SF-FWA: A Self-adaptive Fast Fireworks Algorithm for Effective Large-scale Optimization

Maiyue Chen\textsuperscript{a,b}, Ying Tan\textsuperscript{a,b,c,}\textsuperscript{*}

\textsuperscript{a}School of Artificial Intelligence, Peking University, Beijing, 100871, China
\textsuperscript{b}Key Laboratory of Machine Perception (MOE), Peking University, Beijing, 100871, China
\textsuperscript{c}Institute for Artificial Intelligence, Peking University, Beijing, 100871, China

Abstract
Computationally efficient algorithms for large-scale black-box optimization have become increasingly important in recent years due to the growing complexity of engineering and scientific problems. In this paper, a novel algorithm called the Self-adaptive Fast Fireworks Algorithm (SF-FWA) is proposed to effectively conduct large-scale black-box optimization. The main idea is to utilize a set of expressive and computationally efficient search distributions to cope with different function landscapes while tuning the hyperparameters of the search distributions in an online fashion. To achieve this, the Expressive Fast Explosion (EFE) mechanism is designed to achieve effective and efficient sampling, and the Inter-Fireworks Competitive Cooperation (IFCC) mechanism is designed to adapt hyperparameter distributions. This new optimization paradigm equips the population with the ability to automatically adjust to a rich set of function landscapes with linear computational complexity in terms of problem dimensionality. Experimental studies show that SF-FWA can not only exploit the separability of the problem efficiently but can also deal with rotational transformations to the coordinate system. The numerical results on the standard large-scale optimization benchmark suite indicate that SF-FWA outperforms current state-of-the-art large-scale optimization algorithms. The outstanding performance of SF-FWA on optimizing neural network controllers for solving reinforcement learning tasks demonstrates its great potential to be applied to a wider range of real-world problems.

Keywords: Black-box optimization, Large-scale optimization, Fireworks algorithm, Continuous control, Reinforcement learning

1. Introduction
The problem of large-scale black-box optimization encompasses a wide range of scenarios that require the optimization of a vast number of decision variables. Conventional metaheuristic algorithms, including Evolutionary Algorithms (EA) and Swarm Intelligence (SI) algorithms, are primarily designed for small-scale problems, typically with less than one hundred variables. However, the computational complexity of these algorithms increases significantly with increasing problem dimensions, making it difficult to apply them to large-scale problems that are prevalent in real-world engineering and scientific domains such as complex filter design [1], large-scale scheduling [2], route planning [3], bioinformatics [4] and reinforcement learning [5, 6].

Numerous valuable contributions have been made to tackle large-scale optimization problems. One approach, as described in [7, 8], involves directly exploiting the problem structure through Cooperative Co-evolution (CC) techniques, which seek to identify the connections and dependencies between decision variables by various methods, such as perturbation, statistical models and domain knowledge [9–14]. This approach is effective in cases where the relationship between variables is simple, such as the additive separable relationship [9]. However, when the underlying structure of the problem is more complex, CC-based algorithms may not be adequate. Another avenue of research aims to modify operators within existing evolutionary algorithms [15–20] or combine them with local search heuristics [21, 22] to address the curse of dimensionality while avoiding assumptions about the objective function. This approach is more versatile and suitable for a wider range of problems. Last, the use of covariance models for metric learning, as outlined in [23, 24], is also a viable strategy. The primary challenge in this approach lies in overcoming the quadratic computational complexity associated with covariance matrices, as highlighted in [25].

Exploiting problem separability and handling coordinate rotation is critical for efficient and effective optimization in large-scale black-box problems. For example, consider an optimization problem with many variables. Exploiting separability allows for solving independent subproblems, making the optimization process more efficient
To clearly distinguish these two types of problems, in this work, and the cooperation mechanism, which defines the interactions between fireworks. By carefully engineering these mechanisms, FW A has already demonstrated its effectiveness in addressing a variety of optimization problems in large-scale optimization. FW A offers a promising framework to design advanced optimization algorithms to automatically adapt both the hyperparameters and search distribution parameters effectively.

The motivation of this work is to leverage the unique properties of the Fireworks Algorithm (FW A) framework, which combines advantages from both evolutionary algorithms (EA) and swarm intelligence (SI) [30–33], to address the challenges of handling both Type-S and Type-R problems in large-scale optimization. FW A offers a promising framework to design advanced optimization algorithms due to its two key mechanisms: the explosion operation, which governs the individual search behavior of each firework, and the cooperation mechanism, which defines the interactions between fireworks. By carefully engineering these mechanisms, FW A has already demonstrated its effectiveness in addressing a variety of optimization problems, including multi-modality [31], multi-objectives [34] and multi-scale landscapes [32].

To fill the aforementioned research gap with FW A, we introduce two novel mechanisms in this paper. Firstly, the Expressive Fast Explosion (EFE) operation combines a diagonal model for exploiting separability with a low-rank model for handling rotated coordinate systems. Secondly, the Inter-Fireworks Competitive Cooperation (IFCC) mechanism adapts both the search distributions and hyperparameters online to dynamically respond to different types of problems. The combination of these two mechanisms results in the novel Self-Adaptive Fast Fireworks Algorithm (SF-FWA), which is the first large-scale optimization algorithm that can automatically adapt to both Type-S and Type-R problems and achieve great performance.

To prove the efficacy of SF-FWA, a number of experiments were conducted. Firstly, we compared SF-FWA to other algorithms to demonstrate its effectiveness on both Type-S and Type-R problems. Secondly, the superiority of IFCC was shown by comparing it with other fixed hyperparameter settings. Thirdly, we tested and compared the scalability of SF-FWA on problem dimensions and found that it performs well even up to 12800 dimensions. Fourthly, we evaluated SF-FWA against current state-of-the-art large-scale optimization methods using standard benchmarks, where SF-FWA emerged as the best performer with the highest average rank. Finally, SF-FWA was applied to reinforcement learning problems, where it was used to optimize neural network controllers for simulated robots with up to 66961 variables, and it was found to significantly outperform existing gradient-free reinforcement learning algorithms.

To summarize, the main contributions of this paper are as follows:

- An expressive and computationally efficient explosion operation called Expressive Fast Explosion (EFE) is proposed, which can exploit separable structures and deal with coordinate system rotations.
- A novel cooperation mechanism called Inter-Fireworks Competitive Cooperation (IFCC) is proposed, which can dynamically adapt both the hyperparameters and search distribution parameters effectively.
- Application to reinforcement learning problems is investigated, which provides potential value to real-world machine learning problems.

2. Background and Related Works

2.1. Overview of Large-scale Optimization Algorithms

Generally, current state-of-the-art large-scale algorithms mainly belong to five categories: cooperative co-evolution (CC) algorithms, memetic algorithms, swarm intelligence algorithms, covariance model-based algorithms and zeroth-order optimization methods.

The goal of cooperative co-evolution (CC) algorithms is to tackle LSGO problems through a divide-and-conquer approach. The process of CC algorithms can be reduced to problem decomposition, subcomponent optimization, and cooperative combination. The core challenge of these algorithms lies in determining how to decompose problems and group variables. There have been several notable works in this field, including static variable grouping algorithms such as CCGA [8], random variable grouping algorithms like DECC [12], and interaction-based algorithms like LINC [35], CCVIL [36], DG2 [37], RDG3 [9], DECC-MDG [38], and CCFR-ERDG-CMA-ES [39], and DGSC2 [40], and fuzzy decomposition methods such as DGCS2 [40].

For memetic algorithms, the core idea is to combine higher-level population-based evolutionary procedures with lower-level individual local search procedures. This
two-stage design allows great flexibility. Representative works include DMS-PSO [41], MA-SW-Chains [21], MOS [42], MPS [43], MLSHADE-SPA [44], BICCA [45], and SHADEILS [22]. Recent LSGO competitions have proven that memetic algorithms generally perform better among different categories due to their hybrid nature.

Swarm Intelligence algorithms are also viable choices to solve large-scale optimization problems due to their ability to solve complex and challenging optimization problems effectively and efficiently. The SDLSSO [46] algorithm is a stochastic dominant learning swarm optimizer that balances swarm diversity and convergence speed while consuming minimal computing resources. Another example is the TPSLO-L [47] algorithm, which is a two-phase learning-based swarm optimizer that combines mass learning and elite learning to achieve better performance on large-scale problems. The CO algorithm [48], on the other hand, is inspired by the hunting strategies of cheetahs and adopts a nature-inspired approach to optimize large-scale problems. These algorithms are examples of the efforts made in the direction of leveraging SI for optimizing large-scale problems and demonstrate the potential of SI-based approaches in solving complex optimization problems.

Another important line of work is to maintain and evolve an efficient covariance model. The covariance matrix adaptation evolution strategy (CMA-ES) [25] [49] has achieved remarkable success on low-dimensional problems by learning a quadratic covariance model that can approximate the local landscape of the objective function. However, the computational complexity brought by the quadratic covariance model is prohibitive for large-scale optimization. To address this problem while preserving the benefits of the covariance model, many efforts have been made. The most straightforward way to restrict the covariance matrix is to approximate the full covariance matrix with a diagonal separable model. This formulation leads to the separable CMA-ES (sep-CMA) [23], which lowers the computational complexity to $O(n)$. Then, the efforts by [29] extended the covariance structure to a diagonal plus rank-one formulation and proved to be able to perform better than sep-CMA. Later, in [24] and [50], inspired by the famous L-BFGS [51], the limited-memory covariance matrix adaptation (LM-CMA) and the limited-memory matrix adaptation (LM-MA) were proposed to store search directions to reproduce the Cholesky factor or the transformation matrix. Formulating the covariance model as a low-rank model, which is composed of an isotropic diagonal component and a linear combination of several rank-one models, also gained significant momentum over the years. MVA-ES [52] maintained a main mutation vector as the component of a rank-one model and captured the principle direction of the covariance matrix. Later, in [53], the Rank One NES (R1-NES) was proposed with a rank one low-rank model on top of the natural evolution strategies. Run-ES was [27] later extended to a model of $m$ rank in a simple ES framework. Fast CMA-ES [54] utilizes a mixture of low-rank models while keeping the sampling complexity low. SDA-ES [55] tried to capture more eigenvector directions in the low-rank model by a heuristic similar to the PCA procedure. MM-ES [28] utilizes a mixture low-rank model to approximate the full covariance matrix with historical evolution paths.

Different from the previous four categories, zeroth-order optimization algorithms resort to estimating the gradient by sampling in a local neighborhood and then plug this estimated gradient into an established first-order optimization algorithm such as Adam [56] [57] [58] [59]. These algorithms are typically utilized on machine learning problems and tend to enjoy better theoretical guarantees compared to evolutionary methods. However, as these algorithms usually employ simple and locally-concentrated sampling distributions, they cannot deal with complex black-box optimization problems with properties such as multi-modality and ill-conditioning as effectively as evolutionary methods.

2.2. Fireworks Algorithm Framework

Fireworks Algorithm (FWA) was first proposed in [30] as a swarm intelligence algorithm that mimics the real-world firework explosion process. FWA features two important components that can induce complex search behaviors: the explosion operation and the cooperation mechanism. On a high level, the firework in FWA is a local population whose search behavior is defined by the explosion operation. Then the cooperation mechanism defines how the fireworks should interact and how the information should be exchanged between local populations.

Seemingly simple explosion operations and cooperation mechanisms can result in rich search behaviors. For example, in [31], fireworks conduct local searches with uniform distributions and collaborate with other fireworks with a simple linear estimation-based restart signal, resulting in the LoT-FWA, which is very competitive in terms of multi-modal problems. In [33], a multi-scale fireworks algorithm (MC-FWA) was proposed to utilize two groups of fireworks on the global and local scales to capture the global landscape while coping with local variations. In [60], multiple fireworks collaboratively partition the search space and conduct two levels of optimization, which significantly increases the sample efficiency and can easily address problems with multi-scale patterns. In the case where first-order gradient information is available, the core idea behind FWA can be extended into a new learning framework called Fireworks Swarm Learning (FSL).

3. Proposed Algorithm

The goal of the considered black-box optimization problem is simple and straightforward, a function or oracle $f$ with only zeroth-order function evaluation is to be minimized

$$\min_{x \in \mathbb{R}^n} f(x) \quad (1)$$
For large-scale problems, the number of dimensions \( n \) is considered greater or equal to 1000 [61].

Figure 1 offers a clear visual representation of the SF-FWA pipeline. The process begins with each fireworks generating samples efficiently using the Expressive Fast Explosion (EFE) method. Afterward, the fireworks participate in Inter-Fireworks Competitive Cooperation (IFCC) to continuously adapt their search distributions and hyperparameters based on the problem structure in real-time.

The number of generations is denoted by \( g \) and starts from 0. The number of fireworks is denoted by \( k \). The population size of each firework is denoted by \( \lambda^{\text{loc}} \), and the corresponding elite population size is \( \mu^{\text{loc}} \). The “loc” in the superscript means “local”. The population size of the whole population is denoted by \( \lambda^{\text{slo}} \), and the corresponding elite population size is \( \mu^{\text{slo}} \). The “slo” in the superscript means “global”. The variance vector is denoted by \( \mathbf{v} \) and the search direction archive is denoted by \( \mathcal{A} \). The population mean is denoted by \( \bar{x} \).

### 3.1. Expressive Fast Explosion (EFE)

In EFE, the multivariate Gaussian distribution is chosen as the search distribution. To cope with large-scale problems, these Gaussian distributions are not parameterized by the full covariance matrix with \( O(n^2) \) complexity. Instead, a diagonal plus low-rank covariance model formulation is utilized to circumvent the quadratic complexity while staying expressive:

\[
\mathbf{C}_{\text{EFE}} = (1 - \phi) \left[ (1 - \gamma) \text{diag}(\mathbf{v}) + \gamma \mathbf{I} \right] + \phi \sum_{i=1}^{m} p_i \mathbf{p}_i^T \tag{2}
\]

In the above equation, \( \phi \in [0, 1] \) is a hyperparameter balancing the contribution of the diagonal component and the low-rank component. Meanwhile, \( \gamma \in [0, 1] \) is a hyperparameter balancing the contribution of the variance component and the unit isotropic component. The notations \( \mathbf{v}, \mathbf{I}, m, \) and \( \mathbf{p} \) represent the variance vector, the diagonal identity matrix, the number of search directions, and a search direction vector, respectively.

EFE’s formulation simply tries to model the covariance structure with a linear combination of the diagonal component which can efficiently express separable structure and the low-rank component which can adapt to rotated coordinate systems.

### 3.2. Inter-Fireworks Competitive Cooperation (IFCC)

The adaptation processes of fireworks distributions and hyperparameters are encapsulated by IFCC. The cooperative aspect manifests itself in the sense that fireworks share their population mean \( \bar{x} \), step size \( \sigma \), variance component \( \mathbf{v} \) and the search direction archive \( \mathcal{A} \). The competitive aspect is shown during the hyperparameter optimization where fireworks compete against each other and hyperparameter distributions are evolved as the result of this competition.

#### 3.2.1. Mean Adaptation

The globally shared mean vector in generation \( g \) for all fireworks is updated with 4, which is essentially a weighted selection in the global population pool. The solutions are sorted according to their fitness values so that \( f(x_1^g) \leq f(x_2^g) \leq \ldots \leq f(x_n^g) \) and weights \( \omega \) are calculated according to [49] and satisfy \( \sum_{i=1}^{n} w_i = 1 \), \( w_1 \geq w_2 \geq \ldots \geq w_{\mu^{\text{slo}}} > 0 \).

\[
\mathbf{x}_i^{g+1} = \sum_{i=1}^{\mu^{\text{slo}}} \omega_i \mathbf{x}_i^g \tag{4}
\]

#### 3.2.2. Diagonal Component Adaptation

The variance explosion component in EFE is updated to estimate the variance of successful sampling steps. When only the variances need to be considered, no matrix multiplication is needed. Denoting the learning rate for
Unbiased Exploration
Self-correlation Exploitation
Search Directions Exploitation

Mix
Isotropic \ldots \text{aka fg}
best <\text{fg}^{-1}\text{best}>, a constant \(c_2 > 1\) will be multiplied: \(\sigma_{g+1} = c_2\sigma_g\). The rationale behind this mechanism

\begin{equation}
\sigma_{g+1} = (1 - c_2)\sigma_g + c_2 \sum_{i=1}^{\mu_{\text{eff}}} \omega_i \left(\frac{x_i^g - \hat{x}^g}{\sigma_g}\right)^2
\end{equation}

3.2.3. Guided Component Adaptation

For search directions in the guided explosion component of EFE, we adopted the evolution path mechanism\cite{25} \cite{49} \cite{24} \cite{27}. Denoting the learning rate for evolution path accumulation as \(c_v\), the evolution path \(p_{v,\text{eff}}^{g+1}\) for the next generation is calculated as follows:

\begin{equation}
p_{v,\text{eff}}^{g+1} = (1 - c_v)\bar{p}_v + c_v \left(2 - c_v\right)\mu_{\text{eff}} \left(\frac{x_{\text{best}}^{g+1} - \hat{x}^g}{\sigma_{g+1}}\right)
\end{equation}

where \(\mu_{\text{eff}} = \sum_{i=1}^{\mu_{\text{eff}}} \frac{1}{\sigma_i}\) is a normalization coefficient as suggested by\cite{49} to satisfy the stationary condition which assures both \(p_{v}^{g+1}\) and \(\bar{p}_v\) follow the same Gaussian distribution under random selection. \(x_{\text{best}}^{g+1}\) is the estimated population mean in the next generation. Although the sampling distribution in SF-FWA is no longer strictly Gaussian, since the discrepancy between firework distributions is not that large, this evolution path update should still be relatively stable, which was also confirmed in later experimental studies.

For the guided explosion component to be expressive, an archive of \(m\) search directions is considered helpful\cite{27} \cite{28} \cite{24}. To build this archive \(A\), consider an array of principle search directions \(A = [p_1, p_2, \ldots, p_m]\), and their corresponding generation timestamps are recorded as \(T = [t_1, t_2, \ldots, t_m]\). In each generation, the generation gaps in \(A\) are calculated, and then if the minimal generation gap is larger than a predefined threshold, the corresponding search direction will be removed from \(A\). If the minimal gap is still larger than the threshold, the oldest direction will be removed. The current \(\bar{p}_v\) will always be injected into \(A\). This process originated from the observation that search directions with small generation gaps also tend to be similar. Imposing a generation gap restriction can diversify the archive and potentially capture more eigenvectors of the local Hessian matrix\cite{62} \cite{27}. For easier manipulation, another array of indices \(Z = [i_1, i_2, \ldots, i_m]\) is introduced. The following Algorithm 1 summarizes this procedure in detail.

3.2.4. History-based Dynamic Explosion Amplitude

The classic dynamic explosion amplitude (DEA) method from FWA\cite{63} \cite{31} \cite{64} serves as a competitive yet extremely simple solution. In DEA, the best solution in the current generation \(f_{\text{best}}^g\) is compared to the best solution \(f_{\text{best}}^{g-1}\) from the previous generation. And if \(f_{\text{best}}^g \geq f_{\text{best}}^{g-1}\), which means no significant progress has been made, a constant \(c_1 < 1\) will be multiplied on \(\sigma^g\): \(\sigma^{g+1} = c_1\sigma^g\). In contrast, if significant progress has been detected, aka \(f_{\text{best}}^g < f_{\text{best}}^{g-1}\), a constant \(c_2 > 1\) will be multiplied: \(\sigma^{g+1} = c_2\sigma^g\). The rationale behind this mechanism

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{An illustration of the SF-FWA framework. (a): Each firework generates its samples with the Expressive Fast Explosion (EFE) in a computationally efficient manner. (b): Fireworks undergo Inter-Fireworks Competitive Cooperation (IFCC) to adapt their search distributions along with hyperparameters to adjust their behaviors according to problem structures in an online fashion.}
\end{figure}
Algorithm 1: Update search directions archive

Input: $A, T, I, T_{min}, p^{g+1}, g, m$
Output: Updated $A, T, I$

1. $G = [t_{0} - t_{1}, t_{0} - t_{2}, ..., t_{m} - t_{m-1}]$
2. $i^* = \arg \min(G)$
3. if $G[i^*] > T_{min}$ then
4. \hspace{1em} $i^* = 1$
5. else
6. \hspace{1em} $i^* = i^* + 1$
7. $\pi_i = I[i^*]$
8. for $j = i^*, ..., m - 1$ do
9. \hspace{1em} $k = j + 1$
10. \hspace{1em} $I[m] = k$
11. $T[m] = i^*$
12. $T[i^*] = g + 1$
13. $A[i^*] = p^{g+1}$

is to encourage the step size to bounce around the state where significant progress can just be made. However, this mechanism requires a relatively large population size to be stable as it greedily compares the best solutions. Another potentially limiting property is that DEA is myopic in the sense it only cares about consecutive progress.

In this paper, a new variant of DEA called History-based Dynamic Explosion Amplitude (HDEA) is proposed to overcome the aforementioned problems. The basic idea of HDEA is to measure the quality of the current generation samples by the advantage they can bring to the historical best solutions. A sorted archive $H^g$ at generation $g$ is constructed for storing the historical best fitness values: $H^g = [b_1^g, ..., b_m^g]$, where $b_i^g$ stands for the $i$th best historical solution up until generation $g$. In generation $g$, the fitness values of the population are merged with $H^g$, and $H^{g+1} = [b_1^{g+1}, ..., b_m^{g+1}]$ is updated by sorting and selecting the best $\mu b$ solutions in the merged selection pool.

Since both $H^g$ and $H^{g+1}$ are sorted, a scalar measure $Q$ of the solution quality can be constructed as an indicator of the improvement brought by the new samples:

$$Q = \sum_{i=1}^{\mu b} \omega_i (H^{g+1}[i] < H^g[i])$$

(7)

In the above equation, $1$ is the indicator that evaluates to 1 when the condition inside the parenthesis holds. In the best case where new samples completely dominate historical samples, $Q$ will evaluate to 1. In the worst case where new samples cannot enter the archive, $Q$ will evaluate to 0. The weighting scheme makes better solutions have a larger impact. This new performance measure is no longer greedy, so it is much more stable. At the same time, this measure takes cross-generation performance into account, which can potentially induce more long-term improvement.

The following Algorithm 2 summarizes the detailed process of HDEA:

Algorithm 2: History-based Dynamic Explosion Amplitude (HDEA)

Input: $\sigma^g, H^g, \{f(x_1^g), ..., f(x_m^g)\}, s^g, Q^g, c_1, c_2$
Output: $\sigma^{g+1}, H^{g+1}$

1. Construct merged selection pool:
   \hspace{1em} $S = H^g \cup \{f(x_1^g), ..., f(x_m^g)\}$
2. $S = \text{sorted}(S)$, so that $S[1] \leq ... \leq S[\lambda^g]$
3. $H^{g+1} = S[1 : \mu b^\phi g^\phi]$
4. $Q = \sum_{i=1}^{\mu b^\phi} \omega_i (H^{g+1}[i] < H^g[i])$
5. $s^{g+1} = (1 - c_\phi)s^g + c_\phi(Q - Q^g)$
6. if $s^{g+1} < 0$ then
   \hspace{1em} $\sigma^{g+1} = c_1\sigma^g$
   \hspace{1em} else
   \hspace{2em} $\sigma^{g+1} = c_2\sigma^g$

3.2.5. Hyperparameter Self-adaptation by Competition

Dynamically adapting the hyperparameters in an online fashion has proven to be very successful in differential evolution-based methods [65] [66]. The core idea underlying these methods essentially resembles a meta-learning procedure [67] [68] [69], where an outer-level optimization procedure inspects the performance difference from the inner optimization process. IFCC exploits this observation and directly takes advantage of multiple firework instances to adapt those hyperparameters that affect the structures of the covariance model.

Specifically, the balancing hyperparameters $\phi$ and $\gamma$ of the diagonal plus low-rank covariance model described in 2 are considered adaptive. For $\phi$, since it determines the ratio of the low-rank model, which cannot be too large or too small, we choose to conduct its adaptation on the logarithmic domain.

$$\begin{aligned}
\psi &= \log_{10} \phi \\
\phi &= 10^\psi
\end{aligned}$$

(8)

For $\gamma$, it can safely take values in $[0, 1]$, and it should be able to approach 0 or 1 with high precision. We use the following sigmoid-like function as a transformation between the $\gamma$ used for sampling and the $\zeta$:

$$\begin{aligned}
\zeta &= -\log_{10}(1 - \frac{\gamma}{C}) \\
\gamma &= \frac{\zeta + 10}{C + 10}
\end{aligned}$$

(9)

$\psi$ and $\gamma$ are stochastically sampled from two separate univariate Gaussian distributions in every iteration. These two Gaussian distributions’ means $\mu_\psi, m_\gamma$ are adaptive, and the standard deviations $\sigma_\psi, \sigma_\gamma$ are predefined. They are denoted as $N_\psi(\mu_\psi, \sigma_\psi), N_\gamma(\mu_\gamma, \sigma_\gamma)$.

In generation $g$, each firework first randomly samples its own $\psi$ and $\zeta$ according to $N_\psi$ and $N_\gamma$. Note that
the number of fireworks is usually very small, for stable adaptation, it is preferable to mirror conduct sampling for these hyperparameters. Once $\psi$ and $\zeta$ are sampled, they are then transformed into $\phi$ and $\gamma$ according to 8 and 9. Then for firework $j$, it builds a covariance model with the globally shared mean, step size, variance vector, direction archive and the locally different $\phi_j$ and $\gamma_j$. As different combinations of $\phi_j$ and $\gamma_j$ directly result in structurally different search distributions, the sample quality of these fireworks will differ. Certain structures are more likely to better reflect the local landscapes of the function, which means samples from these models will exhibit a greater competitive advantage. Then these advantages can then serve as the optimization signal for the hyperparameter distributions.

Considering that the population of solutions is evaluated, their fitness values are first arranged into the follow-

```matlab
Algorithm 3: SF-FW A
Input: $f, \hat{x}^0, \sigma^0$
1. $g = 0$
2. $s = 0, H = \emptyset$
3. $A = [p_1, ..., p_m] = [0, ..., 0]$
4. $T = [1, ..., m], T = [0, ..., 0]$
5. $p_0 = 0, v_0 = 0$
6. $\mu_0 = \mu_0^{\text{init}}, \mu_0^0 = \mu_0^{\text{init}}$
7. while termination condition not met do
8.     $\psi_1, ..., \psi_k \sim \mathcal{N}(\mu_0^g, \sigma^g)$
9.     $\zeta_1, ..., \zeta_k \sim \mathcal{N}(\mu_0^g, \sigma^g)$
10. for $i = 1, ..., k$ do
11.     $\phi^g_i = 10^{\psi_i^g}$
12.     $\gamma^g_i = \frac{1}{1 + 10^{\psi_i^g}}$
13.     for $j = 1, ..., \lambda^{\text{loc}}$ do
14.         $z_1, z_2 \sim \mathcal{N}(0, 1)$
15.         $r_1, ..., r_m \sim \mathcal{N}(0, 1)$
16.         $x_j^{g,i} = \hat{x}_j^i + \sigma_i (\sqrt{1 - \phi^g_i} \sqrt{(1 - \gamma^g_i)^{\frac{1}{2}}}, \sqrt{\gamma^g_i} z_1) + \sqrt{c_i} \sum_{k=1}^{m} r_k p_k$
17.     Sort the whole global solutions such that $f_i(x_j^{g,i}) \leq ... \leq f_i(x_j^{g,i})$
18.     $\hat{x}^{g+1} = \sum_{i=1}^{\mu^{\text{loc}}} \omega_i x_j^{g,i}$
19.     $v^{g+1} = (1 - c_i) v^g + c_i \sum_{i=1}^{\mu^{\text{loc}}} \omega_i (x_j^{g,i} - \hat{x}^{g+1})^2$
20.     $p^{g+1} = (1 - c_i) p^g + c_i (2 - c_i) \mu^{\text{loc}} (x_j^{g+1} - \hat{x}^{g+1})$
21. Update $\sigma^{g+1}, H^{g+1}$ according to Algorithm 2
22. Update $A, I, T$ according to Algorithm 1
23. Gather solutions from different fireworks into $F$ according to 10
24. Conduct sorting and selection on $F$ into $F^g$ according to 11
25. $B = \arg \min(F^g, \text{axis}=0)$
26. $\mu^{\text{loc}} = (1 - c_i) \mu^g + c_i \sum_{i=1}^{\mu^{\text{loc}}} \omega_i \psi_i^g$
27. $\mu^{\text{loc}} = (1 - c_i) \mu^g + c_i \sum_{i=1}^{\mu^{\text{loc}}} \omega_i \psi_i^g$
28. $g = g + 1$
```

(10)
$F^*$, which holds the elite solutions within each fireworks:

$$F^* = \begin{bmatrix} f^1_1 & f^1_2 & \cdots & f^1_{\mu_{max}} \\ \vdots & \vdots & \ddots & \vdots \\ f^k_1 & f^k_2 & \cdots & f^k_{\mu_{max}} \end{bmatrix}$$

(11)

With this array, the performance of each fireworks can be directly compared. An arg min operation is performed along the column axis to extract the index of the best-performing fireworks on each ranked column, which results in a one-dimensional vector holding displays the best indices:

$$B = \arg \min(F^*, \text{axis}=0) = [j_1, j_2, \ldots, j_{\mu_{max}}]$$

(12)

where $j_k \in \{1, 2, \ldots, k\}$. With $B$, it is then possible to weigh between different values of $\psi$ and $\zeta$. Denote the array holding sampled values for $\psi$ and $g$ as $\Psi$, and the array holding sampled $\zeta$s as $Z^g$:

$$\Psi^g = [\psi^g_1, \ldots, \psi^g_k]$$

$$Z^g = [\zeta^g_1, \ldots, \zeta^g_k]$$

(13)

Then the corresponding Gaussian means $\mu^{g+1}_{\psi}$ and $\mu^{g+1}_{\zeta}$ are updated in this manner:

$$\mu^{g+1}_{\psi} = (1 - c_a) \mu^g_{\psi} + c_a \sum_{i=1}^{\lambda_{loc}} \omega^g_i [\Psi^g[B[i]]]$$

$$\mu^{g+1}_{\zeta} = (1 - c_a) \mu^g_{\zeta} + c_a \sum_{i=1}^{\lambda_{loc}} \omega^g_i [Z^g[B[i]]]$$

(14)

where $c_a$ is the learning rate for hyperparameter adaptation.

3.3. Detailed Implementation of SF-FWA

The final implementation of SF-FWA is summarized by Algorithm 3.1. In the beginning, a random mean vector $\mathbf{X}^0$ and an initial step size $\sigma^0$ are provided. The dynamic parameters are initialized. After that, a set of $\psi$ and $\zeta$ are randomly sampled for fireworks. Then for each fireworks, its $\phi$ and $\gamma$ are obtained by the predefined transformation functions in 8 and 9. Next, each fireworks samples its solutions according to the sampling procedure described in 3.1. After all solutions are evaluated, the globally shared mean, variance component, evolution path, search direction, and search direction archive are updated accordingly. Finally, the performance of different fireworks is gathered and compared, and the means of the hyperparameters’ distributions are updated with the procedures described in 3.2.5. These steps are repeated until the termination condition is met.

3.3.1. Complexity

The space complexity of SF-FWA mainly comes from maintaining the search direction archive $A$ with $m$ search directions and storing $\lambda_{loc}$ solutions. The overall space complexity is $O(n \cdot m)$. For the time complexity, all sampling and adaptation operations can be performed in $O(n)$.

4. Experimental Studies

In this section, the behavior and performance of SF-FWA were analyzed through experimentation. Firstly, the two benchmark datasets utilized in the experiments are introduced. Subsequently, the hyperparameter configuration of SF-FWA is studied. Afterward, comprehensive comparisons of SF-FWA on basic functions and standard benchmarks are conducted. Lastly, the application of SF-FWA in reinforcement learning is investigated.

4.1. Benchmark Settings

4.1.1. Basic Functions

To study the behavior of SF-FWA, a set of 5 commonly used basic functions along with their rotated version are utilized. These functions are the bent cigar function $f_{\text{BentCigar}}$, the discus function $f_{\text{Discus}}$, the ellipsoid function $f_{\text{Ellipsoid}}$, the different powers function $f_{\text{DiffPow}}$, and the Rosenbrock’s function $f_{\text{Rosen}}$. The unrotated version of these functions is either separable or the linkage between variables is relatively weak, which makes them Type-S problems. Their rotated version is constructed by applying a random normalized rotation matrix to the solution vector: $f_{\text{rotated}}(X) = f_{\text{unrotated}}(RX)$, which makes them Type-R problems. Although generally simple and easy to solve, this set of basic functions is commonly used across literature [24] [50] [55] [28] [27] [54] [70] [71] since they well-represent different function landscapes. For example, $f_{\text{BentCigar}}$ and $f_{\text{Rosen}}$ have landscapes that can be well-approximated by a few search directions, and $f_{\text{Ellipsoid}}$ requires a set of widely distributed eigenvalues to effectively capture the local structure. The detailed definition of these basic functions is shown in Table 1.

4.1.2. Standard Large-scale Benchmark

To further test the performance of our method on more complex and universal problems, the standard large-scale optimization benchmark CEC’2013 LSGO [61] suite was utilized. The number of decision variables for this benchmark suite is 1000. A brief description of the properties of the test functions is depicted in Table 2. The benchmark consists of four types of large-scale problems: fully-separable functions, partially separable functions, overlapping functions and fully- nonseparable functions. With the introduction of ill-conditioning, irregularities and symmetry breaking, CEC’2013 LSGO benchmark models real-world large-scale problems quite well.
4.2. Static Hyperparameter Settings for SF-FWA

In SF-FWA, hyperparameters can be grouped into dynamic hyperparameters and static ones. As dynamic hyperparameters are adapted online, this section describes the static hyperparameters used in SF-FWA.

To identify the optimal static hyperparameters for SF-FWA, we conducted comparative experiments using the CEC’2013 LSGO benchmark suite. The hyperparameters studied include the number of fireworks $m$, the dynamic adjustment rate $c$, and the standard deviations $\sigma_{\xi}$ and $\sigma_{\zeta}$. A range of values for each hyperparameter was tested, and for each setting, the standard deviations $\sigma_{\xi}$ and $\sigma_{\zeta}$ were adapted online, this section describes the static hyperparameters used in SF-FWA.

The results of the static hyperparameter comparisons are depicted in Figure 2. Each row represents the results for a specific hyperparameter setting, with the first column indicating the number of wins, draws, and losses compared to other settings. The second column provides a detailed visualization of the number of wins for each setting, with the darkness of the squares indicating the number of wins (darker squares indicating better performance). To determine the optimal hyperparameter value, the number of significant wins was selected as the metric (represented by the green bars in the first column). As can be observed from Figure 2(a), the setting with $k = 4$ achieved the largest number of significant wins compared to other settings. For the dynamic adjustment rate $c_1$, a value greater than 0.90 generally performed well, as shown in Figure 2(b), with $c_1 = 0.93$ delivering the best overall result. The performance of the HDEA target threshold $q^*$ was found to be sensitive to its value, with values of 0.6 and 0.7 generally performing well, as shown in Figure 2(c). The rate of hyperparameter adaptation was found to be crucial for good performance, with the best result achieved for $c_a = 0.1$, as shown in Figure 2(d). The results for the standard deviations $\sigma_\xi$ and $\sigma_\zeta$ showed a limited differences between different settings, with 0.1 being the optimal setting for $\sigma_\xi$ and 0.3 and 0.2 performing similarly well for $\sigma_\zeta$, as shown in Figure 2(e) and Figure 2(f). Given that $\sigma_\zeta = 0.3$ resulted in only 11 significant losses (compared to 17 losses for $\sigma_\zeta = 0.2$), $\sigma_\zeta = 0.3$ was chosen as the optimal value.

For the local population size, since SF-FWA utilizes multiple firework instances, the population size for each firework distribution $\lambda^{\text{firework}}$ is set to grow logarithmically as the problem dimension grows, which is widely used in other covariance model-based algorithms [27, 28, 49, 55]. For the low-rank model, the number of search directions $m$ is usually determined as an important hyperparameter as it directly determines the expressiveness of the low-rank model. Typically, out of the computational complexity concern, this number is not set too large. In [27], $m$ is set to 2 for RmES, in MMES [28], a similar parameter called the mixing strength is set to 4. In SF-FWA, as our vectorized imple-
Figure 2: This figure shows the comparison of SF-PWA’s important static hyperparameters. Each row shows the result for a static hyperparameter. The first column shows the win/draw/lose numbers of a specific setting compared to other settings. The second column shows the detailed number of wins of each setting for a specific function. The darkness of the square represents the number of wins. The darker the better.

(a)

(b)

(c)

(d)

(e)

(f)
mentation is sufficiently efficient, \( m \) is set to 10 to better capture the low-rank structure. The generation gap for archive update \( T_{\text{min}} \) is set to be the number of dimensions \( n \), which is the recommended setting suggested by [27]. The \( \mu_{\text{null}} \) and \( \mu_{\text{mix}} \) are set to \(-2.0\) and \(0.0\) respectively to make the initial EFE distributions unbiased.

\[
\lambda^{\text{loc}} = \left(\frac{4 + 3 \ln n}{\Delta} \right), \quad \mu^{\text{loc}} = \left[ \lambda^{\text{loc}} \right],
\]

\[
\lambda^g = k \lambda^{\text{loc}}, \quad \mu^{g} = \left[ \lambda^{g} \right],
\]

\[
w_i = \ln \left( \frac{\lambda^g}{\lambda^{g}-1} \right), \quad i = 1, \ldots, \mu
\]

\[
k = 4, \quad m = 10, \quad Q^* = 0.7
\]

\[
c_1 = 0.93, \quad c_2 = \frac{1}{c_1}, \quad c_3 = 0.3
\]

\[
T_{\text{min}} = n, \quad c_v = \frac{10}{c_3 + 1}, \quad c_v = \frac{10}{c_3 + 1}
\]

\[
c_a = 0.01, \quad \mu_{\text{null}} = -2.0, \quad \mu_{\text{mix}} = 0.0, \quad \sigma_v = 0.1, \quad \sigma_\lambda = 0.3
\]

Table 3: Static hyperparameters for SF-FWA

4.3. Experiments on Basic Functions

For comparison on those basic functions, four representative large-scale algorithms based on covariance model adaptation were selected and carefully implemented under the same environment. These algorithms are the separable CMA-ES algorithm (sep-CMA) [23] which restricts the covariance model to the diagonal form, the limited-memory matrix adaptation algorithm (LM-MA) [50] which directly scales up the MA-ES [72] by utilizing the sequential product of a limited number of supporting vectors, the rank-m evolution strategies (Rm-ES) [27] and the mixture model-based evolution strategies (MMES) [28] which are two state-of-the-art low-rank model-based large-scale ES variants. These algorithms are implemented by strictly following their optimal hyperparameter settings as reported in their original paper.

4.3.1. Effectiveness on Both Type-S and Type-R Problems

We conducted an experiment where SF-FWA, sep-CMA, LM-MA, Rm-ES, and MMES were tested on the 1000-dimensional basic functions as described in Table 1. For robust evaluation, each algorithm was run repeatedly 10 times, and the median results were plotted. Their optimization process is recorded in Figure 3, where each algorithm was allowed to consume at most \(10^8\) function evaluations at most and the problem was considered solved when the precision threshold of \(1 \times 10^{-8}\) was reached and the algorithm was stopped.

As can be seen from Figure 3, on Type-S problems, SF-FWA and sep-CMA are generally the two strongest algorithms, which indicates they can well exploit the separable structure. This kind of separability exploitation is especially helpful on problems such as the ellipsoid function where a wide eigenvalue spectrum is shown. Rotation invariant methods are essentially not capable of capturing the variance as effectively as SF-FWA and sep-CMA.

4.3.2. Effectiveness of Hyperparameter Self-adaptation

Although the search distribution defined in EFE is naturally expressive, the hyperparameter configuration can greatly affect the performance of SF-FWA on different problems. The proposed self-adaptive mechanism tackles this problem by optimizing the hyperparameters through performance feedback in an online fashion. In this sec-

\[
\begin{array}{cccccc}
\text{Algorithms} & \text{LM-MA} & \text{MMES} & \text{Rm-ES} & \text{Sep-CMA} & \text{SF-FWA} \\
\hline
f_{\text{RotCigar}} & 5^\text{c} & 3^\text{c} & 2^\text{c} & 4^\text{c} & 1^\text{c} \\
f_{\text{RotDiffPowers}} & 5^\text{c} & 3^\text{c} & 4^\text{c} & 2^\text{c} & 1^\text{c} \\
f_{\text{Discus}} & 5^\text{c} & 3^\text{c} & 4^\text{c} & 1^\text{c} & 2^\text{c} \\
f_{\text{Ellipsoid}} & 5^\text{c} & 3^\text{c} & 4^\text{c} & 2^\text{c} & 1^\text{c} \\
f_{\text{Rosen}} & 1^\text{c} & 3^\text{c} & 2^\text{c} & 5^\text{c} & 4^\text{c} \\
\end{array}
\]

Table 4: The number of function evaluations to reach the precision threshold \(1 \times 10^{-8}\) is recorded and ranked. A smaller rank indicates a faster convergence rate. The Wilcoxon rank-sum test is performed to test whether the mean performance of SF-FWA is significantly better than comparing algorithms (p-value threshold is 0.05). \(\oplus\) indicates that SF-FWA is significantly better, while \(\ominus\) indicates the opposite scenario. \(\odot\) indicates no significant difference detected. The average rank is computed by averaging across rows for each algorithm, and is a reflection of relative advantage within an algorithm portfolio.

On the other hand, it is also obvious that on Type-R problems, sep-CMA’s performance dramatically deteriorated and could only barely solve the rotated Discus function. This is because the variance component simply cannot model the covariance between rotated variables. SF-FWA along with two low-rank models Rm-ES and MMES generally showed similar performance across problems, which indicates that SF-FWA can utilize the low-rank model equally well as typical low-rank methods.

The final performance ranking is reported in Table 4, where the function evaluations required to reach the precision threshold of \(1 \times 10^{-8}\) are compared and ranked among five algorithms. A smaller rank indicates a better result. The average ranking (AR) is calculated by averaging the ranks of a specific algorithm under different problems. SF-FWA achieved the best result for Type-S problems with an AR of 1.80 as well as the best result for overall problems with an AR of 2.20.

The above results suggest that SF-FWA can effectively tackle both Type-S and Type-R problems.
Figure 3: Optimization process of SF-FWA, LM-MA, sep-CMA, Rm-ES and MMES on the basic functions. The top row shows results on Type-S problems where SF-FWA and sep-CMA tend to perform better. The bottom row shows results on Type-R problems where sep-CMA significantly performs worse but SF-FWA (blue) manages to keep up with other methods.

Figure 4: The optimization process of $\gamma$ on the basic test functions.

In Figure 4, the Type-S problems are colored red colors and Type-R problems are colored blue. The $\gamma$-axis indicates the sigmoid input value of the $\gamma$ hyperparameter. The higher this value is, the larger portion the unit Gaussian component accounts for. In fact, this unit Gaussian component is crucial for effective exploration when the coordinate system is rotated. Without this term, the variance component and the low-rank component may accumulate bias toward certain directions together and fail to pick up search directions in other dimensions. It can be observed that the evolution curves of $\gamma$ on Type-S problems except for the Rosenbrock’s function tend to go in the negative direction, and on the other hand on Type-R problems, these curves have a tendency to go up. The interesting exception here is the Rosenbrock’s function, which showed similar behavior with other Type-R problems. This phenomenon can be well explained by the fact that the Rosenbrock’s function is not fully separable like other problems, and its parabolic landscape requires effective rotation handling capabilities like other Type-R problems, which indicates that our method does not naively detect whether the coordinate system is rotated.

Potentially, fixing the hyperparameters for the diagonal plus low-rank covariance model may also lead to good performance under both Type-S and Type-R problems. To verify the effectiveness of our self-adaptation mechanism, another experiment was conducted where we predefined a set of fixed combinations of gammas and $\gamma$ and compared their optimization curves against the adaptive version. The $\phi$ hyperparameter took values from $\{10^{-3}, 10^{-2}, 10^{-1}\}$, the $\gamma$ hyperparameter took values from $\{0.0, 0.5, 1.0\}$. This configuration resulted in a total of nine fixed combinations and can well-represent a variety of different fixed combinations. For a fair comparison, the median results of 10 repeated runs of each configuration are recorded and the ranks on each problem were reported in Figure 5. A larger area indicates a better overall performance. As can be seen from Figure 5, for those fixed configurations, they can only perform well either on the unrotated problems or on the rotated problems, the adap-
Figure 5: This radar plot shows the performance of different hyperparameter settings. The adaptive setting corresponds to the proposed self-adaptive hyperparameter evolution method, while other settings are fixed. The radius indicates the relative ranking of a particular setting, the larger the radius, the lower the rank. As the adaptive (red) setting clearly covers more area, it proves that the self-adaptation of $\phi$ and $\gamma$ provides benefits beyond the expressive fast explosion.

Figure 6: An illustration of the sample efficiency scalability of different algorithms on basic functions. The top row shows the scalability of Type-S problems. Generally, SF-FWA (blue) tends to enjoy the best scalability as the dimension grows. The bottom row shows the scalability of Type-R problems. SF-FWA (blue) shows significantly better scalability compared to LM-MA and sep-CMA and stays close to MMES and Rm-ES.

4.3.3. Dimension Scalability

Another crucial aspect of all large-scale algorithms is scalability with regard to dimensions. The term scalability manifests itself in two ways: sample efficiency and runtime efficiency. The sample efficiency is measured by how much function evaluations are consumed to reach a certain preci-
sion threshold as the dimension grows. A good large-scale algorithm should be able to avoid exponential scaling as much as possible. For real-world practical applications, runtime scalability also requires sufficient attention. As our implementation of SF-FWA is capable of utilizing both CPU and GPU, a comparative study was conducted to show how long a single generation of SF-FWA took under different computation device setups.

### 4.3.4. Sample Efficiency

To investigate how the sample efficiency of SF-FWA scales with regard to dimensions, an experiment where the function evaluations to reach the precision threshold of $1 \times 10^{-8}$ of different algorithms were compared. The set of algorithms and basic functions introduced in 4.3 were utilized in this experiment. The tested dimensions were \{100, 200, 400, 800, 1600, 3200, 6400, 12800\}, which should serve as good coverage of dimensions. Since it may take too many function evaluations to reach $1 \times 10^{-8}$ in higher dimensions, a maximum evaluation bar was set to be $10^8$ function evaluations. The median functions evaluations to reach the precision of $1 \times 10^{-8}$ across 10 runs are illustrated in Figure 6.

On Type-S problems (top row in Figure 6), sep-CMA generally showed good performance as expected. It performs the best on low dimensions on the unrotated DiffPowers function, the Discus and the Ellipsoid function. However, as the dimension grows, its performance gaps with Rm-ES and MMES are gradually narrowed and on BentCigar and DiffPowers function, it was even outperformed by low-rank methods. Since it may take too many function evaluations to reach $1 \times 10^{-8}$ in higher dimensions, a maximum evaluation bar was set to be $10^8$ function evaluations. The median functions evaluations to reach the precision of $1 \times 10^{-8}$ across 10 runs are illustrated in Figure 6.

On Type-S problems (top row in Figure 6), sep-CMA generally showed good performance as expected. It performs the best on low dimensions on the unrotated DiffPowers function, the Discus and the Ellipsoid function. However, as the dimension grows, its performance gaps with Rm-ES and MMES are gradually narrowed and on BentCigar and DiffPowers function, it was even outperformed by low-rank methods. Since it may take too many function evaluations to reach $1 \times 10^{-8}$ in higher dimensions, a maximum evaluation bar was set to be $10^8$ function evaluations. The median functions evaluations to reach the precision of $1 \times 10^{-8}$ across 10 runs are illustrated in Figure 6.

On Type-R problems (top row in Figure 6), sep-CMA generally showed good performance as expected. It performs the best on low dimensions on the unrotated DiffPowers function, the Discus and the Ellipsoid function. However, as the dimension grows, its performance gaps with Rm-ES and MMES are gradually narrowed and on BentCigar and DiffPowers function, it was even outperformed by low-rank methods. Since it may take too many function evaluations to reach $1 \times 10^{-8}$ in higher dimensions, a maximum evaluation bar was set to be $10^8$ function evaluations. The median functions evaluations to reach the precision of $1 \times 10^{-8}$ across 10 runs are illustrated in Figure 6.

On Type-R problems (top row in Figure 6), sep-CMA generally showed good performance as expected. It performs the best on low dimensions on the unrotated DiffPowers function, the Discus and the Ellipsoid function. However, as the dimension grows, its performance gaps with Rm-ES and MMES are gradually narrowed and on BentCigar and DiffPowers function, it was even outperformed by low-rank methods. Since it may take too many function evaluations to reach $1 \times 10^{-8}$ in higher dimensions, a maximum evaluation bar was set to be $10^8$ function evaluations. The median functions evaluations to reach the precision of $1 \times 10^{-8}$ across 10 runs are illustrated in Figure 6.

### 4.3.5. Runtime Efficiency

For real-world applications, the wall-clock running time for solving large-scale algorithms is crucial. Since the running time for function evaluations is outside the scope of algorithm design, only the algorithm’s internal running time is recorded. The sphere function was chosen as the target problem for its simplicity.

As SF-FWA was implemented with the modern efficient vectorization computation framework PyTorch, we tried to maximize the computation efficiency with the "vectorization trick" as much as possible. In addition, our Pytorch implementations are capable of utilizing other modern accelerators other than CPUs without code modification.

To investigate how SF-FWA’s internal computation runtime scales with regard to problem dimensions, an experiment was conducted under different computation device settings under increasing dimensions. For the CPU, the AMD-EPYC 7783 platform was used. The internal computation cost was recorded under 1, 4, 16, 32, 64, and 128 cores. For GPU, the NVIDIA RTX 3090 was used. To effectively reflect the dimensional scalability of the runtime, the tested dimensions were set to be \{1, 10, 100, 1000, 10000, 100000\} with each dimension, a total of $1 \times 10^4$ generations were run and the median runtime to generate a single sample was recorded. As can be seen from Figure 7, for dimensions under $1 \times 10^3$, utilizing the CPU is faster but no significant difference between different numbers of CPU cores was observed. In higher dimensions (larger than 1000 dimensions), GPU showed significantly better scalability than CPU and the performance gap widened as the dimension grew. In $1 \times 10^6$ dimensions, utilizing GPU is nearly 100 times faster than utilizing CPU.

### 4.4. Comparative Experiments on the Standard Large-scale Benchmark

For a comprehensive comparative study, we selected 17 state-of-the-art LSGO-capable algorithms from five categories, including cooperative co-evolution algorithms (SGCC [77], CC-RDG3 [9], DGSC2 [74], BICCA [45], DECC-MDG [38], CC-CMA-ES [11], and CCFR-ERDG-CMA-ES [39]), memetic algorithms (SHADEILS [22], MOS-CEC’2013 [75, 78], and MLSHADE-SPA [44]), and covariance model-based algorithms (sep-CMA [23], LM-MA [50], Rm-ES [27], and MMES [28]), as well as two state-of-the-art swarm intelligence-based optimizers (SDLSO [46] and TPSLO-L [47]) and a recent FW A variant for large-scale optimization (EDFEA [76]). All algorithms were tested with 25 repeated runs and a maximum of $3 \times 10^6$ function evaluations, and their median results were reported. The Wilcoxon rank-sum test was applied.
Method Description
SGCC It addresses the problem of grouping decision variables through soft assignments controlled by a probability distribution function.
CC-RDG3 It solves overlapping components using a modified Recursive Differential Grouping method.
DSGC2 A cooperative co-evolution fuzzy decomposition algorithm that groups decision variables based on their interaction degree.
BICCA It uses evolutions in two spaces (pattern space and search space) to adaptively carry out cooperative co-evolution and global search.
DE-CMA-ES A cooperative co-evolution algorithm that scales up the CMA-ES by using a new sampling scheme, two new decomposition strategies and an adaptive scheme.
CCFR-ERDG-CMA-ES An efficient recursive differential grouping method for cooperative co-evolution by exploiting historical information.
SHADE-ILS It combines a Differential Evolution algorithm with different local search methods in an iterative way.
MOS-CE2013 A multiple offspring sampling to combine DE and the multiple trajectory search (MTS).
MLSHADE-SPA A memetic algorithm that combines LSHADE-SPA and multiple trajectory search (MTS).
sep-CMA A CMA-ES variant with diagonal covariance matrix.
LM-MA The limited-memory variant of the Matrix Adaptation (MA) algorithm.
Rm-ES A variant of ES that utilizes a rank-m low-rank covariance model.
MMES An improved low-rank method with a mixture model.
SDL-PSO An adaptive stochastic dominant learning variant of PSO.
TFSO-L A two-phase mass learning and elite learning-based PSO.

Table 5: Short description of the algorithms used in the comparative standard LSGO benchmark experiments.

Figure 7: This figure shows the runtime scalability of SF-FWA. As dimension grows, all settings take a longer time to generate a single sample, but GPU takes significantly less time in higher dimensions. To assess the statistical significance of performance differences, short descriptions of all comparison algorithms are given in Table 5. And the final benchmark results as shown in Table 6.

Within the cooperative co-evolution category, the CC-RDG3 and CCFR-ERDG-CMA-ES algorithms demonstrate relatively strong performance, with average rankings of 6.07 and 5.47, respectively. Generally, CC methods excel in solving partially separable functions, especially F8, F9 and F11, which align well with their design goals of decomposing decision variables into groups.

Regarding the memetic algorithms category, the overall performance of these algorithms is robust, particularly on multi-modal functions (F2, F3, and F12), highlighting their strong exploration ability. It is worth noting that SF-FWA outperforms all memetic algorithms on F7.

Among covariance model-based algorithms, sep-CMA generally performed better than LM-MA and pure low-rank methods, suggesting that simply modeling the variances of the decision variables can serve as a simple yet strong baseline. MMES again outperformed Rm-ES on this benchmark suite, which suggested that a larger search direction archive could improve performance under both basic test functions and more complex functions.

In the swarm intelligence category, the performance of the two algorithms is comparable across all functions, which may indicate the efficacy of the base particle swarm optimization (PSO) algorithm. Nonetheless, SF-FWA outperforms these algorithms on 11 out of the 15 functions, only losing on multi-modal functions (F2, F5, and F9).

As the newest fireworks algorithm variant, SF-FWA outperforms the fireworks algorithm (EDFWA) on every function evaluated.

Overall, SF-FWA has achieved the highest average ranking of 4.40 among the 18 algorithms evaluated. SF-FWA demonstrates superior performance on F1, F5, F7, F12, F13, F14 and F15 and there is no problem where SF-FWA performed particularly badly. This indicated
Table 6: The table shows fitness values of 14 algorithms on the standard benchmark suite CEC’2013 LSGO after three million function evaluations. The best performer on each function is indicated by the bold font. The Wilcoxon rank-sum test is performed to test whether the mean performance of SF-FWA is significantly better than comparing algorithms (p-value threshold is 0.05). ◆ indicates that SF-FWA is significantly better, while ◎ indicates the opposite scenario. ⊖ indicates no significant difference detected. The average rank is computed by averaging across rows for each algorithm and is a reflection of relative advantage within an algorithm portfolio. It can be seen from the table that SF-FWA achieved the best average across a wide range of algorithm categories, indicating the effectiveness of the proposed mechanisms.

that SF-FWA can achieve universally good performance on a wide range of problems.

4.5. Application to Reinforcement Learning

To examine whether SF-FWA has the potential to be applied to real-world problems, we chose the Mujoco continuous control benchmark which is often used in studying reinforcement learning problems as our test suite.

Applying black-box optimization techniques to directly search for high-dimensional neural network controllers to solve reinforcement learning problems has proven to be competitive against more complex Markov Decision Process (MDP) based policy gradient methods, as they may lead to better exploration and easier parallelization [5] [59] [79] [58].

Four representative tasks within the Mujoco suite...
were selected. For each task, a neural network controller (whose number of parameters varies from 17922 to 66961) is utilized to output actions given the states of the environment. The objective is to maximize the episodic reward accumulated along a maximum of 1000 time steps. For the purpose of comparison, we have selected four established gradient-free reinforcement learning algorithms to serve as benchmarks: The Gradient-Less Descent (GLD) algorithm [59], a heuristic-based random search method that is parameter-free; the OpenAI-ES algorithm [5], a streamlined version of Natural Evolution Strategies (NES) that incorporates fixed variances; the Cross-Entropy Method (CEM) [80] [81], which utilizes a variance model to guide its search; the Parameter-Exploring Policy Gradients (PGPE) algorithm [82], which calculates likelihood gradients through direct sampling of the parameter space; and finally, the MMES algorithm [28] that was introduced in previous sections. All algorithms were run on each

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Variables</th>
<th>GLD</th>
<th>OpenAI-ES</th>
<th>CEM</th>
<th>PGPE</th>
<th>MMES</th>
<th>SF-FWA (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant-v4</td>
<td>21128</td>
<td>1710.98</td>
<td>90.95</td>
<td>1111.09</td>
<td>56.88</td>
<td>4322.87</td>
<td>166.19</td>
</tr>
<tr>
<td>HalfCheetah-v4</td>
<td>19590</td>
<td>381.56</td>
<td>916.91</td>
<td>456.50</td>
<td>472.36</td>
<td>4124.47</td>
<td>818.83</td>
</tr>
<tr>
<td>Humanoid-v4</td>
<td>66961</td>
<td>1043.09</td>
<td>147.23</td>
<td>1031.41</td>
<td>152.49</td>
<td>1580.73</td>
<td>151.35</td>
</tr>
<tr>
<td>Swimmer-v4</td>
<td>17922</td>
<td>356.01</td>
<td>5.10</td>
<td>364.53</td>
<td>0.24</td>
<td>364.93</td>
<td>0.11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Complexity</th>
<th>O(n)</th>
<th>O(n)</th>
<th>O(n)</th>
<th>O(n)</th>
<th>O(n)</th>
<th>O(n)</th>
<th>O(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊙ / ⊙ / ⊙</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
</tr>
</tbody>
</table>

Table 7: Final performance comparisons on selected Mujoco tasks between SF-FWA and state-of-the-art gradient-free algorithms after 100000 episodes, the maximum mean rewards across 10 runs and the standard deviations are shown in the table. The Wilcoxon rank-sum test is performed to test whether the mean performance of SF-FWA is significantly better than comparing algorithms. ⊙ indicates that p-value is less than 0.05, while ⊙ ⊙ ⊙ indicates no significant difference detected.

Figure 8: Visualizations of the selected Mujoco continuous control tasks

Figure 9: Results on optimizing neural network controllers for Mujoco continuous control tasks
task repeatedly 10 times for up to 100,000 episodes. The optimization curve was plotted in Figure 9 where the lines indicate the mean reward and the shades indicate standard deviations. The final performance is reported in Table 7, where the Wilcoxon rank sum test is conducted to compare SF-FWA against other algorithms.

From Figure 9 and Table 7, SF-FWA outperformed all other algorithms significantly. The lead is especially obvious on Ant-v4, where SF-FWA reached twice the maximum reward as the second-place algorithm. On HalCheetah-v4, SF-FWA managed to reach around 5500 maximum rewards while all other algorithms ended at around 4000. On Humanoid-v4, which features a large parameter space, SF-FWA showed outstanding exploration ability by quickly reaching 1500 rewards and finally managed to score around 2000. Finally, on swimmer-v4, all algorithms climbed up to around 360 rewards quickly except PGPE, and SF-FWA is not only one of the fastest algorithms to reach convergence, but it also ended up being significantly better than other algorithms.

This experiment demonstrates that SF-FWA can serve as a strong candidate to be applied to reinforcement learning problems.

5. Conclusion

In this study, we present the Self-adaptive Fast Fireworks Algorithm (SF-FWA) as a solution for large-scale optimization problems. By combining two novel mechanisms: Expressive Fast Explosion (EFF) and Inter-Fireworks Competitive Cooperation (IFCC), SF-FWA balances the ability to exploit separable problems with the ability to tackle rotated coordinate systems. Our experiments confirmed the efficacy of SF-FWA, demonstrating its good scalability and state-of-the-art performance on complex LSGO problems. Our results also showed that SF-FWA outperforms gradient-free methods in optimizing policy neural networks for reinforcement learning, demonstrating its potential for wider applications in machine learning and real-world problems.

For future work, several directions hold promising potential for further development and improvement of SF-FWA. One area of focus is to analyze the impact of different covariance model structures on the efficiency of SF-FWA for different types of problems. Additionally, there is room for exploration into whether other hyperparameters besides $\gamma$ and $\phi$ can be adapted for improved performance. Furthermore, integrating other collaboration mechanisms such as loser-out-tournament, search space partition, and hierarchical collaboration into SF-FWA could lead to even richer search behavior and capabilities. Overall, the introduction of SF-FWA presents exciting opportunities for advancing the field of large-scale optimization and its application in machine learning and real-world problems.

Author Contributions

Ying Tan: Funding acquisition, investigation, conceptualization, supervision, manuscript revision. Maiyue Chen: Investigation, conceptualization, algorithm design, algorithm implementation, experimental study, manuscript writing.

Acknowledgement

This work was financially supported by the National Natural Science Foundation of China under grant 62250037. It was also supported by the National Natural Science Foundation of China grant 62076010, 62276008 and the Science and Technology Innovation 2036—“New Generation Artificial Intelligence” Major Project (Grant Nos.: 2018AAA0102302). We thank Xiangrui Meng, Yifan Liu and Shipeng Cen for their valuable suggestions on our work.

References


18


[58] M. A. Akbari, M. Zare, R. Antizipah-abarghoee, S. Mirjalili,


[69] M. Molga, C. Smutnicki, Test functions for optimization needs 43.


• A self-adaptive and computationally efficient algorithm for large-scale optimization.
• Explicit separability exploitation while keeping the ability to deal with rotations.
• State-of-the-art performance on standard large-scale optimization benchmark.
• Superior performance on optimizing neural networks for reinforcement learning.
Declaration of interests

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

☒ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Ying Tan reports financial support was provided by National Natural Science Foundation of China.