Received December 31, 2020, accepted January 12, 2021, date of publication January 26, 2021, date of current version February 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054636

An Adaptive Multi-Population Optimization Algorithm for Global Continuous Optimization

ZHIXI LI, (Graduate Student Member, IEEE), VINCENT TAM, (Senior Member, IEEE), AND LAWRENCE K. YEUNG, (Senior Member, IEEE)
Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong

Corresponding author: Zhixi Li (lizx@eee.hku.hk)

ABSTRACT Nowadays, there are various optimization problems that exact mathematical methods are not applicable. Metaheuristics are considered as efficient approaches for finding the solutions. Yet there are many real-world problems that consist of different properties. For instance, financial portfolio optimization may contain many dimensions for different sets of assets, which suggests the need of a more adaptive metaheuristic method for tackling such problems. However, few existing metaheuristics can achieve robust performance across these variable problems even though they may obtain impressive results in specific benchmark problems. In this paper, a metaheuristic named the Adaptive Multi-Population Optimization (AMPO) is proposed for continuous optimization. The algorithm hybridizes yet modifies several useful operations like mutation and memory retention from evolutionary algorithms and swarm intelligence (SI) techniques in a multi-population manner. Furthermore, the diverse control on multiple populations, solution cloning and reset operation are designed. Compared with other metaheuristics, the AMPO can attain an adaptive balance between the capabilities of exploration and exploitation for various optimization problems. To demonstrate its effectiveness, the AMPO is evaluated on 28 well-known benchmark functions. Also, the parameter sensitivity analysis and search behavior study are conducted. Finally, the AMPO is validated on its applicability through a portfolio optimization problem as a challenging example of real-world applications. The benchmark results show that the AMPO achieves a better performance than those of nine state-of-the-art metaheuristics including the IEEE CEC winning algorithms, recent SI and multi-population/hybrid metaheuristics. Besides, the AMPO can consistently produce a good performance in portfolio optimization.

INDEX TERMS Continuous optimization, evolutionary algorithm, metaheuristic, multi-population, portfolio optimization, swarm intelligence.

I. INTRODUCTION

Optimization problems are prevalent in various scientific and engineering domains. For instance, in the field of artificial intelligence, researchers often seek to optimize various machine learning models, such as the hyper-parameter optimization of Support Vector Machines [1], [2] and network architecture search of Neural Networks [3]–[5], to obtain a better performance. In finance, investors usually pursue an optimal portfolio aiming to maximize the return while minimizing the risk [6], [7]. In industrial design and manufacturing, engineers always encounter numerous optimization problems for various products and scenarios, such as optimizing aerodynamic shapes for aircraft, cars, bridges [8], and optimizing supply chain management [9]. Also, there are many optimization problems in our daily lives, like finding the shortest vehicle route to a destination [10] and resource allocation to satisfy performance goals [11].

Since many optimization problems are too complex to be solved with a satisfactory solution by exact mathematical approaches in a reasonable time, metaheuristic optimization algorithms have recently captured extensive attention and achieved some success in solving such problems [12]. In the past decades, a number of metaheuristic optimization algorithms were proposed, among which most are inspired by some natural phenomena and developed based on all kinds of metaphors. Such algorithms can be roughly categorized into evolutionary algorithms (EAs) [13]–[15], swarm intelligence (SI) [16]–[18] approaches, physics/chemistry-based metaheuristics [19]–[22] and others [23]–[25]. EAs are inspired by biological evolutionary processes. Genetic Algorithm (GA) [26], Differential Evolution (DE) [27]

The associate editor coordinating the review of this manuscript and approving it for publication was Jagdish Chand Bansal.

and Evolution Strategies [28] can be regarded as the well-known algorithms. SI algorithms imitate intelligent behaviors of creatures in nature. Among them, Particle Swarm Optimization (PSO) [29] derived from the swarm behavior of birds and fishes is an early work. Until now, the research of SI algorithms has been very active such that new algorithms using various metaphors were proposed from time to time. Some representative examples include the Ant Colony Optimization (ACO) [30], Artificial Bee Colony (ABC) algorithm [31], Grey Wolf Optimizer (GWO) [32], Social Spider Algorithm (SSA) [33], Whale Optimization Algorithm (WOA) [34], Grasshopper Optimization Algorithm (GOA) [35], Earthworm Optimization Algorithm (EOA) [36], Moth Search (MS) [37], Naked Mole-Rat Algorithm (NMA) [38], Monarch Butterfly Optimization (MBO) [39], Harris Hawks Optimization (HHO) [40], Sailfish Optimizer (SFO) [41] and Slime Mould Algorithm (SMA) [42].

Although some metaheuristics exist, the number of effective algorithms is still small relative to the board spectrum of optimization problems. Undoubtedly, this motivates researchers to continuously develop new algorithms to fill the gap. On the other hand, some recently proposed metaphor-based algorithms are criticized due to the abuse of metaphor [43], and to some extent their ultimate performance particularly in solving real-world problems is questionable. One of the main obstacles is that many practical problems have different characteristics. For instance, financial portfolio optimization can be with different dimensions given by different sets of assets, which implies an optimization algorithm with a high adaptivity is required for resolving such problems. In spite of some studies [44]–[47] applied some metaheuristics to portfolio optimization, the scalability and adaptivity of such approaches are not examined carefully yet. Also, the scale of the datasets the OR-Library [48] frequently used by many researchers is relatively small, i.e. up to 225 assets only. Accordingly, many existing metaheuristics that obtain impressive results in some benchmark problems usually fail to solve such challenging practical problems.

In fact, the capabilities of exploration and exploitation should be carefully considered when designing an optimization algorithm [49]. Both can be balanced through a diversity control on the population [50]. Clearly, most of the aforementioned metaheuristics are single-population-based algorithms. While in recent years, devising an optimization algorithm in a multi-population framework is an emerging research direction for developing a more effective metaheuristic because the population diversity can be better maintained, and also different areas can be searched simultaneously [51]. Some recently proposed multi-population-based metaheuristics include the Multi-Swarm Particle Swarm Optimization [52], Multi-Population Ensemble Differential Evolution (MPEDE) [53], and Multi-Population Fruit Fly Outpost Optimization Algorithm (MPFOA) [54]. However, most of the existing studies construct sub-populations only relying on the same search strategy of the original algorithm with minor change(s), thus limiting their adaptivity and effectiveness. Besides, the communication strategy between sub-populations should be explored to prevent from any premature convergence [51].

The above issues motivate us to propose the presented metaheuristic called the Adaptive Multi-Population Optimization Algorithm (AMPO). The AMPO algorithm is a metaheuristic for continuous optimization problems. The algorithm combines some useful operations from EAs and SI approaches. More importantly, the AMPO can achieve an adaptive balance between exploitation and exploration through the unique design of the diversification of the search strategies in a multi-population manner. The solution cloning and reset operations serve as a more efficient communication mechanism between the sub-populations. Therefore, the algorithm can be applied to solve various optimization problems more effectively and efficiently.

To demonstrate its effectiveness and efficiency, the AMPO is evaluated on 28 well-known benchmark functions in terms of solution quality measure, convergence rate, scalability, stability and computational cost. Furthermore, the parameter sensitivity analysis and search behavior study are conducted to investigate the factors that contribute to the success of the algorithm, which can provide useful guidelines for future research and applications. Lastly, the AMPO is applied to a significant computational finance application, i.e. portfolio optimization. In the benchmark function tests, the optimization results of the AMPO are compared with nine state-of-the-art metaheuristics, namely three recent SI approaches, three IEEE CEC winning algorithms and three advanced multi-population and hybrid metaheuristics. In the portfolio optimization problem, the AMPO algorithm competes with three other algorithms that achieved outstanding performance in this problem reported in the literature.

The rest of this paper is organized as follows. In Section II, the AMPO is described in detail, including its definitions, operations and implementations. The design ideas of the AMPO and its differences from other metaheuristics are discussed there as well. Section III covers the experimental results and the related discussion on the benchmark function tests. The parameter sensitivity analysis and search behavior study of the AMPO are analyzed and discussed in Section IV and Section V respectively. The AMPO is applied to solve the financial portfolio optimization in Section VI. We conclude this work and shed light on various potential future directions in Section VII.

II. THE PROPOSED ALGORITHM

A. DEFINITIONS

The AMPO, originally from our preliminary study [55], is formally proposed here for solving unconstrained, static and single-objective continuous optimization problems. Unlike most of the population-based metaheuristic algorithms, the AMPO is essentially a multi-population-based metaheuristic that diversifies the search strategies to enhance the optimization capability and adaptivity. Such so-called adaptivity is that the algorithm is able to dynamically change its search behavior at the time it is run based on information available so as to adapt to different problems.

The AMPO algorithm comprises individuals, which are further classified into five sub-populations (also called groups) including the *random search group*, *global search group*, *local search group*, *leader group* and *migrating group*. Except for the *leader group* and *migrating group*, the sizes of these sub-populations are dynamic during the search. Also, different search strategies are assigned to the individual(s) in each sub-population. For convenience, the *global search group*, *local search group* and *leader group* are uniformly called the *source group*.

- **Definition 1: An Individual**
  In the AMPO, the population size is denoted by N_{pop}. Each individual has a feasible solution as shown below.
  $$\boldsymbol{X}_i^t = [x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,n}^t]^T \tag{1}$$
  where $i \in \{1, 2, \ldots, N_{pop}\}$ represents the index of an individual, $\boldsymbol{X}_i^t$ denotes the feasible solution of the i^{th} individual at the t^{th} iteration, and n is the dimensionality of the problem, i.e. the total number of the decision variables.
- **Definition 2: The Random Search Group**
  The individuals of the *random search group* basically carry out random search in the entire search space, i.e.
  $$\boldsymbol{X}_i^t = U(\boldsymbol{LB}, \boldsymbol{UB}) \tag{2}$$
  where $\boldsymbol{LB} = [lb_1, lb_2, \ldots, lb_n]^T$ and $\boldsymbol{UB} = [ub_1, ub_2, \ldots, ub_n]^T$ each denote the lower bounds and upper bounds of the corresponding domain of the variables being considered, and $U(\cdot)$ is a random number generator function based on the uniform distribution.
  An individual of the *random search group* can possibly be transformed into the *global search group* or the *local search group* by the *transformation* operation that will be introduced in Section II-B4.
- **Definition 3: The Global Search Group**
  The individuals of the *global search group* with relatively larger search step sizes perform global search. The solution of a *global search group* individual at the t^{th} iteration is updated by the following equations.
  $$\boldsymbol{GS}_i^{t+1} = w \times \boldsymbol{GS}_i^t + r \times (\boldsymbol{gbest}^t - \boldsymbol{X}_i^t) \tag{3}$$
  $$\boldsymbol{X}_i^{t+1} = \boldsymbol{X}_i^t + \boldsymbol{GS}_i^{t+1} \tag{4}$$
  where $\boldsymbol{GS}_i^t = [gs_{i,1}^t, gs_{i,2}^t, \ldots, gs_{i,n}^t]^T$ is called the search step size of the *global search group* individual i at the t^{th} iteration, $w \in (0, 1)$ is the weighting factor, $\boldsymbol{gbest}^t$ is the best solution obtained by the population up to the t^{th} iteration, and r is a random number produced by the standard uniform distribution of $U(0, 1)$
- **Definition 4: The Local Search Group**
  The solution of a *local search group* individual is updated by
  $$\sigma_i^{t+1} = \gamma \times \sigma_i^t \tag{5}$$
  $$\boldsymbol{LS}_i^{t+1} = Gaussian(0, \sigma_i^{t+1}) \times \boldsymbol{X}_i^t \tag{6}$$
  $$\boldsymbol{X}_{i+1} = \boldsymbol{X}_i + \boldsymbol{LS}_i^{t+1} \tag{7}$$
  where $\boldsymbol{LS}_i^t$ is the search step size of the *local search group* individual i at the t^{th} iteration, $\gamma \in (0, 1)$ is the constant decay rate, and $Gaussian(0, \sigma_i^{t+1})$ is a random number generator based on a Gaussian distribution with a mean of zero and a standard deviation of σ_i^{t+1}.
  As the value of γ is smaller than 0, the updated standard deviation σ_i^{t+1} input to the Gaussian random number generator will decrease with the number of iterations going up. Then the updated search step size will be more likely smaller (in absolute value). Thus, a *local search group* individual will tend to search for a better solution around its previous one.
  Also, the *local search group* is designed with the transformation probabilities donated by
  $$\begin{aligned} &P_{LS}^{LS} \in (0, 1)\\ &P_{GS}^{LS} \in (0, 1)\\ \text{s.t.}\quad &P_{LS}^{LS} + P_{GS}^{LS} = 1 \end{aligned} \tag{8}$$
  where P_{LS}^{LS} and P_{GS}^{LS} represent the transformation probabilities of an individual of the *random search group* paired up with a *local search group* individual transforming to an individual of the *local search group* (LS) or the *global search group* (GS) in the *transformation* operation.
- **Definition 5: The Leader Group**
  In the AMPO, there is always only one individual in the *leader group*. Such individual represents the best solution obtained by the population so far, possibly except for the *migrating group*. As shown in (9), the update of its solution is paused.
  $$\boldsymbol{X}_i^{t+1} = \boldsymbol{X}_i^t \tag{9}$$
  Similarly, the *leader group* is also associated with the transformation probabilities P_{LS}^{LD} and P_{GS}^{LD} representing the transformation probabilities of a *random search group* individual after being paired up with the individual of the *leader group*.
  $$\begin{aligned} &P_{LS}^{LD} \in (0, 1)\\ &P_{GS}^{LD} \in (0, 1)\\ \text{s.t.}\quad &P_{LS}^{LD} + P_{GS}^{LD} = 1 \end{aligned} \tag{10}$$
- **Definition 6: The Migrating Group**
  The *migrating group* conducts the search independently and possibly contributes some potentially better solutions to the *leader group* by the *migration* operation that will be described in Section II-B5.
  The sub-population size of the *migrating group* is determined by the partition rate PR. Specifically, its size is $N_{pop} \times (1 - PR)$. In this thesis, the *migrating group* is powered by a classical Differential Evolution (DE) algorithm called DE/rand/1/bin [27], [56]. Its user-controlled parameters follow the suggested values in [57].

B. OPERATIONS

In the AMPO algorithm, the *initialization*, *selection*, *update*, *transformation*, *migration* and *reset* are six essential operations.

1) INITIALIZATION

At the starting point of the algorithm, the iteration number t is initialized to 0. The solution of each individual is randomly generated in the search space as shown below.

$$\boldsymbol{X}_i^{t=0} = \boldsymbol{LB} + r \times (\boldsymbol{UB} - \boldsymbol{LB}) \tag{11}$$

where $r \sim U(0, 1)$ is a randomly generated number.

Then the values of $\boldsymbol{S}^G$ and δ in the update functions of the *global search group* and *local search group* are initialized by (12) and (13) respectively.

$$\boldsymbol{GS}_i^{t=0} = \frac{U(\boldsymbol{LB}, \boldsymbol{UB})}{10} \tag{12}$$

$$\delta_i^{t=0} = U(0.1, 1) \tag{13}$$

Lastly, apart from the *migrating group* individuals, all the other individuals are initialized as the *random search group*.

Algorithm 1 AMPO Initialization Operation

```
Input: User-controlled parameters: $N_{pop}$, $P_{LS}^{LD}$, $P_{LS}^{LS}$, $PR$,
       $\gamma$ and $\omega$ Optimization problem information
Output: Newly created individuals: individuals
1  individuals ← ∅;
2  $i \leftarrow 1$;
3  while ($i \leq N_{pop}$) do
4      Initialize a new individual ind with a solution to (11);
5      if ($i < int(N_{pop} * PR)$) then
6          Initialize the control factors in the update functions according to (12) & (13);
7          ind.type ← 'random search';
8      else
9          ind.type ← 'migrating';
10     end
11     Insert ind into individuals;
12     $i \leftarrow i + 1$;
13 end
14 return individuals;
```

Algorithm 1 shows the detailed implementation of such initialization process. The inputs include the aforementioned user-controlled parameters as well as information of the problem to be optimized, i.e. the population size (N_{pop}), partition rate for the *migrating group* (PR), transformation probabilities (P_{LS}^{LD} and P_{LS}^{LS}), control factors used in the search functions (γ and w), dimensionality of the optimization problem (D), and lower and upper bounds ($bound_l$ and $bound_u$). The outputs are the initialized individuals.

2) SELECTION

In the emphselection operation, the individual with the best solution obtained so far among the *random search group*, *global search group* and *local search group* is selected as the *leader group* individual after fitness evaluations for all individuals at each iteration. Meanwhile, the previous individual in the *leader group* is downgraded to the *local search group* individual. Finally, the stored variable $\boldsymbol{gbest}^t$ used by the *global search group* individuals in (3) is updated as well.

3) UPDATE

Algorithm 2 clearly manifests the pseudo-code of the *update* operation. The *update* operation is carried out at each iteration to make individuals search solutions. Depending on the sub-population type of individuals, different search strategies are performed according to (2)—(9).

During the *update* operation, the boundary constraints are handled as shown in (14).

$$\hat{x}_{i,j}^t = \begin{cases} x_{i,j}^t, & lb_j \leq x_{i,j}^t \leq ub_j \\ lb_j, & x_{i,j}^t < lb_j \\ ub_j, & x_{i,j}^t > ub_j \end{cases} \tag{14}$$

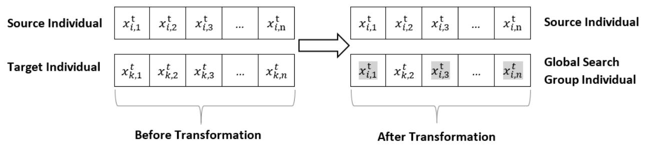
where $\hat{x}_{i,j}^t$ is the constrained value of the j^{th} variable of the solution of the individual i at the t^{th} iteration.

Algorithm 2 AMPO Update Operation

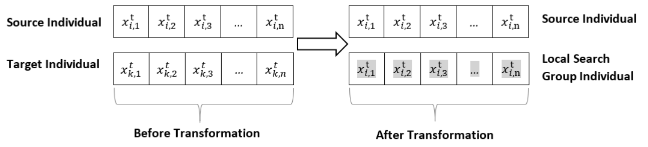
```
Input: All individuals: individuals
Output: All individuals with updated solutions:
        individuals
1  for each ind in individuals do
2      switch ind.type do
3          case random search
4              Update the individual's solution according to (2);
5          end
6          case global search
7              Update the control facotrs of the individual according to (3);
8              Update the individual's solution according to (4);
9          end
10         case local search
11             Update the control facotrs of the individual according to (5) & (6);
12             Update individual's solution according to (7);
13         end
14         case leader
15             Update the individual's solution according to (9);
16         end
17         case migrating
18             Update the individual's solution according to the DE/rand/1/bin algorithm;
19         end
20     endsw
21     Handle the bound constraints according to (14);
22 end
23 return individuals;
```

4) TRANSFORMATION

The *transformation* operation, as the communication strategy between the sub-populations, is a critical factor for the success of the AMPO. Essentially, the goal of the *transformation*



(a) An Individual Transformed to the *Global Search Group* Individual



(b) An Individual Transformed to the *Local Search Group* Individual

FIGURE 1. AMPO Solution Cloning Mechanism.

operation is to spread the solution information among all the individuals so as to empower the search effectiveness.

The detailed implementation of the *transformation* operation is demonstrated in Algorithm 3, where a three-step mechanism is performed:

First of all, the individuals belonging to the *leader group*, *global search group* and *local search group* assemble the *source group* with all the solution qualities sorted in descending order while the individuals of the *random search group* form the *target group* with all the solution qualities sorted in ascending order.

Secondly, each sorted individual of the *source group* is paired up with the corresponding individual in the *target group*. Then, each individual of the *target group* will be transformed in the *local search group* or the *global search group* depending on the transformation probabilities (as defined in Section II-A) of that paired individual in the *source group*.

Lastly, a solution cloning mechanism is executed on each transformed individual as provided below.

- **Transform to a global search group individual**
  When a *random search group* individual is transformed to a *global search group* individual by a *source group* individual, a single-side crossover is adopted. As illustrated in Fig. 1a, each variable of the n-dimensional solution of the *random search group* individual is randomly replaced by that of the *source* one with a fixed probability of 0.5.
- **Transform to a local search group individual**
  When a *random search group* individual transformed by a *source group* individual becomes a *local search group* individual, the whole solution of the *source group* individual is duplicated to the *random search group* one as shown in Fig. 1b.

Such process is conducted at each iteration until all the individuals of the *random search group* are transformed, or all the individuals in the *source group* have been paired up with the individuals of the *random search group*.

5) MIGRATION

The *migration* operation is introduced as an additional operation of the AMPO framework to further diversify the population so that its search capability of solving more complex optimization problems can be enhanced.

Since the *migrating group* works independently, it may discover better solutions compared with all the other sub-populations. One simple migration strategy is that once a better solution is found, the solution will be migrated to the

Algorithm 3 AMPO Transformation Operation

```
Input: The source individuals: sourceInds
       ⊆ {local search, global search, leader} The
       target individuals: targetInds ⊆ {random search}
       The problem dimensionality: n
Output: The individuals with updated solutions
// A solution with a smaller fitness
   value means a higher quality
   solution.
Sort sourceInds by the ascending order of the fitness
values;
Sort targetInds by the descending order of the fitness
values;
for each sourceInd ∈ sourceInds do
    if (|targetInds| > 0) then
        Select the first targetInd from the targetInds;
        Get the transformation probabilities of the
        sourceInd;
        switch sourceInd.type do
            case leader
                if ($rand(0, 1) \le P_{LS}^{LD}$) then
                    transformingTtype ← 'local search';
                else
                    transformingType ← 'global
                    search';
                end
            end
            case local search
                if ($rand(0, 1) \le P_{LS}^{LS}$) then
                    transformingType ← 'local search';
                else
                    transformingType ← 'global
                    search';
                end
            end
            case global search
                transformingType ← 'global search';
            end
        endsw
        targetInd.type ← transformingType;
        switch transformingType do
            case global search
                for $idx = 0$ to $n - 1$ do
                    if $rand(0, 1) \le 0.5$ then
                        targetInd.solution[idx] ←
                        sourceInd.solution[idx];
                    end
                end
            end
            case local search
                targetInd.solution ←
                sourceInd.solution;
            end
        endsw
        Remove targetInd from targetInds;
    end
end
```

whole population directly. Yet this heuristic operation may break search trajectories of the concerned AMPO algorithm by itself, thus possibly leading to an unsatisfactory and unstable performance. Therefore, whenever the *migrating group* finds a better solution, an adaptive probability is predefined to migrate such solution to replace the current best solution maintained by the *leader group* individual. Such solution migrating is under the present condition as illustrated in Algorithm 4.

Algorithm 4 AMPO Migration Operation

```
Input: The leader group individual: $leaderInd$ The
       migrating group individuals: $migratingInds$ The
       current iteration no: $t$ The total number of
       iterations: $T$
Output: The updated leader group individual: $leaderInd$
        The updated stored best solution: $gbest^t$
1 $\langle bestSolution, bestFitness \rangle \leftarrow migratingInds$ ;
2 if ($rand(0, 1) \leq (0.5 * t/T)$) then
3     if ($bestFitness < leaderInd.fitness$) then
4         $leaderInd$.solution $\leftarrow bestSolution$;
5         $leaderInd$.fitness $\leftarrow bestFitness$;
6         $gbest^t \leftarrow bestSolution$
7     end
8 end
9 return $leaderInd, gbest^t$;
```

6) RESET

Due to the powerful *transformation* operation, all the individuals of the *random search group* may be transformed into other groups very soon, thus possibly resulting in a low exploration capability. Accordingly, once no *random search group* individuals are available, the *reset* operation as a restart mechanism will be performed to carefully reset a specified percentage of the individuals from the *global search group* and *local search group* to continue with the exploration process. This control parameter is denoted by *resetPercent* as a randomly generated number between 0.1 and 0.9 at each iteration. This design can help avoid losing all the search information accumulated so far during the search process. In other words, the *leader group* that stores the best-so-far solution is never reset.

Furthermore, a downgrading mechanism is implemented in the *reset* operation: an individual picked up from the *local search group* moves into the *global search group* while another selected individual of the *global search group* is shifted into the *random search group* during the *reset* process. With the relaxation of the search restrictions through resetting individuals, the search capability of the algorithm can be improved gradually to explore the other parts of the search space.

The detailed implementation of this operation is given in Algorithm 5. The individuals of the *global search group* and *local search group* are the input. They may be selected to be reset. In the operation, their solutions as well as the parameters used in the *update* operation are re-initialized. Then a downgraded sub-population type is assigned. The output is the reset individuals.

Algorithm 5 AMPO Reset Operation

```
Input: Individuals: $resetIndividuals$
       $\subseteq \{local\ search, global\ search\}$ Random search
       individuals: $randomIndividuals$
       $\subseteq \{random\ search\}$
Output: Updated reset individuals: $resetIndividuals$
1  if ($|randomIndividuals| = 0$) then
2      $resetPercent \leftarrow rand(0.1, 0.9)$;
3      $resetNum \leftarrow |resetIndividuals| * resetPercent$ ;
4      $rIns \leftarrow$ {the first $resetNum$ individuals} $\subseteq$
       $resetIndividuals$;
5      for each $ind \in rIns$ do
6          Re-initialize the solution of the $ind$ referring to
           (11) ;
7          Re-initialize the control factors of the $ind$
           according to (12) & (13);
8          switch $ind.type$ do
9              case local search
10                 $ind$.type $\leftarrow$ 'global search' ;
11             end
12             case global search
13                 $ind$.type $\leftarrow$ 'random search' ;
14             end
15         endsw
16     end
17 end
18 return $resetIndividuals$;
```

C. THE ALGORITHMIC FLOW OF THE AMPO

Fig. 2 manifests the algorithmic flow of the AMPO. At first, the user-controlled parameters of the algorithm and the objective function of the optimization problem are given as the input to run the AMPO.

First, the *initialization* operation presented in Algorithm 1 is performed.

Then, during the optimization process, the operations are executed successively: i) the *function evaluation* is to calculate the results of the objective function for all individuals. Such results are also called fitness values; ii) the *selection* operation is conducted; iii) the *transformation* operation described in Algorithm 3 is executed to spread the solution information among the population; iv) the *migration* is to possibly migrate a better solution from the *migrating* group to the *leader group* (see Algorithm 4); v) the *update* operation (see Algorithm 2) is to search solutions using different search strategies; vi) the *reset* operation (see Algorithm 5) is triggered once no *random search group* individuals are available.

Such process is executed iteratively until the predefined termination criteria is satisfied, e.g. the maximum number of iterations is reached. Finally, the best solution obtained by the algorithm is provided as the output.

The source code of the AMPO can be found at https://github.com/rayzxli/AMPO.

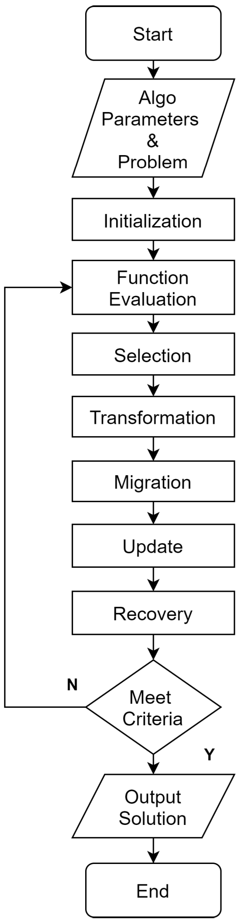


FIGURE 2. The Algorithmic Flow of the AMPO.

D. THE DESIGN IDEAS AND DIFFERENCES FROM EXISTING METAHEURISTICS

Exploration and exploitation are two important factors in designing an optimization algorithm [49], [50]. Exploration related to global search is the process of searching for new solutions in the search space. Exploitation related to local search is the process of refining good solutions that have been discovered. In order to dynamically balance both capabilities for tackling different optimization problems, the design of the AMPO algorithm has absorbed the strengths of EAs and SI approaches. Several advantageous operations are hybridized and modified in the AMPO, thereby constructing different search strategies in a multi-population manner.

Most of the existing metaheuristics are SI algorithms. A common feature of such algorithms is that they have memory retention that can store the (sub-)optimal solutions obtained by individuals in the population. The search can then be partly guided by the memory. For example, in the PSO, the position of each particle represents a feasible solution, which is updated by the best previous position of the particle (local memory) and the position attained by the best particle in the swarm (global memory). The PSO has a high global search capability and fast computing speed [58]. Also, it is known to solve large-scale nonlinear optimization problems effectively [59]. Nevertheless, it also has several shortcomings. For instance, it will converge prematurely and have low exploitation capability in the later search stage where the particles are close to each other or global optimum [60]. Thus, keeping the advantages of the PSO, we first design a memory retention mechanism that the *leader group* acts as; second, the search guided by the global memory in the PSO is equipped with the search strategy of the *global search group* to perform global search.

For some newly proposed SI algorithms, such as the WOA [34] and SSA [33], both simulate the foraging behaviors of animals. Basically, the inventors also followed the basic idea of the PSO but partially modified the search strategies to enhance the optimization ability. Taking the WOA as an example, the WOA mathematically models the search for prey and bubble-net attacking behavior of humpback whales as the exploration and exploitation processes, respectively. To a certain extent, this design readily improves the optimization capability, but it still suffers premature convergence, resulting in trapping in local optima easily [61].

Besides, the design of the AMPO is also motivated by the GA, which is a typical EA algorithm. Since the GA has no memory retention, and its crossover and mutation operators are performed randomly, an individual's good solution may be lost if that individual is not selected, thus causing a lack of fast convergence towards the optimal solution [62]. The operations of the AMPO refer to the selection, mutation and crossover operations in the GA. However, the AMPO has many differences from the GA. First, in the GA, the individual with a higher quality solution may be selected with a higher probability to participate in the crossover and mutation while only the individual with the best-so-far solution is selected and stored in the memory in the AMPO, which overcomes the aforementioned drawback of the GA. Second, the Gaussian mutation with a declining standard deviation is adopted for the *local search group* in the *update* operation in the AMPO so that the exploitation ability can be adjusted dynamically. Third, the crossover operator is modified to the single-side crossover in the *transformation* operation.

Since the exploration and exploitation balance can be achieved through diversity maintenance of the population [50], we build the algorithm based on a multi-population framework. Compared with other multi-population metaheuristics, the AMPO contains an effective communication strategy between the sub-populations, in which the diversity of the population can be dynamically maintained by the *selection*, *transformation*, *migration* and *reset* operations for resolving different problems adaptively. By contrast, many existing studies adopted simple multi-population designs such as the latest developed MOFOA [54]. The MOFOA has no communication strategy between sub-populations, and it only outputs the best solution selected from all sub-populations at the end of iterations. Besides, the sizes of such sub-populations are fixed all the time.

Moreover, the design of the *migration* in the AMPO also provides a useful interface for researchers and users to

TABLE 1. Benchmark Functions.

Function	Name	Expression	Search Range	Global Optimum $f(\boldsymbol{X}^*)$
F1	Bartels Conn	$f(x,y) = \lvert x^2+y^2+xy\rvert + \lvert\sin(x)\rvert + \lvert\cos(y)\rvert$	[-500,500]	0
F2	Bird	$f(x,y) = \sin(x)e^{(1-\cos(y))^2} + \cos(y)e^{(1-\sin(x))^2} + (x-y)^2$	[-2π,2π]	-106.764537
F3	Easom	$f(x,y) = -\cos(x_1)\cos(x_2)\,e^{\left(-(x-\pi)^2-(y-\pi)^2\right)}$	[-100,100]	-1
F4	Egg Crate	$f(x,y) = x^2+y^2+25(\sin^2(x)+\sin^2(y))$	[-5,5]	0
F5	Himmelblau	$f(x,y) = (x^2+y-11)^2+(x+y^2-7)^2$	[-6,6]	0
F6	Sphere	$f(\boldsymbol{X}) = \sum_{i=1}^n x_i^2$	[-5.12,5.12]	0
F7	Schwefel 1.2	$f(\boldsymbol{X}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j\right)^2$	[-100,100]	0
F8	Schwefel 2.20	$f(\boldsymbol{X}) = \sum_{i=1}^n \lvert x_i\rvert$	[-100,100]	0
F9	Schwefel 2.21	$f(\boldsymbol{X}) = \max_{i=1,\ldots,n} \lvert x_i\rvert$	[-100,100]	0
F10	Schwefel 2.22	$f(\boldsymbol{X}) = \sum_{i=1}^n \lvert x_i\rvert + \prod_{i=1}^n \lvert x_i\rvert$	[-100,100]	0
F11	Ackley	$f(\boldsymbol{X}) = -20 * e^{-0.02\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^n \cos(2\pi x_i)} + 20 + e$	[-35,35]	0
F12	Alpine N1	$f(\boldsymbol{X}) = \sum_{i=1}^n \lvert x_i \sin(x_i) + 0.1x_i\rvert$	[-10,10]	0
F13	Csendes	$f(\boldsymbol{X}) = \sum_{i=1}^n x_i^6\left(2+\sin\frac{1}{x_i}\right)$	[-1,1]	0
F14	Griewank	$f(\boldsymbol{X}) = 1 + \sum_{i=1}^n \frac{x_i^2}{4000} - \prod_{i=1}^n \cos(\frac{x_i}{\sqrt{i}})$	[-100,100]	0
F15	Quartic	$f(\boldsymbol{X}) = \sum_{i=1}^n ix_i^4 + \text{random}[0,1)$	[-1.28,1.28]	0 + random noise
F16	Rastrigin	$f(\boldsymbol{X}) = 10n + \sum_{i=1}^n (x_i^2 - 10\cos(2\pi x_i))$	[-5.12,5.12]	0
F17	Salomon	$f(\boldsymbol{X}) = 1 - \cos(2\pi\sqrt{\sum_{i=1}^n x_i^2}) + 0.1\sqrt{\sum_{i=1}^n x_i^2}$	[-100,100]	0
F18	Rotated Discus Function	See [63]	[-100,100]	300
F19	Shifted and Rotated Ackley's Function	See [63]	[-100,100]	500
F20	Shifted Rastrigin's Function	See [63]	[-100,100]	800
F21	Hybrid Function 1	See [63]	[-100,100]	1700
F22	Hybrid Function 2	See [63]	[-100,100]	1800
F23	Composition Function 1	See [63]	[-100,100]	2300
F24	Composition Function 2	See [63]	[-100,100]	2400
F25	Composition Function 3	See [63]	[-100,100]	2500
F26	Composition Function 4	See [63]	[-100,100]	2600
F27	Composition Function 5	See [63]	[-100,100]	2700
F28	Composition Function 6	See [63]	[-100,100]	2800

integrate their own algorithms to the algorithmic framework conveniently in tackling specific problems, which greatly increases the flexibility of the AMPO algorithm. In other words, another metaheuristic can directly serve as the search algorithm to empower the *migrating group* without any modifications. As the *migration* is a one-way operation, i.e. passing the solution from the *migrating group* to the *leader group*, this means the solution obtained by the AMPO framework is not be worse than that generated by the integrated algorithm itself.

III. THE BENCHMARK FUNCTION TESTS

A. THE BENCHMARK FUNCTIONS

In this work, a suite of 28 well-known functions are selected from [63], [64]. These benchmark functions have been widely used in many previous studies to evaluate all kinds of metaheuristic algorithms. Table 1 lists the information about these functions that can be divided into the following groups.

- Group I: $F01$—$F05$ are two-dimensional multi-modal functions;
- Group II: $F06$—$F10$ are scalable uni-modal functions;
- Group III: $F11$—$F17$ are scalable multi-modal functions;
- Group IV: $F18$—$F20$ are shifted and/or rotated functions;
- Group V: $F21$—$F28$ are hybrid and composition functions.

Such benchmark functions cover the most possible characteristics (uni/multi-modal, convex/non-convex, differentiable/non-differentiable and separable/non-separable) of real-world optimization problems. Generally, in $F01$—$F05$, each function has a fixed dimension with a lot of local optima. $F06$—$F17$ are scalable so that the optimizing quality and scalability of the algorithm can be measured. $F18$—$F20$ are some complicated functions in which the shift and rotation transformations are assigned. $F21$—$F28$ are employed to evaluate the algorithm in handling the problems consisting of different sub-components with different features.

B. THE EVALUATION METHOD

In this part of the experiment, each benchmark function is tested for 31 runs for each algorithm. In each run, a maximum number of function evaluations (MaxFEs) that is set

TABLE 2. The Comparative Results of the AMPO and Recent Swarm Intelligence Algorithms on the Benchmark Test.

	AMPO	SFO	SSA	WOA
F01	**0.0000E+00 ± 0.0000E+00**	9.5286E-05 ± 1.0196E-04 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	**7.1627E-18 ± 3.9232E-17** ⊜
F02	2.5074E-07 ± 9.9170E-15	3.4868E+00 ± 7.5791E+00 ⊖	**2.5074E-07 ± 6.9824E-15** ⊕	3.9588E-06 ± 6.5015E-06 ⊖
F03	4.3142E-13 ± 2.1008E-12	1.1235E-06 ± 1.7758E-06 ⊖	**6.4292E-02 ± 2.4482E-01** ⊕	5.3948E-01 ± 4.9188E-01 ⊖
F04	**3.6338E-114 ± 1.9903E-113**	1.4018E-06 ± 2.8277E-06 ⊖	2.3860E-27 ± 4.0755E-27 ⊖	**1.0202E-111 ± 5.5876E-111** ⊜
F05	**1.4314E-120 ± 7.8377E-120**	1.1704E-06 ± 2.6003E-06 ⊖	3.3388E-27 ± 4.6797E-27 ⊖	1.2515E-113 ± 6.8513E-113 ⊖
F06	**0.0000E+00 ± 0.0000E+00**	2.5865E-09 ± 4.0825E-09 ⊖	2.5029E-24 ± 5.3224E-25 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜
F07	**0.0000E+00 ± 0.0000E+00**	3.8106E-07 ± 7.0377E-07 ⊖	5.7462E+01 ± 2.6967E+01 ⊖	9.6874E+00 ± 3.7051E+01 ⊖
F08	**0.0000E+00 ± 0.0000E+00**	1.4878E-04 ± 1.6173E-04 ⊖	8.8668E-24 ± 9.8144E-25 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜
F09	**0.0000E+00 ± 0.0000E+00**	5.7888E-06 ± 4.8180E-06 ⊖	6.8415E-01 ± 4.2005E-01 ⊖	3.9308E+01 ± 2.5662E+01 ⊖
F10	**0.0000E+00 ± 0.0000E+00**	1.7139E-04 ± 1.4748E-04 ⊖	1.8257E+36 ± 3.4947E+36 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜
F11	**4.4409E-16** ± 0.0000E+00	1.9839E-05 ± 2.8169E-05 ⊖	7.8933E-15 ± 1.3848E-15 ⊖	2.9654E-15 ± 2.4203E-15 ⊖
F12	**0.0000E+00 ± 0.0000E+00**	2.4465E-05 ± 2.4658E-05 ⊖	1.9417E-03 ± 6.2866E-04 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜
F13	**0.0000E+00 ± 0.0000E+00**	7.0716E-28 ± 3.2274E-27 ⊖	2.3893E-25 ± 8.8833E-26 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜
F14	**0.0000E+00 ± 0.0000E+00**	1.4986E-10 ± 2.5101E-10 ⊖	2.2598E-15 ± 1.0867E-14 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜
F15	**6.9340E-05 ± 8.1517E-05**	2.7339E-04 ± 1.9020E-04 ⊖	3.0092E-03 ± 8.1616E-04 ⊖	1.4258E-04 ± 1.2936E-04 ⊖
F16	**0.0000E+00 ± 0.0000E+00**	7.8856E-07 ± 1.2950E-06 ⊖	1.3441E-12 ± 3.1196E-12 ⊖	**3.6673E-15 ± 2.0087E-14** ⊜
F17	**0.0000E+00 ± 0.0000E+00**	3.1676E-06 ± 2.5482E-06 ⊖	2.8697E-01 ± 3.3524E-02 ⊖	1.3215E-01 ± 7.3530E-02 ⊖
F18	**0.0000E+00 ± 0.0000E+00**	7.3132E+04 ± 4.5425E+03 ⊖	8.6441E+02 ± 1.2838E+03 ⊖	6.8545E+04 ± 1.7194E+04 ⊖
F19	**2.0034E+01 ± 1.0749E-01**	2.0986E+01 ± 4.6309E-02 ⊖	2.0554E+01 ± 5.9461E-02 ⊖	2.0378E+01 ± 1.6115E-01 ⊖
F20	7.0610E+00 ± 2.6668E+00	3.0613E+02 ± 2.3793E+01 ⊖	**2.1435E-11 ± 6.3535E-11** ⊕	1.8766E+02 ± 3.4942E+01 ⊖
F21	**3.3405E+05 ± 2.0746E+05**	3.8546E+07 ± 1.4963E+07 ⊖	5.9093E+05 ± 2.6104E+05 ⊖	5.4892E+06 ± 3.8540E+06 ⊖
F22	8.2537E+02 ± 8.3400E+02	1.0095E+09 ± 4.7065E+08 ⊖	**1.3246E+02 ± 4.8046E+01** ⊕	4.9257E+03 ± 5.7981E+03 ⊖
F23	**2.0000E+02 ± 0.0000E+00**	2.0000E+02 ± 5.2575E-04 ⊖	3.1401E+02 ± 1.7212E-04 ⊖	3.4381E+02 ± 3.0861E+01 ⊖
F24	**2.0000E+02 ± 0.0000E+00**	2.0002E+02 ± 6.1605E-03 ⊖	2.2420E+02 ± 5.7202E+00 ⊖	2.1835E+02 ± 2.6356E+01 ⊖
F25	**2.0000E+02 ± 0.0000E+00**	2.0000E+02 ± 1.2121E-05 ⊖	2.0194E+02 ± 2.7135E-01 ⊖	2.1667E+02 ± 1.7297E+01 ⊖
F26	**1.0027E+02 ± 6.1524E-02**	1.3413E+02 ± 4.2541E+01 ⊖	1.0039E+02 ± 9.2093E-02 ⊖	1.0043E+02 ± 1.1097E-01 ⊖
F27	**2.0000E+02 ± 0.0000E+00**	2.0000E+02 ± 1.5465E-04 ⊖	4.2161E+02 ± 6.3044E+00 ⊖	9.5831E+02 ± 4.5021E+02 ⊖
F28	**2.0000E+02 ± 0.0000E+00**	2.0000E+02 ± 3.0160E-04 ⊖	4.1601E+02 ± 5.5726E+00 ⊖	2.5829E+03 ± 4.8980E+02 ⊖
	Statistical Results ⊕|⊖|⊜	0|28|0	4|23|1	0|19|9

to $10^4 \times n$ (n is the dimension of the problem) serves as the stopping criteria [63]. Except for the functions in Group I, n is set to 30.

The experimental results are examined via the means and standard deviations of the optimal fitness values attained by each algorithm in all the runs. For convenience, the results are converted to $f(\boldsymbol{X}) - f(\boldsymbol{X}^*)$ where $f(\boldsymbol{X}^*)$ is the real global optimum of the function as listed in Table 1. Thus, the global optima for all the functions should be 0. Besides, a series of Wilcoxon rank-sum tests [65] with a significance level 0.05 are performed so as to validate whether such results are statistically significant or not. The null hypothesis is that the AMPO performs similarly with the competitor when solving each benchmark function.

In all the following experiments, the user-controlled parameters of each competing algorithm refer to its tuned or suggested values reported in its corresponding literature unless otherwise specified. Concerning the AMPO, the parameter combination (N_{pop} =50, P_{LS}^{LD} =0.8, P_{LS}^{LS} =0.8, PR =0.6, γ =0.9 and w =0.1) is utilized in all the remaining experiments of this paper. It is noted that this parameter set may not be the best one for solving all optimization problems. Therefore, the parameter tuning is still recommended for dealing with any problem to achieve a better performance.

All algorithms are implemented in Python 3.6.9 and executed on a desktop computer installed with the Intel®i9-7900X processor running at 3.3 GHz and 64 GB of RAM.

C. THE COMPARISONS WITH THE STATE-OF-THE-ART ALGORITHMS

1) A COMPARISON WITH RECENT SWARM INTELLIGENCE ALGORITHMS

Three successful SI algorithms proposed recently, including the aforementioned SSA and WOA and SFO [41], are compared with the AMPO.

Table 2 presents the comparative results of the AMPO and the three competing algorithms. The statistical result of the Wilcoxon rank-sum test is presented next to the mean value for each algorithm: the '⊕' represents that the performance of the competing algorithm is better than that of the proposed AMPO algorithm with statistical significance; the '⊖' indicates that the competing algorithm is significantly inferior to the proposed algorithm; the '⊜' means that the performance of the two algorithms is consistent. It can be observed from the table:

- The AMPO overall outperforms the compared algorithms in terms of the statistical test results. In particular, it discovers the exact best solutions in 12 cases, i.e. $F01$, $F06$—$F10$, $F12$—$F14$ and $F16$—$F18$. This reveals that the AMPO is able to completely solve all uni-modal and a few multi-modal problems in this test;
- Among all the benchmark functions, the AMPO produces significantly better results when compared with the SFO;
- The optimization results of AMPO is better than those of the SSA in 23 functions. For Group I functions, the statistical tests show that the AMPO outperforms the

TABLE 3. The Comparative Results of the AMPO and CEC Winning Algorithms on the Benchmark Test.

	AMPO	LSHADE	LSHADE_CNEPSIN	SHADEILS
F01	**0.0000E+00** ± **0.0000E+00**	**0.0000E+00** ± **0.0000E+00** ⊜	**0.0000E+00** ± **0.0000E+00** ⊜	**0.0000E+00** ± **0.0000E+00** ⊜
F02	2.5074E-07 ± 9.9170E-15	5.0203E+00 ± 8.5123E+00 ⊜	**6.2754E-01** ± **3.4372E+00** ⊕	2.5074E-07 ± 7.6981E-15 ⊜
F03	4.3142E-13 ± 2.1008E-12	6.4511E-02 ± 2.4565E-01 ⊕	2.5804E-01 ± 4.3753E-01 ⊜	**0.0000E+00** ± **0.0000E+00** ⊕
F04	3.6338E-114 ± 1.9903E-113	1.6506E-259 ± 0.0000E+00 ⊕	**4.2817E-315** ± **0.0000E+00** ⊕	5.3278E-40 ± 2.9179E-39 ⊕
F05	1.4314E-120 ± 7.8377E-120	1.1920E-269 ± 0.0000E+00 ⊕	**4.2130E-316** ± **0.0000E+00** ⊕	1.2539E-39 ± 4.4946E-39 ⊖
F06	**0.0000E+00** ± **0.0000E+00**	4.4077E-73 ± 1.5770E-72 ⊖	5.1171E-159 ± 2.7784E-158 ⊖	1.5274E-38 ± 3.9752E-38 ⊖
F07	**0.0000E+00** ± **0.0000E+00**	1.1681E-30 ± 2.0576E-30 ⊖	4.8650E-15 ± 8.3960E-15 ⊖	4.8287E-07 ± 6.6280E-07 ⊖
F08	**0.0000E+00** ± **0.0000E+00**	3.1854E-41 ± 9.7434E-41 ⊖	3.0226E-76 ± 1.6540E-75 ⊖	1.3659E-16 ± 8.8452E-17 ⊖
F09	**0.0000E+00** ± **0.0000E+00**	7.8902E-10 ± 8.5777E-10 ⊖	5.8177E-74 ± 1.6069E-73 ⊖	2.0389E-10 ± 1.1168E-09 ⊖
F10	**0.0000E+00** ± **0.0000E+00**	5.8873E-41 ± 2.0429E-40 ⊖	9.3144E-78 ± 5.0964E-77 ⊖	2.1254E-16 ± 2.3533E-16 ⊖
F11	**4.4409E-16** ± **0.0000E+00**	3.9968E-15 ± 0.0000E+00 ⊖	3.9968E-15 ± 0.0000E+00 ⊖	3.9524E-14 ± 4.4208E-15 ⊖
F12	**0.0000E+00** ± **0.0000E+00**	4.7433E-04 ± 1.6511E-04 ⊖	1.0124E-03 ± 4.4240E-04 ⊖	4.2871E-19 ± 4.0031E-19 ⊖
F13	**0.0000E+00** ± **0.0000E+00**	4.0497E-119 ± 2.2175E-118 ⊖	**0.0000E+00** ± **0.0000E+00** ⊜	3.1476E-106 ± 2.1617E-106 ⊖
F14	**0.0000E+00** ± **0.0000E+00**	**0.0000E+00** ± **0.0000E+00** ⊜	**0.0000E+00** ± **0.0000E+00** ⊜	**0.0000E+00** ± **0.0000E+00** ⊜
F15	**6.9340E-05** ± **8.1517E-05**	7.7391E-04 ± 2.8894E-04 ⊖	6.2570E-04 ± 2.3630E-04 ⊖	6.6303E-03 ± 2.3235E-03 ⊖
F16	**0.0000E+00** ± **0.0000E+00**	2.0653E-04 ± 3.9554E-04 ⊖	4.5701E-02 ± 2.2402E-01 ⊖	8.8016E-14 ± 3.1760E-14 ⊖
F17	**0.0000E+00** ± **0.0000E+00**	1.0955E-01 ± 2.9565E-02 ⊖	9.9873E-02 ± 5.2215E-09 ⊖	2.0955E-01 ± 7.3418E-02 ⊖
F18	**0.0000E+00** ± **0.0000E+00**	4.4008E-14 ± 2.3767E-14 ⊖	6.2344E-14 ± 1.6806E-14 ⊖	1.7288E-06 ± 1.0694E-06 ⊖
F19	**2.0034E+01** ± **1.0749E-01**	2.0702E+01 ± 1.2897E-01 ⊖	2.0098E+01 ± 2.4130E-02 ⊖	**2.0000E+01** ± **9.7667E-05** ⊜
F20	7.0610E+00 ± 2.6668E+00	4.4100E+01 ± 3.9962E+01 ⊖	1.6645E+02 ± 3.3924E+00 ⊖	**3.0805E-13** ± **7.1864E-14** ⊕
F21	3.3405E+05 ± 2.0746E+05	2.8209E+06 ± 6.8037E+06 ⊕	**1.0766E+03** ± **3.4769E+02** ⊕	7.3022E+03 ± 3.9345E+03 ⊕
F22	8.2537E+02 ± 8.3400E+02	8.7444E+07 ± 3.7210E+08 ⊕	5.8550E+01 ± 9.3905E+00 ⊕	**4.6493E+01** ± **1.2026E+01** ⊕
F23	**2.0000E+02** ± **0.0000E+00**	2.0000E+02 ± 1.9127E-13 ⊖	2.0000E+02 ± 1.5246E-13 ⊖	3.1524E+02 ± 2.5353E-08 ⊖
F24	**2.0000E+02** ± **0.0000E+00**	2.0000E+02 ± 1.3458E-04 ⊖	2.0000E+02 ± 4.4770E-04 ⊖	2.2744E+02 ± 1.1998E+00 ⊖
F25	**2.0000E+02** ± **0.0000E+00**	2.0000E+02 ± 1.1369E-13 ⊖	**2.0000E+02** ± **8.1675E-14** ⊜	2.0339E+02 ± 4.0575E-01 ⊖
F26	**1.0027E+02** ± **6.1524E-02**	2.0000E+02 ± 1.8929E-13 ⊖	2.0000E+02 ± 1.8235E-13 ⊖	**1.0025E+02** ± **3.8064E-02** ⊜
F27	**2.0000E+02** ± **0.0000E+00**	2.0000E+02 ± 2.0412E-13 ⊖	2.0000E+02 ± 1.3506E-13 ⊖	3.3627E+02 ± 3.8494E+01 ⊖
F28	**2.0000E+02** ± **0.0000E+00**	2.0000E+02 ± 1.5143E-13 ⊖	**2.0000E+02** ± **0.0000E+00** ⊜	8.3803E+02 ± 3.2781E+01 ⊖
	Statistical Results ⊕|⊖|⊜	5|20|3	5|17|6	5|18|5

SSA in $F4$ and $F5$ while it is inferior to the SSA in $F2$ and $F3$. It is noted that the AMPO attains the mean fitness (i.e. 4.3142E-13) that is much better than the value (i.e. 6.4292E-02) generated by the SSA in $F03$; however, the inferential statistics gives an opposite result. When investigating it further, we find that the SSA gets the exact global optima in 29 runs but falls into local optima that is far from the global optimum in 2 runs. By contrast, the AMPO achieves the global optimum in 12 runs, and it is close to the global optimum in the other runs. Except for $F02$, $F03$, $F20$ and $F22$, all the results generated by the AMPO are better than those of the SSA with a statistical significance. On the other hand, the optimization capability of the SSA seems to be out of work in tackling $F10$ where it produces a large fitness value. On the contrary, the AMPO gets the global optimal solution exactly;

- No functions in which the WOA outperforms the AMPO are found. The performances of the two algorithms are consistent in 9 functions;
- In addition to the statistical test, the average rank is calculated for each algorithm on all functions. The average ranks are 1.11, 2.57, 2.86 and 3.21 for the AMPO, WOA, SSA and SFO, respectively. Given such results, the WOA is ranked second, next only to the AMPO in this test.

2) A COMPARISON WITH THE CEC WINNING ALGORITHMS

According to [66], the variants of the Successful-History based Adaptive Differential Evolution (SHADE) [67] algorithm have achieved remarkable success in the IEEE CEC competitions. In this part of the experiment, the AMPO is compared with some winning algorithms: LSHADE [68], LSHADE-cnEpSin (denoted by LS-cnEPsin later) [69] and SHADE-ILS [70].

Table 3 indicates such results in which the results of the AMPO are put together to make the comparison more straightforward and convenient. From the table, we have the following observations:

- According to the statistical results, the performance of the AMPO algorithm is as a whole better than those of all the three CEC winning algorithms;
- Each competing algorithm itself outperforms the AMPO in 5 functions. Most of the better results generated by the LSHADE, LS-cnEpSin and SHADE-ILS algorithms are located in Group I. By contrast, the performance of the AMPO is fairly good in coping with those 30-Dimensional functions. This suggests that the AMPO enjoys few advantages to dealing with such low dimensional problems;
- In terms of the descriptive statistics, the average rankings of the AMPO, LS-cnEpSin, LSHADE and SHADE-ILS are respectively 1.43, 2.86, 2.87 and 2.89.

3) A COMPARISON WITH THE ADVANCED HYBRID AND MULTI-POPULATION ALGORITHMS

The AMPO algorithm is also compared with three advanced hybrid and multi-population algorithms, i.e.

TABLE 4. The Comparative Results of the AMPO and Hybrid and Multi-population Algorithms on the Benchmark Test.

	AMPO	HFPSO	MOFOA	MPEDE
F01	**0.0000E+00 ± 0.0000E+00**	5.3054E-14 ± 1.9405E-13 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	8.6947E-10 ± 8.8797E-10 ⊖
F02	**2.5074E-07 ± 9.9170E-15**	**1.8826E+00 ± 5.7515E+00** ⊜	1.6358E+00 ± 1.4535E+00 ⊖	6.6371E-06 ± 2.4320E-05 ⊖
F03	**4.3142E-13 ± 2.1008E-12**	8.3871E-01 ± 3.6780E-01 ⊖	6.7555E-03 ± 6.6069E-03 ⊖	1.2258E-11 ± 1.2067E-11 ⊖
F04	**3.6338E-114 ± 1.9903E-113**	3.0607E-01 ± 1.6764E+00 ⊖	2.5339E-31 ± 1.3750E-30 ⊖	8.2012E-13 ± 1.0834E-12 ⊖
F05	**1.4314E-120 ± 7.8377E-120**	3.0607E-01 ± 1.6764E+00 ⊖	2.0197E-31 ± 8.2523E-31 ⊖	1.7206E-12 ± 3.0285E-12 ⊖
F06	**0.0000E+00 ± 0.0000E+00**	1.3614E-19 ± 1.6753E-19 ⊖	**1.7932E-233 ± 0.0000E+00** ⊜	4.3997E-21 ± 2.1305E-20 ⊖
F07	**0.0000E+00 ± 0.0000E+00**	9.6803E+02 ± 2.3484E+03 ⊖	4.5945E-272 ± 0.0000E+00 ⊖	6.5211E-22 ± 1.5940E-21 ⊖
F08	**0.0000E+00 ± 0.0000E+00**	1.7975E+01 ± 1.9722E+01 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	3.5162E-10 ± 5.0701E-10 ⊖
F09	**0.0000E+00 ± 0.0000E+00**	3.0461E+00 ± 1.9310E+00 ⊖	**5.4489E-143 ± 2.9845E-142** ⊜	5.6982E-13 ± 9.2508E-13 ⊖
F10	**0.0000E+00 ± 0.0000E+00**	5.2914E+02 ± 1.4183E+02 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	4.2077E-10 ± 5.4289E-10 ⊖
F11	**4.4409E-16 ± 0.0000E+00**	4.9185E+00 ± 1.2101E+00 ⊖	**4.4409E-16 ± 0.0000E+00** ⊜	1.3583E-11 ± 1.9539E-11 ⊖
F12	**0.0000E+00 ± 0.0000E+00**	3.9104E+00 ± 1.9326E+00 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	2.5690E-03 ± 1.4959E-03 ⊖
F13	**0.0000E+00 ± 0.0000E+00**	5.9306E-11 ± 1.6738E-10 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	5.8364E-42 ± 2.3525E-41 ⊖
F14	**0.0000E+00 ± 0.0000E+00**	1.3021E-02 ± 1.3077E-02 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	**0.0000E+00 ± 0.0000E+00** ⊜
F15	**6.9340E-05 ± 8.1517E-05**	2.5760E-02 ± 1.2973E-02 ⊖	**2.5260E-04 ± 6.7048E-04** ⊜	2.7235E-03 ± 1.0010E-03 ⊖
F16	**0.0000E+00 ± 0.0000E+00**	6.1887E+01 ± 1.9322E+01 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	4.8157E+00 ± 3.1470E+00 ⊖
F17	**0.0000E+00 ± 0.0000E+00**	1.2773E+00 ± 1.7360E-01 ⊖	**0.0000E+00 ± 0.0000E+00** ⊜	**8.2059E-05 ± 4.4946E-04** ⊜
F18	**0.0000E+00 ± 0.0000E+00**	5.4158E+03 ± 1.2634E+04 ⊖	8.2465E+04 ± 4.4179E+03 ⊖	1.0546E+00 ± 8.9419E-01 ⊖
F19	**2.0034E+01 ± 1.0749E-01**	**2.0037E+01 ± 8.5514E-02** ⊜	2.0963E+01 ± 4.5682E-02 ⊖	2.0549E+01 ± 3.4983E-02 ⊖
F20	**7.0610E+00 ± 2.6668E+00**	1.3636E+02 ± 2.9273E+01 ⊖	3.2937E+02 ± 1.2300E+01 ⊖	1.2206E+01 ± 3.2418E+00 ⊖
F21	3.3405E+05 ± 2.0746E+05	2.4306E+06 ± 3.7620E+06 ⊖	1.1807E+08 ± 2.5327E+07 ⊖	**4.7061E+02 ± 2.0622E+02** ⊕
F22	8.2537E+02 ± 8.3400E+02	7.6334E+04 ± 2.8893E+05 ⊜	4.6796E+09 ± 1.0153E+09 ⊖	**5.3594E+02 ± 2.1742E+03** ⊕
F23	**2.0000E+02 ± 0.0000E+00**	3.2886E+02 ± 1.8302E+01 ⊖	**2.0000E+02 ± 0.0000E+00** ⊜	2.0000E+02 ± 3.8074E-10 ⊖
F24	**2.0000E+02 ± 0.0000E+00**	2.4850E+02 ± 1.2187E+01 ⊖	**2.0000E+02 ± 0.0000E+00** ⊜	2.0000E+02 ± 1.1152E-07 ⊖
F25	**2.0000E+02 ± 0.0000E+00**	2.2100E+02 ± 1.2334E+01 ⊖	**2.0000E+02 ± 0.0000E+00** ⊜	2.0000E+02 ± 8.0476E-12 ⊖
F26	**1.0027E+02 ± 6.1524E-02**	1.5674E+02 ± 6.6068E+01 ⊖	1.6364E+02 ± 3.5607E+01 ⊖	2.0000E+02 ± 1.6807E-13 ⊖
F27	**2.0000E+02 ± 0.0000E+00**	1.1751E+03 ± 2.7164E+02 ⊖	**2.0000E+02 ± 0.0000E+00** ⊜	2.0000E+02 ± 4.2874E-11 ⊖
F28	**2.0000E+02 ± 0.0000E+00**	2.5354E+03 ± 8.5186E+02 ⊖	**2.0000E+02 ± 0.0000E+00** ⊜	2.0000E+02 ± 3.4604E-10 ⊖
	Statistical Results ⊕|⊖|⊜	0|25|3	0|11|17	2|24|2

the Multi-population Ensemble Differential Evolution (MPEDE) [53], Hybrid Firefly and Particle Swarm Optimization (HFPSO) [71], and MOFOA that is discussed in Section II-D.

The detailed results are presented in Table 4, in which we can see that:

- The performance of AMPO still exceeds all the three algorithms here;
- The competing algorithms are also designed in hybrid and multi-population manners. Nevertheless, only the MPEDE algorithm achieves better results than our proposed algorithm in 2 out of all 28 functions. On the other hand, its results are worse than the AMPO in other 24 functions;
- For the functions in Group II & Group III, the performance between the MOFOA and AMPO algorithms is almost consistent except for $F07$. Yet the MOFOA fails to deliver satisfactory results for the function in the Group I & Group IV compared to the AMPO. According to [54], the MOFOA also adopts the Gaussian mutation (called outpost mechanism) that is similar to our design but with a difference that the variance is fixed as 1 in the MOFOA. Such results imply that the MOFOA has excellent convergence speed; however, a lack of sufficient adaptivity of working on a wider range of problems. Generally speaking, the MOFOA is the second best in this comparison in light of the statistical result.
- Regarding the descriptive statistics, the average rankings of the AMPO, MOFOA, MPEDE and HFPSO are respectively 1.07, 1.96, 2.73, and 3.68.

D. THE CONVERGENCE TEST

In addition to the solution quality, we are also interested in the rate of convergence. Therefore, the convergence tests on all the compared algorithms are carried out.

Since each function is tested for 31 runs for each algorithm, the run with the median fitness result is selected to plot the convergence curve. Fig. 3—Fig. 7 display the convergence results, in which the x-axis is the fitness function evaluations (FE) consumed and the y-axis is the best fitness value obtained so far. Also, as some algorithms may produce large fitness values at the beginning of running, the maximum value of y-axis is limited to the median value of such finesses to make the observation clearer. The findings are summarized in the following points:

- The AMPO generally converges faster than other algorithms and hence possesses a better convergence capability for the tested optimization problems;
- For Group I functions, the convergence speed of the AMPO is a bit slower than a few compared algorithms. Also, the optimization capability of the AMPO on these functions is weaker than those of the competitors, as stated in Section III-C. This finding suggests that the AMPO algorithm may have relatively few advantages in solving these low-dimensional problems;

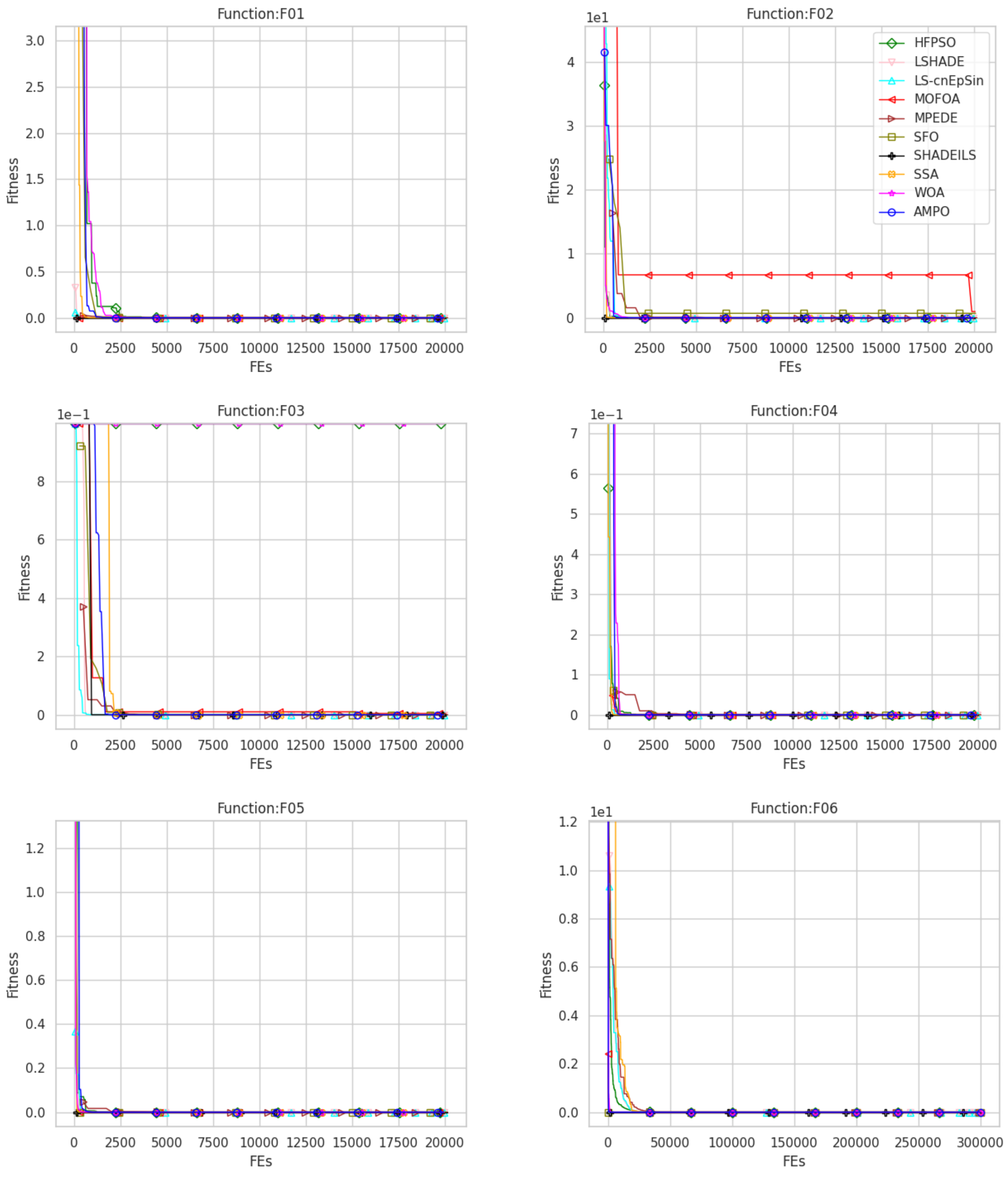


FIGURE 3. The Results of Convergence Test (F01-F06).

- For $F19$ and $F20$, the convergence rates of the AMPO are only next to those of the SHADE-ILS algorithm.
- In the remaining functions, the AMPO achieves the best performance over all the competitors in terms of the convergence rate.

E. THE STABILITY ANALYSIS

Despite the standard deviations shown in Section III-C, the box plots of fitness variation are illustrated in Fig. 8—Fig. 12, in which we may observe the distribution of the results generated by each algorithm, especially outliers

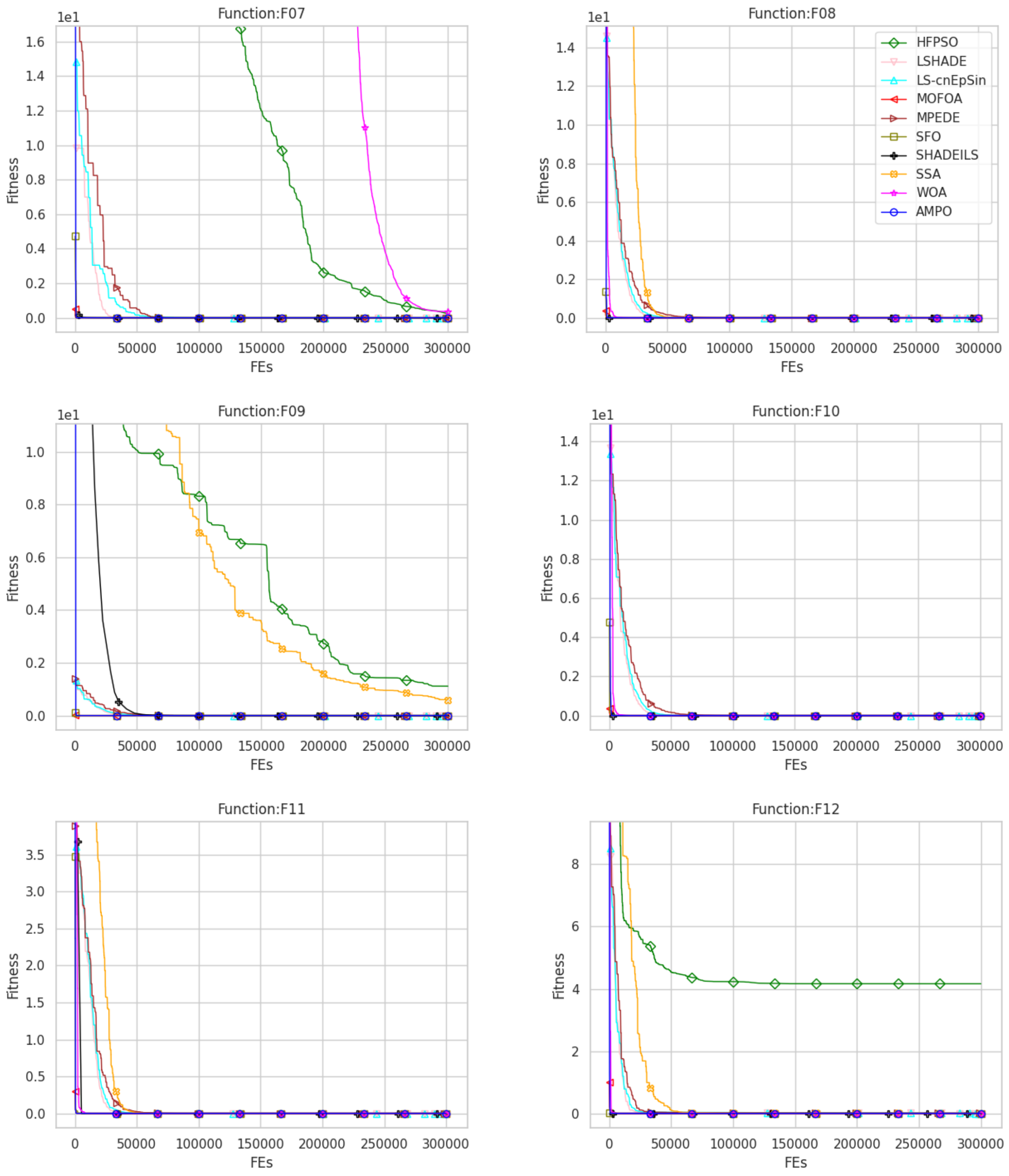


FIGURE 4. The Results of Convergence Test (F07-F12).

in all the runs on the function. We have a few observations that:

- The body of the box of the AMPO is the shortest one among all algorithms in $F1$—$F18$ and $F23$—$F28$. The AMPO even has no outlier points in these cases. This demonstrates that the performance of AMPO is very stable;
- Only the AMPO algorithm gets zero values of the standard deviation in $F27$ and $F28$;
- In $F19$, the AMPO optimizer produces several outliers. On the other hand, except for the SHADEILS, all the other algorithms are unstable in optimizing this problem as well.

Due to the effective multi-population design in which the computational resources can be dynamically allocated to different sup-population supported by different search strategies (such search behavior is further revealed in Section V). In each run, the AMPO algorithm is able to adapt to the

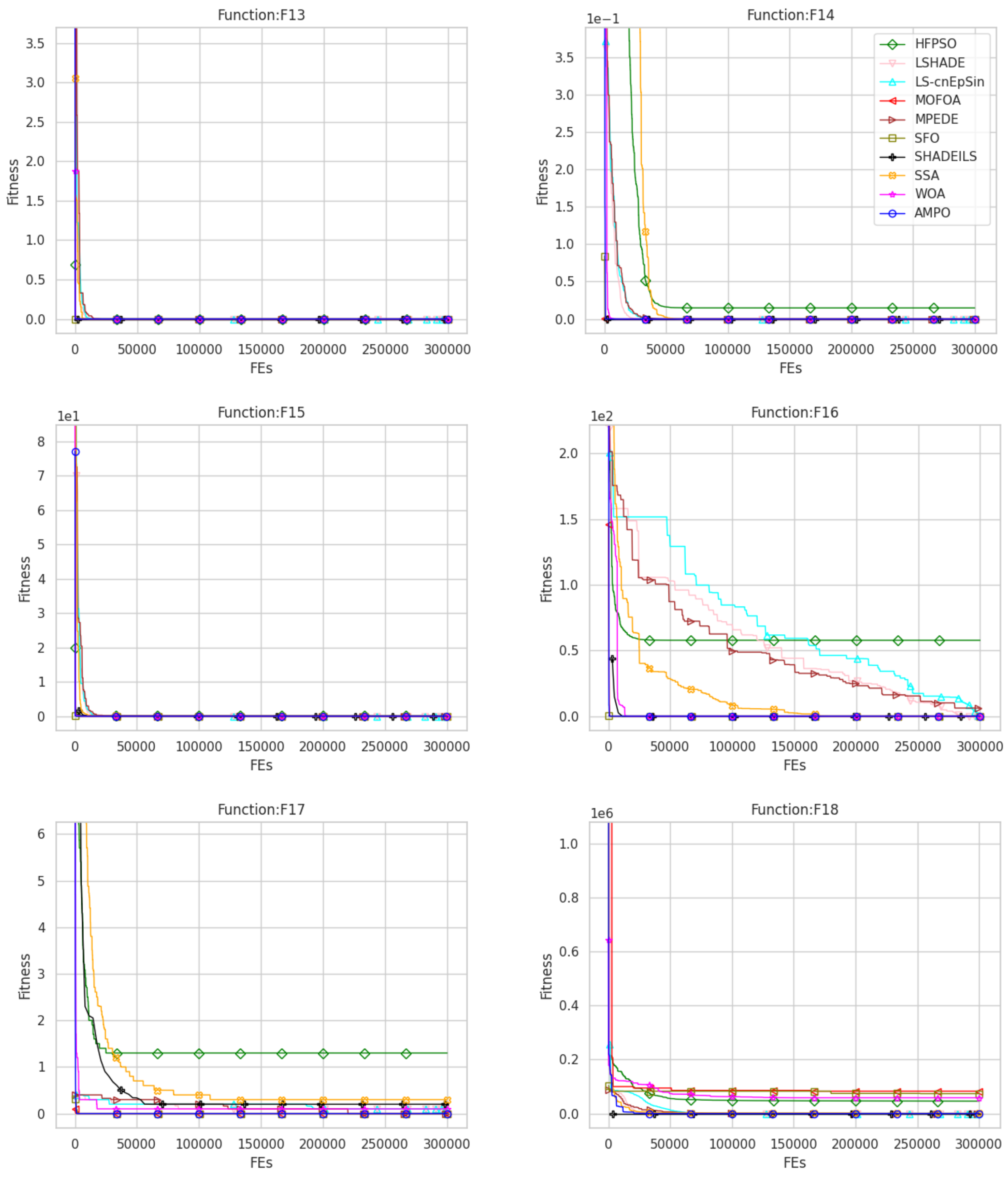


FIGURE 5. The Results of Convergence Test (F13-F18).

problem based on the obtained information. Thus, although the search trajectories of all the runs are different from each other, the AMPO still has a quite stable performance.

F. A COMPARISON OF THE COMPUTATIONAL COSTS

During running the tests presented in Section III-C, we also record the time performance to compare the computational costs between all optimizers. Table 5 lists the average execution time of all the runs in terms of CPU seconds. It can be seen that:

- The AMPO runs the fastest in three cases, i.e. $F7$, $F14$ and $F17$;
- Two other metrics are calculated to evaluate the overall time performance of each algorithm. The average time

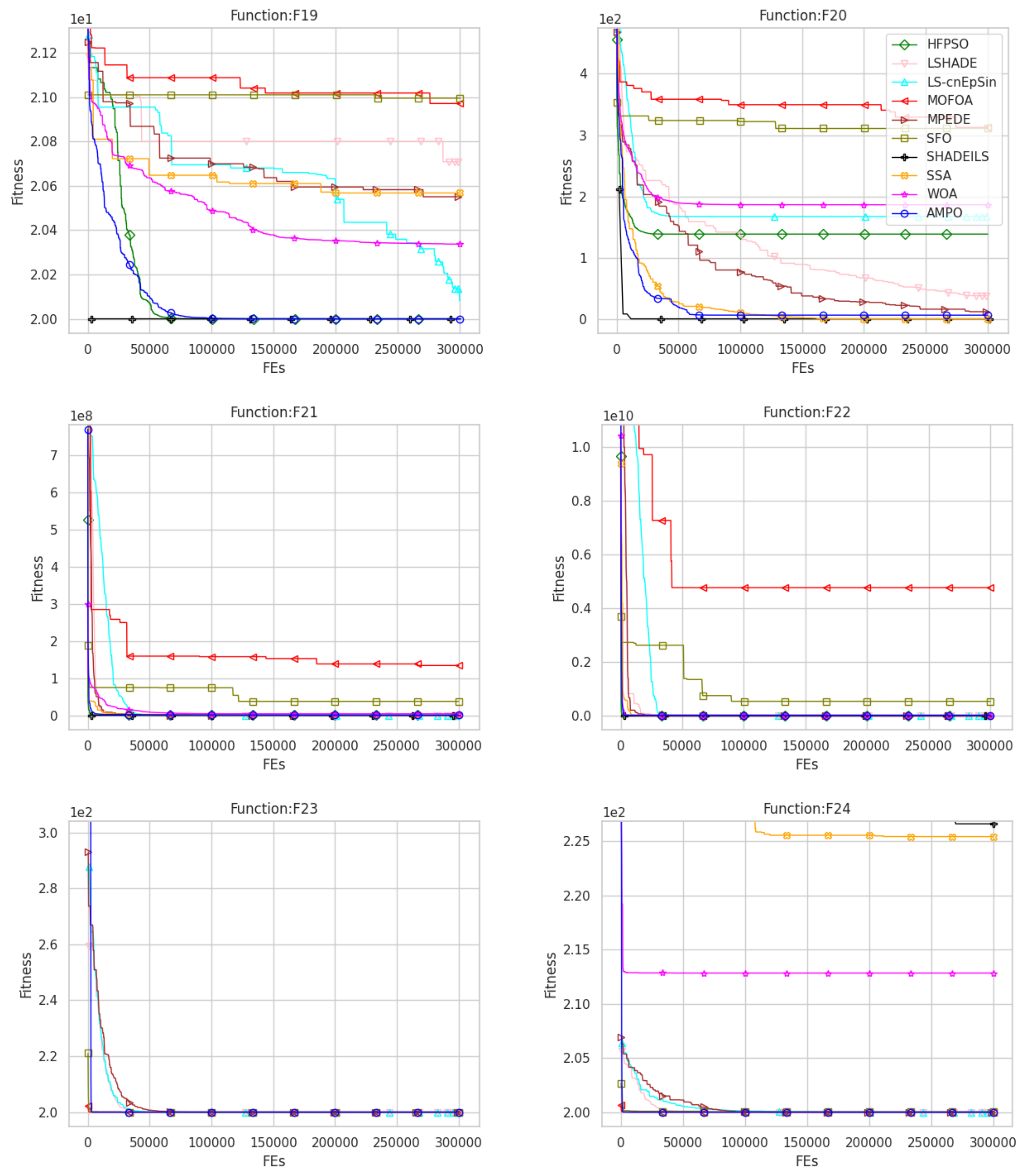


FIGURE 6. **The Results of Convergence Test (F19-F24).**

is the mean value of all times, and the average rank is the mean ranking of all times. In terms of the average rank, the AMPO is the second fastest algorithm, only following the SSA, while it ranks first on the basis of the average time;

- Given the overall ranking, the slowest algorithm is the LS-cnEpSin algorithm, which involves the time-consuming calculation of the Euclidean distance and the covariance matrix.

G. THE SCALABILITY TEST

Many practical optimization problems in computational finance as well as other areas have various dimensions.

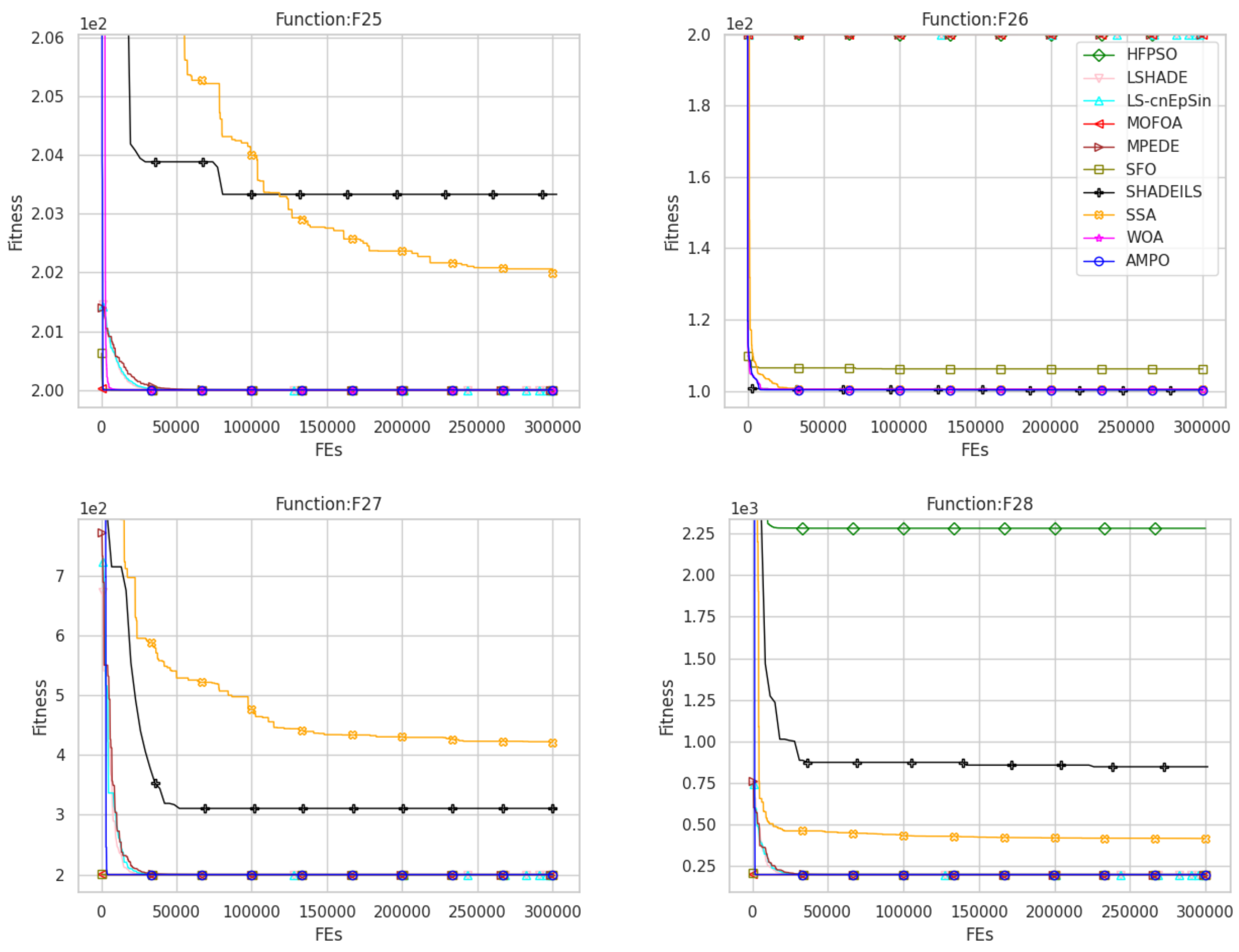


FIGURE 7. The Results of Convergence Test (F25-F28).

The good scalability is very significant for a metaheuristic algorithm to solve such problems.

Therefore, in addition to the 30-dimension benchmark function tests, the simulations on 50-D, 100-D and 200-D benchmark problems in Group II & III are performed to evaluate the scalability of the AMPO. In order to make a thorough comparison, three compared algorithms that attain good results in the above test, i.e. the WOA, LS-cnEpSin and MOFOA, are involved in the test. The results are presented in Table 6—Table 8, where the measurement is the mean fitness of all the 31 runs.

From the tables, we can see that the AMPO algorithm still achieves the best performance in all the problems no matter with 50, 100 and 200 dimensions. Except for $F11$ and $F15$, the AMPO has exactly found the global optimal solutions. In fact, the exploitation of an optimizer may dramatically affect the ultimate fitness of the searched solution particularly in solving some high-dimensional problems like $F07$ and $F16$. That is, due to a large-scale set of the decision variables, the fitness value of the optimization problem may become very large as long as some variables of a solution are not accurate enough. This means that the algorithm should be able to find more precise solutions around the best one at hand. As the AMPO is empowered by the adaptive search framework, the algorithm can allocate more resources to the *local search group*, which helps the algorithm search for more precise solutions. Such behavior is observed in Section V.

IV. THE PARAMETER SENSITIVITY ANALYSIS

Apart from the max number of iterations (or MaxFEs), there are six user-controlled parameters in the AMPO: N_{pop}, w, γ, PR, P_{LS}^{LD} and P_{LS}^{LS} that are defined in Section II-A. As the selection of user-controlled parameters may dramatically affect the performance of one metaheuristic algorithm, we systematically investigate the possible impacts of different parameter settings on the performance of the AMPO according to the practice in our previous work [72].

The parameter set used in the above experiment (see Section III-B) is considered as the standardized parameter set for the subsequent discussion. Since it is very computationally expensive to explore all combinations of such control parameters, the parameter sensitivity analysis test focuses on a specific parameter while fixing all other parameters. For example, the population size in the standardized

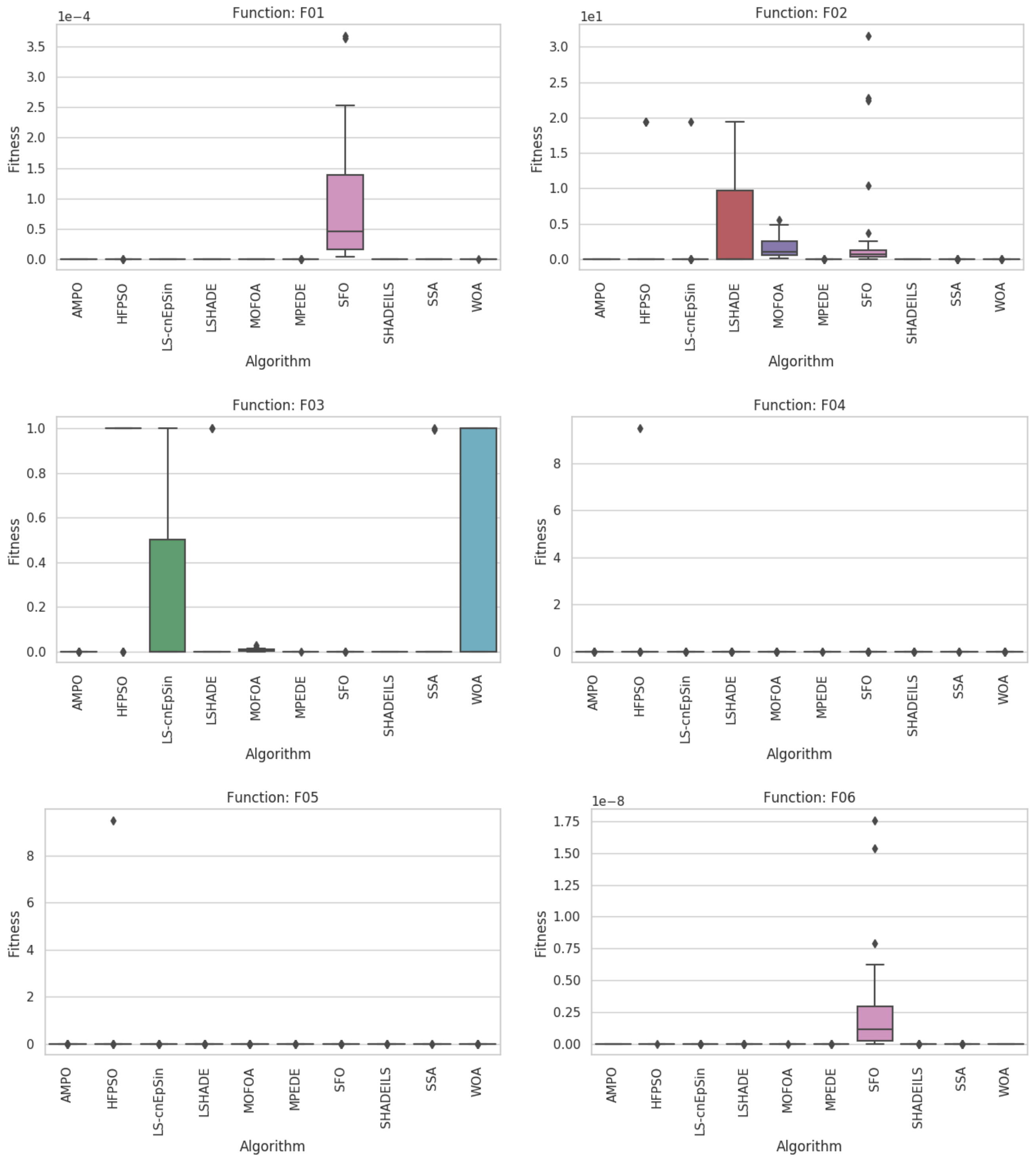


FIGURE 8. The Box Plot of Fitness Variation (F01-F06).

set is 50, whereas it is set to 10, 30, 100, or 200 for the subsequent sensitivity analysis with all other parameters remained unchanged. The test is conducted on a total of 15 representative functions, i.e. $F02$, $F03$ and $F04$, $F06$, $F08$ and $F10$, $F12$, $F15$ and $F17$, $F19$ and $F20$ and $F23$, $F25$, $F26$ and $F26$ selected from Groups I, II, III, IV and V respectively. The AMPO with the standardized parameter set acts as the baseline algorithm whose results are presented in Section III-C. Also, a series of statistical tests similar to those practices in Section III-B are performed to compare the tested algorithms with the baseline algorithm.

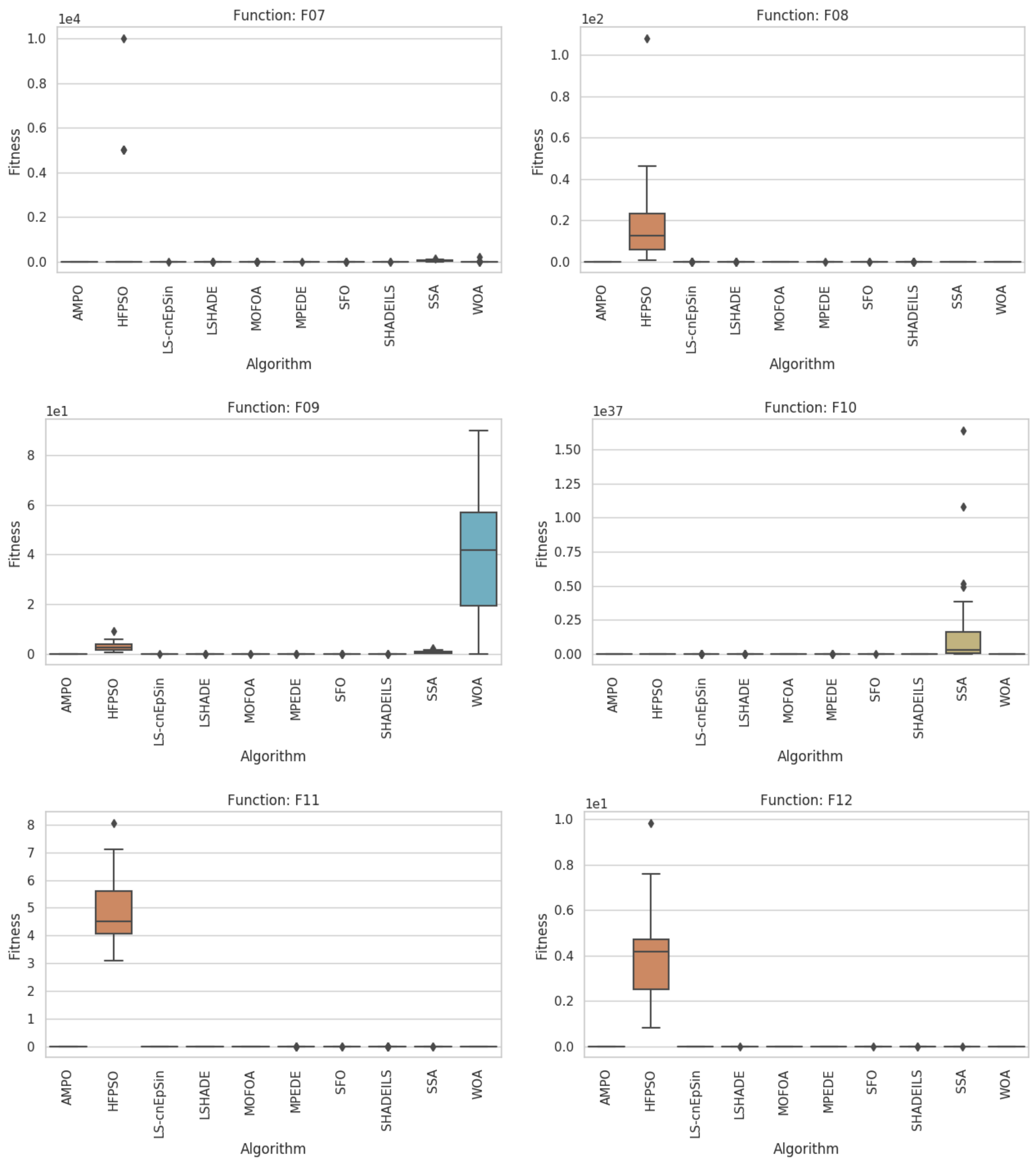


FIGURE 9. **The Box Plot of Fitness Variation (F07-F12).**

The results are presented in Table 9—Table 14, where we can see that:

- Under the same MaxFEs, the algorithm with the population size of 30 overall achieves a better performance than that of the baseline algorithm in terms of the statistical results. It appears that the AMPO with a population size that is much smaller or larger than the problem dimensionality tends to produce worse results for these problems. For example, in the cases of $N_{pop} = 10$ and $N_{pop} = 200$, the algorithm is inferior to the baseline in five and six functions respectively;

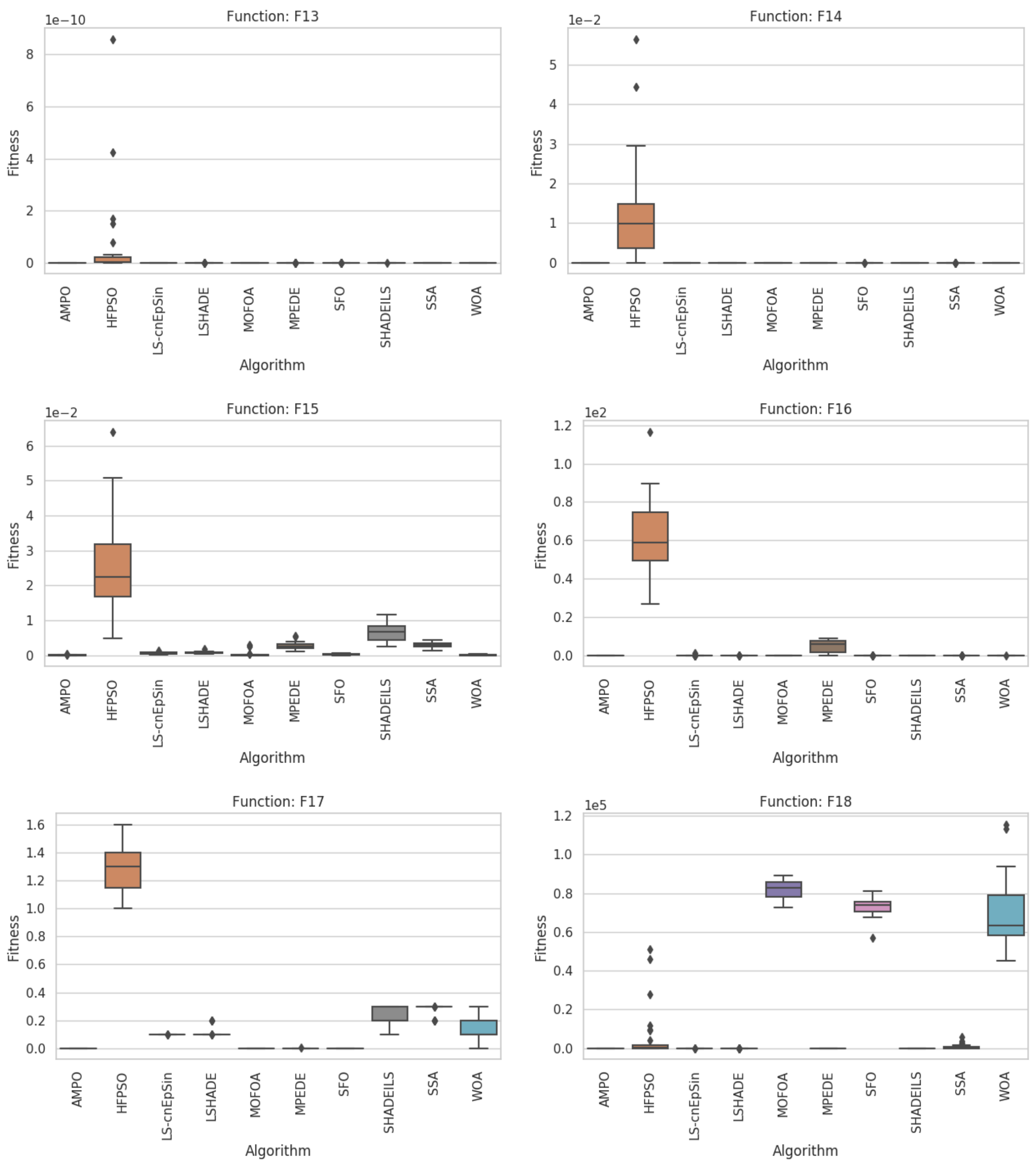


FIGURE 10. **The Box Plot of Fitness Variation (F13-F18).**

- At least in these functions, the performance of the AMPO is not very sensitive to the weighting factor w that is used in the search step size update function of the *global search group*;
- The selection of the constant decay rate γ has a significant impact on the performance when it is set to a small value, e.g. 0.1, 0.3 or 0.5 in this test. This parameter is used to decrease the standard deviation of the Gaussian function of the *local search group*. If the value is too small, the search step size of the *local search group* will decrease quickly, thus easily leading to a slow convergence to the global optima;
- In the case of $P_{LS}^{LD} = 0.6$ or $P_{LS}^{LS} = 0.6$, no difference in the performance is found between the tested algorithm

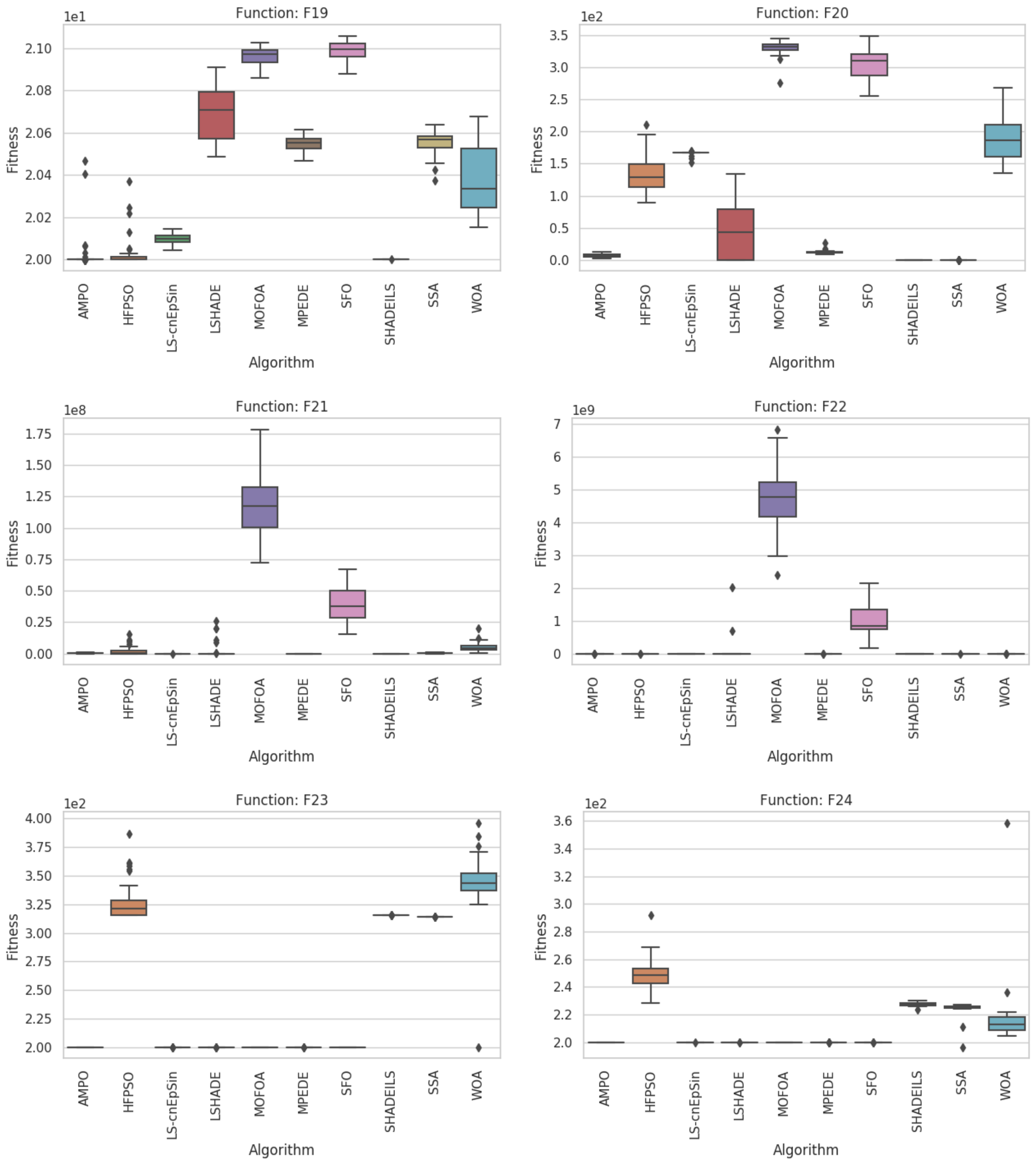


FIGURE 11. **The Box Plot of Fitness Variation (F19-F24).**

and baseline algorithm. However, setting the parameter of P_{LS}^{LD} or P_{LS}^{LS} to small values has a slight negative effect on the performance of the algorithm observed in this experiment. Thus, setting these parameters to small values is not advisable;

- A small or large value of *PR* may produce worse results in this test.

The above observations indicate that selecting medium values of N_{pop} (e.g. 30 & 50), large values of γ (e.g. 0.9), and medium to large values of P_{LS}^{LD} (e.g. 0.6 & 0.8) and

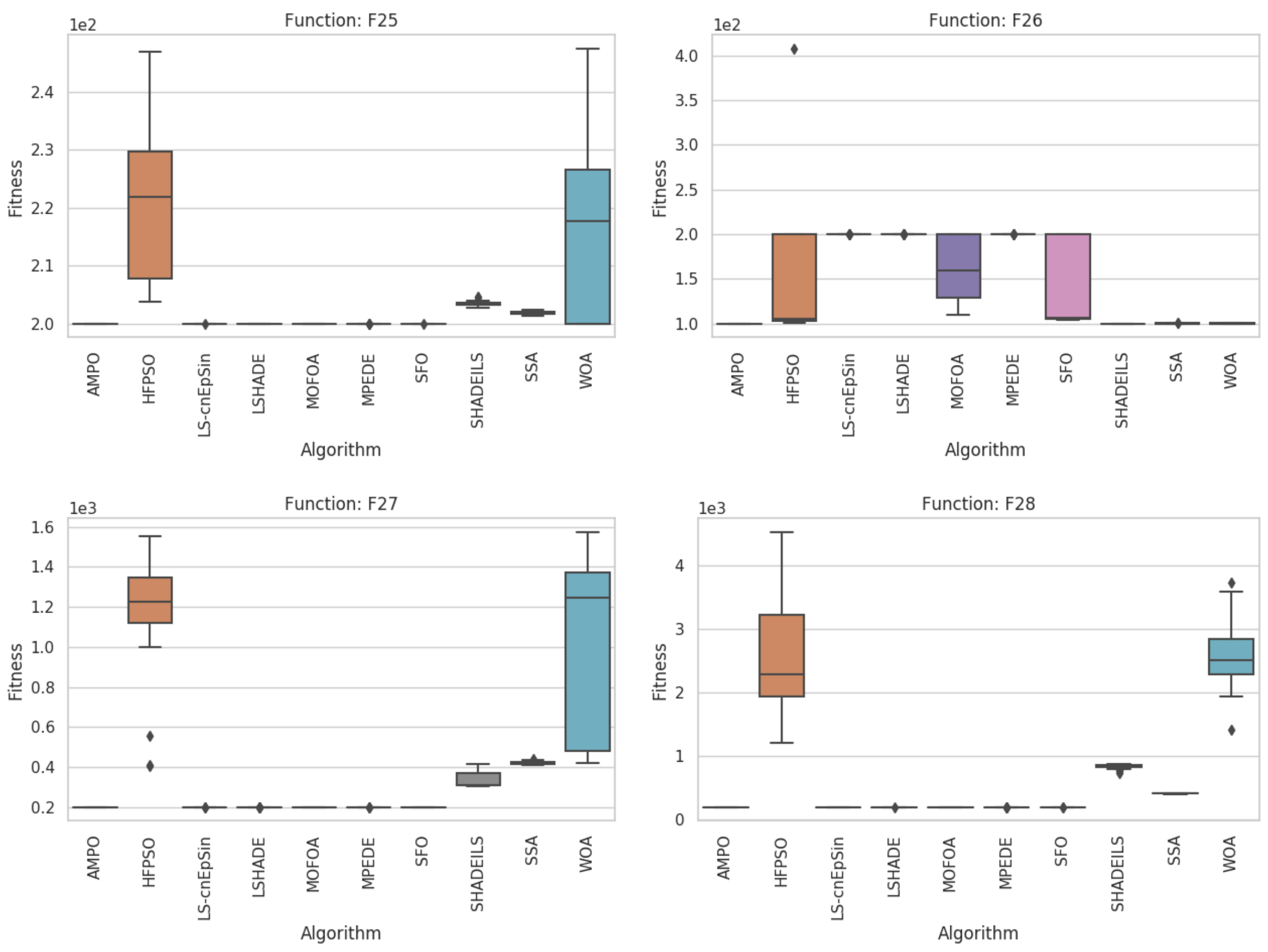


FIGURE 12. **The Box Plot of Fitness Variation (F25-F28).**

P_{LS}^{LS} (e.g. 0.6 & 0.8) are suggested to improve the adaptivity and effectiveness of the algorithm for optimizing various problems. To determine the value of *PR*, the trial-and-error method is advised when solving different problems.

V. THE SEARCH BEHAVIOR STUDY

In this section, we further investigate the search behavior of the AMPO, including the dynamics of the search step size and sub-population size. Ten representative benchmark functions are involved in the study. We re-run the AMPO on these functions with the same Experimental Setup as stated in Section III-B except for the stopping criteria. In this study, the search will be terminated once the MaxFEs or global optimal solution is reached so that the observation can be conducted more clearly.

Firstly, for each function, the search step sizes of the *local search group* and *global search group* individuals are recorded on each iteration in each run. These metrics are then averaged on all the runs as illustrated in (15)—(18). Such averaging mechanism is also applied to the following metrics observed. In other words, the sub-population sizes of the *local search group*, *global search group* and *random search group* are recorded in this way.

$$\overline{\mathrm{S}_{\mathrm{r}ij}} = \frac{\sum_{\mathrm{d}=1}^{\mathrm{D}} \left|\mathrm{S}_{\mathrm{rijd}}\right|}{\mathrm{D}} \tag{15}$$

$$\overline{\mathrm{S}_{\mathrm{r}i}} = \frac{\sum_{j=1}^{\mathrm{n}} |\mathrm{S}_{\mathrm{rij}}|}{\mathrm{n}} \tag{16}$$

$$\overline{\mathrm{S}_{\mathrm{r}}} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{I}} \overline{\mathrm{S}_{ri}}}{\mathrm{I}} \tag{17}$$

$$\overline{\mathrm{S}} = \frac{\sum_{\mathrm{r}=1}^{\mathrm{R}} \overline{\mathrm{S}_{\mathrm{r}}}}{\mathrm{R}} \tag{18}$$

where r, i, j and d represent the run no, iteration no, individual no and problem dimension, respectively; n is the sub-population size, D means the total dimension of the problem, I is the total number of iterations, R is the number of runs, and S_{rijd} is a real value of the search step size for one dimension.

Secondly, the contribution rate is constructed to investigate which sub-population contributes better solutions to the algorithm during the search. In each run, all iterations in which

TABLE 5. The Comparison of the Average Execution Time (in CPU Seconds).

	AMPO	SFO	SSA	WOA	LSHADE	LSHADE-cnEpSin	SHADEILS	HFPSO	MOFOA	MPEDE
F01	0.30	0.48	**0.24**	1.12	1.91	2.44	3.24	0.57	0.69	0.88
F02	0.67	**0.26**	0.54	1.23	2.03	2.49	2.86	0.64	0.75	0.95
F03	0.55	0.43	**0.42**	1.08	1.76	2.34	2.66	0.72	0.61	0.78
F04	0.64	0.51	**0.48**	1.22	1.92	2.52	4.72	0.60	0.68	0.93
F05	0.64	0.52	**0.50**	1.25	2.03	2.47	3.22	0.59	0.70	0.87
F06	8.04	40.15	**6.26**	15.69	21.64	37.87	30.92	6.90	8.40	11.95
F07	**110.85**	198.43	158.07	187.47	167.36	176.36	180.16	154.80	155.90	167.15
F08	7.97	28.40	**6.01**	15.76	21.11	37.09	27.65	6.80	8.31	11.40
F09	7.97	40.39	**6.14**	15.72	21.17	32.62	29.09	6.88	8.07	11.10
F10	9.35	42.13	**7.82**	17.45	23.87	39.84	29.21	8.46	10.28	12.93
F11	13.13	46.48	**12.41**	23.13	26.76	41.91	34.22	13.05	14.84	18.24
F12	9.45	41.94	**7.54**	17.74	22.27	26.12	30.66	8.25	9.74	13.21
F13	11.31	43.15	**8.88**	18.88	24.54	40.80	35.43	9.66	11.11	14.71
F14	**9.14**	46.75	12.66	23.41	27.32	33.92	36.75	13.29	15.16	17.62
F15	18.45	51.31	**16.79**	26.75	31.75	35.34	37.59	17.38	18.89	20.99
F16	10.82	43.36	**8.98**	18.66	24.32	27.36	32.35	9.64	11.25	14.08
F17	**8.49**	44.92	10.67	20.63	31.65	29.08	35.62	11.10	12.99	16.03
F18	9.14	41.42	**7.63**	18.34	22.19	32.95	29.44	8.36	9.80	12.52
F19	9.70	41.62	**7.71**	17.84	20.88	26.99	31.63	8.35	9.97	12.71
F20	9.20	41.33	**7.49**	17.45	23.72	45.86	31.79	7.90	10.06	13.22
F21	10.14	42.09	**7.35**	18.70	25.45	28.10	30.68	9.03	11.19	13.97
F22	9.81	41.91	**7.97**	18.47	26.12	27.87	30.83	8.62	10.69	13.74
F23	13.20	45.02	**11.63**	22.45	28.20	44.46	35.59	12.74	14.88	17.67
F24	11.68	42.60	**9.88**	20.43	27.00	30.10	32.69	10.62	13.19	15.71
F25	12.42	43.45	**11.06**	21.13	27.27	42.93	34.61	11.55	13.83	16.42
F26	49.42	81.64	**47.96**	62.13	64.09	79.49	72.07	49.86	52.15	54.02
F27	49.40	67.25	**48.54**	61.76	64.29	83.72	72.48	49.40	52.14	52.89
F28	14.41	26.74	**10.63**	19.14	31.55	48.69	41.62	16.43	18.55	21.30

TABLE 6. The Results of the Scalability Test for 50-D Benchmark Functions.

	AMPO	LS-cnEpSin	MOFOA	WOA
F06	**0.0000E+00**	5.2907E-156	2.6183E-199	**0.0000E+00**
F07	**0.0000E+00**	5.2673E-08	**0.0000E+00**	1.1046E+02
F08	**0.0000E+00**	1.8182E-78	**0.0000E+00**	**0.0000E+00**
F09	**0.0000E+00**	1.7339E-68	2.4361E-225	3.4242E+01
F10	**0.0000E+00**	7.7269E-77	**0.0000E+00**	**0.0000E+00**
F11	**4.4409E-16**	3.9968E-15	4.4409E-16	2.7362E-15
F12	**0.0000E+00**	6.8181E-03	4.5607E-156	**0.0000E+00**
F13	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**
F14	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	4.5338E-04
F15	**4.6748E-05**	7.4546E-04	3.7683E-04	1.3304E-04
F16	**0.0000E+00**	1.9866E+01	**0.0000E+00**	1.1002E-14
F17	**0.0000E+00**	1.0633E-01	2.2866E-04	1.3536E-01

TABLE 7. The Results of the Scalability Test for 100-D Benchmark Functions.

	AMPO	LS-cnEpSin	MOFOA	WOA
F06	**0.0000E+00**	2.7716E-156	1.4036E-05	**0.0000E+00**
F07	**0.0000E+00**	6.8631E-04	**0.0000E+00**	3.7261E+03
F08	**0.0000E+00**	5.6451E-78	**0.0000E+00**	**0.0000E+00**
F09	**0.0000E+00**	9.6348E-68	**0.0000E+00**	4.1506E+01
F10	**0.0000E+00**	1.2258E-75	**0.0000E+00**	**0.0000E+00**
F11	**4.4409E-16**	3.9968E-15	4.4409E-16	3.3092E-15
F12	**0.0000E+00**	2.9484E-02	1.1199E-275	**0.0000E+00**
F13	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**
F14	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**
F15	**1.4970E-05**	6.5125E-04	1.2647E-03	4.4834E-05
F16	**0.0000E+00**	1.1428E+02	0.0000E+00	0.0000E+00
F17	**0.0000E+00**	1.0310E-01	0.0000E+00	1.2891E-01

TABLE 8. The Results of the Scalability Test for 200-D Benchmark Functions.

	AMPO	LS-cnEpSin	MOFOA	WOA
F06	**0.0000E+00**	3.0784E-178	9.4679E-07	**0.0000E+00**
F07	**0.0000E+00**	3.3900E-02	**0.0000E+00**	4.0206E+04
F08	**0.0000E+00**	5.2785E-76	**0.0000E+00**	**0.0000E+00**
F09	**0.0000E+00**	2.9436E-69	3.0090E-04	4.6442E+01
F10	**0.0000E+00**	1.0371E-76	**0.0000E+00**	0.0000E+00
F11	**4.4409E-16**	3.5384E-15	1.7639E-10	2.9654E-15
F12	**0.0000E+00**	4.4114E-10	5.3293E-203	**0.0000E+00**
F13	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**
F14	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**
F15	**8.4249E-06**	4.0659E-04	1.2276E-03	2.4608E-05
F16	**0.0000E+00**	1.2658E-02	**0.0000E+00**	**0.0000E+00**
F17	**0.0000E+00**	9.9873E-02	1.8630E-03	1.4182E-01

the algorithm achieves a better solution than the last iteration are counted, which is called the effective iteration, and then the sub-population type of the individual who generates such a solution is recorded as one contribution for this sub-population. At the end of the run, the contribution rate of each sub-population is calculated based on its percentage of total contributions in all effective iterations.

Lastly, the reset rate is also recorded. Such rate is defined as the percentage of the iterations where the *reset* operation is performed in all iterations.

Table 15 shows the statistical results of the metrics above. From the table, we may observe that:

- The search step sizes of the *local search group* are much smaller than those of the *global search group*. For example, they are 0.0056 versus 0.6335 in $F04$. However, the search step size of the *local search group* is even a bit larger than that of the *global search group* in some cases, namely $F02$, $F19$ and $F12$. This is because the so-called *local search group* individuals enlarge their search step

TABLE 9. The parameter sensitivity analysis of the results on N_{pop}.

	10	30	100	200
F02	2.5074E-07 ⊖	2.5074E-07 ⊕	2.5074E-07 ⊖	2.5074E-07 ⊖
F03	4.8671E-04 ⊖	3.1874E-16 ⊕	5.5464E-12 ⊜	1.9232E-07 ⊜
F04	1.5561E-284 ⊕	1.6621E-168 ⊕	3.9864E-71 ⊖	4.5574E-39 ⊖
F06	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F08	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	4.7445E-290 ⊖
F10	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	3.3220E-291 ⊖
F12	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	2.0526E-291 ⊖
F15	3.2653E-04 ⊖	1.0841E-04 ⊜	6.2513E-05 ⊜	2.8820E-05 ⊕
F17	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F19	2.0021E+01 ⊜	2.0066E+01 ⊜	2.0017E+01 ⊜	2.0000E+01 ⊜
F20	1.2600E+02 ⊖	2.2948E+01 ⊖	3.2098E-01 ⊕	2.5721E+01 ⊖
F23	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F25	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F26	1.0043E+02 ⊖	1.0032E+02 ⊖	1.0030E+02 ⊜	1.0030E+02 ⊜
F27	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
⊕|⊖|⊜	1|5|9	3|2|10	1|2|12	1|6|8

TABLE 10. The parameter sensitivity analysis of the results on w.

	0.3	0.5	0.7	0.9
F02	2.5074E-07 ⊜	2.5074E-07 ⊜	2.5074E-07 ⊜	2.7254E-07 ⊜
F03	3.7506E-09 ⊖	2.7149E-04 ⊖	9.5813E-04 ⊖	2.3185E-03 ⊖
F04	6.4752E-113 ⊜	1.7643E-116 ⊜	3.1541E-110 ⊜	1.9125E-115 ⊜
F06	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F08	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F10	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F12	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F15	5.0774E-05 ⊜	2.7691E-05 ⊕	1.9236E-05 ⊕	1.5075E-05 ⊕
F17	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F19	2.0082E+01 ⊖	2.0303E+01 ⊖	2.0747E+01 ⊖	2.0762E+01 ⊖
F20	6.0661E+00 ⊜	6.4512E+00 ⊜	6.6441E+00 ⊜	6.7145E+00 ⊜
F23	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F25	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F26	1.0026E+02 ⊜	1.0027E+02 ⊜	1.0026E+02 ⊜	1.0032E+02 ⊖
F27	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
⊕|⊖|⊜	0|2|13	1|2|12	1|2|12	1|3|11

TABLE 11. The parameter sensitivity analysis of the results on γ.

	0.1	0.3	0.5	0.7
F02	2.5074E-07 ⊜	2.5074E-07 ⊜	2.5074E-07 ⊜	2.5074E-07 ⊜
F03	3.3757E-05 ⊖	4.6981E-07 ⊖	1.9487E-13 ⊜	7.4149E-14 ⊜
F04	7.7000E-30 ⊖	2.1832E-29 ⊖	1.1323E-38 ⊖	5.9718E-70 ⊖
F06	1.6429E-73 ⊖	8.3207E-265 ⊖	0.0000E+00 ⊜	0.0000E+00 ⊜
F08	5.1117E-46 ⊖	7.1959E-132 ⊖	3.1812E-293 ⊖	0.0000E+00 ⊜
F10	6.4154E-45 ⊖	1.0138E-128 ⊖	4.2827E-297 ⊖	0.0000E+00 ⊜
F12	2.3286E-06 ⊖	5.3596E-58 ⊖	2.4480E-301 ⊖	0.0000E+00 ⊜
F15	2.4326E-03 ⊖	1.1070E-03 ⊖	5.5501E-04 ⊖	1.7718E-04 ⊜
F17	1.9987E-01 ⊖	5.7991E-02 ⊖	0.0000E+00 ⊜	0.0000E+00 ⊜
F19	2.0004E+01 ⊜	2.0000E+01 ⊜	2.0036E+01 ⊜	2.0020E+01 ⊜
F20	6.0660E+00 ⊜	6.7079E+00 ⊜	6.3549E+00 ⊜	6.2907E+00 ⊜
F23	3.1524E+02 ⊖	3.0409E+02 ⊖	2.0000E+02 ⊜	2.0000E+02 ⊜
F25	2.0286E+02 ⊖	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F26	1.0027E+02 ⊜	1.0030E+02 ⊜	1.0027E+02 ⊜	1.0028E+02 ⊜
F27	3.2703E+02 ⊖	3.3060E+02 ⊖	2.1998E+02 ⊖	2.0000E+02 ⊜
⊕|⊖|⊜	0|11|4	0|10|5	0|6|9	0|1|14

sizes via the *reset* operation when the algorithm falls into local optima in these cases;

- The sub-population sizes of the *local search group*, *global search group* and *random search group* are dynamic. Once the algorithm gets close to the global optimal, the *local search group* will expand for a better exploitation ability via the *transformation* and *reset* operations. The examples include $F04$, $F06$ and $F08$.

TABLE 12. The parameter sensitivity analysis of the results on *PR*.

	0.2	0.4	0.8
F02	2.5074E-07 ⊖	2.5074E-07 ⊖	2.5074E-07 ⊖
F03	4.9896E-05 ⊖	3.0807E-08 ⊖	1.6976E-15 ⊕
F04	5.1950E-57 ⊖	1.5701E-91 ⊖	1.0780E-146 ⊕
F06	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F08	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F10	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F12	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F15	3.9699E-04 ⊖	3.1777E-04 ⊖	1.1864E-04 ⊜
F17	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F19	2.0042E+01 ⊖	2.0014E+01 ⊜	2.0017E+01 ⊜
F20	6.7420E-01 ⊕	1.8615E+00 ⊕	3.4182E+01 ⊖
F23	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F25	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F26	1.0026E+02 ⊜	1.0029E+02 ⊜	1.0036E+02 ⊖
F27	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
⊕|⊖|⊜	1|5|9	1|4|10	2|3|10

TABLE 13. The parameter sensitivity analysis of the results on P_{LS}^{LD}.

	0.2	0.4	0.6
F02	2.5074E-07 ⊜	2.5074E-07 ⊜	2.5074E-07 ⊜
F03	3.2984E-13 ⊜	5.5654E-15 ⊜	1.3560E-12 ⊜
F04	4.5111E-94 ⊖	3.2982E-101 ⊖	1.2915E-113 ⊜
F06	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F08	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F10	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F12	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F15	2.6408E-04 ⊖	2.5563E-04 ⊖	1.5563E-04 ⊜
F17	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F19	2.0007E+01 ⊜	2.0020E+01 ⊜	2.0021E+01 ⊜
F20	6.4191E+00 ⊜	6.3228E+00 ⊜	7.0931E+00 ⊜
F23	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F25	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F26	1.0029E+02 ⊜	1.0024E+02 ⊜	1.0028E+02 ⊜
F27	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
⊕|⊖|⊜	0|2|13	0|2|8	0|0|10

TABLE 14. The parameter sensitivity analysis of the results on P_{LS}^{LS}.

	0.2	0.4	0.6
F02	2.5074E-07 ⊜	2.5074E-07 ⊖	2.5074E-07 ⊜
F03	1.0239E-14 ⊜	3.5808E-12 ⊜	1.0818E-12 ⊖
F04	2.1423E-85 ⊖	1.7544E-102 ⊖	1.4130E-108 ⊜
F06	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F08	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F10	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F12	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F15	1.1812E-03 ⊖	5.6644E-04 ⊖	2.0186E-04 ⊜
F17	0.0000E+00 ⊜	0.0000E+00 ⊜	0.0000E+00 ⊜
F19	2.0004E+01 ⊜	2.0010E+01 ⊜	2.0002E+01 ⊜
F20	6.2586E+00 ⊜	6.0019E+00 ⊜	6.6759E+00 ⊜
F23	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F25	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
F26	1.0028E+02 ⊜	1.0028E+02 ⊜	1.0028E+02 ⊜
F27	2.0000E+02 ⊜	2.0000E+02 ⊜	2.0000E+02 ⊜
⊕|⊖|⊜	0|2|13	0|3|12	0|1|14

By contrast, the algorithm enlarges the size of the *global search group* and *random search group* to improve the exploration, as seen in $F15$, $F19$ and $F20$;

- Different groups contribute better solutions to the search in optimizing different problems. For example, the biggest contributor in $F06$ or $F12$ (uni-modal functions) is the *local search group*. The opposite phenomenon can be seen in $F02$ and $F19$. In $F15$, the contribution rates of the *global search group* and *local search group* are closing to each other, and also the *random search group* contributes 9.73% of better solutions during the search. The *migrating group* is able to contribute good solution(s) to the search as well, such as in $F02$;
- During the search, the *reset* operation is performed in 40%+ of the iterations. As described in Section II-B6, this operation is vital to the success of the AMPO.

Similar to the convergence test, the run with the median fitness is selected to plot Fig. 13, where four functions are illustrated. Generally, the search step size of the *local search group* declines quickly with finding better solutions (see Fig. 13a & Fig. 13b). On the other hand, as stated above, the *local search group* individuals themselves may also increase their search step sizes once no better solutions are found (see 13d). Specifically, there is no search step size of the *local search group* recorded in some periods shown in Fig. 13c. This is because no *local search group* individuals are available during that period. In fact, $F15$ involves a random noise, which produces different fitness values in each evaluation, even with the same solution input. It is difficult for the AMPO to converge into the optimal solution, thereby downsizing the *local search group* while upsizing the *global search group*. The contribution rates of the *local search group* and the *global search group* are around 40% and 50%. Owing to this orchestration, the AMPO beats all the other competitors in this problem, as presented in Section III-C.

By the study, the AMPO has demonstrated its powerful adaptivity in tackling various problems.

VI. REAL-WORLD APPLICATION: PORTFOLIO OPTIMIZATION

A. THE PROBLEM FORMULATION

Portfolio optimization is a significant problem in computational finance. Investors usually want to maximize returns and minimize risks through portfolio diversification, i.e. allocating a fixed amount of capital into a collection of assets.

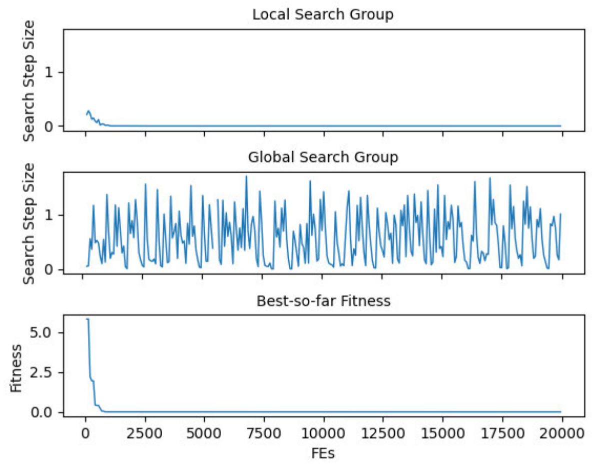
According to the mean-variance model formulated by [73], the variance is regarded as a measure of risks that investors expose to. The optimization problem is presented in (19) as below.

$$\max \ E(R(\boldsymbol{X})) = \sum_{i=1}^{n} x_i r_i$$

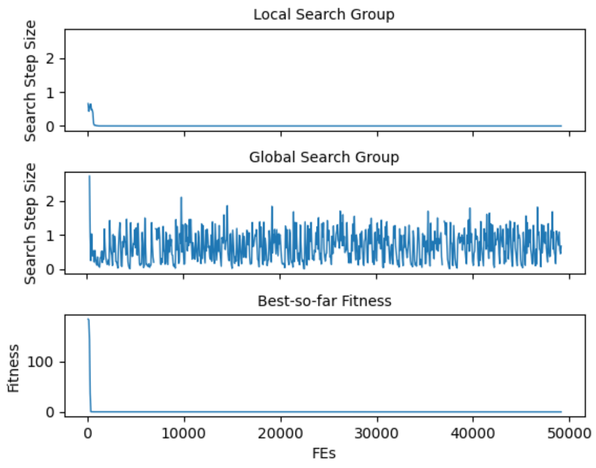
$$\min \ V(R(\boldsymbol{X})) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \sigma_{ij}$$

$$\text{s.t. } x_i \in \boldsymbol{X} = \{x_i \in R \,|\, \sum_{i=1}^{n} x_i = 1, \ 0 \leq x_i \leq 1\} \tag{19}$$

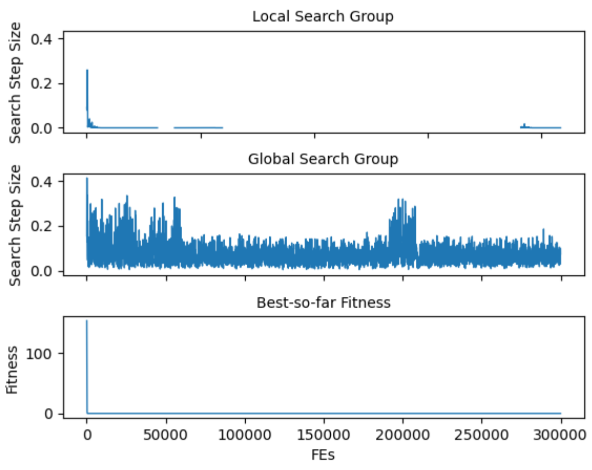
where $\boldsymbol{X}$ is the proportion weight vector, $E(R(\boldsymbol{X}))$ and $V(R(\boldsymbol{X}))$ are the expected return and variance of the whole portfolio respectively, x_i is the weight of the initial capital



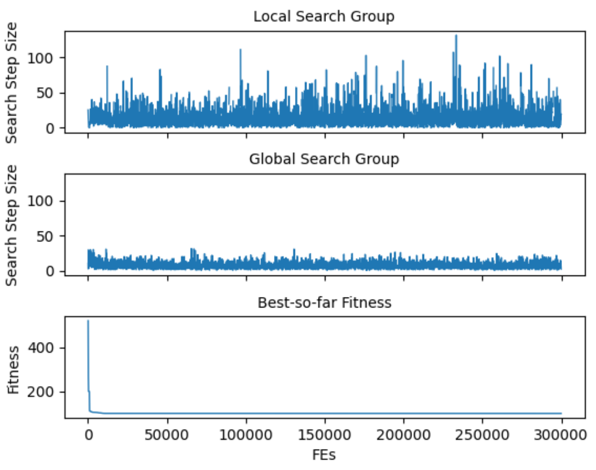
(a) Function: F04



(b) Function: F06



(c) Function: F15



(d) Function: F26

FIGURE 13. **The Analysis of Search Step Sizes of Global and Local Search Groups.**

TABLE 15. **The Statistics of the Search Behavior of the AMPO on the Representative Functions.**

Metric*	F02	F04	F06	F08	F12	F15	F19	F20	F23	F26
LS Group Search Size	1.0030	0.0056	0.0060	0.0646	0.0067	0.0077	13.4324	12.6264	0.0255	14.5455
GS Group Search Size	0.5315	0.6335	0.6641	13.0427	1.3054	0.0674	10.1060	8.6124	13.1193	7.4181
LS Group Number	3.1513	17.5770	17.8379	17.8840	17.8416	0.6752	4.8329	4.1029	17.9777	1.9769
GS Group Number	24.0877	10.7669	10.6916	10.6798	10.7327	26.8408	23.1349	23.6986	10.6403	25.2566
RS Group Number	1.7610	0.6560	0.4705	0.4361	0.4257	2.4686	1.0323	1.1986	0.3821	1.7664
LS Group Contribution Rate	0.0145	0.9616	0.9852	0.9919	0.9931	0.4019	0.0007	0.0041	0.8517	0.1138
GS Group Contribution Rate	0.9154	0.0280	0.0118	0.0064	0.0052	0.5008	0.9981	0.9915	0.1147	0.8660
MG Group Contribution Rate	0.0431	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0037	0.0000	0.0077
RS Contribution Rate	0.0269	0.0104	0.0030	0.0017	0.0017	0.0973	0.0012	0.0007	0.0336	0.0125
Reset Rate	0.4103	0.4651	0.4713	0.4719	0.4712	0.3859	0.4337	0.4286	0.4731	0.4053

* LS: Local Search; GS: Global Search; RS: Random Search; MG: Migrating.

that will be allocated in the ith asset, r_i is the return of the ith asset, n is the total number of assets, and σ_{ij} stands for the covariance of the returns of the ith and jth assets.

The portfolio based on the above model is restricted. When the constrained condition $0 \le x_i \le 1$ is removed, it means that the weights can be negative. Such a portfolio becomes unrestricted, and both longing and shorting on stocks are included. As a matter of fact, short selling is allowed in real-world markets, where investors can sell stocks that they do not hold and must repurchase them later.

Portfolio optimization is fundamentally a multi-objective optimization problem. One feasible approach is to transfer the problem into a single-objective optimization problem. There are two transfer methods: the first one is to select the return or risk as the objective function to be optimized while the rest are defined as constraints; an alternative method is to establish only one objective function by assigning weights to multiple objectives. In the first conventional approach, the mean-variance efficient frontier of the portfolio can be constructed for decision-making. This thesis seeks to optimize the Sharpe Ratio (SR) of the portfolio [74], which belongs to the above second transfer approach. The SR combines the information from the mean and variance of an asset. It is a risk-adjusted measure of the investment return. Until now, the SR has become a critical tool in the modern financial industry to evaluate the performance of various investment portfolios.

The definition of the SR is shown in (20).

$$SR = \frac{E(R(\boldsymbol{X})) - R_f}{V(R(\boldsymbol{X}))} = \frac{\sum_{i=1}^{n} x_i r_i - R_f}{\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j \sigma_{ij}} \tag{20}$$

where SR is the SR of the portfolio, R_f is a risk-free rate, the representations of $\boldsymbol{X}$, x_i, σ_{ij} and r_i are same with those in (19).

Since the AMPO and other comparative algorithms are designated for handling minimization problems, the problem should be changed to the minimization problem as given in (21).

$$\min \mathit{fitness} = \frac{1}{SR} \tag{21}$$
$$\text{s.t. } SR = 10^{-10} \text{ if } SR \leq 0$$

As investment returns probably are zero or even negative in the financial market, a tiny number 10^{-10} is assigned to the SR for dealing with this case.

To avoid handling the equality constraint, the problem is converted to the unconstrained form as shown below.

$$x_i' = \frac{x_i}{\sum_{i=1}^{n} |x_i|} \tag{22}$$

In order to perform more accurate and realistic simulations on the real-world markets, both restricted and unrestricted scenarios of the portfolio are studied in this work.

B. THE EXPERIMENTAL SETUP

In this part of the experiment, the real-world datasets recently released by [75] for portfolio optimization are used as our experimental targets. The datasets consist of six major stock indexes: Dow Jones Industrial Average (DowJones), Nasdaq 100 (NASDAQ100), Financial Times Stock Exchange 100 (FTSE100), S&P 500(SP500), Fama and French 49 Industry (FF49Industries) and NASDAQ Composite (NASDAQComp). The datasets contain cleaned weekly return values that have been adjusted for dividends and stock splits. Table 16 shows the details of such datasets, where the maximum number of stocks is from 28 up to 1, 203 so that the scalability of the metaheuristic can be well tested in solving this real-world problem as well.

Following the previous practice, the information of mean and covariance is acquired from such weekly historical data.

TABLE 16. A Concise Description of the Portfolio Datasets.

Dataset	No. of Stocks	No. of Instances	Time Interval
DowJones	28	1363	Feb 1990-Apr 2016
FF49Industries	49	2325	Jul 1969-Jul 2015
NASDAQ100	82	596	Nov 2004-Apr 2016
FTSE100	83	717	Jul 2002-Apr 2016
SP500	442	595	Nov 2004-Apr 2016
NASDAQComp	1203	685	Feb 2003-Apr 2016

In the experiment, such values are calculated through all recorded prices for each dataset. More specifically, for each stock, the average of the weekly returns is computed as the expected return of the stock. The daily values of the yields of the U.S. and the U.K. 5-year treasuries during the time interval that the dataset involved are averaged to act as the risk-free rates for the U.S. and U.K. markets, respectively. These risk-free rates (yearly) are 4.20%, 5.68%, 2.42%, 2.94%, 2.40% and 2.43% for DowJones, FF49Industries, NASDAQ100, FTSE100, SP500 and NASDAQComp, respectively. By convention, the SR is converted into the annualized SR for better understanding and comparison, as shown in (23).

$$\mathit{Annualized}\ SR = \frac{E(R(\boldsymbol{X})) - R_f/52}{V(R(\boldsymbol{X}))} \times \sqrt{52} \tag{23}$$

where 52 stands for the number of weeks in a year.

Apart from longing on stocks (restricted portfolio), we study a real-world scenario in which short-selling is allowed (unrestricted portfolio), i.e. x_i can be negative, which enlarges the searching space of the problem.

In this part, the experiment involves three representative algorithms that perform well in the benchmark function test to verify their effectiveness in tackling these practical problems. They are the WOA, LS-cnEPSin and MOFOA. Among them, the excellent performance of WOA in portfolio optimization is also reported by [47]. In addition, referring to [76], [77], we select three other competitive algorithms that demonstrate their excellent performance in optimizing the same problem in the literature, namely, Adaptive Particle Swarm Optimization (APSO) [44], Fireworks Algorithm (FA) [45] and Harmony Search Algorithm (HSA) [46]. Due to a large dimension of some datasets, the MaxFEs is set to 5×10^5 for each algorithm to achieve a trade-off between computational cost and performance. Other user-controlled parameters of all algorithms are the same as those used in the benchmark function test.

C. SIMULATION RESULTS AND DISCUSSION

The simulation results of the portfolio optimization are shown in Table 17 and Table 18, where the mean values and standard deviations of the optimized annualized SR in 30 runs are presented. Also, the comparison of computational cost measured in CPU seconds is provided there.

From the tables, we can observe that the AMPO generates the best SR over all the competing algorithms in 11 out of 12 portfolios. Notably, the AMPO algorithm delivers

TABLE 17. The Comparative Results of the Portfolio Optimization between AMPO and Selected Algorithms (Restricted).

Dataset	Metric	AMPO	LS-cnEpSin	MOFOA	WOA	APSO	FA	HS
DowJones	Sharpe Ratio (Mean)	**0.9362**	0.9360	0.8976	0.8575	0.8047	0.8578	0.9189
	Sharpe Ratio (Std)	0.0008	0.0008	0.0069	0.0324	0.0332	0.0124	0.0040
	Avg Time (s)	51.68	76.78	55.45	73.00	42.60	112.92	228.61
FF49Industries	Sharpe Ratio (Mean)	**1.1498**	1.1338	1.0946	0.9978	0.9458	1.0017	1.0821
	Sharpe Ratio (Std)	0.0022	0.0028	0.0078	0.0267	0.0181	0.0149	0.0064
	Avg Time (s)	44.68	324.59	47.19	59.86	43.49	127.55	345.85
NASDAQ100	Sharpe Ratio (Mean)	**1.5970**	1.5765	1.3731	1.1203	0.6822	0.8321	0.9997
	Sharpe Ratio (Std)	0.0013	0.0057	0.0232	0.0902	0.0160	0.0301	0.0134
	Avg Time (s)	54.72	556.65	56.91	74.24	45.00	158.67	536.39
FTSE100	Sharpe Ratio (Mean)	**1.3417**	1.3182	1.1139	0.8659	0.9971	1.1279	1.3181
	Sharpe Ratio (Std)	0.0025	0.0064	0.0258	0.1064	0.0335	0.0243	0.0135
	Avg Time (s)	54.84	570.75	56.98	73.35	44.35	156.80	530.40
SP500	Sharpe Ratio (Mean)	**1.2508**	0.9018	1.2158	0.7309	0.5622	0.5969	0.6703
	Sharpe Ratio (Std)	0.0280	0.0166	0.0422	0.1422	0.0101	0.0097	0.0048
	Avg Time (s)	2302.49	4617.62	2310.88	2744.47	3283.40	10992.76	51986.24
NASDAQComp	Sharpe Ratio (Mean)	1.3159	0.9303	**1.3195**	0.8854	0.8109	0.8381	0.9050
	Sharpe Ratio (Std)	0.0207	0.0247	0.0293	0.0675	0.0095	0.0100	0.0035
	Avg Time (s)	5179.65	16440.75	4943.72	5423.06	4532.96	25017.23	142506.28

TABLE 18. The Comparative Results of the Portfolio Optimization between AMPO and Selected Algorithms (Unrestricted).

Dataset	Metric	AMPO	LS-cnEpSin	MOFOA	WOA	APSO	FA	HS
DowJones	Sharpe Ratio (Mean)	**0.9360**	0.9348	0.8663	0.8448	0.7118	0.8656	0.9167
	Sharpe Ratio (Std)	0.0006	0.0015	0.0112	0.0454	0.0249	0.0133	0.0040
	Avg Time (s)	51.76	77.79	55.43	73.22	43.18	112.65	228.41
FF49Industries	Sharpe Ratio (Mean)	**1.1537**	1.1274	1.0093	0.9603	0.8367	0.9946	1.0676
	Sharpe Ratio (Std)	0.0034	0.0030	0.0194	0.0345	0.0302	0.0141	0.0066
	Avg Time (s)	43.46	327.72	46.04	58.66	43.49	130.75	346.99
NASDAQ100	Sharpe Ratio (Mean)	**2.3324**	2.3038	1.3750	1.2321	1.4270	1.7641	2.0959
	Sharpe Ratio (Std)	0.0107	0.0055	0.0326	0.0768	0.1210	0.0575	0.0179
	Avg Time (s)	54.98	623.16	57.03	74.31	44.44	156.98	531.74
FTSE100	Sharpe Ratio (Mean)	**1.7252**	1.7039	1.0610	0.9470	0.9004	1.1981	1.4874
	Sharpe Ratio (Std)	0.0076	0.0104	0.0337	0.0815	0.0822	0.0557	0.0199
	Avg Time (s)	55.15	595.65	57.30	73.46	45.02	157.66	537.26
SP500	Sharpe Ratio (Mean)	**3.0476**	2.8831	1.0936	0.6901	0.6705	0.8779	1.5928
	Sharpe Ratio (Std)	0.0557	0.0466	0.0548	0.0730	0.0987	0.0587	0.0292
	Avg Time (s)	2304.76	4811.22	1676.18	3033.20	3434.69	11299.45	51825.56
NASDAQComp	Sharpe Ratio (Mean)	**4.7972**	3.4231	0.9792	0.8922	0.7929	1.0029	1.8566
	Sharpe Ratio (Std)	0.1684	0.0854	0.0428	0.0666	0.0937	0.0604	0.0373
	Avg Time (s)	5123.95	15787.81	4947.05	5522.45	4536.46	26277.65	142595.23

TABLE 19. The Statistical Test Results of the Portfolio Optimization in *p*-value.

Portfolio	APSO	FA	HAS	LS-cnEpSin	MOFOA	WOA
DowJones (Restricted)	0.0000	0.0000	0.0000	0.1916	0.0000	0.0000
DowJones (Unrestricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
FF49Industries (Restricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
FF49Industries (Unrestricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
FTSE100 (Restricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
FTSE100 (Unrestricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NASDAQ100 (Restricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NASDAQ100 (Unrestricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NASDAQComp (Restricted)	0.0010	0.0010	0.0010	0.0010	0.5391	0.0010
NASDAQComp (Unrestricted)	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010
SP500 (Restricted)	0.0000	0.0000	0.0000	0.0000	0.0004	0.0000
SP500 (Unrestricted)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

much better values than the second-best values in optimizing large-scale unrestricted portfolios on the SP500 (3.0476 versus 2.8831) and NASDAQComp (4.7972 versus 3.4231).

The average ranks are respectively 1.08, 2.17, 3.50, 3.67, 5.00, 5.75 and 6.83 for the AMPO, LS-cnEpSin, HSA, MOFOA, FA, WOA and APSO algorithms. For the second-best optimizer, namely LS-cnEpSin algorithm, its performance on the small-scale restricted portfolios, i.e. DowJones, FF49Industries, NASDAQ100 and FTSE10, is only next to the AMPO, but it gets much worse results in constructing restricted portfolios for the large-scale SP500 and NASDAQComp datasets. On the other hand, the LS-cnEpSin algorithm produces the contrary phenomenon for unrestricted portfolios. These findings suggest that the adaptivity of LS-cnEpSin is in question. Also, the LS-cnEpSin becomes much computationally expensive when processing large-scale problems. This is associated with the time-consuming calculation that is discussed in Section III-F. Here the AMPO is 3, even 10 times faster than the LS-cnEpSin.

Concerning the MOFOA optimizer, it produces low-quality solutions for such small-scale problems but much better results for the restricted NASDAQComp portfolio, in which it gets the best value 1.3195 among all algorithms.

In regard to the WOA algorithm that achieves a satisfactory performance in the benchmark function test, it fails to produce good results here, which implies it contains limited adaptivity and effectiveness of dealing with these practical problems.

Another inferential statistics test based on the Wilcoxon signed-rank test is performed in addition to the descriptive statistics. The alternative hypothesis is that AMPO outperforms the competing algorithm in terms of solution quality when resolving these tasks. When the p-value is less than 0.05, the alternative hypothesis is accepted. As shown in Table 19, almost all the p values are much smaller than 0.05, even most are rounded to 0. Regarding the only two exceptions, a few extra tests still cannot distinguish the performance between the AMPO and the compared ones. Such outputs have demonstrated that the AMPO is on the whole good at tackling these portfolio problems compared to all the compared algorithms. Also, the excellent scalability of the AMPO has been verified in coping with these practical problems.

VII. CONCLUSION

This paper proposes a competitive metaheuristic optimization algorithm, namely the AMPO, for tackling challenging continuous optimization problems. The algorithm is carefully designed with different operations to diversify the search strategies so as to improve its optimization capability. The AMPO powered by the multi-population design has an excellent adaptivity when compared with the existing metaheuristics. The algorithm can dynamically allocate its search power to different sub-populations that conduct local search or global search during the optimization process for different problems.

In this work, the AMPO is critically evaluated on a total of 28 well-known benchmark functions covering a broad range of characteristics of optimization problems. All the obtained results are carefully compared and analyzed with those of nine state-of-the-art optimization algorithms, including the recent SI approaches, the IEEE CEC winning algorithms and the latest developed multi-population and hybrid optimization algorithms. The results demonstrate the outstanding performance of our proposed algorithm in terms of solution fitness, convergence rate, scalability, stability and computational cost. In particular, the AMPO shows a unique potential for high-dimensional continuous optimization problems. Additionally, the parameter sensitivity analysis and search behavior of the AMPO are investigated. Lastly, the AMPO is applied to solve the challenging portfolio optimization problem with different numbers of assets from ranging 28 up to $1,203$. The result shows that the AMPO can achieve robust performance in constructing these portfolios over other algorithms.

The drawbacks of the AMPO are summarized as follows. First, the number of user-controlled parameters is larger than those of the existing optimization approaches like the GA and PSO. In addition, the algorithm can only handle unconstrained optimization problems at the moment.

Concerning some future work, making the parameters of the AMPO as self-adaptive is worth exploring. Besides, a more thorough investigation should be conducted on applying the AMPO to solve constrained optimization problems. Furthermore, it is interesting to apply the AMPO to different real-world applications for comparison with more recently proposed metaheuristics like the HHO and SMA. Last but not least, the AMPO has a great potential to be extended to tackle multi-objective optimization problems.

The source code of the AMPO is released at https://github.com/rayzxli/AMPO.

REFERENCES

[1] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, ''Optimizing deep learning hyper-parameters through an evolutionary algorithm,'' in *Proc. Workshop Mach. Learn. High-Perform. Comput. Environ.*, Nov. 2015, pp. 1–5.

[2] A. Tharwat, A. E. Hassanien, and B. E. Elnaghi, ''A BA-based algorithm for parameter optimization of support vector machine,'' *Pattern Recognit. Lett.*, vol. 93, pp. 13–22, Jul. 2017.

[3] F. Friedrichs and C. Igel, ''Evolutionary tuning of multiple SVM parameters,'' *Neurocomputing*, vol. 64, pp. 107–117, Mar. 2005.

[4] Y. Nalçakan and T. Ensari, ''Decision of neural networks hyperparameters with a population-based algorithm,'' in *Proc. Int. Conf. Mach. Learn., Optim., Data Sci.* Volterra, Italy: Springer, Sep. 2018, pp. 276–281.

[5] V. K. Ojha, A. Abraham, and V. Snášel, ''Metaheuristic design of feedforward neural networks: A review of two decades of research,'' *Eng. Appl. Artif. Intell.*, vol. 60, pp. 97–116, Apr. 2017.

[6] K. T. Lwin, R. Qu, and B. L. MacCarthy, ''Mean-VaR portfolio optimization: A nonparametric approach,'' *Eur. J. Oper. Res.*, vol. 260, no. 2, pp. 751–766, Jul. 2017.

[7] K. Metaxiotis and K. Liagkouras, ''Multiobjective evolutionary algorithms for portfolio management: A comprehensive literature review,'' *Expert Syst. Appl.*, vol. 39, no. 14, pp. 11685–11698, Oct. 2012.

[8] S. N. Skinner and H. Zare-Behtash, ''State-of-the-art in aerodynamic shape optimisation methods,'' *Appl. Soft Comput.*, vol. 62, pp. 933–962, Jan. 2018.

[9] O. Bozorg-Haddad, M. Solgi, and H. A. Loáiciga, *Meta-Heuristic and Evolutionary Algorithms for Engineering Optimization*. Hoboken, NJ, USA: Wiley, 2017.

[10] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, ''A review of dynamic vehicle routing problems,'' *Eur. J. Oper. Res.*, vol. 225, no. 1, pp. 1–11, 2013.

[11] Z. Huang, X. Lu, and H. Duan, ''A task operation model for resource allocation optimization in business process management,'' *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 5, pp. 1256–1270, Sep. 2012.

[12] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Frome, U.K.: Luniver Press, 2008. [Online]. Available: https://dl.acm.org/doi/book/10.5555/1628847

[13] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*. London, U.K.: Springer-Verlag, 2010.

[14] P. A. Vikhar, ''Evolutionary algorithms: A critical review and its future prospects,'' in *Proc. Int. Conf. Global Trends Signal Process., Inf. Comput. Commun. (ICGTSPICC)*, Dec. 2016, pp. 261–265.

[15] H. R. Maier, S. Razavi, Z. Kapelan, L. S. Matott, J. Kasprzyk, and B. A. Tolson, ''Introductory overview: Optimization using evolutionary algorithms and other metaheuristics,'' *Environ. Model. Softw.*, vol. 114, pp. 195–213, Apr. 2019.

[16] R. S. Parpinelli and H. S. Lopes, ''New inspirations in swarm intelligence: A survey,'' *Int. J. Bio-Inspired Comput.*, vol. 3, no. 1, pp. 1–16, Jan. 2011.

[17] J. Kennedy, ''Swarm intelligence,'' in *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models With Emerging Technologies*, A. Y. Zomaya, Ed. Boston, MA, USA: Springer, 2006, pp. 187–219, doi: 10.1007/0-387-27705-6_6.

[18] X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu, *Swarm Intelligence and Bio-Inspired Computation: Theory and Applications*. Oxford, U.K.: Elsevier, 2013. [Online]. Available: https://www.sciencedirect.com/book/9780124051638/swarm-intelligence-and-bio-inspired-computation?via=ihub=

[19] P. J. M. Van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht, The Netherlands: Springer, 1987. [Online]. Available: https://www.springer.com/gp/book/9789027725134

[20] N. H. Siddique and H. Adeli, *Nature-Inspired Computing: Physics and Chemistry-Based Algorithms*. Boca Raton, FL, USA: CRC Press, 2017.

[21] A. Y. S. Lam and V. O. K. Li, ''Chemical-reaction-inspired metaheuristic for optimization,'' *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 381–399, Jun. 2010.

[22] H. Abedinpourshotorban, S. Mariyam Shamsuddin, Z. Beheshti, and D. N. A. Jawawi, ''Electromagnetic field optimization: A physics-inspired Metaheuristic optimization algorithm,'' *Swarm Evol. Comput.*, vol. 26, pp. 8–22, Feb. 2016.

[23] X.-S. Yang, ''Harmony search as a metaheuristic algorithm,'' in *Music-Inspired Harmony Search Algorithm: Theory and Applications*, Z. W. Geem, Ed. Berlin, Germany: Springer, 2009, pp. 1–14.

[24] S. J. Mousavirad and H. Ebrahimpour-Komleh, ''Human mental search: A new population-based Metaheuristic optimization algorithm,'' *Int. J. Speech Technol.*, vol. 47, no. 3, pp. 850–887, Oct. 2017.

[25] J. Zhang, M. Xiao, L. Gao, and Q. Pan, ''Queuing search algorithm: A novel Metaheuristic algorithm for solving engineering optimization problems,'' *Appl. Math. Model.*, vol. 63, pp. 464–490, Nov. 2018.

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[27] R. Storn and K. Price, ''Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces,'' *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[28] H. G. Beyer and H. P. Schwefel, ''Evolution strategies—A comprehensive introduction,'' *Natural Comput.*, vol. 1, no. 1, pp. 3–52, 2002.

[29] J. Kennedy and R. Eberhart, ''Particle swarm optimization,'' in *Proc. IEEE ICNN*, vol. 4. Nov./Dec. 1995, pp. 1942–1948.

[30] M. Dorigo, M. Birattari, and T. Stutzle, ''Ant colony optimization,'' *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[31] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, ''A comprehensive survey: Artificial bee colony (ABC) algorithm and applications,'' *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, Jun. 2014.

[32] S. Mirjalili, S. M. Mirjalili, and A. Lewis, ''Grey wolf optimizer,'' *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[33] J. J. Q. Yu and V. O. K. Li, ''A social spider algorithm for global optimization,'' *Appl. Soft Comput.*, vol. 30, pp. 614–627, May 2015.

[34] S. Mirjalili and A. Lewis, ''The whale optimization algorithm,'' *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[35] S. Saremi, S. Mirjalili, and A. Lewis, ''Grasshopper optimisation algorithm: Theory and application,'' *Adv. Eng. Softw.*, vol. 105, pp. 30–47, Mar. 2017.

[36] G.-G. Wang, S. Deb, and L. dos S. Coelho, ''Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems,'' *Int. J. Bio-Inspired Comput.*, vol. 12, no. 1, pp. 1–22, Jan. 2018.

[37] G.-G. Wang, ''Moth search algorithm: A bio-inspired Metaheuristic algorithm for global optimization problems,'' *Memetic Comput.*, vol. 10, no. 2, pp. 151–164, Jun. 2018.

[38] R. Salgotra and U. Singh, ''The naked mole-rat algorithm,'' *Neural Comput. Appl.*, vol. 31, no. 12, pp. 8837–8857, Dec. 2019.

[39] G. G. Wang, S. Deb, and Z. Cui, ''Monarch butterfly optimization,'' *Neural Comput. Appl.*, vol. 31, no. 7, pp. 1995–2014, 2019.

[40] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, ''Harris hawks optimization: Algorithm and applications,'' *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, Aug. 2019.

[41] S. Shadravan, H. R. Naji, and V. K. Bardsiri, ''The sailfish optimizer: A novel nature-inspired Metaheuristic algorithm for solving constrained engineering optimization problems,'' *Eng. Appl. Artif. Intell.*, vol. 80, pp. 20–34, Apr. 2019.

[42] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, ''Slime mould algorithm: A new method for stochastic optimization,'' *Future Gener. Comput. Syst.*, vol. 111, pp. 300–323, Oct. 2020.

[43] K. Sörensen, ''Metaheuristics—The metaphor exposed,'' *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, 2015.

[44] H. Zhu, Y. Wang, K. Wang, and Y. Chen, ''Particle swarm optimization (PSO) for the constrained portfolio optimization problem,'' *Expert Syst. Appl.*, vol. 38, no. 8, pp. 10161–10169, Aug. 2011.

[45] N. Bacanin and M. Tuba, ''Fireworks algorithm applied to constrained portfolio optimization problem,'' in *Proc. IEEE Congr. Evol. Comput. (CEC)*, May 2015, pp. 1242–1249.

[46] K. H. Lai, W. J. Siow, and A. A. M. N. Kaw, ''Sharpe ratio-based portfolio optimization using harmony search algorithm,'' *Commun. Comput. Appl. Math.*, vol. 1, no. 1, Mar. 2019.

[47] Q. H. Zhai, T. Ye, M. X. Huang, S. L. Feng, and H. Li, ''Whale optimization algorithm for multiconstraint second-order stochastic dominance portfolio optimization,'' *Comput. Intell. Neurosci.*, vol. 2020, pp. 1–19, Aug. 2020.

[48] J. E. Beasley, ''OR-library: Distributing test problems by electronic mail,'' *J. Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, Nov. 1990.

[49] A. E. Eiben and C. A. Schippers, ''On evolutionary exploration and exploitation,'' *Fundamenta Informaticae*, vol. 35, nos. 1–4, pp. 35–50, 1998.

[50] M. Črepinšek, S.-H. Liu, and M. Mernik, ''Exploration and exploitation in evolutionary algorithms: A survey,'' *ACM Comput. Surv.*, vol. 45, no. 3, pp. 1–33, Jun. 2013.

[51] H. Ma, S. Shen, M. Yu, Z. Yang, M. Fei, and H. Zhou, ''Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey,'' *Swarm Evol. Comput.*, vol. 44, pp. 365–387, Feb. 2019.

[52] X. Xia, L. Gui, and Z.-H. Zhan, ''A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting,'' *Appl. Soft Comput.*, vol. 67, pp. 126–140, Jun. 2018.

[53] G. Wu, R. Mallipeddi, P. N. Suganthan, R. Wang, and H. Chen, ''Differential evolution with multi-population based ensemble of mutation strategies,'' *Inf. Sci.*, vol. 329, pp. 329–345, Feb. 2016.

[54] H. Chen, S. Li, A. Asghar Heidari, P. Wang, J. Li, Y. Yang, M. Wang, and C. Huang, ''Efficient multi-population outpost fruit fly-driven optimizers: Framework and advances in support vector machines,'' *Expert Syst. Appl.*, vol. 142, Mar. 2020, Art. no. 112999.

[55] Z. Li and V. Tam, ''A novel meta-heuristic optimization algorithm inspired by the spread of viruses,'' 2020, *arXiv:2006.06282*. [Online]. Available: http://arxiv.org/abs/2006.06282

[56] S. Das and P. N. Suganthan, ''Differential evolution: A survey of the state-of-the-art,'' *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[57] A. W. Mohamed, H. Z. Sabry, and T. Abd-Elaziz, ''Real parameter optimization by an effective differential evolution algorithm,'' *Egyptian Informat. J.*, vol. 14, no. 1, pp. 37–53, Mar. 2013.

[58] Y. Zhuang, K. Gao, X. Miu, L. Han, and X. Gong, ''Infrared and visual image registration based on mutual information with a combined particle swarm optimization—Powell search algorithm,'' *Optik*, vol. 127, no. 1, pp. 188–191, Jan. 2016.

[59] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, ''Particle swarm optimization: Basic concepts, variants and applications in power systems,'' *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 171–195, Apr. 2008.

[60] T. T. Ngo, A. Sadollah, and J. H. Kim, ''A cooperative particle swarm optimizer with stochastic movements for computationally expensive numerical optimization problems,'' *J. Comput. Sci.*, vol. 13, pp. 68–82, Mar. 2016.

[61] S. Mostafa Bozorgi and S. Yazdani, ''IWOA: An improved whale optimization algorithm for optimization problems,'' *J. Comput. Design Eng.*, vol. 6, no. 3, pp. 243–259, Jul. 2019.

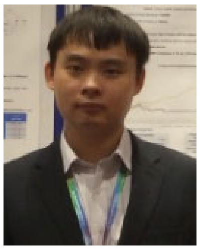
[62] M. N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, ''A comprehensive review of swarm optimization algorithms,'' *PLoS ONE*, vol. 10, no. 5, May 2015, Art. no. e0122827.

[63] J. J. Liang, B. Y. Qu, and P. N. Suganthan, ''Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization,'' Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China, Nanyang Technol. Univ., Singapore, Tech. Rep., 2013, vol. 635, p. 490.

[64] M. Jamil and X.-S. Yang, ''A literature survey of benchmark functions for global optimization problems,'' *Int. J. Math. Model. Numer. Optim.*, vol. 4, no. 2, p. 44, 2013.

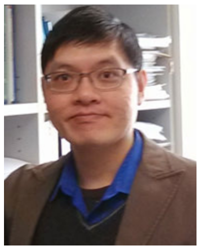
[65] J. Derrac, S. García, D. Molina, and F. Herrera, ''A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms,'' *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, Mar. 2011.

[66] D. Molina, A. LaTorre, and F. Herrera, "An insight into bio-inspired and evolutionary algorithms for global optimization: Review, analysis, and lessons learnt over a decade of competitions," *Cognit. Comput.*, vol. 10, no. 4, pp. 517–544, Aug. 2018.
[67] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 71–78.
[68] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2014, pp. 1658–1665.
[69] N. H. Awad, M. Z. Ali, and P. N. Suganthan, "Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2017, pp. 372–379.
[70] D. Molina, A. LaTorre, and F. Herrera, "SHADE with iterative local search for large-scale global optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
[71] I. B. Aydilek, "A hybrid firefly and particle swarm optimization algorithm for computationally expensive numerical problems," *Appl. Soft Comput.*, vol. 66, pp. 232–249, May 2018.
[72] Z. Li, V. Tam, and L. K. Yeung, "A study on parameter sensitivity analysis of the virus spread optimization," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2020, pp. 1535–1542.
[73] A. Stuart and H. M. Markowitz, "Portfolio selection: Efficient diversification of investments," *OR*, vol. 10, no. 4, p. 253, Dec. 1959.
[74] W. F. Sharpe, "Mutual fund performance," *J. Bus.*, vol. 39, no. 1, pp. 119–138, Jan. 1966.
[75] R. Bruni, F. Cesarone, A. Scozzari, and F. Tardella, "Real-world datasets for portfolio selection and solutions of some stochastic dominance portfolio models," *Data Brief*, vol. 8, pp. 858–862, Sep. 2016.
[76] J. Doering, R. Kizys, A. A. Juan, À. Fitó, and O. Polat, "Metaheuristics for rich portfolio optimisation and risk management: Current state and future trends," *Oper. Res. Perspect.*, vol. 6, 2019, Art. no. 100121.
[77] D. A. Milhomem and M. J. P. Dantas, "Analysis of new approaches used in portfolio optimization: A systematic literature review," *Production*, vol. 30, Jul. 2020.

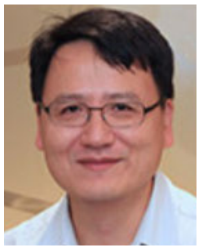


ZHIXI LI (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Electrical and Electronic Engineering, The University of Hong Kong.

He was a Research Staff Member at the Hong Kong Applied Science and Technology Research Institute. He has solid experience in academic research as well as industrial application development. His research interests include machine learning, computational finance, and computational intelligence. He is a Graduate Student Member of the Computational and Financial Econometrics Network with several international refereed publications and academic awards from universities and academic organizations.



VINCENT TAM (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Melbourne, in 1998. He is currently a Principal Lecturer and also Honorary Associate Professor with the Department of Electrical and Electronic Engineering, The University of Hong Kong. He has over 150 internationally refereed publications, including 10 book chapters. His main research interests include big data analytics, computational intelligence, cloud computing, machine learning, and information visualization. Besides, he served as the Chairman of the IEEE (HK) Computational Intelligence Chapter (2014–2017).



LAWRENCE K. YEUNG (Senior Member, IEEE) was born in 1969. He received the B.Eng. and Ph.D. degrees in information engineering from The Chinese University of Hong Kong, in 1992 and 1995, respectively. He joined the Department of Electrical and Electronic Engineering, The University of Hong Kong, in July 2000, where he is currently a Professor. His research interests include next-generation internet, packet switch/router design, all-optical networks, and wireless data networks.