution of symmetric frames using graph factorization, *Commun Numer Methods Eng* 2004; **20:** 889-910. https://doi.org/10.1002/cnm.711.
24. Kaveh A, Rahami H. Block diagonalization of adjacency and Laplacian matrices for graph product; applications in structural mechanics, *Int J Numer Methods Eng* 2006; **68:** 33-63. https://doi.org/10.1002/nme.1696.
25. Kaveh A, Nemati F. Eigensolution of rotationally repetitive space structures using a canonical form, *Int J Numer Method Biomed Eng* 2010; **26:** 1781-96. https://doi.org/10.1002/cnm.1265.
26. Kaveh A, Rahami H. An efficient analysis of repetitive structures generated by graph products, *Int J Numer Methods Eng* 2010; **84:** 108-26. https://doi.org/10.1002/nme.2893.
27. Kaveh A, Fazli H. Approximate eigensolution of locally modified regular structures using a substructuring technique, *Comput Struct* 2011; **89:** 529-37. https://doi.org/10.1016/j.compstruc.2010.12.013.
28. Kaveh A, Rahmani P. Canonical forms and rotationally repetitive matrices for eigensolution of symmetric structures, *Sci Iran* 2020; **28:** 192-208. https://doi.org/10.24200/SCI.2020.56639.4827.
29. Kaveh A, Rahami H, Shojaei I. *Swift Analysis of Civil Engineering Structures Using Graph Theory Methods*, Springer, 1st edition, Cham, Switzerland, 2020.
30. Kaveh A, Rahami H. Block circulant matrices and applications in free vibration analysis of cyclically repetitive structures, *Acta Mech* 2011; **217:** 51-62. https://doi.org/10.1007/s00707-010-0382-x.
31. Kaveh A, Koohestani K. Combinatorial optimization of special graphs for nodal ordering and graph partitioning, *Acta Mech* 2009; **207:** 95-108.

https://doi.org/10.1007/s00707-008-0107-6.
32. Koohestani K, Kaveh A. Efficient buckling and free vibration analysis of cyclically repeated space truss structures, *Finite Elem Anal Des* 2010; **46:** 943-8. https://doi.org/10.1016/j.finel.2010.06.009.
33. Kaveh A, Kamalinejad M, Hamedani KB, Arzani H. Quantum Teaching-Learning-Based Optimization algorithm for sizing optimization of skeletal structures with discrete variables, *Structures* 2021; **32:** 1798-819. Elsevier. https://doi.org/10.1016/j.istruc.2021.03.046.
34. Kruse R, Borgelt C, Braune C, Mostaghim S, Steinbrecher M, Klawonn F, Moewes C. *Computational intelligence*, Springer, 1st edition, London, 2013.
35. Talbi EG. *Metaheuristics: from design to implementation*, John Wiley & Sons, 1st edition, USA, 2009.
36. Kaveh A. *Advances in metaheuristic algorithms for optimal design of structures*, Springer, 1st edition, Cham, Switzerland, 2014.
37. Joines JA, Houck CR. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. InProceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence 1994 Jun 27 (pp. 579-584). IEEE. https://doi.org/10.1109/ICEC.1994.349995
38. Kulkarni O, Kulkarni N, Kulkarni AJ, Kakandikar G. Constrained cohort intelligence using static and dynamic penalty function approach for mechanical components design, *Int J Parallel Emergent Distrib Syst* 2018; **33:** 570-88. https://doi.org/10.1080/17445760.2016.1242728
39. Jordehi AR. A review on constraint handling strategies in particle swarm optimisation, *Neural Comput Appl* 2015; **26:** 1265-75. https://doi.org/10.1007/s00521-014-1808-5.
40. Rowe WB. *Principles of modern grinding technology*, William Andrew, 2nd edition, USA, 2013.
41. Yang B. *Stress, strain, and structural dynamics: an interactive handbook of formulas, solutions, and MATLAB toolboxes*, Academic Press, 1st edition, USA, 2005.
42. Chopra AK. *Dynamics of structures*, Pearson Education, 1st edition, India, 2007.
43. Zemaityte M. *Theory and Algorithms for Linear Eigenvalue Problems*, The University of Manchester, United Kingdom, 2020.
44. Bathe KJ, Wilson EL. Solution methods for eigenvalue problems in structural mechanics, *Int J Numer Methods Eng* 1973; **6:** 213-26. https://doi.org/10.1002/nme.1620060207
45. Kaveh A, Biabani Hamedani K,, Joudaki A, Kamalinejad M. Optimal analysis for optimal design of cyclic symmetric structures subject to frequency constraints, Structure, in print, 2021.
46. Kaveh A. *Applications of Metaheuristic Optimization Algorithms in Civil Engineering*, Springer, 1st edition, Cham, Switzerland, 2017.
47. Kaveh A, Ghazaan MI. Vibrating particles system algorithm for truss optimization with multiple natural frequency constraints, *Acta Mech* 2017; **228:** 307-22. https://doi.org/10.1007/s00707-016-1725-z.

48. Kaveh A, Ilchi Ghazaan M. Meta-heuristic algorithms for optimal design of real-size structures, Springer, 1st edition, Cham, Switzerland, 2018.
49. Kaveh A, Ilchi Ghazaan M. A new hybrid meta-heuristic algorithm for optimal design of large-scale dome structures, *Eng Optim* 2018; **50:** 235-52. https://doi.org/10.1080/0305215X.2017.1313250.
50. Degertekin SO, Bayar GY, Lamberti L. Parameter free Jaya algorithm for truss sizing-layout optimization under natural frequency constraints, *Comput Struct* 2021; **245:** 106461. https://doi.org/10.1016/j.compstruc.2020.106461.
51. Carvalho JP, Lemonge AC, Carvalho ÉC, Hallak PH, Bernardino HS. Truss optimization with multiple frequency constraints and automatic member grouping, *Struct Multidiscip Optim* 2018; **57:** 547-77. https://doi.org/10.1007/s00158-017-1761-x.
52. Degertekin SO, Yalcin Bayar G, Lamberti L. Jaya algorithm for sizing and layout optimization of truss structures with natural frequency constraints. In: Topping BHV, Ivany P, editors, Proceedings of the sixteenth international conference on civil, structural & environmental engineering computing, Civil-Comp Press, Stirlingshire (UK); 2019.