Chapter 11
Tug of War Optimization Algorithm

11.1 Introduction

Kaveh and Zolghadr [1] developed a novel population-based metaheuristic algorithm inspired by the game tug of war. Utilizing a sports metaphor, the algorithm, denoted as tug of war optimization (TWO), considers each candidate solution as a team participating in a series of rope pulling competitions. The teams exert pulling forces on each other based on the quality of the solutions they represent. The competing teams move to their new positions according to Newtonian laws of mechanics. Unlike many other metaheuristic methods, the algorithm is formulated in such a way that considers the qualities of both of the interacting solutions. TWO is applicable to global optimization of discontinuous, multimodal, non-smooth, and non-convex functions. Viability of the TWO in solving engineering COPs currently is examined using some structural optimization problems [2–4]. It should be noted that TWO has a very simple structure and the available studies indicate its potential to be used for many real-size COPs.

Like other metaheuristics, TWO starts from a set of randomly generated initial candidate solutions. Each candidate solution is considered as a team, and all population form a league. In each iteration of the algorithm, after evaluation, teams are assigned by a value of weight proportional to their merit. Accordingly, the best team has the highest weight, and the worst one has the lowest weight. Consider two candidate solutions or two teams as two rival sides pulling a rope. As a rule, the lightest team will lose the competition and move toward the heaviest team. TWO idealizes the tug of war framework, to determine the resultant force affecting the defeated team due to its interaction with the heavier team. As a result, the lightest team accelerates toward the heaviest team according to Newton's second law with a calculable displacement rate based on the constant acceleration movement equation. TWO also added randomness to the formulation extracted from the Newtonian laws of mechanics. Analogously, in each iteration, the league is updated by a series of team-team rope pulling competitions with the aim of attracting teams toward the

A. Kaveh, T. Bakhshpoori, *Metaheuristics: Outlines, MATLAB Codes and Examples*, https://doi.org/10.1007/978-3-030-04067-3_11

optimum position. This forms the convergence operator of TWO. It should be noted that TWO additionally benefits a heuristic strategy to regenerate the teams are moved outside the search space which is most likely to be accrued for the lightest ones.

11.2 Formulation and Framework of TWO

In the following, after presenting the idealized tug of war framework as the essence of the TWO, the concepts and framework of TWO algorithm, its pseudo code, and the related flowchart are presented.

11.2.1 Idealized Tug of War Framework

Tug of war or rope pulling is a strength contest in which two competing teams pull on the opposite ends of a rope in an attempt to bring the rope in their direction against the pulling force of the opposing team. The activity dates back to ancient times and has continued to exist in different forms ever since. There has been a wide variety of rules and regulations for the game, but the essential part has remained almost unaltered. Naturally, as far as both teams sustain their grips of the rope, movement of the rope corresponds to the displacement of the losing team.

Triumph in a real game of tug of war generally depends on many factors and could be difficult to analyze. However, an idealized framework is utilized in TWO where two teams having weights W_i and W_j are considered as two objects lying on a smooth surface as shown in Fig. 11.1.

As a result of pulling the rope, the teams experience two equal and opposite forces (F_p) according to Newton's third law. For object i, as far as the pulling force is smaller than the maximum static friction force ($W_i\mu_s$), the object rests in its place. Otherwise the non-zero resultant force can be calculated as:

$$F_r = F_p - W_i\,\mu_k \tag{11.1}$$

As a result, the object i accelerates toward the object j according to Newton's second law:

$$a = F_r/(W_i/g) \tag{11.2}$$

Since the object i starts from a zero velocity, its new position can be determined as:

$$X_i^{new} = \frac{1}{2}a\,t^2 + X_i^{old} \tag{11.3}$$

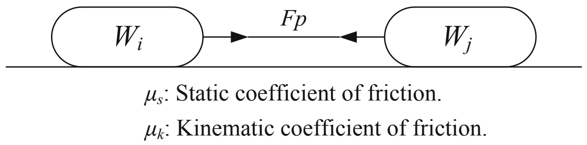
W_i
Fp
W_j
μ_s: Static coefficient of friction.
μ_k: Kinematic coefficient of friction.

Fig. 11.1 An idealized tug of war framework

11.2.2 TWO Algorithm

Looking at the idealized tug of war framework overviewed at the previous subsection, there is an analogy between it and a population-based metaheuristic. Each agent of the algorithm can be considered as a team engaged in a series of tug of war competitions. The weight of teams is determined based on the quality of the corresponding solutions, and the amount of pulling force that a team can exert on the rope is assumed to be proportional to its weight. Naturally, the opposing team will have to maintain at least the same amount of force in order to sustain its grip on the rope. The lighter team accelerates toward the heavier team, and this forms the convergence operator of TWO. The algorithm improves the quality of the solutions iteratively by maintaining a proper exploration/exploitation balance using the described convergence operator. The rules of TWO can be stated as follows:

Rule 1: Initialization
TWO starts from a set of candidate solutions or teams randomly generated within the search space. The number of teams is considered as *nT*. These particles construct the matrix of teams (*T*) or the league matrix. After evaluating the teams, the corresponding objective function (*Fit*) and the penalized objective function (*PFit*) vectors are produced.

Rule 2: Weight Assignment
Each solution is considered as a team with the following weight:

$$W_i = \frac{PFit_i - \min(PFit)}{\min(PFit) - \max(PFit)} + 1 \tag{11.4}$$

where min and max return the minimum and the maximum element of the penalized objective vector, respectively. According to this definition, the weights of the teams range between 1 and 2 so that the best is the heaviest.

Rule 3: Competition and Displacement
In TWO each of the teams of the league competes against all the others one at a time to move to its new position in each iteration. The pulling force exerted by a team is assumed to be equal to its static friction force ($W\mu_s$). μ_s is the static coefficient of friction and can be considered as 1. Hence the pulling force between teams i and j ($F_{p,ij}$) can be determined as $\max\{W_i\mu_s, W_j\mu_s\}$. Such a definition keeps the position of the heavier team unaltered. Then the resultant force affecting team i due to its interaction with heavier team j can be calculated as follows:

$$F_{r,ij} = F_{p,ij} - W_i\,\mu_k \tag{11.5}$$

where μ_k is the kinematic coefficient of friction and can be considered equal to 1. Consequently, team *i* accelerates toward team *j*:

$$a_{ij} = \left(F_{r,ij}/W_i\mu_k\right)g_{ij} \tag{11.6}$$

where g_{ij} is the gravitational acceleration constant defined as:

$$g_{ij} = T_j - T_i \tag{11.7}$$

in which T_j and T_i are the position vectors for teams *j* and *i*. Finally, the displacement of the team *i* after competing with team *j* can be derived as:

$$stepsize_{ij} = \frac{1}{2}a_{ij}\Delta T^2 + \alpha\,\beta\,(Lb - Ub)\circ randn \tag{11.8}$$

The second term of this formulation introduces randomness into the algorithm. This term can be interpreted as the random portion of the search space traveled by team *i* before it stops after the applied force is removed. The role of α is to gradually decrease the random portion of the team's movement. For most of the applications, α could be considered as a constant chosen from the interval [0.9, 0.99]; bigger values of α decrease the convergence speed of the algorithm and help the candidate solutions explore the search space more thoroughly. β is a scaling factor which can be chosen from the interval (0,1). This parameter controls the steps of the candidate solutions when moving in the search space. When the search space is supposed to be searched more accurately with smaller steps, smaller values should be chosen for this parameter. *Lb* and *Ub* are the vectors containing the lower and upper bounds of the permissible ranges of the design variables, respectively; $\circ$ denotes element by element multiplication; *randn* is a vector of random numbers drawn from a standard normal distribution. ΔT is the time step and can be considered equal to 1.

It should be noted that when team *j* is lighter than team *i*, the corresponding displacement of team *i* will be equal to zero. Finally, the total displacement of team *i* is as follows (*i* not equal *j*):

$$stepsize_i = \sum_{j=1}^{nT} stepsize_{ij} \tag{11.9}$$

The new position of the team *i* is then calculated as:

$$T_i^{new} = T_i + stepsize_i \tag{11.10}$$

Rule 4: Updating the League

Once the teams of the league compete against each other for a complete round, the league should be updated. This is done by comparing the new candidate solutions (the new positions of the teams) with the current teams of the league. That is to say if the new candidate solution *i* is better than the worst team of the league in terms of penalized objective function value, the worst team is removed from the league, and the new solution takes its place.

Rule 5: Handling the Side Constraints

It is possible for the candidate solutions to leave the search space, and it is important to deal with such solutions properly. This is especial case for the solutions corresponding to lighter teams for which the values of *stepsize* are usually bigger. Different strategies might be used in order to solve this problem. For example, such candidate solutions can be simply brought back to their previous feasible position (flyback strategy), or they can be regenerated randomly. TWO uses a new strategy incorporating the best-so-far solution (*bestT*) to handle the dimensions of teams which exit side constraints. This strategy is utilized with a certain probability (0.5). For the rest of the cases, the violated limit is taken as the new value of the *j*th design variable. Based on this strategy, which is named as best team strategy (BTS), the new value of the *j*th design variable of the *i*th team that violated side constraints in the *NITs* iteration of the algorithm is defined as:

$$T_{ij} = bestT_j + (randn/NITs)(bestT_j - T_{ij}) \tag{11.11}$$

where *randn* is a random number drawn from a standard normal distribution. There is a very slight possibility for the newly generated variable to still be outside the search space. In such cases, a flyback strategy is used.

Rule 6: Termination Criteria

A maximum number of objective function evaluations (*maxNFEs*) or a maximum number of algorithm iterations (*maxNITs*) are considered as the terminating criterion.

Based on the six rules as the essence of the TWO, the pseudo code of TWO is given as follows, and the flowchart is illustrated in Fig. 11.2.

The pseudo code of TWO algorithm for solving COPs:

```
Define the algorithm parameters: $nT$, $\mu_s$, $\mu_k$, $\alpha$, $\beta$, $\Delta T$, and $maxNFEs$.
Generate random initial teams or the random initial league ($T$).
Evaluate the initial league and form its corresponding vectors of the objective
    function ($Fit$) and penalized objective function ($PFit$).
Monitor the best candidate solution or team ($bestT$) and its corresponding objective
    function ($minFit$) and penalized objective function ($minPFit$).
While $NFEs < maxNFEs$
    Update the number of algorithm iterations ($NITs$).
    Determine the weights of the teams of the league using Eq. (11.4).
    Determine the total displacement of teams by a series of rope pulling compe-
        titions based on the equations presented in Rule 3.
```

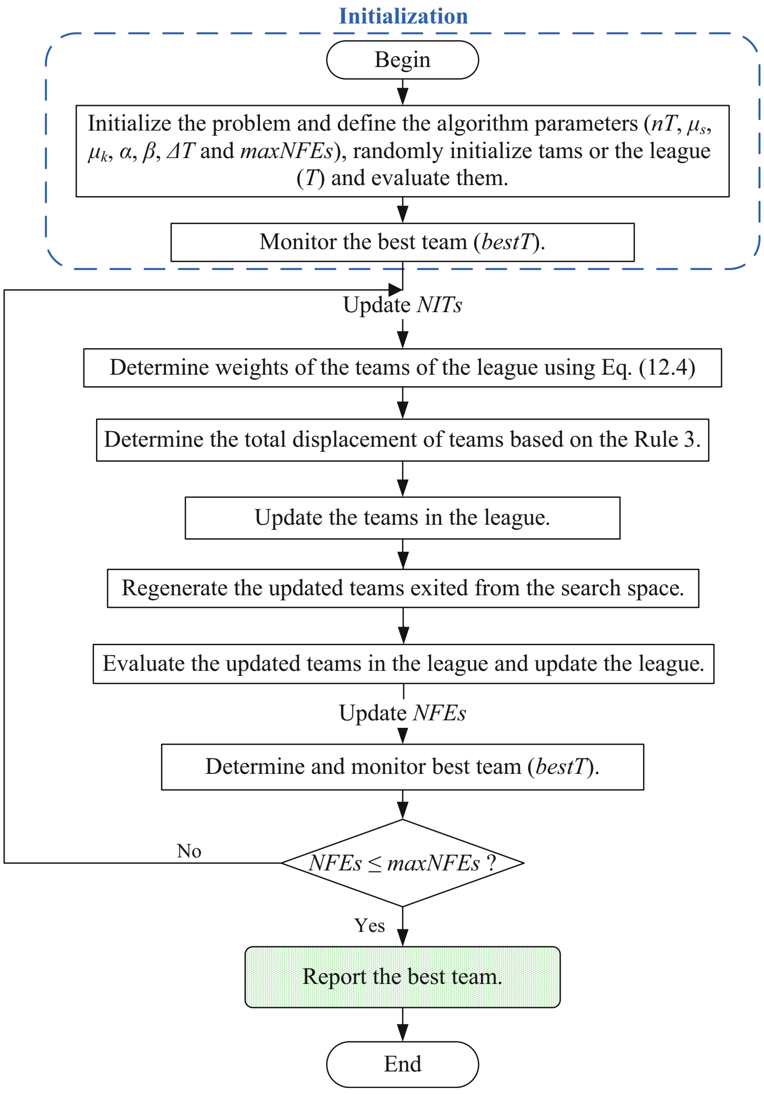
Initialization
Begin
Initialize the problem and define the algorithm parameters (nT, μ_s, μ_k, α, β, ΔT and $maxNFEs$), randomly initialize tams or the league (T) and evaluate them.
Monitor the best team ($bestT$).
Update $NITs$
Determine weights of the teams of the league using Eq. (12.4)
Determine the total displacement of teams based on the Rule 3.
Update the teams in the league.
Regenerate the updated teams exited from the search space.
Evaluate the updated teams in the league and update the league.
Update $NFEs$
Determine and monitor best team ($bestT$).
No
$NFEs \leq maxNFEs$?
Yes
Report the best team.
End

Fig. 11.2 Flowchart of the TWO algorithm

Update the teams in the league by adding the total displacement to their current position.
Regenerate the updated teams exited from the search space using the best team strategy.
Evaluate the updated teams in the league.
Update *NFEs*.

Update the league by replacing the current worst teams with the better teams produced.

Determine and monitor the best team (*bestT*) and its corresponding objective function (*minFit*) and penalized objective function (*minPFit*).

end While

11.3 Matlab Code for the Tug of War Optimization (TWO) Algorithm

According to the previous section, five functions are considered for coding the algorithm. First, these functions are presented with the order of their recall in the algorithm, and at the end, the algorithm is coded. The required comments are presented. Note that the percent sign (%) represents a comment.

The first function is the objective function evaluation (*fobj*) which was presented in Chap. 2.

The second function is named the *Weight* function to determine the weights of the teams of the league (*W*). The input argument is the penalized objective function vector (*PFit*). The output argument is the weight vector (*W*).

```
% Define the weights of the teams of the league.
function [W]=Weight(PFit)
nT=size(PFit,2);

for i=1:nT
W(i)=(PFit(i)-min(PFit))/(max(PFit)-min(PFit))+1;
end
```

The third function is named *Rope_Pulling* function in which the total displacement of teams will be determined by a series of rope pulling competitions. The input arguments are the current positions of the teams or the league (*T*), the weight vector of the league (*W*), lower and upper bound vectors of the design variables (*Lb* and *Ub*), static and kinematic coefficients of friction (μ_s and μ_k), controlling parameters (α and β) and the time step parameter (ΔT) of the step size equation, and the current number of algorithm iterations (*nIT*). The output argument is matrix of the total displacement of teams (*stepsize*). The *Rope_Pulling* function is coded as follows:

```
% Determine the total displacement of teams by a series of rope pulling
competitions.

function
[stepsize]=Rope_Pulling(T,W,Lb,Ub,meus,meuk,deltat,alpha,beta,NITs)

nT=size(T,1);
nV=size(T,2);

stepsize=zeros(size(T));

for i=1:nT
    for j=1:nT
        if W(i)<W(j)
            PF=max(W(i)*meus,W(j)*meus);  % Pulling force between teams i
and j.
            RF=PF-W(i)*meuk; % The resultant force affecting the team i due
to its interaction with the heavier team j.
            g=T(j,:)-T(i,:); % Gravitational acceleration.
            a=RF/(W(i)*meuk).*g; % Acceleration of team i towards team j.
            stepsize(i,:)=stepsize(i,:)+0.5*a*deltat^2+alpha^NITs*beta*(Lb-
Ub).*randn(1,nV);
        end
    end
end
```

The fourth function, named as *BTS* function by which the updated teams exited from the search space, regenerated using the best team strategy (Rule 5). Input arguments are the updated set of teams (*newt*), the current set of the teams of the league (*T*), the so far best-monitored team (*bestT*), lower and upper bound vectors of the design variables (*Lb* and *Ub*), and the current number of algorithm iterations (*nIT*). The output argument is the updated set of teams (*newt*) with corrected positions. The *BTS* function is coded as follows:

```
% Regenerate the teams exited from the search space using the Best Team
Strategy (BTS).

function [newT]=BTS(newT,T,bestT,Lb,Ub,NITs)

nT=size(newT,1);
nV=size(newT,2);

for i=1:nT
    for j=1:nV
        if newT(i,j)<Lb(j)||newT(i,j)>Ub(j)
            if rand<=0.5
            newT(i,j)=bestT(j)+randn/NITs*(bestT(j)-newT(i,j)); % Best Team
Strategy.
            if newT(i,j)<Lb(j)||newT(i,j)>Ub(j)
                newT(i,j)=T(i,j); % If the component is still outside the
search space, return it using the flyback strategy.
            end
            else
                if newT(i,j)<Lb(j)
                newT(i,j)=Lb(j);
                end
                if newT(i,j)>Ub(j)
                    newT(i,j)=Ub(j);
                end
            end
        end
    end
end
```

The fifth function is the *Replacement* function by which the league updated comparing the new teams with current teams of league. The input arguments are current league (*T*), its corresponding objective (*Fit*) and penalized objective (*PFit*) function vectors, updated set of teams (*newt*) and their corresponding objective (*newFit*) and penalized objective (*newPFit*) function vectors. The output arguments are updated league and its corresponding objective and penalized objective function vectors. *Replacement* function is coded as follows:

```
% Updating the league by comparing the updated teams with current teams of
league.

function [T,Fit,PFit]=Replacement(T,Fit,PFit,newT,newFit,newPFit)
nT=size(T,1);

helpT=[T;newT];helpFit=[Fit newFit];helpPFit=[PFit newPFit];

[~,order]=sort(helpPFit);

T=helpT(order(1:nT),:);
Fit=helpFit(order(1:nT));
PFit=helpPFit(order(1:nT));
```

TWO algorithm is coded in the following composed of three parts: initialization, algorithm cyclic body, and monitoring the results.

```
clc
clear

%% Initialization

% Define the properties of COP (tension/compression spring design problem).
nV=3; % Number of design variables.
Lb=[0.05 0.25 2]; % Lower bounds of design variables.
Ub=[2 1.3 15]; % Upper bounds of design variables.

% Define the parameters of TWO algorithm.
nT=10; % Number of Teams.
meus=1;meuk=1; % Static and Kinematic coefficients of friction.
deltat=1;  %  Time  step  in  the  movement  equation  with  a  constant
acceleration.
alpha=0.99; % Controlling parameter of the algorithm's randomness.
beta=0.1; % Scaling factor of team's movement.
maxNFEs=20000; % Maximum Number of Objective Function Evaluations.

%Generate random initial solutions.
for i=1:nT
    T(i,:)=Lb+(Ub-Lb).*rand(1,nV); % Teams matrix or the league or matrix
of the initial candidate solutions or the initial population.
end

%  Evaluate  initial  population  (League)  calling  the  fobj  function
constructed in the second chapter and form its corresponding vectors of
objective function (Fit) and penalized objective function (PFit). It should
be noted that the design vectors all are inside the search space.
for i=1:nT
    [X,fit,pfit]=fobj(T(i,:),Lb,Ub);
    T(i,:)=X;
    Fit(i)=fit;
    PFit(i)=pfit;
end

% Monitor the best candidate solution or team (bestT) and its corresponding
objective function (minFit) and penalized objective function (minPFit).
[minPFit,m]=min(PFit);
minFit=Fit(m);
bestT=T(m,:);

%% Algorithm Body

NFEs=0;  %  Current  Number  of  Objective  Function  Evaluations  used  by  the
algorithm until yet.
NITs=0; % Number of algorithm iterations

while NFEs<maxNFEs
    NITs=NITs+1; % Update the number of algorithm iterations.

    % Define the weights of the teams of the league.
    W=Weight(PFit);

    % Determine the total displacement of teams by a series of rope pulling
competitions.
    [stepsize]=Rope_Pulling(T,W,Lb,Ub,meus,meuk,deltat,alpha,beta,NITs);

    % Update the teams in the league.
    newT=T+stepsize;

    % Regenerate the teams exited from the search space using the Best Team
Strategy (BTS).
```

```
    [newT]=BTS(newT,T,bestT,Lb,Ub,NITs);

    % Evaluate the updated teams calling the fobj function. It should be
noted
    % that the existed dimensions are corrected before within the BTS
function. However, to do not change the form of fobj function it is checked
again here which is not needed.
    for i=1:nT
        [X,fit,pfit]=fobj(newT(i,:),Lb,Ub);
        newT(i,:)=X;
        newFit(i)=fit;
        newPFit(i)=pfit;
    end
    NFEs=NFEs+nT;

    % Update the league by the replacement strategy.
    [T,Fit,PFit]=Replacement(T,Fit,PFit,newT,newFit,newPFit);

    % Monitor the best candidate solution (bestT) and its corresponding
objective function (minFit) and penalized objective function (minPFit).
    minPFit=PFit(1);minFit=Fit(1);bestT=T(1,:);



    % Display desired information of the iteration.
    disp(['NITs= '  num2str(NITs) '; minFit = '  num2str(minFit) '; minPFit
= '  num2str(minPFit)]);

    % Save the required results for post processing and visualization of
the algorithm performance.
    output1(NITs,:)=[minFit,minPFit,NFEs];
    output2(NITs,:)=[min(PFit),max(PFit),mean(PFit)];
    output3(NITs,:)=[bestT,NFEs];
end

%% Monitoring the results
figure;
plot((1:1:NITs),output2(:,1),'g',(1:1:NITs),output2(:,2),'r--
',(1:1:NITs),output2(:,3),'b-.')
legend('min','max','mean');
xlabel('NITs');
ylabel('PFit');
```

11.4 Experimental Evaluation

TWO has seven parameters which are as follows: number of teams or the size of league (*nT*), static and kinematic coefficients of friction (μ_s and μ_k), controlling parameters (α and β) and the time step parameter (ΔT) used in the step size equation, and the maximum number of objective function evaluations (*maxNFEs*) as the stopping criteria of the algorithm.

Considering a large enough value for *maxNFEs* to ensure convergence of the algorithms is essential. This parameter almost is problem dependent and should be considered larger for complicated and larger problems. It is considered here equal to 20,000 for TWO like other chapters.

Static and kinematic coefficients of friction (μ_s and μ_k) and the time step parameter ($\varDelta T$) have no essential effect on the algorithm performance and are considered equal to 1 as recommended before.

The number of algorithm individuals in the population-based metaheuristics can be a very important parameter that influences the algorithm performance. Figure 11.3 shows the convergence history for a single run of TWO algorithm from a same initial population with a different number of teams (5–100). In these runs, α and β parameters are considered as 0.99 (the largest value of the recommended range) and 0.1 (the smallest value of the recommended range), respectively, to deep search with small step sizes. According to this figure and statistical results (not presented here) obtained from more independent runs, values of nT between 10 and 30 are efficient in making a balance between exploration and exploitation. It is worth to mention that TWO gets trapped in local optima when the nT is less than 10. On the other hand, larger values lead to a reduction of the convergence rate and accuracy.

Simulation results show that controlling parameters (α and β) used in the step size equation are the most important parameters of TWO. The recommended values and their role in the algorithm performance are completely observed. Especially the α parameter with the values outside the recommended range completely interrupts efficiency of the algorithm.

At the end it is worth to mention that although TWO defines seven parameters in its framework, however, as it is discussed above, TWO has only two easily tunable parameters (α and β) in addition to the number of algorithm agents and the maximum number of objective function evaluations as the common parameters of all metaheuristics. Therefore, TWO has a great potential in solving many COPs because of its simple structure, easy to be tuned for the problem at hand, and its special formulation in such a way that considers the qualities of both of the candidate solutions in updating the population.

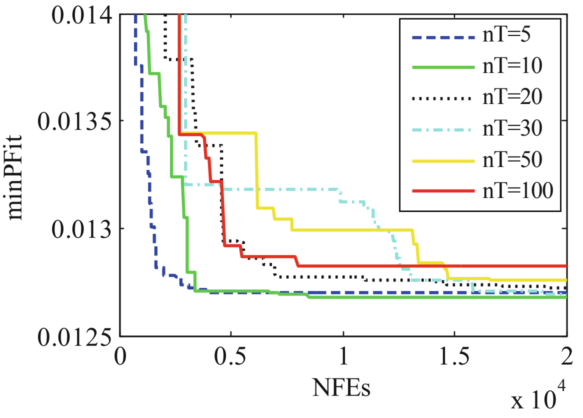

Fig. 11.3 Convergence histories of the TWO for different values of nT

References

1. Kaveh A, Zolghadr A (2016) A novel meta-heuristic algorithm: tug of war optimization. Int J Optim Civ Eng 6:469–492
2. Kaveh A, Shokohi F (2016) Optimum design of laterally-supported castellated beams using tug of war optimization algorithm. Struct Eng Mech 58:533–553
3. Kaveh A, Bijari S (2017) Bandwidth, profile and wavefront optimization using PSO, CBO, ECBO and TWO algorithms. Iran J Sci Trans Civ Eng 41:1–12
4. Kaveh A, Zolghadr A (2017) A guided modal strain energy-based approach for structural damage identification using tug-of-war optimization algorithm. J Comput Civ Eng 31(4):04017016