Information Sciences 571 (2021) 358–374

Contents lists available at ScienceDirect

Information Sciences

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

An adaptive fuzzy penalty method for constrained evolutionary optimization

Bing-Chuan Wang [a,b], Han-Xiong Li [c], Yun Feng [d,e], Wen-Jing Shen [f,*]

[a] School of Automation, Central South University, Changsha, China
[b] Hunan Xiangjiang Artificial Intelligence Academy, Changsha, China
[c] Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong
[d] College of Electrical and Information Engineering, Hunan University, Changsha 410082, China
[e] National Engineering Laboratory for Robot Visual Perception & Control Technology, Hunan University, Changsha 410082, China
[f] Sino-German College of Intelligent Manufacturing, Shenzhen Technology University, Shenzhen, China

A R T I C L E I N F O

Article history:
Received 17 March 2020
Received in revised form 21 March 2021
Accepted 29 March 2021
Available online 6 April 2021

Keywords:
Constrained optimization
Adaptive fuzzy penalty
Individual level
Population level
Mutation scheme

A B S T R A C T

Penalty function is well-known for constrained evolutionary optimization. An open question in the penalty function is how to tune the penalty coefficient. This paper proposes an adaptive fuzzy penalty method to address this issue, where the coefficient is adjusted at both the individual level and the population level. At the individual level, each individual chooses a penalty coefficient from a predefined domain according to some fuzzy rules. At the population level, the domain of the crisp output is adjusted adaptively by using population information. To enhance the population diversity, an effective mutation scheme is developed. Due to its numerous merits, differential evolution is used to design a search algorithm. By the above processes, a constrained optimization evolutionary algorithm called AFPDE is proposed. Since the objective function value and the degree of constraint violation are normalized, AFPDE is less problem-dependent than the seminal work of the fuzzy penalty method. AFPDE introduces a lower penalty value in the early stage of AFPDE while a higher one in the later stage. Thus, it can escape local optima in the infeasible region. Experiments on three well-known benchmark test sets and two mechanical design problems validate that AFPDE is competitive.

1. Introduction

Constrained optimization problems (COPs) exist widely in the real world [7]. A typical COP can be formulated as follows:

$$\text{minimize} f(\vec{x}), \vec{x} = (x_1, \dots, x_D) \in S, L_d \leqslant x_d \leqslant U_d$$

$$\text{subject to} : g_j(\vec{x}) \leqslant 0, j = 1, \dots, n_p$$

$$h_j(\vec{x}) = 0, j = n_p + 1, \dots, n_q,$$

* Corresponding author.
E-mail address: shenwenjing@sztu.edu.cn (W.-J. Shen).

https://doi.org/10.1016/j.ins.2021.03.055
0020-0255/© 2021 Elsevier Inc. All rights reserved.

Nomenclature

Notations and abbreviations

AFPDE	adaptive fuzzy penalty differential evolution
ATM	adaptive tradeoff model
CEC	congress on evolutionary computation
COEA	constrained optimization evolutionary algorithm
COP	constrained optimization problem
DE	differential evolution
EA	evolutionary algorithm
FES	function evaluations
MaxFEs	maximal function evaluations
Mean OFV	average of the objective function values obtained over 25 independent runs
MF	membership function
Std Dev	standard deviation
$\vec{x}$	decision vector/target vector/solution/individual
$\vec{x}_{best}$	best solution in the current population
$\vec{x}^*$	true optimum
$\vec{x}_i$	ith target vector
$\vec{u}_i$	trial vector
$\vec{v}_r$	offspring in the mutation scheme
$\vec{o}$	solution with the maximal degree of constraint violation
$f(\vec{x})$	objective function
$F(\vec{x})$	expanded objective function
F	scaling factor
CR	crossover control parameter
$G(\vec{x})$	degree of constraint violation
$g_j(\vec{x})$	jth inequality constraint
$h_j(\vec{x})$	jth equality constraint
D	dimensionality
n_p	number of inequality constraints
n_q	number of constraints
S	decision space
δ	tolerance value
r_f	penalty coefficient
f_{max}	maximal $f(\vec{x})$ in the current population
f_{min}	minimal $f(\vec{x})$ in the current population
$f^{norm}(\vec{x})$	normalized $f(\vec{x})$
G_{max}	maximal $G(\vec{x})$ in the current population
G_{min}	minimal $G(\vec{x})$ in the current population
$G^{norm}(\vec{x})$	normalized $G(\vec{x})$
p_f	reformulated penalty coefficient
p_{max}	upper bound of the output domain of p_f
$\mathbf{R}_i$	ith fuzzy rule
P_S	population size
$\vec{x}_{r1}$,$\vec{x}_{r2}$,$\vec{x}_{r3}$	solution randomly selected from the population
j_{rand}	integer randomly selected from $\{1,\ldots,D\}$
x_d	dth dimension of a solution
U_d	upper bound of the dth dimension of a solution
L_d	lower bound of the dth dimension of a solution
P	population
t	current generation number
T	maximal generation number
ε	threshold in the ε level controlling method
ε_0	initial value of ε
p,η	algorithm-specific parameter

where $\vec{x}$ is a D-dimensional decision vector (i.e., solution or individual); each dimension $x_d, (d = 1, \ldots, D)$ is bounded between L_d and U_d; $S = \prod_{d=1}^{D}[L_d, U_d]$ is the decision space; $f(\vec{x})$ is the objective function; $g_j(\vec{x})$ is the jth inequality constraint; $h_j(\vec{x})$ is the $(j - n_p)$th equality constraint; n_p is the number of inequality constraints, and $(n_q - n_p)$ is the number of equality constraints. Note that the black-box objective function and constraints have no simple closed forms. Classical approaches that need an analytical expression are not suitable to solve this kind of COP. Additionally, single-point-based approaches are easy to be trapped in a local optimum. Nature-inspired algorithms are population-based and do not need an analytical expression of a COP. Thus, we focus on using nature-inspired algorithms, especially evolutionary algorithms (EAs), to solve COPs.

When tackling COPs, we must quantify the degree of constraint violation (i.e., $G(\vec{x})$). Conventionally, it is calculated as follows [40]:

$$G(\vec{x}) = \sum_{j=1}^{n_q} G_j(\vec{x}), \tag{1}$$

$$G_j(\vec{x}) = \begin{cases} \max(0, g_j(\vec{x})), & 1 \leqslant j \leqslant n_p \\ \max(0, |h_j(\vec{x})| - \delta), & n_p + 1 \leqslant j \leqslant n_q \end{cases}, \tag{2}$$

where $\delta > 0$ is a tolerance value. A solution which satisfies $G(\vec{x}) = 0$ is called a feasible solution. The feasible region is composed of all feasible solutions. A constrained optimization algorithm is to seek the optimum in the feasible region.

Due to their various merits, EAs [3] have been widely used for optimization. In addition, constrained optimization has attracted much attention, and thus numerous kinds of constraint-handling techniques have been proposed [5,7,26,27,33]. In general, they can be classified into four categories:

- methods based on penalty function [22,34],
- methods based on separation of constraints and objective function [39],
- methods based on multiobjective optimization [6],
- hybrid methods [41,43].

A penalty function combines $f(\vec{x})$ with $G(\vec{x})$ by using a penalty coefficient. A method based on separation of constraints and objective function uses $f(\vec{x})$ and $G(\vec{x})$ alternatively. As its name infers, the third kind of method transforms a COP into a multiobjective optimization problem. A hybrid method tries to leverage the distinct advantages of several constraint-handling techniques. Traditionally, methods based on penalty function are frequently used because they are intuitive and have a simple structure. An open question in these methods is how to set the penalty coefficient properly.

According to the manner of setting the penalty coefficient, methods based on penalty function can be further divided into three groups:

- methods based on static penalty [14],
- methods based on dynamic penalty [22],
- methods based on adaptive penalty [19].

In a method based on static penalty, the coefficient is kept unchanged throughout the optimization process. As we know, different degrees of penalty would be needed for different problems or in different optimization stages. A constant value would impair the performance of a method based on static penalty significantly. Thus, this kind of method may not be effective to complex COPs. In a method based on dynamic penalty, the coefficient is varied with generations according to a predefined function. Although a dynamic method would be better than a static one in many cases, it seems not easy to define a general function which would be problem-dependent. In a method based on adaptive penalty, the coefficient is tuned by using population information. Since the information can reflect the evolution stage of the considered COP to some degree, a method based on adaptive penalty would be more effective than the former two kinds of methods. Most state-of-the-art methods based on penalty function fall into this category [4].

Among various methods based on adaptive penalty [19,34,48], the fuzzy penalty method is promising, since it can take advantage of the fuzzy theory. The method in [45] is recognized as the first fuzzy penalty method, where the penalty coefficient is decided by some fuzzy rules. In these rules, the original $f(\vec{x})$ and $G(\vec{x})$ are used as antecedent variables. The performance of this method is problem-dependent, because the scales of $f(\vec{x})$ and $G(\vec{x})$ are often different. IF-THEN rules are also used in [25], where a penalty value is calculated for each constraint. However, this method contains numerous parameters which must be set empirically. In [20], a fuzzy membership function (MF) is utilized to define "the solution with small $G(\vec{x})$". However, advantages of the fuzzy theory are not used adequately in this method. A recent study [34] incorporates the feasible proportion, that is, the percentage of feasible solutions in the population, into the IF-THEN rules. To be specific, a low feasible proportion would lead to a high penalty value, while a high feasible proportion would cause a low penalty value. In the early stage,[1] the feasible proportion would be very low. Thus, a high penalty value would be assigned. However, as shown in [22,41], less information of constraints must be used in the early stage. A high penalty value would trap the population into a local optimum in the infeasible region easily. If the constraints are complicated, the situation would be even worse.

In summary, the fuzzy penalty method owns numerous advantages such as ease of implementation and good generality. Moreover, it can take advantage of the fuzzy theory. In most fuzzy penalty methods, only the individual information is used to tune the coefficient, while the population information is neglected to some extent. Thus, there is an urgent need for designing a fuzzy penalty method where the penalty coefficient can be tuned at both the individual level and the population level. Based on these observations, an adaptive fuzzy penalty method which consists of two levels is proposed. At the individual level, the penalty coefficient of $\vec{x}$ is decided by some fuzzy rules, where the normalized $f(\vec{x})$ and normalized $G(\vec{x})$ are utilized as antecedent variables. At the population level, the domain of the crisp output is adjusted adaptively. By utilizing both the individual information and the population information, the adaptive fuzzy penalty method can set a penalty coefficient properly.

As we know, a constrained optimization evolutionary algorithm (COEA) includes two ingredients: a constraint-handling technique and a search algorithm. In order to use the adaptive fuzzy penalty method to solve COPs, we must combine it with a search algorithm. Due to its numerous advantages [9,28], differential evolution (DE) serves as the search algorithm. Moreover, population diversity is also critical to a COEA [41,44]. The ability of mutation schemes to introduce diversity is well-recognized [11]. Thus, a simple yet effective mutation scheme is designed to further enhance the population diversity. By using these ingredients, we propose an *a*daptive-*f*uzzy-*p*enalty-based *DE* (AFPDE). The main contributions of this paper are summarized as follows:

- An adaptive fuzzy penalty method is proposed to set a penalty coefficient properly.
- An effective search algorithm based on DE is proposed to generate solutions.
- A simple yet effective mutation scheme is proposed to enhance the population diversity.
- Since the objective function value and degree of constraint violation are normalized, AFPDE is less problem-dependent than the seminal work [45] of the fuzzy penalty method. Compared with a recent study [34], AFPDE uses a lower penalty value in the early stage while a higher one in the later stage. Thus, it can escape local optima in the infeasible region.
- Experiments on three well-known benchmark sets and two mechanical design problems demonstrate that AFPDE is competitive.

The rest of this paper is organized as follows. Section 2 introduces some basic knowledge. AFPDE is presented in Section 3. In Section 4, extensive experiments and discussions are conducted. Section 5 summarizes the conclusions and gives some future directions.

2. Preliminary knowledge

2.1. Methods based on the penalty function

In a method based on penalty function, an expanded objective function $F(\vec{x})$ used to compare solutions is formulated as follows:

$$F(\vec{x}) = f(\vec{x}) + r_f \cdot G(\vec{x}). \tag{3}$$

where r_f is a positive coefficient. It is worth noting that a more general $F(\vec{x})$ is provided in [5] where two penalty coefficients are used for inequality and equality constraints, respectively. In order to increase robustness, $f(\vec{x})$ and $G(\vec{x})$ are normalized:

$$f^{norm}(\vec{x}) = \frac{f(\vec{x}) - f_{min}}{f_{max} - f_{min}}, \tag{4}$$

$$G^{norm}(\vec{x}) = \frac{G(\vec{x}) - G_{min}}{G_{max} - G_{min}}, \tag{5}$$

where f_{max} and f_{min} are the maximal and minimal values of $f(\vec{x})$ in the current population, respectively; G_{max} and G_{min} are the maximal and minimal values of $G(\vec{x})$ in the current population, respectively.

As shown in Fig. 1(a), a solution $\vec{x}$ can divide the space that uses $f(\vec{x})$ and $G(\vec{x})$ as two coordinates into four subspaces. From the perspective of $\vec{x}$, Subspace I contains no promising information because solutions in this subspace are worse than $\vec{x}$ [39] in terms of both f and G. Inversely, Subspace III includes promising information of both constraints and objective function. Additionally, Subspace II has some promising information of constraints because solutions in this subspace own better

[1] Note that the early stage and the later stage are two frequently used terms in the community of evolutionary computation. To the best of our knowledge, there are no strict definitions of these two terms [8,34]. It is well-known that they are closely related to the number of generations. Thus, the parameters related to these two stages are adjusted according to the number of generations.

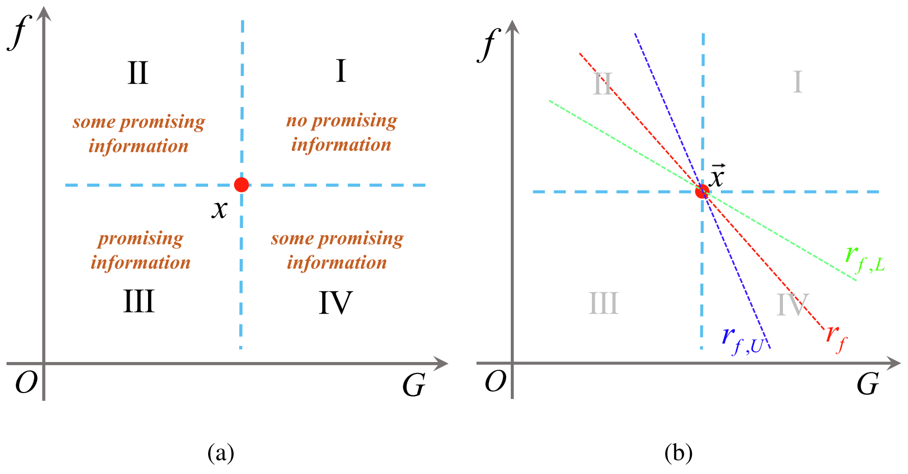

Fig. 1. Methodology of methods based on the penalty function.

values of G than $\vec{x}$. On the contrary, Subspace IV contains some promising information of objective function. The key task of a constraint-handling technique is to take advantage of the promising information in Subspaces II and IV.

As described in Fig. 1(b), in a method based on penalty function, solutions below the straight line $F = f + r_f \cdot G$ which passes through $\vec{x}$ are better than $\vec{x}$. Thus, some promising information in Subspaces II and IV can be used. Moreover, a bigger r_f (i.e., $r_{f,U} > r_f$) can leverage more information of constraints while less information of objective function. Inversely, a smaller r_f (i.e., $r_{f,L} < r_f$) can use more information of objective function while less information of constraints. Many researchers [5] have stated that the amount of information of constraints/objective function will decide the amount of exploration of infeasible regions. The exploration is important to approach a feasible optimum lying on the boundary of the feasible region or in a decision space with disjoint feasible regions. Thus, how to set a proper r_f is critical to a method based on penalty function.

2.2. Fuzzy logic theory

As shown in Fig. 2, a fuzzy logic system includes three steps:

1) Fuzzification: fuzzify crisp inputs by fuzzy MFs.
2) Fuzzy inference: infer fuzzified outputs by fuzzy rules. The Mamdani type inference engine [13,34] which uses the max–min operator for calculation is adopted conventionally.
3) Defuzzification: defuzzify fuzzified outputs.

For example, a typical system with multiple inputs and a single output is explained. It includes k rules: $\mathbf{R}_i, (i = 1, \ldots, k)$. The ith rule is described as follows:

$\mathbf{R}_i$: IF q_1 is F_{i1} AND q_2 is F_{i2} AND $\cdots$ AND q_n is F_{in} THEN y is C_i;

where q_j $(j = 1, \ldots, n)$ is the input; F_{ij} is the label of q_j; y is the output, and C_i is its label. Let $\mu_{ij}(q_j)$ be the MF of q_j on F_{ij}. Similarly, $\mu_i(y)$ is the MF of y on C_i. As a result, the firing strength of $\mathbf{R}_i$ (i.e., w_i) is calculated as follows:

$$w_i = \min_{j=1,\ldots,n}\{\mu_{ij}(q_j)\}. \tag{6}$$

Subsequently, the fuzzy implication is calculated as follows:

$$\alpha_i = \min\{w_i, \mu_i(y)\}. \tag{7}$$

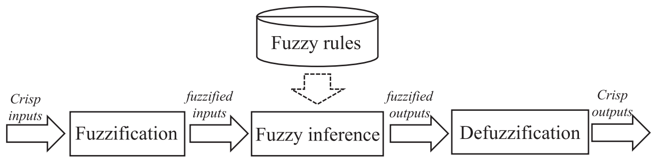

Fig. 2. Configuration of the fuzzy logic system.

Next, the fuzzy rules are aggregated as follows:

$$\mu(y) = \max_{i=1,\dots,k}\alpha_i. \tag{8}$$

Finally, the defuzzified output can be calculated by the centroid of area method:

$$\hat{y} = \int_y y\mu(y)d_y \Big/ \int_y \mu(y)d_y. \tag{9}$$

3. Proposed method–AFPDE

First, the framework of AFPDE is described in Section 3.1. Afterward, its main elements including the adaptive fuzzy penalty method, the search algorithm, and the mutation scheme are explained in Sections 3.2, 3.3, and 3.4, respectively. Finally, some critical issues are discussed in Section 3.5.

3.1. Framework

AFPDE maintains a population of P_S individuals: $P = \{\vec{x}_1, \dots, \vec{x}_{P_S}\}$. The population is updated according to the following steps until the stopping criterion is satisfied.

Step 1): Initialization
Step 1.1) Initialize and calculate related parameters.
Step 1.2) Generate a population of P_S individuals in the decision space uniformly.
Step 2): Population updating:
Step 2.1) Execute the **search algorithm** to generate offsprings.
Step 2.2) Execute the **adaptive fuzzy penalty method** to decide a penalty coefficient for each individual.
Step 2.3) Select individuals with good performance according to $F(\vec{x})$.
Step 2.4) Execute the **mutation scheme** to enhance the population diversity.
Step 3): Stopping criteria: If the stopping criterion is satisfied, then stop the procedure; otherwise, go to **Step 2)**.

3.2. Adaptive fuzzy penalty method

It seems not easy to decide a general range of r_f for all COPs. To address this issue, we set $r_f = \frac{p_f}{1-p_f}$ where $0 \leqslant p_f < 1$. Thus, Eq. (3) can be reformulated as:

$$F(\vec{x}) = \frac{(1-p_f)\cdot f^{norm}(\vec{x}) + p_f \cdot G^{norm}(\vec{x})}{1-p_f}. \tag{10}$$

In this case, a lower value of p_f corresponds to a lower penalty value, while a higher one results in a higher penalty value. It would be easier to adjust p_f than r_f, since p_f is within a certain range. Subsequently, an adaptive fuzzy penalty method including an individual level and a population level is presented to tune p_f.

3.2.1. Individual level

At the individual level, the Mamdani-type fuzzy logic is used to adjust the penalty coefficient for each individual. As described in Section 2.2, a fuzzy logic system includes three steps: fuzzification, fuzzy inference, and defuzzification. Thus, the individual level will be explained according to these three steps.

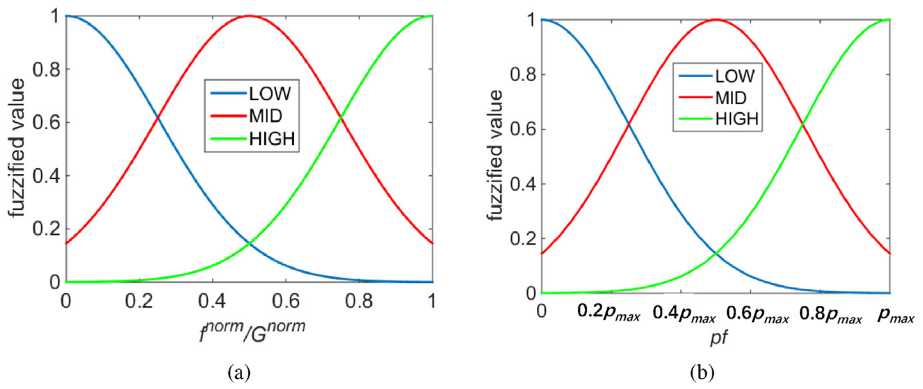

Fig. 3. The membership functions: (a) $f^{norm}(\vec{x})$ or $G^{norm}(\vec{x})$, (b) p_f.

At the step of fuzzification, fuzzy MFs need to be defined for inputs: $f^{norm}(\vec{x})$ and $G^{norm}(\vec{x})$. Similar to [34], a Gaussian-like function is chosen:

$$\mu(t|c,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-c)^2}{2\sigma^2}}, \tag{11}$$

where t is the input variable; c is the center; σ is the standard deviation. As shown in Fig. 3(a), each input variable is decomposed into three fuzzy regions which are called labels (i.e., LOW, MID, and $HIGH$). Note that MID stands for medium. The MFs of these labels are defined by

- $LOW: \mu_L(t) = \sqrt{\mu(t|0,0.18)}$,
- $MID: \mu_M(t) = \sqrt{\mu(t|0.5,0.18)}$,
- $HIGH: \mu_H(t) = \sqrt{\mu(t|1,0.18)}$.

A crisp input (i.e.,$f^{norm}(\vec{x})$ or $G^{norm}(\vec{x})$) can be fuzzified into $(\mu_L(f^{norm}(\vec{x})), \mu_M(f^{norm}(\vec{x})), \mu_H(f^{norm}(\vec{x}))$ or $(\mu_L(G^{norm}(\vec{x})), \mu_M(G^{norm}(\vec{x})), \mu_H(G^{norm}(\vec{x}))$.

At the step of fuzzy inference, the Mamdani-type inference engine is adopted, where some linguistic rules need to be designed. The valuable knowledge summarized in the community of constrained evolutionary optimization is used to design these rules. The goal of constrained optimization is to seek a feasible solution with an expected value of $f(\vec{x})$. To achieve this goal, many researchers [26,34] have found that:

- the individual with a higher value of $G(\vec{x})$ needs to be assigned a higher penalty value;
- the individual with a better value of $f(\vec{x})$ needs to be given a lower penalty value.

Noticing that, the higher the penalty value is, the higher the value of p_f is, and vice versa. Based on these discussions and references [17,18], the linear fuzzy rules for tuning p_f are designed in the following format and shown in Fig. 4:

$\mathbf{R}_i$: IF $f^{norm}(\vec{x})$ is F_{i1} AND $G^{norm}(\vec{x})$ is F_{i2}, THEN the penalty coefficient p_f is C_i.

where F_{i1} and F_{i2} are the labels of $f^{norm}(\vec{x})$ and $G^{norm}(\vec{x})$, respectively; C_i is the label of p_f, which is also chosen from $\{LOW, MID, HIGH\}$. The strength of these rules for p_f (i.e., $\mu(p_f)$) can be calculated according to Eqs. (6) and (8).

At the step of defuzzification, we first decide the domain of the crisp output p_f. It is worth noting that p_f is a value between 0 and 1. Thus, the domain is set as $[0, p_{max}]$, where $p_{max} \in [0, 1)$. Note that p_{max} will be adjusted adaptively at the population level. As shown in Fig. 3, the MFs used for defuzzification are defined as follows:

- $LOW: \mu_L(p_f) = \sqrt{\mu(p_f|0,0.18)}$,
- $MID: \mu_M(p_f) = \sqrt{\mu(p_f|0.5,0.18)}$,
- $HIGH: \mu_H(p_f) = \sqrt{\mu(p_f|1,0.18)}$.

As a result, p_f can be defuzzified according to Eq. (9) and these MFs. In summary, by these three steps, a value of p_f can be derived for $\vec{x}$.

G^{norm} \ f^{norm}	***LOW***	***MID***	***HIGH***
LOW	*LOW*	*LOW*	*MID*
MID	*LOW*	*MID*	*HIGH*
HIGH	*MID*	*HIGH*	*HIGH*

Fig. 4. Linear fuzzy rules for tuning $p_f \in [0, p_{max}]$.

3.2.2. Population level

At the population level, population information is used to adjust p_{max} adaptively. To this end, some prior knowledge and feedback information are taken into consideration. The former represents valuable findings summarized in this area and is used to define a coarse-grained trend for p_{max}, while the latter is extracted from the evolving population and is used for the fine-tuning of p_{max}.

Some well-known knowledge is used to define the coarse-grained trend function. Specifically, two critical findings [5] are used: 1) A low penalty value is beneficial in the early stage; 2) A high penalty value is necessary in the later stage. As shown in Section 3.2.1, a lower value of p_{max} will cause a lower penalty value. Thus, a low value of p_{max} is beneficial in the early stage, while a high one is necessary in the later stage. In summary, the coarse-grained trend function of p_{max} is set as follows:

$$p_{max} = \frac{t}{T}, \tag{12}$$

where t and T are the current and maximal generation numbers, respectively. As described in Eq. (12), the value of p_{max} will increase as the generation increases. Thus, a low value and a high value of p_{max} can be maintained in the early and the later stages, respectively.

Of course, this trend function could not satisfy all COPs. Especially, for some COPs, this kind of penalty increases too slow. In this case, a feasible solution could not be found because the penalty is not enough. In view of this, the coarse-grained p_{max} needs to be fine-tuned by using the feedback information of the population. Herein, the minimal $G(\vec{x})$ in the current population (i.e., G_{min}) is considered as the feedback information. Intuitively, if the penalty is not enough for a COP, G_{min} would not decrease continuously. In order to judge whether the penalty is sufficient, we set a threshold (i.e., ε) for G_{min} at each generation. If G_{min} could not achieve ε, the penalty would not be enough. In this case, p_{max} would be set to a value that is close to 1 to impose a high penalty. Based on [35], the ε level controlling method is utilized to set the threshold:

$$\varepsilon = \begin{cases} \varepsilon_0 (1 - \frac{t}{T})^{cp}, & \text{if } \frac{t}{T} \leqslant p \\ 0, & \text{otherwise} \end{cases}. \tag{13}$$

By setting the truncation precision as $\varepsilon_0(1-p)^{cp} = 10^{-\lambda}$, we obtain:

$$cp = -\frac{\log \varepsilon_0 + \lambda}{\log(1-p)}, \tag{14}$$

where ε_0 is the initial threshold; p is the truncation parameter. Note that ε_0 is set according to the maximal $G(\vec{x})$ in the initial population. Both ε_0 and λ are not very sensitive in the ε level controlling method [35,41]. Additionally, we have done extensive experiments to decide the value of p. As the same in [35], λ is set to 5, and its effective range is investigated experimentally. Therefore, the ε level controlling method can be used easily in practice.

A simple example is given as follows for better understanding. In a typical run for g01 from the IEEE CEC2006 competition [21], ε_0 is initialized as 1064.29. In the empirical study, p is set to 0.6. Thus, cp can be calculated according to Eq. (14): $cp = 20.17$. If $t < 0.6T$, the threshold ε would decrease according to $\varepsilon = 1064.29(1 - \frac{t}{T})^{20.17}$; Otherwise, it would be truncated to 0. In summary, ε decreases as the generation increases. The population information can be used properly to tune p_{max}.

In summary, each solution can obtain a proper value of p_f through the adaptive fuzzy penalty method. Details of this method are summarized in **Algorithm 1**.

Algorithm 1: Adaptive fuzzy penalty method

```
1: Population level:
2: Calculate the threshold $\varepsilon$ according to Eq. (13) and Eq. (14).
3: Set $p_{max}$ according to Eq. (12): $p_{max} = \frac{t}{T}$.
4: Calculate the minimal degree of constraint violation $G_{min}$ in the population.
5: IF $G_{min} > \varepsilon$, THEN set $p_{max}$ to a value that is close to 1.
6: Individual level:
7: Normalize $f(\vec{x})$ and $G(\vec{x})$ for each individual according to Eq. (4) and Eq. (5), respectively.
8: For each individual, fuzzified $f^{norm}(\vec{x})$ and $G^{norm}(\vec{x})$ according to the MFs defined in Fig. 3(a).
9: For each individual, infer the fuzzified outputs according to Eq. (6) and Eq. (8).
10: For each individual, calcualte $p_f$ according to Eq. (9) and the MFs defined in Fig. 3(b).
```

3.3. Search algorithm

The search algorithm, which is used to generate new solutions, is another important ingredient of a COEA. It is known that exploration and exploitation [8] are two critical criteria used to design an effective search algorithm. Because of its simplicity and powerful search ability, DE [9,28] is adopted. Specifically, "DE/current-to-rand/1" is used for exploration, while "DE/rand-to-best/1/bin" is utilized for exploitation:

DE/current-to-rand/1

$$\vec{u}_i = \vec{x}_i + rand \cdot (\vec{x}_{r1} - \vec{x}_i) + F \cdot (\vec{x}_{r2} - \vec{x}_{r3}), \tag{15}$$

DE/rand-to-best/1/bin

$$\vec{v}_i = \vec{x}_{r1} + F \cdot (\vec{x}_{best} - \vec{x}_{r1}) + F \cdot (\vec{x}_{r2} - \vec{x}_{r3}), \tag{16}$$

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } rand_j < CR \text{ or } j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases}, \tag{17}$$

where $\vec{x}_i$, $\vec{v}_i$, and $\vec{u}_i$ are the target vector, mutant vector, and trial vector, respectively; $\vec{x}_{r1}, \vec{x}_{r2}$, and $\vec{x}_{r3}$ are three mutually different solutions chosen from the population; F is the scaling factor; $\vec{x}_{best}$ is the best solution in the current population; *rand* and $rand_j$ are two random values generated from the uniform distribution between 0 and 1; CR is the crossover control parameter; j_{rand} is a random integer chosen from $\{1, \ldots, D\}$.

Algorithm 2: Search algorithm

```
1  for $i = 1$ to $P_S$ do
2      if $rand < \frac{\eta^{t/T}}{\eta}$ then
3          Select $\vec{x}_{r1}$, $\vec{x}_{r2}$, and $\vec{x}_{r3}$ from $P/\vec{x}_i$ randomly;
4          Select $\vec{x}_{best}$ based on $F(\vec{x})$ in Eq.(10);
5          Generate an offspring $\vec{u}_i$ according to Eq.(16) and Eq.(17);
6      else
7          Select $\vec{x}_{r1}$, $\vec{x}_{r2}$, and $\vec{x}_{r3}$ from $P/\vec{x}_i$ randomly;
8          Generate an offspring $\vec{u}_i$ according to Eq.(15);
9      end
10 end
```

As described in Eq. (15), "DE/current-to-rand/1" uses a differential vector $(\vec{x}_{r1} - \vec{x}_i)$ to generate a mutant vector. This differential vector will motivate $\vec{x}_i$ to approach $\vec{x}_{r1}$ which is selected from the current population randomly. Thus, "DE/current-to-rand/1" is able to promote the exploration by using the information of a random solution. As shown in Eq. (16), "DE/rand-to-best/1/bin" uses a differential vector $(\vec{x}_{best} - \vec{x}_{r1})$ to generate a mutant vector. Since the information of the best solution $\vec{x}_{best}$ is used, "DE/rand-to-best/1/bin" can accelerate the convergence.

These two operators are combined in a compact manner. For each solution, if $rand < \frac{\eta^{t/T}}{\eta}$, "DE/rand-to-best/1/bin" would be executed; otherwise, "DE/current-to-rand/1" would be used. Note that $\eta > 1$ is a user-defined parameter and its sensitivity will be analyzed in Section 4.6. In the early stage, $\frac{\eta^{t/T}}{\eta}$ is a small value. "DE/current-to-rand/1" will be utilized more frequently than "DE/rand-to-best/1/bin". Thus, the population diversity can be enhanced. In the later stage, $\frac{\eta^{t/T}}{\eta}$ becomes big. Inversely, "DE/rand-to-best/1/bin" will be used more frequently. Thus, the convergence can be accelerated. In this manner, a tradeoff between exploration and exploitation can be achieved.

Note that $\vec{x}_{best}$ is selected based on $F(\vec{x})$ in Eq. (10). Inspired by our previous studies [41,44] which are effective to solve COPs, we select F randomly from the pool of {0.6, 0.8, 1.0} and CR from the pool of {0.1, 0.2, 1.0}. Details of the search algorithm are described in **Algorithm 2**.

3.4. Mutation scheme

In practice, many COPs have complicated constraints. They would contain some local optima in the infeasible region, where the population can be stuck easily. In fact, the population diversity is critical to escaping a local optimum. In order to tackle complicated constraints, a mutation scheme is developed to increase the population diversity. In this scheme, one dimension of a solution is regenerated randomly between its upper and lower bounds. In this manner, the new solution would be located anywhere in the decision space. Thus, it could introduce some disturbances to the population. In order to avoid too many disturbances, only one solution is mutated at each generation.

In fact, the scheme is similar to a steady state genetic algorithm. First, a solution $\vec{v}_r$ is randomly selected from the population. Subsequently, a random dimension $d, (d \in \{1, \ldots, D\})$ of $\vec{v}_r$ is regenerated between the upper bound U_d and the lower bound L_d:

$$v_{r,d} = L_d + rand \cdot (U_d - L_d). \tag{18}$$

Next, we compare $\vec{v}_r$ with $\vec{o}$ which is the solution with the maximal $G(\vec{x})$. By comparing $\vec{v}_r$ with $\vec{o}$, the risk of eliminating a solution with small $G(\vec{x})$ will be reduced. To be specific, if $f(\vec{v}_r) < f(\vec{o})$ or $G(\vec{v}_r) < G(\vec{o})$, we would replace $\vec{o}$ with $\vec{v}_r$. Details of the mutation scheme are summarized in **Algorithm 3**.

Algorithm 3: Mutation scheme

```
1: Randomly select a solution from the population and copy it to $\vec{v}_r$;
2: Randomly select a dimension from $\{1, \ldots, D\}$ and denote it as $d$;
3: Regenerate the $d$th dimension of $\vec{v}_r$ according to Eq. (18);
4: Select the solution with the maximal $G(\vec{x})$ from the population and denote it as $\vec{o}$;
5: IF $f(\vec{v}_r) < f(\vec{o})$ or $G(\vec{v}_r) < G(\vec{o})$, THEN replace $\vec{o}$ with $\vec{v}_r$
```

3.5. Discussions

- An adaptive fuzzy penalty method that introduces a penalty term into the objective function is presented. The same approach has been used in our previous studies [40,41]. The main difference among these methods is how to tune the penalty coefficient/weight value, which is critical to the tradeoff between constraints and objective function. We design an adaptive fuzzy method to achieve this aim in AFPDE.
- The search algorithm also plays a key role in a COEA. Due to its simple structure and powerful search ability [9,28], DE has been used to design search algorithms in most recent COEAs [23,29,34,41]. Moreover, constrained DE algorithms achieve the first rank in the CEC2017 and CEC2018 competitions [30] for constrained optimization. It implies that DE may be more appropriate for solving COPs than other nature-inspired algorithms. Thus, we take advantage of DE to design the search algorithm.
- The computational time complexity of AFPDE is analyzed as follows. First, we analyze the computational time complexity of the adaptive fuzzy penalty method which is mainly determined by the individual level. At this level, the Mamdani type fuzzy inference system is used to decide a value of p_f for each individual. According to [13,15,16,38], the computational time complexity of the firing strength calculation is $O(2 \cdot l^n \cdot n)$, where n is the number of input variables in a fuzzy rule, and l is the number of linguistic values. The computational time complexity of the fuzzy implication is $O(2 \cdot l^n \cdot N_k)$, where N_k denotes the number of discretized points of the output variable. The computational time complexity of the aggregation is $O(l^n \cdot N_k)$. We assume that addition, multiplication, and division all need only one operation. Thus, the computational time complexity of the defuzzification is $O(3N_k)$. Since the above steps are executed in an additive manner, the computational time complexity of the adaptive fuzzy penalty method is $O(l^n \cdot (3N_k + 2n))$. The computational time complexity of the search algorithm is the same as that of "DE/rand/1/bin" [10] (i.e., $O(D \cdot P_S)$). At each generation, the adaptive fuzzy penalty method is executed P_S times. Thus, its computational time complexity is $O(P_S \cdot (l^n \cdot (3N_k + 2n)))$. In the mutation scheme, only one dimension of one individual is regenerated. Its computational time complexity is $O(1)$. The above steps are executed in an additive manner and repeated for T generations. In summary, the computational time complexity of AFPDE is $O(T \cdot P_S \cdot (l^n \cdot (3N_k + 2n) + D))$.

4. Empirical study

The empirical study contains three parts. The first part (i.e., Section 4.1) introduces the experimental settings. In the second part (i.e., Section 4.2), AFPDE is compared with some state-of-the-art methods based on three benchmark test sets and two mechanical design problems. The third part (i.e., Section 4.3) presents some further analyses.

4.1. Benchmark test functions and parameter settings

We first assessed the performance of AFPDE on three sets of benchmark test functions, which cover various challenging properties. To be specific, they include 22, 36, and 56 test functions, which were used for the IEEE CEC2006 competition [21], the IEEE CEC2010 competition [24], and the IEEE CEC2017 competition [46], respectively. These test functions have been widely used for performance assessment in the community of constrained evolutionary optimization [34,37]. The common parameters used in these three sets were given in [21,24]. As described in Table 1, the maximal function evaluations (i.e., *MaxFEs*) and the population sizes (i.e., P_S) of each test set are different. The maiximal generation number can be approximated as $\left[T = \frac{MaxFEs}{P_S}\right]$ where $[.]$ denotes the rounding function. According to the suggestions in [21,24,46], each algorithm

Table 1
Maximal function evaluations *MaxFEs* and population size P_S.

Test Functions	*MaxFEs*	P_S
twenty-two test functions from the IEEE CEC2006	5.0E+05	80
eighteen 10D test functions from the IEEE CEC2010	2.0E+05	60
eighteen 30D test functions from the IEEE CEC2010	6.0E+05	80
twenty-eight 50D test functions from the IEEE CEC2017	1.0E+06	80
twenty-eight 100D test functions from the IEEE CEC2017	2.0E+06	100

was run 25 times independently for each test function. The tolerance value δ was set to the standard value suggested in [21] (i.e., 10^{-4}). In some studies [23], it was initialized to a big value and adjusted adaptively. Note that it converged to 10^{-4} eventually. The other parameters were set as follows: $\varepsilon_0 = \min\{G_{max}, 10^{D/2}\}, p = 0.6$, and $\eta = 4.5$.

4.2. Performance comparison with other algorithms

4.2.1. Performance comparison on the 22 benchmark test functions from the IEEE CEC2006 competition

First, we evaluated the performance of AFPDE based on test functions used for the IEEE CEC2006 competition. Specifically, we compared it with four recent COEAs: ITLBO [40], DW [29], fpenalty [34], and CACDE [47]. As discussed in Section 3.5, DE reveals excellent performance in solving COPs. Among the four competitors, DW, fpenalty, and CACDE use DE as search algorithms. Additionally, ITLBO is based on another kind of nature-inspired algorithm, which shows satisfactory performance in solving COPs. The experimental results of AFPDE and the four competitors were reported in Table S1 in the supplementary file, where "Mean OFV" and "Std Dev" denote the average and standard deviation of the objective function values obtained over 25 independent runs, respectively. In [21], the condition $f(\vec{x}_{best}) - f(\vec{x}^*) < 10^{-4}$ was used to judge whether a COEA is able to find the known optimum $\vec{x}^*$. For a test function, if a COEA can achieve this condition consistently over all 25 runs, we would record a "*" in the table. As shown in Table S1, AFPDE and CACDE can find the optima of all test functions successfully. Unfortunately, ITLBO, DW, and fpenalty cannot find the optima of six, one, and five test functions, respectively. We compared AFPDE with other COEAs statistically using non-parametric Wilcoxon's signed ranks test [2,12]. Specifically, the test was used to compare each pair of COEAs over 22 test functions simultaneously. The results were summarized in Table 2. Particularly, R^+ denotes the sum of ranks for the functions in which the first COEA outperforms the second one, and R^- the sum of ranks for the opposite. As shown in Table 2, the R^+ values are bigger than the R^- values in the first three cases. We also compared all COEAs based on the Friedman's test [12]. The results in Fig. 5(a) show that both AFPDE and CACDE achieve the first rank. The above analysis shows that AFPDE can solve this set of benchmark test functions successfully. However, the competitors can also solve most of the test functions, because these test functions are relatively easy and well-studied. To further validate the effectiveness of AFPDE, two more complicated sets of benchmark test functions were adopted in the following subsections.

4.2.2. Performance comparison on the 36 benchmark test functions from the IEEE CEC2010 competition

In this subsection, we used the second set of test functions to further compare AFPDE with the competitors. In fact, this set includes eighteen 10-dimension (10D) test functions and eighteen 30-dimension (30D) ones. Since the true optima have not provided in [24], we used "Mean OFV" for comparisons directly. To this end, we calculated the rank value of each COEA. Specifically, we first ranked all COEAs on each test function based on "Mean OFV". Subsequently, for each COEA, we calculated the total rank value by summing its rank values of all test functions. Intuitively, a COEA with a smaller rank value is better than that with a bigger one. In addition, we implemented the Wilcoxon's signed ranks test and the Friedman's test to test statistical significance.

For the 10D test functions, we summarized "Mean OFV", "Std Dev", and the rank values in Table S2. We also reported the number of test functions on which a COEA achieves the best results. In the table, "Δ" represents that a COEA cannot get a feasible solution. As shown in Table S2, AFPDE obtains the smallest rank value. Additionally, it achieves the best results for 10 out of 18 test instances. It implies that AFPDE outperforms the other four peer COEAs. As shown in Table 3, the R^+ values are bigger than the R^- values in all cases. It means that AFPDE outperforms the other four competitors. Moreover, the significant difference at $\alpha = 0.05$ can be observed in one case (i.e., AFPDE versus fpenalty). Results of the Friedman's test

Table 2
Results of the Wilcoxon's signed ranks test for AFPDE and the other four peer methods on twenty-two test functions from the IEEE CEC2006 competition.

Algorithm	R^+	R^-	*p*-value	α = 0.1	α = 0.05
ITLBO	144.5	86.5	$\geqslant$0.2	No	No
DW	126.0	105.0	$\geqslant$0.2	No	No
fpenalty	145.5	85.5	$\geqslant$0.2	No	No
CACDE	126.5	126.5	$\geqslant$0.2	No	No

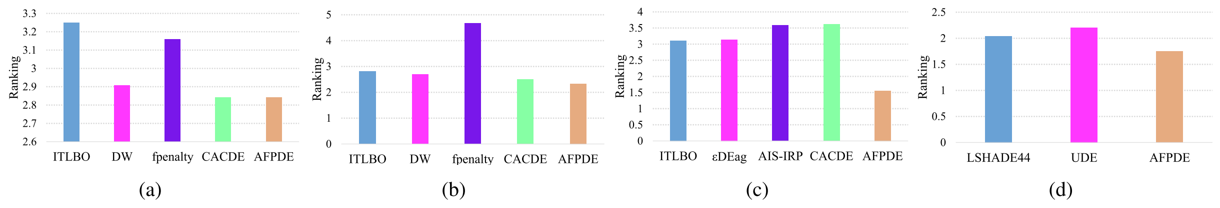

Fig. 5. Ranking of AFPDE and the competitors by the Friedman's test on benchmark test functions: (a) IEEE CEC2006, (b) IEEE CEC2010 with 10D, (c) IEEE CEC2010 with 30D, (d) IEEE CEC2017.

Table 3
Results of the Wilcoxon's signed ranks test for AFPDE and the other four selected methods on eighteen 10D test functions from the IEEE CEC2010 competition.

Algorithm	R^+	R^-	p-value	α = 0.1	α = 0.05
ITLBO	115.0	56.0	$\geqslant$0.2	No	No
DW	100.5	52.5	$\geqslant$0.2	No	No
fpenalty	157.0	14.0	8.392E-04	Yes	Yes
CACDE	94.5	58.5	$\geqslant$0.2	No	No

Table 4
Results of the Wilcoxon's signed ranks test for AFPDE and the other four selected methods on eighteen 30D test functions from the IEEE CEC2010 competition.

Algorithm	R^+	R^-	p-value	α=0.1	α=0.05
ITLBO	155.5	15.5	1.1673E−03	Yes	Yes
εDEag	147.0	6.0	2.1360E−04	Yes	Yes
AIS-IRP	141.5	11.5	9.5370E−04	Yes	Yes
CACDE	165.0	6.0	1.0682E−04	Yes	Yes

in Fig. 5(b) show that AFPDE achieves the first rank. In summary, we can conclude that AFPDE has competitive performance on the 10D test functions.

Results of the 30D test functions were reported in Table S3. Since DW and fpenalty do not provide the experimental results of the 30D test functions, we replaced these two COEAs by εDEag [35] and AIS-IRP [49]. Note that εDEag is the winner in the IEEE CEC2010 competition for single-objective constrained optimization. AIS-IRP is based on the artificial immune system and obtains outperformed results on these functions. As shown in Table S3, the rank value of AFPDE is smaller than that of the other four peer algorithms. AFPDE obtains the best results for 14 test instances. It implies that AFPDE outperforms the other four algorithms. Similarly, results of the Wilcoxon's signed ranks test in Table 4 reflect that AFPDE performs better than the other four algorithms. The significant difference at $\alpha = 0.05$ can be observed in all cases. What's more, as shown in Fig. 5(c), AFPDE achieves the first rank in the Friedman's test. These experimental results and discussions show that AFPDE is able to solve the 30D test functions successfully.

4.2.3. Performance comparison on the 56 benchmark test functions from the IEEE CEC2017 competition

In order to further demonstrate the performance of AFPDE on high-dimensional test functions, we evaluated it based on twenty-eight 50-dimension (50D) and twenty-eight 100-dimension (100D) test functions from the IEEE CEC2017 competition. Afterward, its results were compared with that of LSHADE44[2] [30] and UDE[3] [36], which are the first two winners in the IEEE CEC2017 competition for constrained optimization. To this end, we first ranked these three algorithms on each test function according to the procedure in [46]. A value would be considered as 0 if it is smaller than 10^{-9}. Afterward, the total rank value of each algorithm on all test functions was calculated. Experimental results were reported in Table S4 in the supplementary file. In Table S4, "*voi*" represents the average of the degree of constraint violation obtained over 25 runs. "*FR*" denotes the percentage of runs where a feasible solution can be found. As shown in Table S4, AFPDE obtains the lowest rank on both the 50D and the 100D test functions. Additionally, results of the Wilcoxon's signed ranks test in Table 5 show that AFPDE performs better than LSHADE44 and UDE. The significant difference at $\alpha = 0.05$ can be observed in both cases. Moreover, AFPDE ranks the first in the Friedman's test. The above analyses show that AFPDE is effective to solve high-dimensional test functions.

4.2.4. Performance comparison on two mechanical design problems

Many mechanical design problems have constraints and are formulated as COPs. In this section, AFPDE was applied to solve two mechanical design problems. The step-cone pulley problem, which is described as Example 1.3 in [32], is to

[2] LSHADE44 is a modification of the success-history-based parameter adaptation of differential evolution using linear population size reduction, where the string '44' in the label of the algorithm indicates four competing DE strategies

[3] UDE denotes the unified DE.

Table 5
Results of the Wilcoxon's signed ranks test for AFPDE and the other two selected methods on 56 test functions from the IEEE CEC2017 competition.

Algorithm	R^+	R^-	p-value	α = 0.1	α = 0.05
LSHADE44	976.5	619.5	4.45E−02	Yes	Yes
UDE	1163.5	432.5	2.13E−02	Yes	Yes

minimize the weight of a four step-cone pulley. The explicit formulation and details of the problem are referred to [32]. We applied AFPDE to solve this problem and compared its performance with that of three COEAs. Note that the *MaxFEs* (i.e., 15000) were the same for all COEAs. Each algorithm was run 100 times independently. The best, mean, and worst objective function values were summarized in Table 6. TLBO and ABC are two nature-inspired algorithms that have been used to solve mechanical design problems successfully. FROFI is a state-of-the-art COEA based on DE which has revealed satisfactory performance in engineering optimization. Among these four methods, AFPDE obtains the best performance. Thus, it can solve this problem accurately and robustly.

The second problem is to optimize the parameters of a hydrodynamic thrust bearing with the aim of minimizing the power loss. More details and the explicit formulation of this problem are referred to [31]. Similarly, we applied AFPDE to solve this problem. The *MaxFEs* (i.e., 25000) were the same for all algorithms. Each algorithm was run 100 times independently and the best, mean, and worst objective function values were summarized in Table 6. AFPDE outperforms the other three algorithms consistently. It reflects that AFPDE is able to solve this problem accurately and robustly.

In summary, these experiments show that AFPDE can solve COPs with various characteristics including discontinuous and tiny feasible regions. Since a single value (i.e., $G(\vec{x})$) is used to represent the constraint violation of all constraints as the same in many other studies [34,43], AFPDE would be not proper to tackle COPs in which the constraints are of unequal importance. Additionally, it outperforms some nature-inspired algorithms including ITLBO. ITLBO has shown better performance than some nature-inspired algorithms which do not require hyperparameter tuning [40]. It implies that DE would be more proper than some nature-inspired algorithms for constrained optimization. This is in line with the discussions in Section 3.5.

4.3. Further analysis

4.3.1. Effectiveness of the adaptive fuzzy penalty method

To validate the effectiveness of the adaptive fuzzy penalty method, we compared it with the penalty method in the adaptive tradeoff model (ATM) [43]. The penalty method in ATM can only be used when the population contains feasible solutions. Thus, we implemented a variant called AFPDE-ATM by combining ATM with the adaptive fuzzy penalty method. If the solutions are all infeasible, the adaptive fuzzy penalty method would be used; otherwise, the penalty method in ATM would be adopted. To investigate the impact of the tolerance value δ, AFPDE-ATM is enhanced by setting δ adaptively. The resultant variant is called AFPDE-AATM.

We first evaluated AFPDE, AFPDE-ATM, and AFPDE-AATM on C01–C18 with 30D. "Mean OFV" and "Std Dev" were reported in Table S5 in the supplementary file. Afterward, we compared each of AFPDE-ATM and AFPDE-AATM with AFPDE based on a pairwise statistical test, that is, the Wilcoxon's rank sum test at a 0.05 significance level. In Table S5, "−", "≈", and "+" represent that a variant performs worse than, similarly to, and better than AFPDE, respectively. AFPDE performs better than AFPDE-ATM and AFPDE-AATM on eight and seven test functions, respectively. On the contrary, AFPDE-ATM and AFPDE-AATM are better than AFPDE on zero and two test functions, respectively. In ATM, the degree of constraint violation is used to reformulate the objective function and construct the final fitness function. In this case, too much emphasis would be put on constraints. Thus, AFPDE-ATM performs worse than AFPDE on C05, C09, C10, and C15 which need much information of objective function. Although the manner of setting δ adaptively can introducing some information of objective function, it would induce a negative impact. For example, AFPDE-AATM performs worse than AFPDE and AFPDE-ATM on C04 and C11. The investigation of the impact is out of the scope of this paper and it would be focused on in the future. These discussions show that the adaptive fuzzy penalty method is effective to handle constraints.

4.3.2. Effectiveness of the individual level and the population level

The adaptive fuzzy penalty method includes an individual level and a population level. In order to investigate the impact of these two levels, we removed the individual level and the population level from AFPDE and obtained AFPDE-WoI and

Table 6
The comparison results of different methods on mechanical design problems.

Algorithm	step-cone pulley			hydrodynamic thrust bearing design		
	Best	Mean	Worst	Best	Mean	Worst
TLBO [31]	16.634510	24.011358	74.022951	1625.443000	1797.707980	2096.801270
ABC [1]	16.634655	36.099500	145.470500	1625.442760	1861.554000	5144.836000
FROFI [44]	14.467584	14.467699	14.468038	1625.449568	1663.562923	1869.449075
AFPDE	**14.467560**	**14.467560**	**14.467560**	**1625.442759**	**1652.258219**	**1847.083241**

AFPDE-WoP, respectively. Noticing that, in AFPDE-WoI, all solutions used p_{max} as the penalty coefficient. In AFPDE-WoP, p_{max} was set to a constant value of 1. Next, we evaluated AFPDE, AFPDE-WoI, and AFPDE-WoP based on C01–C18 with 30D. The results were reported in Table S6. If a method could not find a feasible solution over all runs, we would record the feasible rate, that is, the percentage of feasible runs. In Table S6, AFPDE performs better than AFPDE-WoI on four test instances, while it is worse than AFPDE-WoI on two test instances. Moreover, AFPDE-WoI fails to find a feasible solution of C17 over seven runs. It indicates that the individual level is important to the adaptive fuzzy penalty method. Compared with AFPDE-WoP, AFPDE obtains better results on twelve test functions. However, AFPDE-WoP fails to obtain a feasible solution of these twelve test functions consistently. Moreover, AFPDE-WoP also performs worse than AFPDE on the rest test instances. It implies that the population level is critical to the adaptive fuzzy penalty method. Additionally, it seems that the population level is more important than the individual level. The reason may be that the general trend of the penalty coefficient is decided by the population level.

4.3.3. Effectiveness of the mutation scheme

We developed a mutation scheme to enhance the population diversity which can help AFPDE escape a local optimum. In order to investigate its effectiveness, we removed the mutation scheme from AFPDE and obtained a competitor called AFPDE-WoM. Afterward, we evaluated AFPDE and AFPDE-WoM based on g01-g24 and C01-C18. Similarly, the test functions on which AFPDE-WoM and AFPDE have significant performance difference were summarized in Table S7. As shown in Table S7, AFPDE-WoM fails to obtain feasible solutions on seven test functions over all runs. Fortunately, with the aid of the mutation scheme, AFPDE can find a feasible solution on these test instances. Thus, the mutation scheme is able to enhance the population diversity of AFPDE.

4.3.4. Impact of the fuzzy inference engine and the fuzzifier

To investigate the impact of the fuzzy inference engine, we developed three variants (i.e., AFPDE-Luk, AFPDE-Zad, and AFPDE-DR) where Lukasiewicz inference engine, Zadeh inference engine, and Dienes-Rescher inference engine were utilized, respectively. Details of these three inference engines are referred to [42]. We evaluated these three variants and AFPDE by using 18 test functions with 30D from the IEEE CEC2010 competition. The Wilcoxon's rank sum test at a 0.05 significance level was used to compare each of these three variants with AFPDE. The experimental resutls were reported in Table S8 in the supplementary file. In Table S8, "$-$", "$\approx$", and "+" represent that a variant performs worse than, similarly to, and better than AFPDE, respectively. As shown in Table S8, AFPDE-Luk, AFPDE-Zad, and AFPDE-DR perform better than AFPDE on one, two, and one test functions, respectively. Inversely, AFPDE is better than these three variants on zero, zero, and one test functions, respectively. The results show that these three variants perform slightly better than or similarly to AFPDE. To investigate the impact of the Gaussian fuzzifier, we used it in AFPDE-Luk, AFPDE-Zad, AFPDE-DR, and AFPDE. As a result, we obtained four variants called AFPDE-LukGau, AFPDE-ZadGau, AFPDE-DRGau, and AFPDE-Gau, respectively. We compared each of them with AFPDE based on 18 test functions with 30D from the IEEE CEC2010 competition. The experimental results were reported in Table S9 in the supplementary file. As shown in Table S9, AFPDE-LukGau, AFPDE-ZadGau, AFPDE-DRGau, and AFPDE-Gau perform better than AFPDE on two, one, two, and one test functions, respectively. On the contrary, AFPDE is better than these four variants on zero, zero, zero, and one test functions, respectively. In summary, these four variants perform slightly better than or similarly to AFPDE.

These experimental results show that a more complex inference engine and the Gaussian fuzzifier could not improve the performance of AFPDE significantly. The reasons may be twofold. On one hand, when the fuzzy logic is used to set a penalty coefficient, the fuzzy rules may be more important than the inference engine and the fuzzifier. To the best of our knowledge, most fuzzy penalty methods focus on designing effective fuzzy rules, and they adopt the Mamdani type fuzzy logic for inference [34,45] directly. On the other hand, the adaptive fuzzy penalty method includes an individual level and a population level. Experimental results and discussions in Section 4.3.2 show that the population level is more important than the individual level. Since the inference engine and the fuzzifier are used in the individual level, they would have less impact on the performance of AFPDE than the population level.

4.3.5. Convergence analysis

The convergence graphs of AFPDE, ITLBO, and FROFI on C01-C18 with 30D were plotted in Figs. S1 and S2 in the supplementary file. Since the source codes of the other COEAs cannot be obtained, their convergence graphs were not given. In Figs. S1 and S2, f and FES denote the objective function value and function evaluations, respectively. We compared these methods based on the FES used to locate the final solutions and the FES used to find a feasible solution. As shown in these figures, AFPDE converges to its solutions faster than ITLBO and FROFI on most of the test functions. However, it costs more FES to find a feasible solution than FROFI on most of the test functions. Because much information of objective function is used in the early stage of AFPDE. It would have a negative impact on finding a feasible solution of a COP with easy constraints. In summary, although it would take more FES to find a feasible solution, AFPDE can converge to the final solution by using fewer FES.

We also compared AFPDE, ITLBO, and FROFI in terms of running time. The experiments were performed on a computer with Intel Core (TM) i5-7500 (3.40 GHz) processor and Windows 10 (64 bit) system. We recorded the running time of each method in finding the first feasible solution. Since the optima of C01-C18 are not provided in [24], we recorded the running time of finding a solution that can be obtained by all algorithms. The experimental results were reported in Tables S10 and

S11 in the supplementary file. To compare these three methods, we first ranked them based on the running time for each test function. Subsequently, for each algorithm, we calculated the total rank value by summing its rank values of all test functions. These experimental results show that AFPDE runs slower than the other two algorithms in both cases. The reason may be that much time is cost by fuzzification and defuzzification in AFPDE. In the future, more efforts will be devoted to accelerating the speed of AFPDE.

4.3.6. Parameter sensitivity analysis

AFPDE involves two algorithm-specific parameters (i.e., p and η), which must be decided manually. In this subsection, we used experiments to set these two parameters properly. Additionally, we also detected the effective range of λ through experiments.

In terms of p, we set it to five fixed values (i.e., $p = 0.2, p = 0.4, p = 0.6, p = 0.8$, and $p = 0.99$). Note that $p = 0.6$ was used in AFPDE in default. We summarized "Mean OFV", "Std Dev", and the feasible rate of these five variants in Table S12. As depicted in Table S12, AFPDE performs better than the variants with $p = 0.2, p = 0.4, p = 0.8$, and $p = 0.99$ on five, four, two, and five test instances, respectively. It performs worse than these variants on zero, zero, two, and two test functions, respectively. Moreover, the variants with $p = 0.2, p = 0.8$, and $p = 0.99$ fail to obtain a feasible solution consistently on one, two, and four test functions, respectively. In summary, it seems that 0.6 is a good choice for parameter p.

Similarly, to decide a proper value of η, we applied five fixed values (i.e., $\eta = 2.5, \eta = 3.5, \eta = 4.5, \eta = 5.5$, and $\eta = 6.5$) in AFPDE. In AFPDE, $\eta = 4.5$ is adopted in default. We reported "Mean OFV", "Std Dev", and the feasible rate of these five variants in Table S13. AFPDE performs better than the variants with $\eta = 2.5, \eta = 3.5, \eta = 5.5$, and $\eta = 6.5$ on three, two, four, and six test instances, respectively. Inversely, it performs worse than these variants on two, one, zero, and zero test instances, respectively. Thus, $\eta = 4.5$ is recommended to AFPDE.

In order to detect the effective range of λ, we implemented seven variants of AFPDE where different values of λ were used (i.e., $\lambda = 1, \lambda = 3, \lambda = 5, \lambda = 7, \lambda = 9, \lambda = 11$, and $\lambda = 13$). Note that λ was set to 5 in default according to [35]. The experimental results of these seven variants were reported in Table S14. AFPDE performs better than the variants with $\lambda = 1, \lambda = 3, \lambda = 7, \lambda = 9, \lambda = 11$, and $\lambda = 13$ on one, one, two, one, three, and seven test functions, respectively. However, these variants cannot perform better than AFPDE on more than one test functions. As shown in Table S14, AFPDE with $\lambda = 1, \lambda = 3, \lambda = 5, \lambda = 7, \lambda = 9$, and $\lambda = 11$ perform similarly on most of the test functions. Thus, a proper value of λ can be chosen from {1, 3, 5, 7, 9, 11}.

5. Conclusions

This paper proposes an adaptive fuzzy penalty method including two levels for constrained evolutionary optimization. At the individual level, each individual chooses a penalty coefficient according to the designed fuzzy rules. At the population level, the output domain of the penalty coefficient is adjusted by leveraging the population information. Additionally, a mutation scheme is designed to further enhance the population diversity. Furthermore, a search algorithm is developed to generate offsprings by taking advantage of DE. By the above processes, we propose a constrained DE called AFPDE. Systematic experiments on three benchmark test sets and two mechanical design problems indicate that:

- AFPDE exhibits better or at least competitive performance against other state-of-the-art COEAs;
- Both the individual level and the population level are significant to the adaptive fuzzy penalty method;
- The mutation scheme is able to enhance the population diversity which is important to tackle complex COPs.

In the future, we will extend AFPDE to settle COPs in which the constraints are of unequal importance.

CRediT authorship contribution statement

Bing-Chuan Wang: Conceptualization, Methodology, Software. **Han-Xiong Li:** Supervision, Writing - review & editing. **Yun Feng:** Writing - review & editing. **Wen-Jing Shen:** Investigation, Software, Validation, Writing - original draft, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 51906160, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2018A030313747, in part by the Natural Science

Foundation of Top Talent of SZTU under Grant 1814309011180003, and in part by a General Research Fund (GRF) project from Research Grants Council (RGC) of Hong Kong (City U: 11210719).

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, athttps://doi.org/10.1016/j.ins.2021.03.055.

References

[1] B. Akay, D. Karaboga, Artificial bee colony algorithm for large-scale problems and engineering design optimization, Journal of Intelligent Manufacturing 23 (4) (2012) 1001–1014.
[2] J. Alcalá-Fdez, L. Sánchez, S. Garcia, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, et al, KEEL: a software tool to assess evolutionary algorithms for data mining problems, Soft Computing 13 (3) (2009) 307–318.
[3] T. Bäck, D.B. Fogel, Z. Michalewicz, Evolutionary Computation 1: Basic Algorithms and Operators, CRC Press, 2018.
[4] H.J. Barbosa, A.C. Lemonge, H.S. Bernardino, A Critical Review of Adaptive Penalty Techniques in Evolutionary Computation, in: R. Datta, K. Deb (Eds.), Evolutionary Constrained Optimization, Infosys Science Foundation Series, Springer India, New Delhi, 2015, pp. 1–27.
[5] C.A.C. Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Computer Methods in Applied Mechanics and Engineering 191 (11–12) (2002) 1245–1287.
[6] C.A. Coello Coello, Constraint-handling using an evolutionary multiobjective optimization technique, Civil Engineering Systems 17 (4) (2000) 319–346.
[7] C.A. Coello Coello, Constraint-handling techniques used with evolutionary algorithms, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, ACM, 2016, pp. 563–587.
[8] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: A survey, ACM Computing Surveys (CSUR) 45 (3) (2013) 35:1–35:33..
[9] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution–an updated survey, Swarm and Evolutionary Computation 27 (2016) 1–30.
[10] S. Das, P.N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE Transactions on Evolutionary Computation 15 (1) (2010) 4–31.
[11] K. Deb, D. Deb, Analysing mutation schemes for real-parameter genetic algorithms, International Journal of Artificial Intelligence and Soft Computing 4 (1) (2014) 1–28.
[12] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 3–18.
[13] J. Dombi, E. Tóth-Laufer, Reducing the computational requirements in the mamdani-type fuzzy control, Acta Polytechnica Hungarica 17 (3)..
[14] Y.-C. Hsieh, Y.-C. Lee, P.-S. You, Solving nonlinear constrained optimization problems: An immune evolutionary based two-phase approach, Applied Mathematical Modelling 39 (19) (2015) 5759–5768.
[15] Y.H. Kim, S.C. Ahn, W.H. Kwon, Computational complexity of general fuzzy logic control and its simplification for a loop controller, Fuzzy Sets and Systems 111 (2) (2000) 215–224.
[16] K.H. Lee, First course on Fuzzy Theory and Applications, vol. 27, Springer Science & Business Media, 2004.
[17] H.-X. Li, H. Gatland, A new methodology for designing a fuzzy logic controller, IEEE Transactions on Systems, Man, and Cybernetics 25 (3) (1995) 505–512.
[18] H.-X. Li, H. Gatland, Conventional fuzzy control and its enhancement, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26 (5) (1996) 791–797.
[19] X. Li, G. Zhang, Minimum penalty for constrained evolutionary optimization, Computational Optimization and Applications 60 (2) (2015) 513–544.
[20] Z. Li, S. Chen, S. Zhang, S. Jiang, Y. Gu, M. Nouioua, FSB-EA: Fuzzy search bias guided constraint handling technique for evolutionary algorithm, Expert Systems with Applications 119 (2019) 20–35.
[21] J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clerc, P.N. Suganthan, C.C. Coello, K. Deb, Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization, Journal of Applied Mechanics 41 (8) (2006) 8–31.
[22] J. Liu, K.L. Teo, X. Wang, C. Wu, An exact penalty function-based differential search algorithm for constrained global optimization, Soft Computing 20 (4) (2016) 1305–1313.
[23] R. Mallipeddi, P.N. Suganthan, Ensemble of constraint handling techniques, IEEE Transactions on Evolutionary Computation 14 (4) (2010) 561–579.
[24] R. Mallipeddi, P.N. Suganthan, Problem definitions and evaluation criteria for the CEC, competition on constrained real-parameter optimization, Nanyang Technological University, Singapore, 2010, p. 24.
[25] J. Matias, A. Correia, P. Mestre, C. Serodio, P. Couto, C. Teixeira, P. Melo-Pinto, Adaptive penalty and barrier function based on fuzzy logic, Expert Systems with Applications 42 (19) (2015) 6777–6783.
[26] E. Mezura-Montes, C.A.C. Coello, Constraint-handling in nature-inspired numerical optimization: past, present and future, Swarm and Evolutionary Computation 1 (4) (2011) 173–194.
[27] Z. Michalewicz, Genetic Algorithms+ Data Structures= Evolution Programs, Springer Science & Business Media, 2013.
[28] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artificial Intelligence Review 33 (1–2) (2010) 61–106.
[29] C. Peng, H.-L. Liu, F. Gu, A novel constraint-handling technique based on dynamic weights for constrained optimization problems, Soft Computing 22 (12) (2018) 3919–3935.
[30] R. Poláková, L-SHADE with competing strategies applied to constrained optimization, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1683–1689.
[31] R.V. Rao, V.J. Savsani, D. Vakharia, Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems, Computer-Aided Design 43 (3) (2011) 303–315.
[32] S.S. Rao, Engineering Optimization: Theory and Practice, John Wiley & Sons, 2009.
[33] D.J. Reid, Genetic algorithms in constrained optimization, Mathematical and Computer Modelling 23 (5) (1996) 87–111.
[34] C. Saha, S. Das, K. Pal, S. Mukherjee, A fuzzy rule-based penalty function approach for constrained evolutionary optimization, IEEE Transactions on Cybernetics 46 (12) (2016) 2953–2965.
[35] T. Takahama, S. Sakai, Constrained optimization by the ε) constrained differential evolution with an archive and gradient-based mutation, in: IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–9..
[36] A. Trivedi, K. Sanyal, P. Verma, D. Srinivasan, A unified differential evolution algorithm for constrained optimization problems, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1231–1238.
[37] A.S.B. Ullah, E.Z. Elfeky, D. Cornforth, D.L. Essam, R. Sarker, Improved evolutionary algorithms for solving constrained optimization problems with tiny feasible space, in: 2008 IEEE International Conference on Systems, Man and Cybernetics, IEEE, 2008, pp. 1426–1433.
[38] D. Velusamy, G. Pugalendhi, Water cycle algorithm tuned fuzzy expert system for trusted routing in smart grid communication network, IEEE Transactions on Fuzzy Systems 28 (6) (2020) 1167–1177.

[39] B.-C. Wang, Y. Feng, H.-X. Li, Individual-dependent feasibility rule for constrained differential evolution, Information Sciences 506 (2020) 174–195.
[40] B.-C. Wang, H.-X. Li, Y. Feng, An improved teaching-learning-based optimization for constrained evolutionary optimization, Information Sciences 456 (2018) 131–144.
[41] B.-C. Wang, H.-X. Li, J.-P. Li, Y. Wang, Composite differential evolution for constrained evolutionary optimization, IEEE Transactions on Systems, Man, and Cybernetics: Systems (99) (2018b) 1–14..
[42] L.-X. Wang, A course in fuzzy systems and control, Prentice-Hall Inc, 1996..
[43] Y. Wang, Z. Cai, Y. Zhou, W. Zeng, An adaptive tradeoff model for constrained evolutionary optimization, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 80–92.
[44] Y. Wang, B.-C. Wang, H.-X. Li, G.G. Yen, Incorporating objective function information into the feasibility rule for constrained evolutionary optimization, IEEE Transactions on Cybernetics 46 (12) (2016) 2938–2952.
[45] B. Wu, X. Yu, L. Liu, Fuzzy penalty function approach for constrained function optimization with evolutionary algorithms, in: Proceedings of the 8th International Conference on Neural Information Processing, Fudan University Press Shanghai, China, 2001, pp. 299–304.
[46] G. Wu, R. Mallipeddi, P. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization, National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report..
[47] B. Xu, X. Chen, L. Tao, Differential evolution with adaptive trial vector generation strategy and cluster-replacement-based feasibility rule for constrained optimization, Information Sciences 435 (2018) 240–262.
[48] F. Xue, A. Sanderson, P. Bonissone, R.J. Graves, Fuzzy Logic Controlled Multi-Objective Differential Evolution, in: The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ '05, 720–725, 2005..
[49] W. Zhang, G.G. Yen, Z. He, Constrained optimization via artificial immune system, IEEE Transactions on Cybernetics 44 (2) (2014) 185–198.