Computers and Structures 182 (2017) 137–148

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

Efficient truss optimization using the contrast-based fruit fly optimization algorithm

Stratis Kanarachos *, James Griffin, Michael E. Fitzpatrick

Faculty of Engineering, Environment and Computing, Coventry University, Priory Street, Coventry CV1 5FB, United Kingdom

ARTICLE INFO

Article history:
Received 29 April 2016
Accepted 16 November 2016
Available online 22 December 2016

Keywords:
Fruit fly optimization
Multi-parameter
Truss optimization

ABSTRACT

A recent biological study shows that the extremely good efficiency of fruit flies in finding food, despite their small brain, emerges by two distinct stimuli: smell and visual contrast. "contrast-based fruit fly optimization", presented in this paper, is for the first time mimicking this fruit fly behaviour and developing it as a means to efficiently address multi-parameter optimization problems. To assess its performance a study was carried out on ten mathematical and three truss optimization problems. The results are compared to those obtained using twelve state-of-the-art optimization algorithms and confirm its good and robust performance. A sensitivity analysis and an evaluation of its performance under parallel computing were conducted. The proposed algorithm has only a few tuning parameters, is intuitive, and multi-faceted, allowing application to complex n-dimensional design optimization problems.

1. Introduction

Design optimization is a powerful tool widely utilised by engineers to produce better performing, more reliable and cost-effective products. It originated from the aircraft industry and rapidly expanded in multiple domains like structural and mechatronics engineering [1–3]. Its success is mainly due to its inherent merit, delivered in combination with a significant increase in computational power and accessibility to practitioners through commercial engineering software [4]. The mathematical formulation of an optimization problem can be expressed as:

$$\begin{aligned} &\textit{minimize } \mathrm{f}(\mathbf{x}) \\ &\textit{subject to}: \ \mathrm{h}_{\mathrm{eq\_j}}(\mathbf{x}) = 0, \quad \textit{eq\_j} = 1,2,\ldots,n_{eq} \\ &\qquad\qquad \mathrm{g}_{\mathrm{ineq\_j}}(\mathbf{x}) \leqslant 0, \quad \textit{ineq\_j} = 1,2,\ldots,n_{ineq} \end{aligned} \tag{1}$$

where $\mathrm{f}(\mathbf{x})$ is the objective function that expresses the performance of a system, $\mathbf{x}$ is a vector comprised out of m design variables, n_{eq} and n_{ineq} are the number of equalities and inequalities described by the functions $\mathrm{h}_{\mathrm{eq\_j}}$ and $\mathrm{g}_{\mathrm{ineq\_j}}$ respectively.

Initially, optimization technology – including the steepest descent, the conjugate gradient, the Newton and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) methods – was based on mathematical formulations involving the calculation of first and/or second derivatives [5,6]. For example, in the gradient descent method, one starts from an initial point $\mathbf{x}_0$ where the function value $\mathrm{f}(\mathbf{x}_0)$ is calculated and then takes a step in a downward direction, where the function value will be lower. To make such a step, one utilises local information $\nabla f^T(\mathbf{x}_0)$ and explores the immediate vicinity of the current point. The search for the optimum design vector $\boldsymbol{x}^*$ is expressed by the following iterative formula:

$$\mathbf{x}[\mathrm{k}+1] = \mathbf{x}[\mathrm{k}] - a[k] \cdot \nabla f^T(\mathbf{x}[\mathrm{k}]) \tag{2}$$

where $a[k]$ is a scaling parameter, k is the iteration number, $\mathbf{x}[k]$ is the design vector in kth iteration and $\mathbf{x}[\mathrm{k}+1]$ is the new design vector. It is highlighted that gradient-based optimization methods converge to the optimum value in only a few iterations and are considered to be the best approach for solving many optimization problems, at least in a local context.

Although mathematically rigorous, gradient-based algorithms get trapped in local minima in the case of noisy or highly nonlinear problems. By contrast, meta-heuristic optimization algorithms like the *Genetic Algorithm* [7], *Particle Swarm Optimization* [8] and *Harmony Search* [9] do not use gradient information and are better suited for global optimization problems. On the downside, the performance of non-gradient algorithms depends on a number of tuning parameters which are not known *a priori*.

Although, in some cases, for the meta-heuristic algorithms empirical rules exist, they are not always adequate. There is a need for intuitive meta-heuristic algorithms with a minimum number of tuning parameters.

* Corresponding author at: Engineering & Computing Building - EC 4-07, Faculty of Engineering, Environment and Computing, Coventry University, 3, Gulson Road, Coventry CV1 2JH, United Kingdom.

E-mail addresses: stratis.kanarachos@coventry.ac.uk (S. Kanarachos), ac0393@coventry.ac.uk (J. Griffin), ab6856@coventry.ac.uk (M.E. Fitzpatrick).

http://dx.doi.org/10.1016/j.compstruc.2016.11.005
0045-7949/

2. Brief literature review on truss optimization

Trusses are fundamental in structural engineering and applications can be found from nano to macro levels [10,11]. Truss optimization problems are usually multi-parameter optimization problems owing to the large number of members comprising the truss. They are also highly nonlinear because of the multiple constraints considered, including displacement, stress and natural frequency, and the complex interaction between the structural members. In the general case, the truss optimization problem is formulated as a mathematical optimization problem:

$$\begin{aligned} &\textit{find design vector } \mathbf{x} \\ &\textit{that minimizes } \mathrm{R}(\mathbf{x}) = \sum_{q=1}^{n} \rho_q \cdot A_q \cdot L_q \\ &\textit{subject to}: \quad \delta_{qmin} \leqslant \delta_q \leqslant \delta_{qmax}, \quad q = 1,2,\ldots,n_n \\ &\qquad\qquad \sigma_{qmin} \leqslant \sigma_q \leqslant \sigma_{qmax}, \quad q = 1,2,\ldots,n_n \\ &\qquad\qquad \mathrm{A}_{qmin} \leqslant \mathrm{A}_q \leqslant \mathrm{A}_{qmax}, \quad q = 1,2,\ldots,n \\ &\qquad\qquad \omega_{qmin} \leqslant \omega_q \leqslant \omega_{qmax}, \quad q = 1,2,\ldots,n_\omega \end{aligned} \tag{3}$$

where R is the mass of the truss, n is the number of truss members, n_n is the number of nodes, and n_ω the number of desired natural frequencies. ρ_q, A_q and L_q are the density, cross sectional area and length of qth member respectively. δ_q and σ_q are the displacement and stress at the qth node. ω_q is the qth natural frequency. δ_{qmin} and δ_{qmax} are the lower and upper displacement bounds for the qth node, σ_{qmin} and σ_{qmax} are the lower and upper normal stress bounds for the qth node, A_{qmin} and A_{qmax} are the lower and upper cross sectional area bounds for the qth structural member and ω_{qmin} and ω_{qmax} are the lower and upper bounds for the qth natural frequency.

There is an increasing interest in developing efficient algorithms for large-scale truss optimization. The algorithms are mainly meta-heuristic and broadly classified into three categories.

The first category encompasses the Evolutionary Algorithms (EA). EAs use mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection for calculating new candidates. EAs usually suffer from premature convergence and weak exploitation capabilities. Both drawbacks are compensated by choosing bigger populations, however, this leads to larger computational cost. Wei et al. [12] proposed, as a solution to this problem, the Niche Hybrid Parallel Genetic Algorithm (NHPGA). NHPGA aims to effectively combine the robust and global search characteristics of the genetic algorithm, the strong exploitation ability of Nelder-Mead's simplex method, and the computational speed of parallel computing.

The *second category* encompasses physical algorithms that resemble an employed physical process. For example, Kaveh and Bakhshpoori [13] developed an algorithm that mimics the evaporation of a tiny amount of water molecules on a solid surface with different wettability. The "*Water Evaporation Optimization Algorithm*" was tested and analysed in comparison to other existing methods on a set of 17 benchmark unconstrained functions, a set of 13 classical benchmark constraint functions, and three benchmark constraint engineering problems. The results obtained indicate that the proposed technique is highly competitive. The performance of the algorithm depends on a number of parameters, including the assumption of a monolayer and droplet evaporation phase, the number of water molecules and the minimum and maximum values of monolayer and droplet evaporation probabilities. Another example is the modified *Teaching-Learning-based optimization* (*TLBO*) algorithm [14]. *TLBO* mimics the two types of pedagogy in a classroom to find the optimum solution: class-level learning from a teacher, and individual learning between students. *TLBO* uses a relatively simple algorithm with no intrinsic parameters controlling its performance.

The third category includes population-based algorithms, such as *Particle Swarm Optimization* (*PSO*) [15]. *PSO* is formulated by mathematically modelling the social behaviour of birds and fish colonies in finding food resources or escaping from predators. In the standard *PSO* each member of the swarm finds its way based on their own experience and the best particle's position: particles do not exchange any information. This causes *PSO* to get trapped into local optima. In a recent publication by Mortazavi and Toğan [16] a new version of *PSO* was proposed. In this version, the concept of a weighted particle, created by exploiting all particles' experiences, is introduced. This helps to avoid premature convergence.

Other popular population-based algorithms are the artificial bee colony, the ant colony and the bacterial algorithm. In [17] the *Artificial Bee Colony* algorithm (*ABC*) is applied to truss optimization problems. *ABC* models the honeybee foraging behaviour in the natural environment. In a bee colony, the female bees start to search for food randomly. After finding a food source, the bee returns to the hive and informs her nest mates about her findings. The information concerns the food source; the direction in which it can be found; distance from the hive; and its quality. In a decentralised and intelligent manner, some of the bees follow their nest mates to the food source, while others search for food independently.

The ant colony optimization method (*ACO*) is employed in [18] for solving truss optimization problems with cardinality constraint. *ACO* models the food search behaviour of real ants. Ants deposit pheromone on the ground to mark their path and to inform other ants about the food location. The more ants concentrate in an area, the more pheromone is laid, and this will attract even more ants. By contrast, locations with no food have lower levels of pheromone, which diminish over time owing to evaporation.

In [19] three different variants of the bacterial foraging optimization algorithm are presented for solving a 10-bar truss structural optimization problem. The bacterial foraging algorithm models the foraging behaviour of *Escherichia coli* bacteria, which is characterised by three phases: chemotaxis, reproduction, and elimination-dispersal. In the chemotaxis phase, every bacterium moves a single step towards a random position. If an improvement in the objective function is achieved, the bacterium continues moving in the same direction till a stopping criterion is met. In the reproduction phase the bacteria with bad performance are eliminated while those with good performance are replicated. The performance is determined by the sum of all chemotaxis steps performed. In the dispersion phase randomly-selected bacteria are substituted by other randomly generated bacteria.

Fruit fly optimization (*FOA*) is a recently developed population-based optimization algorithm [20]. Fruit flies can very effectively find food at very long distances. This, in combination with the fact that their brain is very simple (it has only 100,000 neurons compared to house fly brains that have 300,000 neurons and human brains which have 100 billion), makes them very interesting from a biological and optimization perspective [21,22]. It is well-known that the main food search mechanism of fruit flies is based on smell. However, a recent biological study reveals that the search mechanism is stimulated also by visual contrasts, which are irrelevant to smell. Furthermore, their motion is described by standardised distinct sensory-motor reflexes, independent of each other. The contrast-based *fruit fly optimization algorithm*, proposed in this paper, mimics those recently discovered elements of fruit fly food search behaviour. The algorithm is, first, evaluated on a set of standard mathematical benchmark tests and then applied to structural truss design benchmark problems.

It is highlighted that fruit fly algorithms have never been tested in structural optimization before. The results show that the algo-

rithm achieves the same or better performance than other optimization algorithms.

The rest of the paper is structured as follows: in Section 3 the *contrast-based fruit fly optimization algorithm* is presented and explained. In Section 4 the results obtained for ten mathematical and three structural optimization problems are discussed and compared to those known from the literature and obtained using standard optimization tools. In Section 5 a sensitivity analysis is performed, including a performance evaluation when parallel computing is used. Finally, Section 6 gives conclusions and future work is proposed.

3. The contrast-based fruit fly optimization algorithm

Fruit flies have a keen sense of smell and use their antennae to detect odours. Their osphresis is so good that a fly can detect a source of food even if this is 40 km away. Furthermore, they have excellent vision and can detect where their fellows gather and fly to that direction. Pan proposed for the first time an optimization algorithm based on the fruit fly food search behaviour [20]. A brief description of Pan's algorithm follows.

For the ease of discussion, a one-dimensional problem is considered. In the first step, a fruit fly is generated at a random position $\mathbf{x}_0 = [X_{01}, Y_{01}]$. Then, a swarm of $N-1$ fruit flies $\mathbf{x_i}$ is randomly generated $\mathbf{x}_\mathrm{i} = [X_{i1}, Y_{i1}] = [X_{01} + rand, Y_{01} + rand]$, where *rand* is a random number $rand \epsilon [0,1]$. For each fruit fly, the distance $D_{i1} = \sqrt{X_{i1}^2 + Y_{i1}^2}$ from the origin of the coordinate system is calculated and used as an input to determine the objective function's value $S_{mi} = \mathrm{f}(D_{i1})$. Subsequently, the fruit fly with the best objective function value is identified, and the rest of the fruit fly swarm is relocated randomly around it. The algorithm iterates until a maximum number of iterations is reached. The simplicity and efficiency of *FOA* made it very popular and different versions were proposed to further improve its performance.

In [23] a modification of the original *FOA* is proposed for solving a steelmaking problem. A new exploitation and exploration strategy are proposed. In the new exploitation procedure, the fruit fly with the worst performance is substituted by the one that achieved the best. In comparison to the original *FOA*, only one fruit fly is replaced instead of the complete swarm. Furthermore, in the proposed exploitation strategy the fruit flies are randomly located around fruit flies that achieved a good performance and not only in the vicinity of the best fruit fly.

In [24] the original FOA is modified for reducing the probability of premature convergence. In particular, the input parameter D_{i1}, which is used to feed the objective function, is modified as $\frac{D_{i1}}{1+D_{i1}^2 \cdot (0.5-\delta)}$, where δ is a problem-dependent tuning parameter. The proposed modification enhances the exploration performance of FOA, however the choice of δ is problem-dependent.

In [25] the so-called improved fruit fly optimization algorithm is discussed. Instead of using fixed values for the food search radius, an adaptive one is suggested $\lambda = \lambda_{max} \cdot \exp\left(\ln\left(\frac{\lambda_{min}}{\lambda_{max}}\right) \cdot \frac{iter}{iter_{max}}\right)$, where λ_{min}, λ_{max} are the minimum and maximum search radius, *iter* is the iteration number and $iter_{max}$ is the maximum number of iterations. The fruit flies are calculated as $\mathbf{x}_\mathrm{i} = [X_{01} + \lambda \cdot rand, Y_{01} + \lambda \cdot rand]$. The proposed change improves the exploration performance of the original *FOA*.

The authors of [26] propose a multi-swarm fruit fly optimization. In this method, the initial swarm is divided into several sub-swarms (usually 4–10) and the sub-swarms explore independently the design space with the aim to find the global optimum. The proposed change makes the original *FOA* more flexible during the exploration phase.

The exploration strategy was also modified in [27]. More specific, the new positions of the fruit fly swarm are generated using a normal cloud generator. The normal cloud generator is dependent on three parameters: namely *Ex* (average value) which stands for possible food position; *En* (entropy) which represents the search range; and *He* (hyper-entropy) which indicates the stability of search. The larger the hyper-entropy is, the bigger the discrete degree. In the proposed algorithm the entropy *En* is continuously decreasing, described by $En = En\_max \cdot \left(1 - \frac{iter}{iter\_max}\right)^a$, where *En_max* and *a* are user defined parameters. The hyper-entropy *He* has a fixed relation to the entropy $He = 0.1 \cdot En$.

In [28] a new exploration strategy is proposed based on the theory of chaos. The authors suggest relocating the fruit flies using the following formula:

$$\mathbf{x_i}[k+1] = \mathbf{x_i}[k] + alpha \cdot (\mathbf{x_i}[k] - \mathbf{x_0}[k]) \tag{4}$$

where $\mathbf{x_i}[k+1]$ and $\mathbf{x_i}[k]$ are the new and old positions respectively, and *alpha* is a chaotic variable. The authors investigated the influence of ten different maps – including Chebyshev, Circle, Gauss/Mouse, Logistic and Tent – used for describing the chaotic behaviour.

In [29] the authors employed a modified version of *FOA* for training a Support Vector Machine utilised in medical data classification. The authors propose a slightly different exploration phase in which the new fruit fly position is determined by $\mathbf{x}_\mathrm{i} = [X_{01} + a_x \cdot rand - b_x, Y_{01} + a_y \cdot rand - b_y]$. The authors found that the proposed FOA improved significantly the generalisation performance of the algorithm and consequently the accuracy of the medical data classifier.

3.1. Contrast-based fruit fly optimization algorithm for multi-parameter problems (c-mFOA)

A recent biological study shows that fruit flies have a more complex food search mechanism [30,31] than the one modelled by Pan. In case the fruit flies cannot find food using osphresis, they start to explore objects with visual contrast. They land, and if there is not something to eat, they continue to forage. As an example, a glass of wine would be a contrasting shape with respect to its background that would merit their attention. Furthermore, it was found that fruit flies surge when the scent is strong and cast when it becomes weaker. Last but not least, it was observed that fruit flies present a response delay. It is believed that fruit flies developed these features to compensate for the chaotic movement of odours, particularly outdoors in the wind. For the first time, in this paper, the particular fruit fly behaviour is idealised, modelled and further developed to address multi-parameter optimization problems.

The basic steps of *c-mFOA* are summarised by the pseudo code shown in Fig. 1 and are explained in detail in the following. A flowchart is provided in Fig. 2.

3.2. Swarm localisation and termination

For the ease of discussion, a one-dimensional problem is considered. A coordinate system and the position of a fruit fly with coordinates $\mathbf{x}_0 = [X_{01}, Y_{01}]$ are defined (Fig. 3). The rest N fruit flies are located, randomly, in the vicinity of $\mathbf{x}_0$ according to Eq. (5).

$$\begin{aligned} X_{i1}[k] &= X_{01}[k] \cdot (1 + M \cdot (2 \cdot rand_{N_{res}} - 1), \quad i = 1, \ldots, N \\ Y_{i1}[k] &= Y_{01}[k] \cdot (1 + M \cdot (2 \cdot rand_{N_{res}} - 1), \quad i = 1, \ldots, N \end{aligned} \tag{5}$$

where $k = 1, 2, \ldots, K$ is the iteration number, N is the size of the swarm and $rand_{N_{res}}$ is a random number from a uniform discrete distribution defined in the interval $[1, N_{\mathrm{res}}]$. The use of a discrete distribution is not observed in nature, but is a feature we introduce to improve the algorithm's performance in multi parameter problems.

Multi-parameter contrast-based fruit fly optimisation algorithm

begin

Objective function f(**x**), **x**=[x_1, x_2, …, x_m]

Generate initial population of fruit flies $\mathbf{x}_i$, i=1,2,…, *N in the vicinity of* $\mathbf{x}_0$ based on a uniform discrete distribution [1, N_{res}]

Smell concentration S_{mi} *at* $\mathbf{x}_i$ *determined by* $S_{mi} = f(\mathbf{x}_i)$

Rank the fruit flies and find the current best $\min(S_{mi}) = S^*_{mi}$ *for* $\mathbf{x}^*_i$

If $S^*_{mi} < S_{m0}$ *then* $\mathbf{x}_0 = \mathbf{x}^*_i$

while (t < *MaxGeneration*)

Reposition the fruit fly swarm $\mathbf{x}_{ki}$, k=1,2, …, K and i=1,2,…, *N in the vicinity of current* $\mathbf{x}_{k0}$ based on a uniform discrete distribution[1, N_{res}]

Smell concentration S_{mki} *at* $\mathbf{x}_{ki}$ *determined by* $S_{mki} = f(\mathbf{x}_{ki})$

Rank the fruit flies and find the current best S^*_{mki} *for* $\mathbf{x}^*_{ki}$

If $S^*_{mki} < S_{mk0}$ *then* $\mathbf{x}_{k0} = \mathbf{x}^*_{ki}$

if (t_k>*delay time*)

if ($S_{mki} < S_{m(k-\kappa)0}$)

reduce the area of attraction (surging phase)

else if ($S_{mki} = S_{m(k-\kappa)0}$)

the worst performing candidate, $\mathbf{x}_{k0} = \mathbf{x}^{\cdot\cdot}_{ki}$ for which $\max(S_{mki}) = S^{\cdot\cdot}_{mki}$ becomes the attraction point (*contrast based vision phase*)

else if ($S_{mki} > S_{m(k-\kappa)0}$)

return to the previous best, $\mathbf{x}_{k0} = \mathbf{x}_{(k-\kappa)0}$ *(casting phase based on memory function)*

end if

end if

Initialise response time t_k =0

end while

Post process results and visualisation

end

Fig. 1. Pseudo-code of the proposed fruit fly optimization algorithm.

M is a scaling parameter that defines how coarse or fine the search strategy is.

Each fruit fly $\mathbf{x_i}[k] = [X_{i1}[k], Y_{i1}[k]]$ is assigned a value $DI_{i1}[k]$ based on the Euclidean distance between the fruit fly and the origin of the coordinate system:

$$D_{i1}[k] = \sqrt{(X_{i1}[k])^2 + (Y_{i1}[k])^2} \tag{6}$$

$$DI_{i1}[k] = \frac{1}{D_{i1}[k]} \tag{7}$$

$DI_{i1}[k]$ is sensitive for fruit flies located in the vicinity of the origin, contrary to those that are positioned far away. This implies that a good search strategy should start close to the origin.

Each fruit fly is assigned a "smell concentration" $Sm_i[k]$ at $\mathbf{x_i}[k]$ determined by the objective function value $Sm_i[k] = \mathrm{f}(\mathbf{x_i}[k]) = \mathrm{f}(DI_{i1}[k])$. A small objective function value corresponds to a position with high smell concentration.

The fruit flies are ranked, on the basis of their smell concentration, and the fruit fly $\boldsymbol{x}^*[k]$ that achieves the highest smell concentration $Sm^*[k]$ (lowest objective function value) at position $[X^*[k], Y^*[k]]$ is identified. In other words, if the smell concentration $Sm^*[k]$ is better than that of the current point of attraction $Sm_0[k]$:

$$\begin{aligned} &\text{if } Sm^*[k] < Sm_0 \\ &\text{then } X_0[k+1] = X^*[k] \text{ and } Y_0[k+1] = Y^*[k] \end{aligned} \tag{8}$$

and $\boldsymbol{x}^*[k]$ becomes the new point of attraction.

The algorithm terminates when the maximum number K iterations is reached.

3.3. Delay, casting, surging and visual contrast phases

When the stimulus changes fruit flies do not respond immediately: a delay is taking place before changing the food search strategy. As presented in [31], the delay is constant and independent of other parameters. This behaviour is idealised and modelled in the *c-mFOA* algorithm.

In case the objective function improves over the last κ iterations, κ represents the response delay, the swarm enters the "surging" phase, during which the flies move towards the attraction point $\mathbf{x}_0[k]$ at a greater speed:

$$\begin{aligned} &\text{if } (Sm_0[k] < Sm_0[k-\kappa]) \\ &M[k+1] = c \cdot M[k] \end{aligned} \tag{9}$$

In case the objective function does not change over the last κ iterations, then the swarm enters the "visual contrast" attraction phase, in which flies are attracted by the point $\mathbf{x}^{\cdot\cdot}[k]$ that achieves the lowest smell concentration $Sm^{\cdot\cdot}[\mathrm{k}] = \max(Sm_i[k])$:

$$\begin{aligned} &\text{if } (Sm_0[k] = Sm_0[k-\kappa]) \\ &X_0[k+1] = X^{\cdot\cdot}[\mathrm{k}] \text{ and } Y_0[k+1] = Y^{\cdot\cdot}[\mathrm{k}] \end{aligned} \tag{10}$$

where k is the current iteration.

In case the objective function worsens over the last κ iterations, the swarm enters the "casting" phase, in which flies return to the previous current best $\mathbf{x}_0[k-\kappa]$ and continue the search at a constant speed:

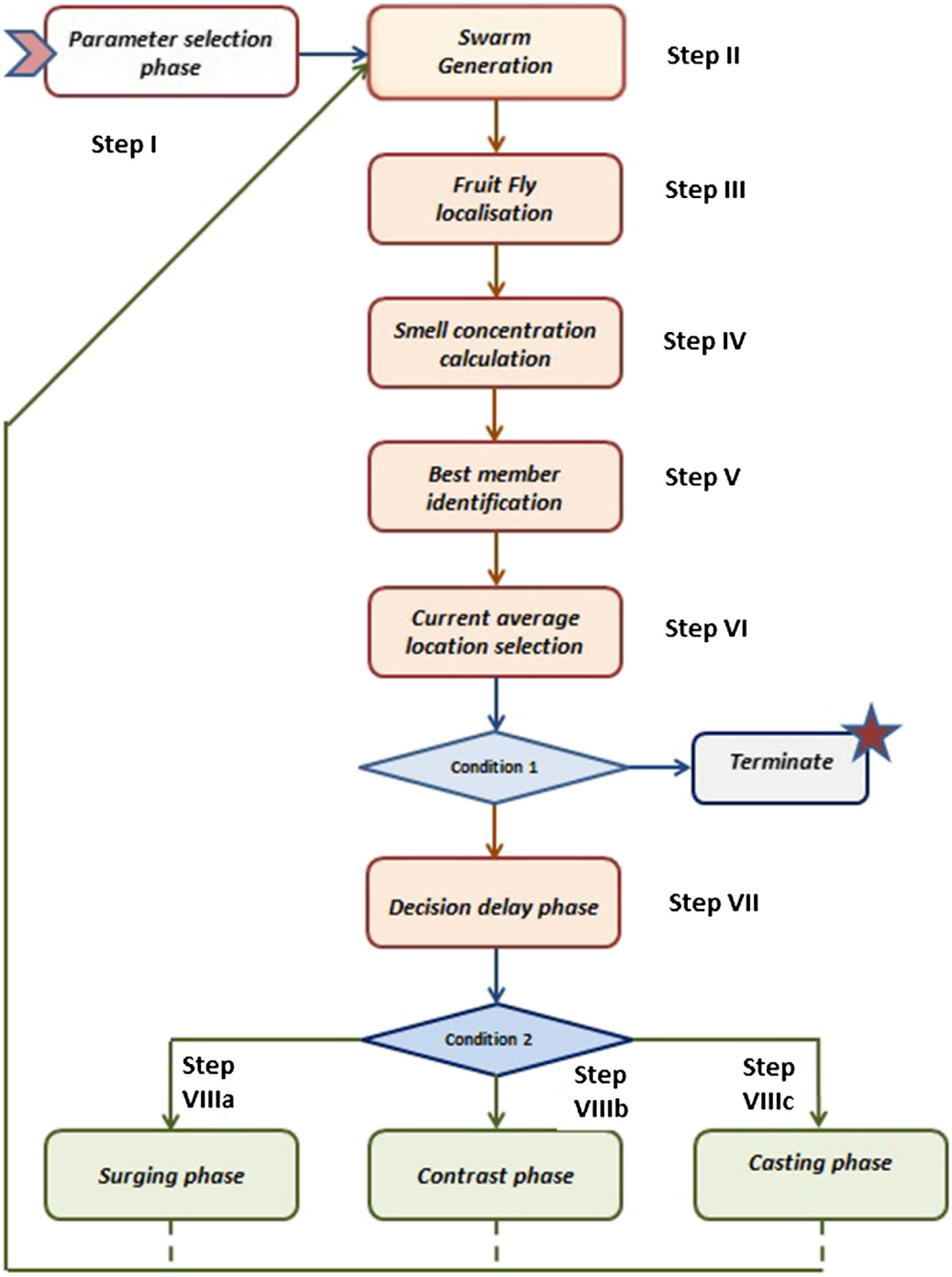

Fig. 2. Flowchart of proposed *c-mFOA* algorithm.

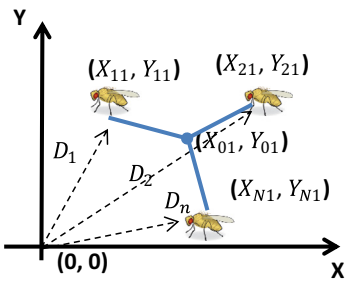

Fig. 3. Fruit fly position (X_i, Y_i) is described in a coordinate system.

$$\begin{aligned} &\text{if } (Sm^*[k] > Sm_0[k-\kappa]) \\ &X_0[k+1] = X_0[k-\kappa] \text{ and } \Upsilon_0[k+1] = Y_0[k-\kappa] \end{aligned} \tag{11}$$

The last step is based on the ability of fruit flies to remember and choose on the basis of how good or bad a memory was [32].

3.4. Constraint handling

In *c-mFOA* the constraints are dealt using the penalty function approach. The constrained optimization problem is formulated as an unconstrained one by augmenting the response function R(**x**) as shown in Eq. (12):

$$minimize\ \mathrm{f}(\mathbf{x}) = \mathrm{R}(\mathbf{x}) + \sum_{q=1}^{n_n} \kappa_q \cdot \varphi_{\mathrm{q}}(\mathbf{x}) + \sum_{q=1}^{n_n} \lambda_q \cdot \psi_{\mathrm{q}}(\mathbf{x}) + \sum_{q=1}^{n} \mu_q \cdot \chi_{\mathrm{q}}(\mathbf{x}) + \sum_{q=1}^{n_\omega} \nu_q \cdot \pi_{\mathrm{q}}(\mathbf{x})$$
$$\kappa_q > 0,\ \ \lambda_q > 0,\ \ \mu_q > 0 \text{ and } \nu_q > 0 \tag{12}$$

where κ_q, λ_q, μ_q, and ν_q are user defined constants and

$$\varphi_{\mathrm{q}}(\mathbf{x}) = \begin{cases} (\delta_q - \delta_{qmax})^2, & \text{if } \delta_q > \delta_{qmax} \\ (\delta_q - \delta_{qmin})^2, & \text{if } \delta_q < \delta_{qmin} \\ 0, & \text{if } \delta_{qmin} \leqslant \delta_q \leqslant \delta_{qmax} \end{cases} \tag{13}$$

$$\psi_{\mathrm{q}}(\mathbf{x}) = \begin{cases} (\sigma_q - \sigma_{qmax})^2, & \text{if } \sigma_q > \sigma_{qmax} \\ (\sigma_q - \sigma_{qmin})^2, & \text{if } \sigma_q < \sigma_{qmin} \\ 0, & \text{if } \sigma_{qmin} \leqslant \sigma_q \leqslant \sigma_{qmax} \end{cases} \tag{14}$$

$$\chi_{\mathrm{q}}(\mathbf{x}) = \begin{cases} (A_q - A_{qmax})^2, & \text{if } A_q > A_{qmax} \\ (A_q - A_{qmin})^2, & \text{if } A_q < A_{qmin} \\ 0, & \text{if } A_{qmin} \leqslant A_q \leqslant A_{qmax} \end{cases} \tag{15}$$

$$\pi_{\mathrm{q}}(\mathbf{x}) = \begin{cases} (\omega_q - \omega_{qmax})^2, & \text{if } \omega_q > \omega_{qmax} \\ (\omega_q - \omega_{qmin})^2, & \text{if } \omega_q < \omega_{qmin} \\ 0, & \text{if } \omega_{qmin} \leqslant \omega_q \leqslant \omega_{qmax} \end{cases} \tag{16}$$

4. Benchmark testing

4.1. Results on multi-parameter mathematical benchmark problems

A set of multi-parameter mathematical functions, commonly used in the literature, is employed to benchmark *c-mFOA*. In Table 1 the mathematical description, the number of variables used, the design space, the optimal position and best value for each function are provided. The functions are characterised by multiple local minima, singular values, and hyper planes ranging from flat to very steep. Solving the benchmark functions is a strong indicator of the robustness and effectiveness of the developed *c-mFOA*. The benchmark is conducted for m = 30 variables, where j = 1, . . . , *m*.*c-mFOA* is benchmarked against the *Genetic* (*GA*), *Particle Swarm Optimization* (*PSO*) and *Simulated Annealing* (*SA*) algorithms, commonly used in engineering practice. It is highlighted that many variants of the aforementioned algorithms exist and it is by no means attempted to compare *c-mFOA* to all variants. The purpose of this exercise is to show the differences between well-known and widely-employed algorithms (one evolutionary, one population-based and one physics-based), when implemented with their default parameter settings. The variants of *GA*, *PSO* and *SA* used in this study are the ones found in Matlab's Global Optimization Toolbox.

In the *GA* a population is comprised of 200 members, default setting in Matlab. The members are selected randomly from a uniform distribution restricted in the design space (*DS*), see Table 1. For each member the fitness value is calculated. The *GA* members are then sorted according to their rank. 80% of the new generation is created by crossover and 5%, progresses from the old generation. A stochastic uniform algorithm is used for the parent selection. The rest of the members are created by mutation. The genetic algorithm terminated when the maximum number of function evaluation generations is reached, unless it stalled. This was set to execute if for over 200 generations of a particular iteration the objective function value did not progress.

The inspiration for the *PSO* algorithm is based on flocks of birds swarming. The first step involves the generation of a population of particles with assigned initial velocities. The particles are uniform randomly created within bounds shown in Table 1. A fitness value is calculated for each particle and then the location that achieves the best value is determined. The algorithm chooses new velocities, based on the current velocity, the particles' individual best locations, and the best locations of their neighbours. It then iteratively updates the particle locations based on their old location, velocity, and their neighbours. The inertia range parameter of the algorithm was set within its standard bound [0.1 1.1]. The self-adjustment and social adjustment weights were set to their standard value 1.49. The swarm size was set to 100. Iterations proceeded until the algorithm reached the maximum number of function evaluations.

The simulated annealing (*SA*) algorithm starts from a random starting vector belonging to *DS* (Table 1). Two parameters – the temperature and re-annealing – determine its behaviour. The first one controls the extent of search. In this study the default initial temperature of 100 was used. The second one emulates the annealing process. After a certain number of new points are accepted, the temperature is raised to a higher value to restart the search and move out of local minima. If re-annealing is performed too fast this may not help the solver identify a minimum. Here, the default interval of 50 was chosen. The procedure terminates when the total number of function evaluations reaches the maximum value.

In *c-mFOA* the starting vector is a random vector bounded by *DS* (Table 1). The other parameters are selected as $K = 320$, $N = 50$, $\kappa = 15$, $M = 0.95$ and $c = 0.92$.

The benchmark is accomplished on the basis of a maximum number of function evaluations $maxfun = 16,000$. The parameters used for the evaluation are the mean value of optimised values found (*Mean*) and their standard deviation (*Std*) following 30 independent optimization runs. The results are summarised in Table 2. Fig. 4 illustrates two convergence rate examples for *c-mFOA*.

A sensitivity analysis for function F9 was conducted by varying the tuning parameter *M* in *c-MFOA*. The results obtained are listed

Table 1
Mathematical benchmark functions.

No	Description	m	$[DI_{jmin}, DI_{jmax}]$	DI_{j*}	f(**x**∗)
F1	$\mathrm{f}(\mathbf{x}) = \sum_{j=2}^{m} j \cdot DI_j^2$	30	$[-5.12, 5.12]$	0	0
F2	$\mathrm{f}(\mathbf{x}) = \sum_{j=2}^{m} j \cdot (2 \cdot DI_j^2 - x_{j-1}^2)^2 + (x_1 - 1)^2$	30	$[-10, 10]$	0	0
F3	$\mathrm{f}(\mathbf{x}) = -\exp(-0.5 \cdot \sum_{j=1}^{m} DI_j^2)$	30	$[-1, 1]$	0	-1
F4	$\mathrm{f}(\mathbf{x}) = \sum_{j=2}^{m} (10^6)^{\frac{j-1}{n-1}} \cdot DI_j^2$	30	$[-100, 100]$	0	0
F5	$\mathrm{f}(\mathbf{x}) = \max(\lvert DI_j \rvert)$	30	$[-100, 100]$	0	0
F6	$\mathrm{f}(\mathbf{x}) = \sum_{j=2}^{m} floor(DI_j + 0.5)^2$	30	$[-100, 100]$	0	0
F7	$\mathrm{f}(\mathbf{x}) = \sum_{j=2}^{m} DI_j^2$	30	$[-100, 100]$	0	0
F8	$\mathrm{f}(\mathbf{x}) = \sum_{j=1}^{m} \lvert DI_j \cdot sin(DI_j) + 0.1 \cdot DI_j \rvert$	30	$[-10, 10]$	0	0
F9	$\mathrm{f}(\mathbf{x}) = \sum_{j=1}^{m} (DI_j^2 - 10 \cdot cos(2 \cdot \pi \cdot DI_j) + 10)$	30	$[-5.12, 5.12]$	0	0
F10	$\mathrm{f}(\mathbf{x}) = 1 - cos(2 \cdot \pi \sum_{j=1}^{m} DI_j^2) + 0.1 \cdot \sum_{j=1}^{m} DI_j^2$	30	$[-100, 100]$	0	0

Table 2
Optimization benchmark results: Mean best value (Mean) and standard deviation (Std) obtained using the *Genetic Algorithm* (*GA*), *Particle Swarm Optimization* (*PSO*), *Simulated Annealing* (*SA*) and *contrast-based multi-parameter fruit fly optimization* (*c-mFOA*). Maximum number of function evaluations *maxfun* = 16,000.

Function	*GA*		*PSO*		*SA*		*c-mFOA*	
	Mean	*Std*	*Mean*	*Std*	*Mean*	*Std*	*Mean*	*Std*
F1	7.35	2.67	$2.10 \cdot 10^{-3}$	$3.00 \cdot 10^{-3}$	2.31	0.91	0	0
F2	1.82	1.32	$5.62 \cdot 10^{-4}$	$7.45 \cdot 10^{-4}$	0.49	0.16	$2.55 \cdot 10^{-4}$	$2.76 \cdot 10^{-4}$
F3	−0.95	0.01	−1	$1.55 \cdot 10^{-6}$	−0.31	0.07	−1	0
F4	$2.43 \cdot 10^{4}$	$2.23 \cdot 10^{4}$	$2.73 \cdot 10^{5}$	$1.30 \cdot 10^{6}$	$6.00 \cdot 10^{6}$	$1.38 \cdot 10^{6}$	$2.55 \cdot 10^{-14}$	$1.16 \cdot 10^{-15}$
F5	1.90	0.67	14.05	4.21	68.35	4.67	$5.63 \cdot 10^{-5}$	$6.47 \cdot 10^{-5}$
F6	12.03	8.38	0.10	0.30	2.86	1.61	0	0
F7	0.72	0.38	0.040	0.034	6.22	1.51	$4.32 \cdot 10^{-19}$	$8.08 \cdot 10^{-20}$
F8	0.67	0.41	0.03	0.06	25.42	1.32	$2.83 \cdot 10^{-10}$	$1.18 \cdot 10^{-11}$
F9	26.74	10.67	60.23	21.91	154.68	31.11	14.31	3.38
F10	$2.00 \cdot 10^{-14}$	$3.80 \cdot 10^{-14}$	$7.26 \cdot 10^{-7}$	$1.51 \cdot 10^{-6}$	$1.57 \cdot 10^{-9}$	$2.95 \cdot 10^{-9}$	$2.95 \cdot 10^{-8}$	$5.25 \cdot 10^{-8}$

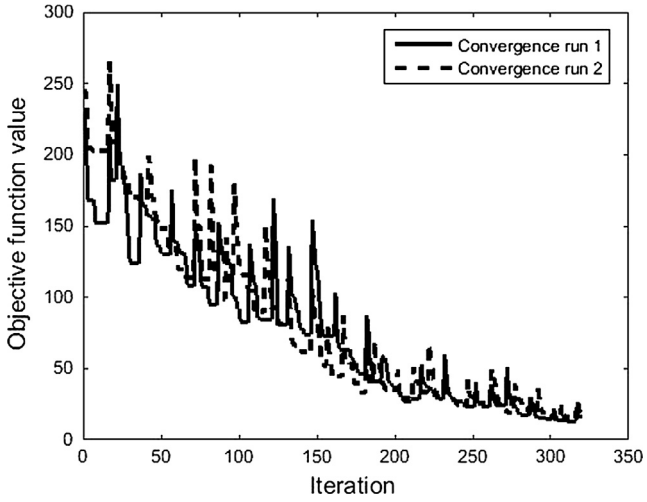

Fig. 4. F9 case study: Convergence rate examples for *c-mFOA*.

in Table 3. For $M > 5$ we achieve the same results as with $M = 5$ (true optimum).

4.2. Results on structural optimization benchmark problems

In this section, *c-mFOA* is compared to state-of-the-art optimization algorithms in solving truss optimization problems. Each problem is solved repetitively and independently thirty times.

4.2.1. Case 1: Twenty-five bar space truss

In Fig. 5 the 25-bar transmission bar used for benchmarking the proposed optimization algorithm is illustrated. It is one of the most common benchmarks for truss optimization algorithms. The coordinates of the nodes of the 25-bar truss are listed in Table 4, while Table 5 shows the element connectivity and Table 6 the load case considered. The structure includes 25 members which are divided into eight groups. Members belonging to the same group have the same cross sectional area, thus the total number of design variables is eight. The members are grouped as follows: (1) El_1, (2) El_2–El_5, (3) El_6–El_9, (4) El_{10}–El_{11}, (5) El_{12}–El_{13}, (6) El_{14}–El_{17}, (7) El_{18}–El_{21} and (8) El_{22}–El_{25}. The design variables are selected from a predefined set

$$\mathbf{S} = [6.45\ 12.90\ 19.35\ 25.80\ 32.26\ 38.70\ 45.16\ 51.61\ 58.06\ 64.52\ 70.96\ 77.41\ 83.87\ 90.32\ 96.77\ 103.22\ 109.67\ 116.12\ 122.58\ 129.03\ 135.48\ 141.93\ 148.38\ 154.83\ 167.74\ 180.64\ 193.54\ 206.45\ 219.35]\ (\mathrm{m}^2).$$

The material considered has a density of $\rho = 2768\ \mathrm{kg/m^3}$ ($0.1\ \mathrm{lb/in.^3}$) and a modulus of elasticity of $\mathrm{E} = 68,950\ \mathrm{MPa}$ ($10,000\ \mathrm{ksi}$).

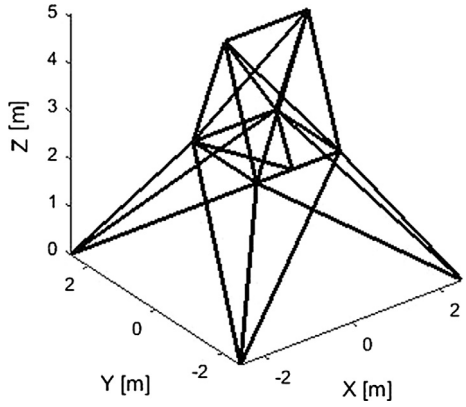

Fig. 5. Cases 1 & 2: 25-bar truss.

Table 4
Coordinates $[x, y, z]$ of the points defining the 25-bar truss.

Node	x (m)	y (m)	z (m)
1	−0.95	0.00	5.08
2	0.95	0.00	5.08
3	−0.95	0.95	2.54
4	0.95	0.95	2.54
5	0.95	−0.95	2.54
6	−0.95	−0.95	2.54
7	−2.54	2.54	0.00
8	2.54	2.54	0.00
9	2.54	−2.54	0.00
10	−2.54	−2.54	0.00

Table 3
c-mFOA sensitivity analysis for different M parameter values in solving F9.

Function	*c-mFOA*							
	$M = 2$		$M = 3$		$M = 4$		$M = 5$	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F9	16.20	6.28	0.56	0.78	0.53	0.50	0	0

Table 5
Element (El.) connectivity for the 25-bar truss.

El.	Node 1	Node 2	El.	Node 1	Node 2	El.	Node 1	Node 2
1	1	2	10	6	3	19	3	8
2	1	4	11	5	4	20	5	10
3	2	3	12	3	4	21	6	9
4	1	5	13	6	5	22	6	10
5	2	6	14	3	10	23	3	7
6	2	4	15	6	7	24	4	8
7	2	5	16	4	9	25	5	9
8	1	3	17	5	8			
9	1	6	18	4	7			

Table 6
Case1: Single load case for the 25-bar truss.

Node	P_x[N]	P_y[N]	P_z[N]
1	4448.22	−44482.22	−44482.22
2	0.00	−44482.22	−44482.22
3	2224.11	0.00	0.00
6	2668.93	0.00	0.00

The allowable stresses for each member are:

$$|\sigma_{qmin}| = \sigma_{qmax} = 275.79 \cdot 10^6 \text{ Pa} (= 40 \text{ ksi}), \quad q = 1, \ldots, 25 \tag{17}$$

and the maximum displacements for each node in x, y and z directions are:

$$|\delta_{qmin}| = \delta_{qmax} = 8.89 \cdot 10^{-3} \text{ m } (= 0.35 \text{ in.}), \quad q = 1, \ldots, 10 \tag{18}$$

The minimum and maximum cross sectional areas are:

$$|A_{qmin}| = 6.45 \cdot 10^{-5} \text{ m}^2 \ (= 0.1 \text{ in.}^2), \quad q = 1, \ldots, 25 \tag{19}$$

and

$$|A_{qmax}| = 219.35 \cdot 10^{-5} \text{ m}^2 \ (= 3.4 \text{ in.}^2), \quad q = 1, \ldots, 25 \tag{20}$$

Three optimization algorithms are utilised to compare *c-mFOA*. A brief description of each algorithm follows.

- The first algorithm is *Differential Evolution* (*DE*) [33].
- The second algorithm is a new version of DE called *adaptive elitist DE* algorithm (*aeDE*) [33]. There are three main differences with respect to the original one. In the mutation phase, an adaptive technique based on the deviation of objective function between the best individual and the whole population in the previous generation is proposed to select a suitable mutation operator. An elitist selection technique is utilised to increase the convergence rate in the selection phase. Finally, a rounding technique is integrated for problems with discrete design variables.
- The third algorithm is the *Mine Blast Algorithm* (*MBA*) [34]. The fundamental concepts and ideas of MBA are derived from the explosion of mine bombs in real world scenarios.

The results are summarised in Table 7. The cross sectional areas listed are the ones obtained for the best run. R_{min} is the truss mass for the best run, R_{avg} the average mass and R_{std} the standard deviation. $N_{analyses}$ is the minimum number of function evaluations that was required for calculating an optimum solution.

4.2.2. Case 2: Twenty-five bar space truss

In the second case the same truss as in Case 1 is utilised but now a multiple loading condition and different allowable stresses/displacements are considered. Furthermore, the cross-sectional areas are allowed to be continuous variables and do not need to take values from a predefined set. Tables 8 and 9 list the loading conditions and the allowable stresses/displacements respectively. The minimum and maximum cross sectional areas are:

$$|A_{qmin}| = 0.65 \cdot 10^{-5} \text{ m}^2 \ (= 0.01 \text{ in.}^2), \quad q = 1, \ldots, 25 \tag{21}$$

and

$$|A_{qmax}| = 219.35 \cdot 10^{-5} \text{ m}^2 \ (= 3.4 \text{ in.}^2), \quad q = 1, \ldots, 25 \tag{22}$$

Here, *c-mFOA* is compared to the:

- *Flower Pollination Algorithm* (*FPA*) [35]. FPA can efficiently combine local and global searches, inspired by cross-pollination and self-pollination of flowering plants to perform local and global searches respectively. In addition, an iterative constraint handling strategy where trial designs are accepted or rejected based on the allowed amount of constraint violation is utilised.
- *Teaching-Learning-based Optimization* (*TLBO*) algorithm [36]. The method makes use of the analogy between the learning process of learners and searching for designs to optimization problems. Learning can be accomplished either from the teacher or by the interaction between learners.
- *Colliding Bodies Optimization* (*CBO*) algorithm [37]. The algorithm is based on the physics describing collisions between bodies. Here each agent represents a body. After a collision between two moving bodies takes place, these are separated and moved to new positions with new velocities. This process is repeated until a termination criterion is satisfied.

The results are summarised in Table 10. The cross sectional areas listed are the ones obtained for the best run. R_{min} is the truss mass for the best run, R_{avg} the average mass and R_{std} the standard deviation. $N_{analyses}$ is the minimum number of function evaluations that was required for calculating an optimum solution.

4.2.3. Case 3: Thirty-seven bar planar truss with frequency constraints

This is considered as a constrained truss optimization problem on size and shape. Fig. 6 shows a sketch of the 37-bar truss. All nodes of the upper chord are allowed to vary in the y-axis (symmetry about Y-axis has to be maintained) and all the diagonal and upper chord bars are allowed to vary their cross-sectional areas. No upper bound exists for the cross sectional area but the minimum is:

$$|A_{qmin}| = 1 \cdot 10^{-4} \text{ m}^2, \quad q = 1, \ldots, 37 \tag{23}$$

The bottom bar elements have fixed cross sectional areas of $A_f = 4 \cdot 10^{-3}$ m^2. Masses of $m_{add} = 10$ kg are attached to each of the bottom nodes. The remaining bars are modelled as elements with initial sectional areas of:

$$|A_q|[k = 0] = 1 \cdot 10^{-4} \text{ m}^2 \tag{24}$$

Table 7
Case 1 - Optimization results for the 25-bar truss.

	Members	Ho-Huu *DE*	Ho-Huu *aeDE*	Sadollah *MBA*	c-MFOA
Variables					
Cross sectional area A_q [10^{-5} m^2]	A_1	6.45	6.45	6.45	6.45
	A_2, A_3, A_4, A_5	19.35	19.35	19.35	19.35
	A_6, A_7, A_8, A_9	219.35	219.35	219.35	219.35
	A_{10}, A_{11}	6.45	6.45	6.45	6.45
	A_{12}, A_{13}	135.48	135.48	135.48	135.48
	$A_{14}, A_{15}, A_{16}, A_{17}$	64.52	64.52	64.52	64.52
	$A_{18}, A_{19}, A_{20}, A_{21}$	32.26	32.26	32.26	32.26
	$A_{22}, A_{23}, A_{24}, A_{25}$	219.35	219.35	219.35	219.35
R_{min} (kg)		219.93	219.93	219.92	219.93
R_{avg} (kg)		219.95	220.00		219.97
R_{std} (kg)		0.06	0.12		0.07
$N_{analyses}$		3500	1440	2150	800

Table 8
Case 2: Multiple load case for the 25-bar truss.

	Node	P_x (N)	P_y (N)	P_z (N)
Case 1	1	4448.22	44482.22	−22241.11
	2	0.00	44482.22	−22241.11
	3	2224.11	0.00	0.00
	6	2224.11	0.00	0.00
Case 2	1	0.00	88964.43	−22241.11
	2	0.00	−88964.43	−22241.11

Table 9
Stresses allowable tensile (σ_{qmax}) and compressive (σ_{qmin}) for each element (*El.*).

El.	σ_{qmax} (MPa)	σ_{qmin} (MPa)	*El.*	σ_{qmax} (MPa)	σ_{qmin} (MPa)	*El.*	σ_{qmax} (MPa)	σ_{qmin} (MPa)
1	275.79	−241.95	10	275.79	−241.95	19	275.79	−47.98
2	275.79	−79.91	11	275.79	−241.95	20	275.79	−47.98
3	275.79	−79.91	12	275.79	−241.95	21	275.79	−47.98
4	275.79	−79.91	13	275.79	−241.95	22	275.79	−76.41
5	275.79	−79.91	14	275.79	−46.60	23	275.79	−76.41
6	275.79	−119.31	15	275.79	−46.60	24	275.79	−76.41
7	275.79	−119.31	16	275.79	−46.60	25	275.79	−76.41
8	275.79	−119.31	17	275.79	−46.60			
9	275.79	−119.31	18	275.79	−47.98			

Table 10
Case2 - Optimization results for the 25-bar truss.

	Members	Bekdas *FPA*	Degertekin *TLBO*	Kaveh *CBO*	c-MFOA
Variables					
Cross sectional area A_q[10^{-5} m^2]	A_1	0.65	0.65	0.65	1.98
	A_2, A_3, A_4, A_5	118.09	133.59	137.37	127.56
	A_6, A_7, A_8, A_9	205.33	190.73	186.18	195.51
	A_{10}, A_{11}	0.65	0.65	0.65	0.65
	A_{12}, A_{13}	0.65	0.65	0.65	0.66
	$A_{14}, A_{15}, A_{16}, A_{17}$	45.26	44.45	43.81	41.33
	$A_{18}, A_{19}, A_{20}, A_{21}$	111.37	104.55	103.70	108.10
	$A_{22}, A_{23}, A_{24}, A_{25}$	165.85	172.65	173.68	174.83
$R_{min}[kg]$		246.73	246.70	246.35	247.04
$R_{avg}[kg]$		246.99	–	246.78	247.19
$R_{std}[kg]$		0.27	–	0.13	0.15
$N_{analyses}$		8149	15,318	9090	8497

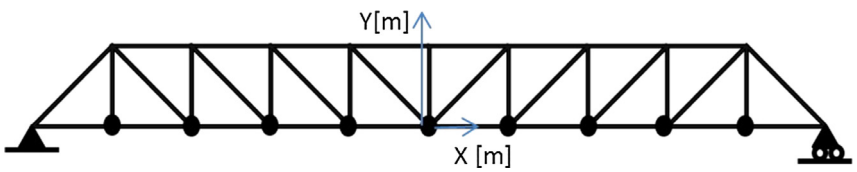

Fig. 6. Case 3: 37-bar truss size and shape optimization problem.

The material properties for the bar elements are $E = 2.1 \cdot 10^{11}$ N/m² and $\rho = 7800$ kg/m³. The first three natural frequencies are constrained, see Eq(25):

$$\begin{aligned} \omega_1 &\geqslant 20 \text{ Hz} \\ \omega_2 &\geqslant 40 \text{ Hz} \\ \omega_3 &\geqslant 60 \text{ Hz} \end{aligned} \tag{25}$$

The optimization problem involves three frequency constraints and 19 design variables (five shape variables + 14 sizing variables).

In the last case study *c-mFOA* is compared to the:

- *Ray Optimization* (*RO*) method [38]. In RO, each agent is modelled as a ray of light that moves in the search space in order to find the global or sub-global optimum solution.
- *Multi-class teaching-learning-based optimization* (*MC-TLBO*) [39]. *MC-TLBO* employs a two-step procedure. In the first step, parallel classes explore the search space; while in the second step, the best solutions of the parallel classes form a super class to be the initial population for a *TLBO*.
- *Particle Swarm Optimization (PSO)* [12].

The optimization results are summarised in Table 11, while the shape of the optimised structure is shown in Fig. 7. The results listed are the ones obtained for the best run. R_{min} is the truss mass for the best run.

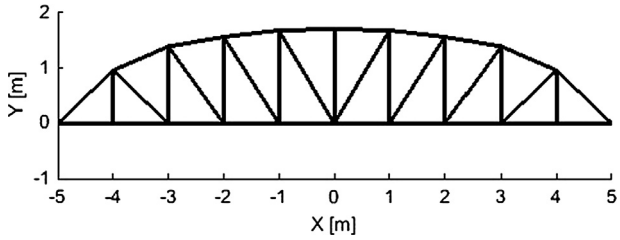

Fig. 7. Optimised truss shape in case 3.

5. Discussion

5.1. Analysis of results

From the benchmark study conducted on the mathematical functions F1-F10, it is concluded that *c-mFOA* performs significantly better than *GA*, *PSO* and *SA*. The calculated mean of best values and standard deviation are lower, an indication that *c-mFOA* presents a better and more robust solution. In all cases but one, mathematical function F9, the proposed algorithm succeeds in finding the optimum within 16,000 iterations. A sensitivity analysis, in which only parameter *M* was varied, showed that it is possible to calculate the optimum value within 16,000 iterations. *M* is a parameter with which we can increase or decrease the search space.

Two structural optimization problems with stress and displacement constraints were studied. In the first case, *c-mFOA* performs as well as the other state-of-the-art optimization algorithms to which it is compared. In the second and more complex case study *c-mFOA* performs better than the other algorithms, except for the *TLBO* algorithm which achieves a slightly better result. In greater detail, the best value calculated is 0.28% worse than *TLBO* and the average of best values is worse by only 0.13%.

In the third case study, which is also the most challenging, *c-mFOA* performs significantly better than the other algorithms except for *MC-TLBO*, which achieves the same result.

In conclusion, the proposed algorithm has been compared to twelve state-of-the-art and three standard optimization algorithms (e.g. those found in the Matlab Optimization Toolbox) for a range of benchmark problems. The benchmarks concern noisy functions with multiple minima, varying gradients and multiple constraints. *c-mFOA* achieves better results than other global optimization algorithms, except for TLBO which it achieves the same performance. It is highlighted that the results using *c-mFOA* are obtained for the same parameter setting. A random starting condition was used to initialise the population; thus, the performance is independent of the starting solution. As shown, only one parameter *M* is required to influence the algorithm's convergence rate.

5.2. Influence of discrete uniform distribution

We analysed the influence of the discrete distribution assumption, in particular parameter N_{res}, for generating the fruit fly swarm. We compared the optimization results found when solving F9 for three different assumptions: one uniform continuous distribution with $N_{res} = \infty$; one uniform discrete distribution with

Table 11
Case 3 - Optimization results for the 37-bar truss.

		Members	Kaveh *RA*	Farshchin *MC-TLBO*	Gomes *PSO*	*c-MFOA*
Variables						
Vertical position y_q (m)	y_3, y_{19}		1.0010	0.98300	0.9637	1.0108
	y_5, y_{17}		1.3909	1.38030	1.3978	1.3860
	y_7, y_{15}		1.5893	1.56450	1.5929	1.5608
	y_9, y_{13}		1.7507	1.68710	1.8812	1.6802
	y_{11}		1.8336	1.75900	2.0856	1.7580
Cross sectional area A_q (10^{-4} m²)		A_1, A_{27}	3.0124	2.99127	2.6797	3.1997
		A_2, A_{26}	1.0623	1.00054	1.1568	1.0025
		A_3, A_{24}	1.0005	1.00415	2.3476	1.0000
		A_4, A_{25}	2.2647	2.59576	1.7182	2.5875
		A_5, A_{23}	1.6339	1.21394	1.2751	1.0895
		A_6, A_{21}	1.6717	1.14226	1.4819	1.1261
		A_7, A_{22}	2.0591	2.31699	4.6850	2.5624
		A_8, A_{20}	1.6607	1.50998	1.1246	1.4121
		A_9, A_{18}	1.4941	1.51723	2.1214	1.5758
		A_{10}, A_{19}	2.4737	2.27223	3.8600	2.2461
		A_{11}, A_{17}	1.5260	1.21117	2.9817	1.0694
		A_{12}, A_{15}	1.4823	1.27385	1.2021	1.3193
		A_{13}, A_{16}	2.4148	2.49338	1.2563	2.3846
		A_{14}	1.0034	1.00000	3.3276	1.0001
R_{min} (kg)			364.04	359.966	377.20	360.07

$N_{res} = 15$ defined in the interval [0 15]; and one uniform discrete distribution with $N_{res} = 10$ defined in the interval [0 10]. Table 12 lists the mean value and standard deviation for the results obtained. Ten independent runs were conducted with the maximum number of function evaluations set to 16,000.

From the results it may be concluded that the discrete distribution assumption is influencing positively the performance of the *c-mFOA* algorithm.

5.3. Parallel computing performance

c-mFOA is a simple algorithm that would probably benefit, in terms of convergence rate, if combined with local search algorithms like the Nelder-Mead search method. On the other hand, *c-mFOA*'s advantage is that, with only a few code changes, it can be processed in parallel.

A study was conducted on a PC, running Windows 7 Enterprise, 64 bit operating system, with an Intel Core i5 processor, CPU running at 3.20 GHz and 4 GB installed memory (RAM). The statistical results obtained for Case 3 optimization, described in Section 4.2.3, are listed in Table 13.

From the results it is evident that by parallel processing the *c-mFOA* algorithm the computational burden can be significantly reduced. In particular, it was reduced by 31% and 61% using 2 parallel processors and 4 parallel processors respectively.

6. Conclusions and future research work

Fruit flies are extremely efficient in finding food, although their brain is rather simple. This is particularly interesting because it means that their performance is achieved using simple means. A recent fruit fly behavioural study found that fruit flies are stimulated not only by smell but also by visually contrasting objects, that their food search behaviour can be analysed in three distinct reflex driven motions, and that their response to stimuli presents a small but constant delay.

In this study, a new meta-heuristic algorithm, *contrast-based fruit fly optimization*, is presented. The proposed version idealises fruit fly behaviour and modifies the initial Fruit Fly Optimization algorithm by introducing the concepts of visual contrast stimulation, surging and casting phases and response delay. Furthermore the proposed algorithm is further developed for solving more efficiently multi-parameter optimization problems.

1. *Contrast-based fruit fly Optimization* was compared to three standard and twelve, state-of-the art optimization algorithms. The benchmark tests employed are ten noisy and unconstrained mathematical functions, two structural optimization problems with stress and displacement constraints, and one structural optimization problem with frequency constraints. *c-mFOA* achieves better results for fewer or the same number of function evaluations compared to other algorithms; except for the *Teaching Learning Based Optimization* algorithm, where it achieves exactly the same performance.

Table 12
Optimization results for three different assumptions: one continuous distribution with $N_{res} = \infty$, one discrete distribution with $N_{res} = 15$ and one discrete distribution with $N_{res} = 10$.

	$N_{res} = \infty$	$N_{res} = 15$	$N_{res} = 10$
Mean	39.41	13.65	8.22
Std	6.79	3.21	2.32

Table 13
Computational cost for optimising Case 3: serial implementation, parallel implementation with 2 processors, and parallel implementation with 4 processors.

Computational burden	Serial (s)	Parallel with 2 processors (s)	Parallel with 4 processors (s)
Mean	1190.8	704.9	473.1
Std	19.2	16.6	15.8

2. The results obtained using *c-mFOA* are achieved for the same set of parameters: no tuning is required. The performance is evaluated by running 30 independent repetitions, in which the initial solution is randomly chosen. As shown, only one parameter M is required to influence the algorithm's convergence rate and this choice is intuitively made. The proposed algorithm is simple to use and thus suitable for implementation in standard engineering tools.
3. The discrete distribution assumption for generating the fruit fly swarm is influencing positively the performance of the algorithm. A comparative analysis, on a complex-to-analyse mathematical function, showed that a uniform discrete distribution defined in the interval [1 10] improved the objective function value by almost 5 times.
4. The computational burden of *c-mFOA* can be significantly reduced if processed in parallel. A comparative study showed that by using four parallel processors it was possible to reduce the computational burden by 61%.

In the future it is foreseen to investigate the hybridisation of *c-MFOA* with local search methods and to study its behaviour in topology truss optimization problems. Last but not least, there are plans to further evaluate and improve the algorithm to work with highly n-dimensional search and more complex domain spaces.

Acknowledgments

MEF is grateful for funding from the Lloyd's Register Foundation, a charitable foundation helping to protect life and property by supporting engineering-related education, public engagement and the application of research.

References

[1] Liu D, Lohse-Busch H, Toropov V, Hühne C, Armani U. Detailed design of a lattice composite fuselage structure by a mixed optimization method. Eng Optim 2015:1–14.
[2] Kanarachos S, Kanarachos A. Intelligent road adaptive suspension system design using an experts' based hybrid genetic algorithm. Expert Syst Appl 2015;42:8232–42.
[3] Reynoso-Meza G, Sanchis J, Blasco X, Freire R. Evolutionary multi-objective optimisation with preferences for multivariable PI controller tuning. Expert Syst Appl 2016;51:120–33.
[4] List of optimization software <https://en.wikipedia.org/wiki/List_of_optimization_software> [21/03/2016].
[5] Papalambros P. Principles of optimal design. S.L.: Cambridge Univ Press; 2016.
[6] Arora J. Introduction to optimum design. Amsterdam: Elsevier/Academic Press; 2004.
[7] Wong K. Evolutionary multimodal optimization: a short survey. Adv Evol Algor Res 2016:1–15.
[8] Engelbrecht A. Particle swarm optimization with crossover: a review and empirical analysis. Artif Intell Rev 2015;45:131–65.
[9] Saka M, Hasançebi O, Geem Z. Metaheuristics in structural optimization and discussions on harmony search algorithm. Swarm Evolution Comput 2016;28:88–97.
[10] Bauer J, Hengsbach S, Tesari I, Schwaiger R, Kraft O. High-strength cellular ceramic composites with 3D microarchitecture. Proc Natl Acad Sci 2014;111:2453–8.
[11] Zhang Y, Hou Y, Liu S. A new method of discrete optimization for cross-section selection of truss structures. Eng Optim 2013;46:1052–73.
[12] Wei L, Tang T, Xie X, Shen W. Truss optimization on shape and sizing with frequency constraints based on parallel genetic algorithm. Struct Multidisc Optim 2010;43:665–82.

[13] Kaveh A, Bakhshpoori T. Water evaporation optimization: a novel physically inspired optimization algorithm. Comput Struct 2016;167:69–85.
[14] Camp C, Farshchin M. Design of space trusses using modified teaching–learning based optimization. Eng Struct 2014;62–63:87–97.
[15] Gomes H. Truss optimization with dynamic constraints using a particle swarm algorithm. Expert Syst Appl 2011;38:957–68.
[16] Mortazavi A, Toğan V. Simultaneous size, shape, and topology optimization of truss structures using integrated particle swarm optimizer. Struct Multidisc Optim 2016;54(4):715–36.
[17] Sonmez M. Artificial bee colony algorithm for optimization of truss structures. Appl Soft Comput 2011;11:2406–18.
[18] Angelo J, Bernardino H, Barbosa H. Ant colony approaches for multiobjective structural optimization problems with a cardinality constraint. Adv Eng Softw 2015;80:101–15.
[19] Parpinelli R, Teodoro F, Lopes H. A comparison of swarm intelligence algorithms for structural engineering optimization. Int J Numer Meth Eng 2012;91:666–84.
[20] Pan W. A new fruit fly optimization algorithm: taking the financial distress model as an example. Knowl-Based Syst 2012;26:6.
[21] Qisong Q, Gening X, Xiaoning F, Jun W. A new type bionic global optimization: const ruction and application of modified fruit fly optimization algorithm. Proc Inst Mech Eng, Part B: J Eng Manuf 2014;229:1614–21.
[22] Lin W. A novel 3D fruit fly optimization algorithm and its applications in economics. Neural Comput Appl 2015;27:1391–413.
[23] Li J, Pan Q, Mao K, Suganthan P. Solving the steelmaking casting problem using an effective fruit fly optimisation algorithm. Knowl-Based Syst 2014;72:28–36.
[24] Pan W. Using modified fruit fly optimisation algorithm to perform the function test and case studies. Connect Sci 2013;25:151–60.
[25] Pan Q, Sang H, Duan J, Gao L. An improved fruit fly optimization algorithm for continuous function optimization problems. Knowl-Based Syst 2014;62:69–83.
[26] Yuan X, Dai X, Zhao J, He Q. On a novel multi-swarm fruit fly optimization algorithm and its application. Appl Math Comput 2014;233:260–71.
[27] Wu L, Zuo C, Zhang H. A cloud model based fruit fly optimization algorithm. Knowl-Based Syst 2015;89:603–17.
[28] Mitić M, Vuković N, Petrović M, Miljković Z. Chaotic fruit fly optimization algorithm. Knowl-Based Syst 2015;89:446–58.
[29] Shen L, Chen H, Yu Z, Kang W, Zhang B, Li H, et al. Evolving support vector machines using fruit fly optimization for medical data classification. Knowl-Based Syst 2016;96:61–75.
[30] van Breugel F, Dickinson M. Plume-tracking behavior of flying drosophila emerges from a set of distinct sensory-motor reflexes. Curr Biol 2014;24:274–86.
[31] How the fruit fly's tiny brain finds food so well <http://www.futurity.org/fruit-flys-tiny-brain-finds-food-well/> [21/03/2016].
[32] Memories of fruit fly larvae are more complex than thought <http://www.nncn.de/en/news/Forschungsergebnisse-en/memories-of-fruit-fly-larvae-are-more-complex-than-thought> [21/03/2016].
[33] 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.
[34] Sadollah A, Bahreininejad A, Eskandar H, Hamdi M. Mine blast algorithm for optimization of truss structures with discrete variables. Comput Struct 2012;102–103:49–63.
[35] Bekdaş G, Nigdeli S, Yang X. Sizing optimization of truss structures using flower pollination algorithm. Appl Soft Comput 2015;37:322–31.
[36] Degertekin S, Hayalioglu M. Sizing truss structures using teaching-learning-based optimization. Comput Struct 2013;119:177–88.
[37] Kaveh A, Mahdavi V. Colliding bodies optimization method for optimum design of truss structures with continuous variables. Adv Eng Softw 2014;70:1–12.
[38] Kaveh A, Khayatazad M. Ray optimization for size and shape optimization of truss structures. Comput Struct 2013;117:82–94.
[39] Farshchin M, Camp C, Maniat M. Multi-class teaching–learning-based optimization for truss design with frequency constraints. Eng Struct 2016;106:355–69.