Revisiting Population Models in Differential Evolution on a Limited Budget of Evaluations

Ryoji Tanabe

September 19, 2020

Revisiting Population Models in Differential Evolution on a Limited Budget of Evaluations*

Ryoji Tanabe

Yokohama National University, Yokohama, Japan
rt.ryoji.tanabe@gmail.com

Abstract. No previous study has reported that differential evolution (DE) is competitive with state-of-the-art black-box optimizers on a limited budget of evaluations (i.e., the expensive optimization scenario). This is true even for surrogate-assisted DEs. The basic framework of DE should be reconsidered to improve its performance substantially. In this context, this paper revisits population models in DE on a limited budget of evaluations. This paper analyzes the performance of DE with five population models on the BBOB function set. Results demonstrate that the traditional synchronous model is unsuitable for DE in most cases. In contrast, the performance of DE can be significantly improved by using the plus-selection model and the worst improvement model. Results also demonstrate that DE with a suitable population model is competitive with covariance matrix adaptation evolution strategy depending on the number of evaluations and the dimensionality of a problem.

1 Introduction

Single-objective black-box numerical optimization involves finding a solution $\boldsymbol{x} = (x_1, ..., x_n)^\top$ that minimizes a given objective function $f : \mathbb{X} \rightarrow \mathbb{R}$. Here, $\mathbb{X}$ is the n-dimensional solution space. Any explicit knowledge of f is unavailable.

This paper considers black-box optimization on a limited budget of evaluations. Optimization with a small number of function evaluations (e.g., $100 \times n$ evaluations) is generally called *expensive optimization*. In contrast, this paper denotes optimization with a relatively large number of function evaluations (e.g., $10\,000 \times n$ evaluations) as *cheap optimization*. Some real-world problems require a long computation time to evaluate a solution $\boldsymbol{x}$ by expensive computer simulations [3,24] (e.g., CFD [6]). Also, expensive optimization frequently appears in the filed of hyperparameter optimization of machine learning models [10].

In the evolutionary computation community, surrogate-assisted evolutionary algorithms are representative approaches for expensive optimization [3,24]. In general, surrogate-assisted evolutionary algorithms replace an expensive objective function with a cheap surrogate model. Then, the objective value of a new solution is predicted by the surrogate model based on past solutions found during

* The final authenticated publication is available online at https://doi.org/10.1007/978-3-030-58112-1_18.

the search process. It is expected that surrogate-assisted evolutionary algorithms can effectively reduce the number of evaluations by an actual objective function.

Covariance matrix adaptation evolution strategy (CMA-ES) is a variant of evolution strategies (ES) for numerical optimization [13,19]. Surrogate-assisted approaches have been intensively studied in the field of ES, especially CMA-ES [4,14,26,33]. Surrogate-assisted CMA-ES has shown state-of-the-art performance for expensive optimization. Even a non-surrogate-assisted CMA-ES performs well for expensive optimization. Although Bayesian optimizers (e.g., EGO [25]) generally show good performance within a very small number of evaluations, a non-surrogate-assisted CMA-ES performs well after that in most cases [32,36].

Similar to CMA-ES, differential evolution (DE) is a numerical optimizer [42]. The results in the annual IEEE CEC competitions have demonstrated that some DEs are competitive with more complex optimizers for *cheap optimization* (e.g., [46]). However, the number of previous studies on DE algorithms for *expensive optimization* is much smaller than that for *cheap optimization*. In contrast to the ES community, as pointed out in [7], surrogate-assisted approaches have not received much attention in the DE community. Also, a surrogate-assisted DE has not been competitive even with a non-surrogate-assisted CMA-ES. Most previous studies "avoided" to compare DE with CMA-ES for expensive optimization.

Nevertheless, the simplicity of DE is still attractive in practice. The easy-to-use property of DE is valuable for a non-expert user who just uses an evolutionary algorithm as a black-box optimization tool. In this context, we want to substantially improve the performance of DE for expensive optimization. To achieve this goal while keeping the simplicity of DE, this paper revisits population models in DE for expensive optimization. A population model determines how to update the population for each iteration. The original DE [42] uses the synchronous model, which updates all individuals in the population simultaneously by using one-to-one survivor selection. In addition to the synchronous model, this paper analyzes the performance of DE with the following four population models: the asynchronous [8,53,54], $(\mu + \lambda)$ [37,49], worst improvement [1], and subset-to-subset [12] models. Although some previous studies (e.g., [8]) investigated the effect of population models in DE, such an analysis is only for cheap optimization, not for expensive optimization. Thus, little is known about the influence of the population model on the performance of DE for expensive optimization.

Our contributions in this paper are at least threefold:

1. We demonstrate that the performance of DE can be significantly improved by replacing the traditional synchronous model with the $(\mu + \lambda)$ or worst improvement models. This means that a more efficient surrogate-assisted DE for expensive optimization could be designed by using the two models.
2. We compare DE with the two models to CMA-ES and other optimizers. We show that DE with a suitable population model performs better than or similar to CMA-ES depending on the number of function evaluations and the dimensionality of a problem. This is the first study to report such a promising performance of DE for expensive optimization.

Algorithm 1: Synchronous model

```
1 Initialize $\boldsymbol{P} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_\mu\}$ randomly;
2 while The termination criteria are not met do
3     for $i \in \{1, ..., \mu\}$ do
4         $\boldsymbol{u}_i \leftarrow$ generateTrialVector($\boldsymbol{P}$);
5     for $i \in \{1, ..., \mu\}$ do
6         if $f(\boldsymbol{u}_i) \leq f(\boldsymbol{x}_i)$ then $\boldsymbol{x}_i \leftarrow \boldsymbol{u}_i$;
```

3. DE with the two models can be viewed as a base-line. Any surrogate-assisted DE should outperform the two non-surrogate-assisted DEs. Some surrogate-assisted DEs have been proposed (e.g., [30,34,51,56]). However, benchmarking a surrogate-assisted DE has not been standardized in the DE community. It is also difficult to accurately reproduce experimental results of most existing surrogate-assisted DEs since their source code is not available through the Internet. Consequently, the progress is unclear. Our base-line addresses this issue, facilitating a constructive development of a surrogate-assisted DE.

The rest of this paper is organized as follows. Section 2 explains the five population models in DE. Section 3 describes the setting of our computational experiments. Section 4 shows analysis results. Section 5 concludes this paper.

2 Five population models in DE

Here, we do not consider any method that aims to maintain the diversity in the population, such as crowding DE [47], island/distributed models [8,52], and the cellular topology [8,35]. These methods aim to prevent the premature convergence of DE to find a good solution with a large number of function evaluations. Thus, these methods are not suitable for expensive optimization, where DE needs to quickly find a good solution with a small number of function evaluations.

We carefully surveyed the literature on population models that aim to accelerate the convergence speed of DE. As a result, we found the following four population models: the asynchronous model [8,53,54], the $(\mu + \lambda)$ model [37,49], the worst improvement model [1], and the subset-to-subset (STS) model [12]. Algorithms 1–5 show the overall procedure of the basic DE with the traditional synchronous model and the four population models. We extracted only the population models from their corresponding original algorithms. For example, the worst improvement model is derived from DE with generalized differential (DEGD) [1], which consists of multiple components. In this study, we want to focus only on the population model. For this reason, we generalized it so that we can examine its effectiveness in an isolated manner. For the same reason, we do not use any surrogate model and any parameter adaptation method for the scale factor F and the crossover rate C [45]. Below, we explain the five models.

• **Synchronous model (Algorithm 1)**

The synchronous model is used in the original DE [42] and most recent DE variants, including jDE [5], JADE [57], CoDE [50], and SHADE [43]. Here, we explain the synchronous model and some basic operations in DE.

At the beginning of the search, the population $\boldsymbol{P} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_\mu\}$ is initialized (line 1 in Algorithm 1), where μ is the population size. For each $i \in \{1, ..., \mu\}$, $\boldsymbol{x}_i$ is the i-th individual in the population $\boldsymbol{P}$. Here, $\boldsymbol{x}_i$ is an n-dimensional solution of a problem. For each $j \in \{1, ..., n\}$, $x_{i,j}$ is the j-th element of $\boldsymbol{x}_i$. According to the DE terminology, we use the terms "individual" and "vector" synonymously.

After the initialization of $\boldsymbol{P}$, the following steps (lines 2–6 in Algorithm 1) are repeatedly performed until a termination condition is satisfied. For each $i \in \{1, ..., \mu\}$, a trial vector (child) $\boldsymbol{u}_i$ is generated (lines 3–4 in Algorithm 1). In this operation, first, a mutant vector $\boldsymbol{v}_i$ is generated by applying a differential mutation to some individuals in $\boldsymbol{P}$. Table 1 shows seven representative mutation strategies in DE. The scale factor F controls the magnitude of the mutation. Parent indices $r_1, r_2, ...$ are randomly selected from $\{1, ..., \mu\} \setminus \{i\}$ such that they differ from each other. In Table 1, $\boldsymbol{x}_{\text{best}}$ is the best individual in $\boldsymbol{P}$. For each $i \in \{1, ..., \mu\}$, $\boldsymbol{x}_{p\text{best}}$ is randomly selected from the top $\max(\lfloor p\,\mu \rfloor, 2)$ individuals in $\boldsymbol{P}$, where $p \in [0, 1]$ controls the greediness of current-to-$p$best/1 [57] and rand-to-$p$best/1 [55]. Also, $\tilde{\boldsymbol{x}}_{r_2}$ and $\tilde{\boldsymbol{x}}_{r_3}$ in current-to-$p$best/1 and rand-to-$p$best/1 are randomly selected from the union of $\boldsymbol{P}$ and an external archive $\boldsymbol{A}$.

After the mutant vector $\boldsymbol{v}_i$ has been generated, a trial vector $\boldsymbol{u}_i$ is generated by applying crossover to $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$. In this study, we use binomial crossover [42]:

$$u_{i,j} := \begin{cases} v_{i,j} & \text{if } q_j \leq C \text{ or } j = j_{\text{rand}} \\ x_{i,j} & \text{otherwise} \end{cases}, \tag{1}$$

where q_j is randomly selected from $[0, 1]$, and j_{rand} is randomly selected from $\{1, ..., n\}$. The crossover rate C in (1) controls the number of inherited elements from the target vector $\boldsymbol{x}_i$ to the trial vector $\boldsymbol{u}_i$.

After the μ trial vectors have been generated, all individuals are updated simultaneously (lines 5–6 in Algorithm 1). If $f(\boldsymbol{u}_i) \leq f(\boldsymbol{x}_i)$ for each $i \in \{1, ..., \mu\}$, $\boldsymbol{x}_i$ is replaced with $\boldsymbol{u}_i$. The individuals that were worse than the trial vectors are stored in the external archive $\boldsymbol{A}$. When $|\boldsymbol{A}|$ exceeds a pre-defined size, randomly selected individuals are deleted to keep the archive size constant.

- **Asynchronous model (Algorithm 2)**

In contrast to the synchronous model, individuals in the population are updated in an asynchronous manner. Thus, immediately after the trial vector $\boldsymbol{u}$ has been generated, $\boldsymbol{x}_i$ can be replaced with $\boldsymbol{u}$ (lines 4–5 in Algorithm 2). It is difficult to find out the first study that proposed the asynchronous model in DE since such an idea is general. In fact, some previous studies did not explicitly describe that they dealt with the asynchronous model (e.g., [53]). While DE with the asynchronous model can immediately use a new superior individual for the search, DE with the synchronous model needs to "wait" by the next iteration. For this reason, the asynchronous model is generally faster than the synchronous model in terms of the convergence speed of the population [7,8].

- $(\mu + \lambda)$ **model (Algorithm 3)**

Table 1: Seven representative mutation strategies for DE.

Strategies	Definitions
rand/1	$\boldsymbol{v}_i := \boldsymbol{x}_{r_1} + F\,(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3})$
rand/2	$\boldsymbol{v}_i := \boldsymbol{x}_{r_1} + F_i\,(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) + F_i\,(\boldsymbol{x}_{r_4} - \boldsymbol{x}_{r_5})$
best/1	$\boldsymbol{v}_i := \boldsymbol{x}_{\text{best}} + F_i\,(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2})$
best/2	$\boldsymbol{v}_i := \boldsymbol{x}_{\text{best}} + F_i\,(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}) + F_i\,(\boldsymbol{x}_{r_3} - \boldsymbol{x}_{r_4})$
current-to-best/1	$\boldsymbol{v}_i := \boldsymbol{x}_i + F_i\,(\boldsymbol{x}_{\text{best}} - \boldsymbol{x}_i) + F_i\,(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2})$
current-to-*p*best/1	$\boldsymbol{v}_i := \boldsymbol{x}_i + F_i\,(\boldsymbol{x}_{p\text{best}} - \boldsymbol{x}_i) + F_i\,(\boldsymbol{x}_{r_1} - \tilde{\boldsymbol{x}}_{r_2})$
rand-to-*p*best/1	$\boldsymbol{v}_i := \boldsymbol{x}_{r_1} + F_i\,(\boldsymbol{x}_{p\text{best}} - \boldsymbol{x}_{r_1}) + F_i\,(\boldsymbol{x}_{r_2} - \tilde{\boldsymbol{x}}_{r_3})$

Algorithm 2: Asynchronous model

```
1 Initialize $\boldsymbol{P} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_\mu\}$ randomly;
2 while The termination criteria are not met do
3     for $i \in \{1, ..., \mu\}$ do
4         $\boldsymbol{u} \leftarrow$ generateTrialVector($\boldsymbol{P}$);
5         if $f(\boldsymbol{u}) \leq f(\boldsymbol{x}_i)$ then $\boldsymbol{x}_i \leftarrow \boldsymbol{u}$;
```

The elitist $(\mu + \lambda)$ model is general in the field of evolutionary algorithms, including genetic algorithm and ES. For each iteration, a set of λ trial vectors $\boldsymbol{Q} = \{\boldsymbol{u}_1, ..., \boldsymbol{u}_\lambda\}$ are generated (lines 3–5 in Algorithm 3). For each $\boldsymbol{u}$, the target vector is randomly selected from the population. Then, the best μ individuals in $\boldsymbol{P} \cup \boldsymbol{Q}$ survive to the next iteration (line 6 in Algorithm 3). Unlike other evolutionary algorithms, the $(\mu + \lambda)$ model has not received much attention in the DE community. Only a few previous studies (e.g., [37,39,49]) considered the $(\mu + \lambda)$ model. As pointed out in [37], the synchronous model may discard a trial vector that performs worse than its parent but performs better than other individuals in the population. The $(\mu + \lambda)$ model addresses such an issue.

• **Worst improvement model (Algorithm 4)**

In the worst improvement model [1], only λ worst individuals can generate trial vectors (lines 3–5 in Algorithm 4). Then, the λ individuals and their λ trial vectors are compared as in the synchronous model (lines 6–7 in Algorithm 4).

Ali [1] demonstrated that a better individual is rarely replaced with its trial vector. In other words, the better the individual $\boldsymbol{x}_i$ ($i \in \{1, ..., \mu\}$) is, the more difficult it is to generate $\boldsymbol{u}_i$ such that $f(\boldsymbol{u}_i) \leq f(\boldsymbol{x}_i)$. In contrast, a worse individual is frequently replaced with its trial vector. Based on this observation, Ali proposed the worst improvement model that allows only the λ worst individuals to generate their λ trial vectors. The number of function evaluations could possibly be reduced by not generating the remaining $\mu - \lambda$ trial vectors that are unlikely to outperform their $\mu - \lambda$ parent individuals.

• **Subset-to-subset (STS) model (Algorithm 5)**

Algorithm 3: $(\mu + \lambda)$ model

```
1 Initialize $\boldsymbol{P} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_\mu\}$ randomly;
2 while The termination criteria are not met do
3     $\boldsymbol{Q} \leftarrow \emptyset$;
4     for $i \in \{1, ..., \lambda\}$ do
5         $\boldsymbol{u} \leftarrow$ generateTrialVector($\boldsymbol{P}$), $\boldsymbol{Q} \leftarrow \boldsymbol{Q} \cup \{\boldsymbol{u}\}$;
6     $\boldsymbol{P} \leftarrow$ Select the $\mu$ best individuals from the union $\boldsymbol{P} \cup \boldsymbol{Q}$;
```

Algorithm 4: Worst improvement model

```
1 Initialize $\boldsymbol{P} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_\mu\}$ randomly;
2 while The termination criteria are not met do
3     $\boldsymbol{K} \leftarrow$ Select indices of the $\lambda$ worst individuals in $\boldsymbol{P}$;
4     for $i \in \boldsymbol{K}$ do
5         $\boldsymbol{u}_i \leftarrow$ generateTrialVector($\boldsymbol{P}$);
6     for $i \in \boldsymbol{K}$ do
7         if $f(\boldsymbol{u}_i) \leq f(\boldsymbol{x}_i)$ then $\boldsymbol{x}_i \leftarrow \boldsymbol{u}_i$;
```

The aim of the STS model [12] is the same as that of the $(\mu + \lambda)$ model. The STS model uses the index-based ring topology. In the STS model, μ individuals and μ trial vectors are grouped based on their indices and the subset size $s \geq 2$. After μ trial vectors have been generated (lines 4–5 in Algorithm 5), an index i is randomly selected from $\{1, ..., \mu\}$ (line 6 in Algorithm 5), where i determines the start position on the index-based ring topology. First, l is set to s or the size of the remaining individuals (line 8 in Algorithm 5). Then, a set of l individuals and l trial vectors are stored into $\boldsymbol{X}$ based on the index-based ring topology (line 9 in Algorithm 5). In Algorithm 5, the function "modulo(a, b)" returns the remainder of the division of a by b. Finally, the l best individuals in $\boldsymbol{X}$ survive to the next iteration (lines 10–11 in Algorithm 5).

For example, suppose that $\mu = 5$, $s = 2$, and $i = 3$. In this case, 5+5 individuals are grouped as follows: $\{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{u}_3, \boldsymbol{u}_4\}$, $\{\boldsymbol{x}_5, \boldsymbol{x}_1, \boldsymbol{u}_5, \boldsymbol{u}_1\}$, and $\{\boldsymbol{x}_2, \boldsymbol{u}_2\}$. When $f(\boldsymbol{x}_3) = 0.3$, $f(\boldsymbol{x}_4) = 0.1$, $f(\boldsymbol{u}_3) = 0.4$, and $f(\boldsymbol{u}_4) = 0.2$, $\boldsymbol{x}_4$ and $\boldsymbol{u}_4$ survive to the next iteration as follows: $\boldsymbol{x}_3 := \boldsymbol{x}_4$ and $\boldsymbol{x}_4 := \boldsymbol{u}_4$. Notice that $\boldsymbol{u}_4$ cannot survive to the next iteration in the synchronous model since $f(\boldsymbol{u}_4) > f(\boldsymbol{x}_4)$.

3 Experimental setup

We performed all experiments using the COCO software [16]. The source code used in our experiments is available at `https://github.com/ryojitanabe/de_expensiveopt`. We used the 24 BBOB noiseless functions $f_1, ..., f_{24}$ [18], which are grouped into the following five categories: separable functions $(f_1, ..., f_5)$, functions with low or moderate conditioning $(f_6, ..., f_9)$, functions with high

Algorithm 5: STS model

```
1  Initialize $\boldsymbol{P} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_\mu\}$ randomly;
2  $h \leftarrow \mu/s$ ;                                   // If modulo$(\mu, s) \neq 0$, $h \leftarrow \mu/s + 1$
3  while The termination criteria are not met do
4      for $i \in \{1, ..., \mu\}$ do
5          $\boldsymbol{u}_i \leftarrow$ generateTrialVector$(\boldsymbol{P})$;
6      $i \leftarrow$ Randomly select an index from $\{1, ..., \mu\}$;
7      for $j \in \{1, ..., h\}$ do
8          $l \leftarrow \min\{\mu - ((j-1) \times s), s\}$;
9          $\boldsymbol{K} \leftarrow \{\text{modulo}(i, \mu), ..., \text{modulo}(i + l - 1, \mu)\}$,
           $\boldsymbol{X} \leftarrow \{\boldsymbol{x}_k | k \in \boldsymbol{K}\} \cup \{\boldsymbol{u}_k | k \in \boldsymbol{K}\}$;
10         for $k \in \boldsymbol{K}$ do
11             $\boldsymbol{x}_k \leftarrow \underset{\boldsymbol{x} \in \boldsymbol{X}}{\arg\min}\{f(\boldsymbol{x})\}$, $\boldsymbol{X} \leftarrow \boldsymbol{X} \setminus \{\boldsymbol{x}_k\}$;
12         $i \leftarrow$ modulo$(i + s, \mu)$;
```

conditioning and unimodal ($f_{10}, ..., f_{14}$), multimodal functions with adequate global structure ($f_{15}, ..., f_{19}$), and multimodal functions with weak global structure ($f_{20}, ..., f_{24}$). The dimensionality n of the functions was set to $2, 3, 5, 10, 20$, and 40. For each function, 15 runs were performed. These settings adhere to the procedure in COCO. According to the expensive optimization scenario in COCO, the maximum number of function evaluations was set to $100 \times n$.

In general, the best parameter setting in EAs depends on the allowed budget of evaluations [9,38,41,44]. Although parameter studies on DE have been well performed for cheap optimization (e.g., [5,11,58]), little is known about a suitable parameter setting for expensive optimization. Thus, it is unclear how to set control parameters in DE. We try to address this issue by "hand-tuning" and "automated algorithm configuration".

We performed "hand-tuning" to select μ, the mutation strategy, and some parameters (i.e., λ and s) in the five population models. We mainly used the Sphere function $f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$ for benchmarking. As a result, we set μ to $\lfloor \alpha_\mu \ln n \rfloor$, and $\alpha_\mu = 13$. The population was initialized using Latin hypercube sampling. We used rand-to-$p$best/1 in Table 1. As in [57], we set the parameters of rand-to-$p$best/1 as follows: $p = 0.05$ and $|\boldsymbol{A}| = \mu$. As in the standard setting in DE for cheap optimization, we set F and C to 0.5 and 0.9, respectively. These settings were suitable for most population models. In the $(\mu + \lambda)$ and worst improvement models, we set λ to 1. We also set s in the STS model to 2.

We performed "automatic algorithm configuration" using SMAC [22], which is a surrogate-model based configurator. We used the latest version of SMAC (version 2.10.03) downloaded from the authors' website. We set the cost function in SMAC (i.e., the estimated performance of a configuration) to the error value $|f(\boldsymbol{x}^{\text{bsf}}) - f(\boldsymbol{x}^*)|$, where $\boldsymbol{x}^{\text{bsf}}$ is the best-so-far solution found by DE, and $\boldsymbol{x}^*$ is the optimal solution of a training problem. We used the 28 CEC2013 functions [28] with $n \in \{2, 5, 10, 20, 40\}$ as the training problems. For each run of DE, the

Table 2: Parameters tuned by SMAC, where $\mu = \max\{\lfloor \alpha_\mu \ln n \rfloor, 6\}$, $|\boldsymbol{A}| = \lfloor \alpha_{\mathrm{arc}} \mu \rfloor$, $\lambda = \max\{\lfloor \alpha_\lambda \mu \rfloor, 1\}$, and $s = \max\{\lfloor \alpha_s \mu \rfloor, 2\}$.

	α_μ	Strategy	p	α_{arc}	F	C	α_λ	α_s
Range	$[5, 20]$	See Table 1	$[0, 1]$	$[0, 3]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 0.2]$
Default	10	rand/1	0.05	1.0	0.5	0.5	0	0
Synchronous	9.27	rand-to-$p$best/1	0.17	0.58	0.51	0.52	-	-
Asynchronous	9.09	rand-to-$p$best/1	0.29	1.81	0.50	0.62	-	-
$(\mu + \lambda)$	9.50	rand-to-$p$best/1	0.34	1.96	0.53	0.65	0.64	-
Worst improvement	7.33	rand-to-$p$best/1	0.89	1.62	0.58	0.75	0.22	-
STS	5.44	rand-to-$p$best/1	0.36	1.95	0.61	0.61	-	0.18

maximum number of function evaluations was set to $100 \times n$. For each run of SMAC, the maximum number of configuration evaluations was set to 5 000. For each population model, five independent SMAC runs were performed. Then, we evaluated the performance of DE with the five configurations found by SMAC on the CEC2013 set with $n \in \{2, 5, 10, 20, 40\}$. Finally, we selected the best one from the five configurations based on their average rankings by the Friedman test. Table 2 shows the range of each parameter, the default configuration, and the best configuration for each model. Interestingly, all configurations include rand-to-$p$best/1. For all configurations, F and C values are relatively similar.

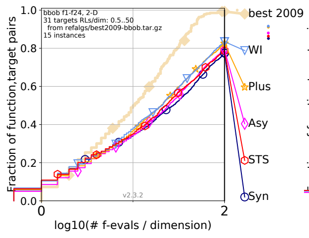
4 Results

This section analyzes the performance of DE with the five population models. We mainly discuss our results based on the anytime performance of optimizers, rather than the end-of-the-run results exactly at $100 \times n$ evaluations. For the sake of simplicity, we refer to "a DE with a population model" as "a population model". We also use the following abbreviations in Figures 1, 2, and 3: the synchronous (Syn), asynchronous (Asy), $(\mu + \lambda)$ (Plus), worst improvement (WI), and subset-to-subset (STS) models. Section 4.1 demonstrates the performance of the five population models with the hand-tuned parameters. Section 4.2 examines the effect of automatic algorithm configuration in the population models. Section 4.3 compares two population models to CMA-ES and other optimizers.

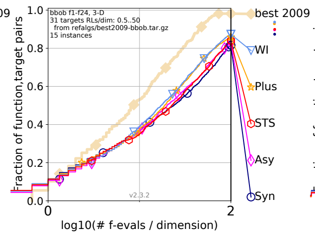
4.1 Comparison of the five population models

Figure 1 shows results of the five population models with the hand-tuned parameters on all 24 BBOB functions with $n \in \{2, 3, 5, 10, 20, 40\}$. In Figure 1, "best 2009" is a *virtual* algorithm portfolio that consists of the performance data of 31 algorithms participating in the GECCO BBOB 2009 workshop [17].

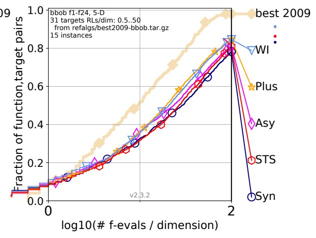
Figure 1 shows the bootstrapped empirical cumulative distribution (ECDF) of the number of function evaluations (FEvals) divided by n (FEvals/n) for 31 targets for all 24 BBOB functions. We used the COCO software with the



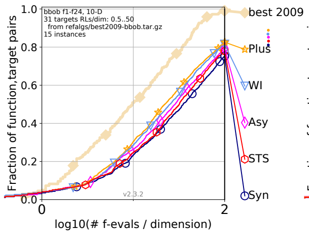
(a) $n = 2$



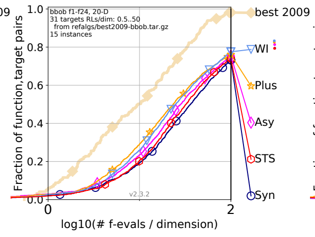
(b) $n = 3$



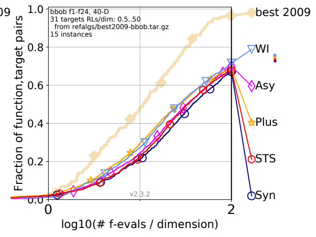
(c) $n = 5$



(d) $n = 10$



(e) $n = 20$



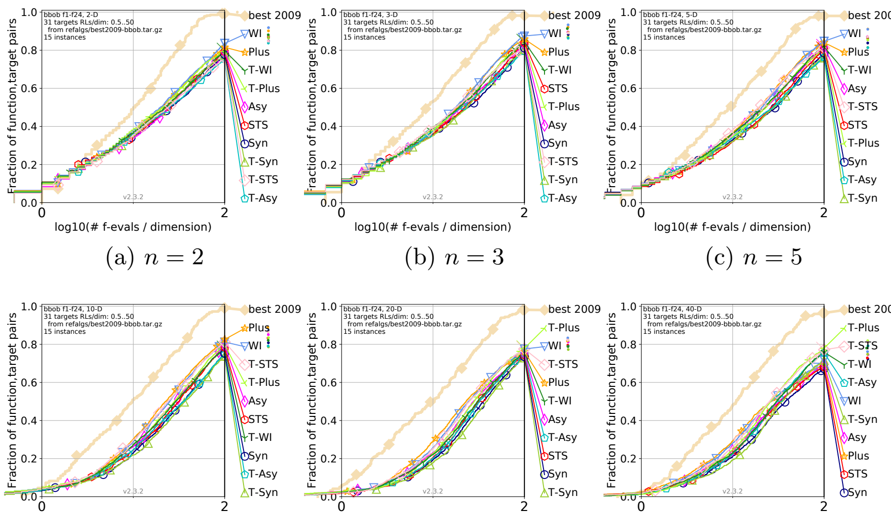
(f) $n = 40$

Fig. 1: Results of the five population models with the hand-tuned parameters. For each abbreviation (Syn, Asy, Plus, WI, and STS), see the beginning of Section 4.

"`--expensive`" option to generate all ECDF figures in this paper. Although the target error values are usually in $\{10^2, ..., 10^{-8}\}$, they are adjusted based on "best 2009". In ECDF figures, the vertical axis indicates the proportion of target error values reached by an optimizer within specified function evaluations. For example, in Figure 1 (e), the synchronous model reaches about 20 percent of all 31 target error values within $10 \times n$ evaluations on all 24 BBOB functions with $n = 20$ in all 15 runs. For more details of ECDF, see [15]. In addition to ECDF, we analyzed the performance of the population models based on average run-time with the rank-sum test ($p = 0.05$), but the results are consistent with the ECDF figures in most cases. For this reason, we show only ECDF figures.

Figure 1 shows that the $(\mu + \lambda)$ and worst improvement models show a good performance on any dimensional problems. Although the $(\mu + \lambda)$ model performs slightly worse than the worst improvement model exactly at $100 \times n$ evaluations, it performs better than the worst improvement model at the early stage, especially for $n \in \{20, 40\}$. The asynchronous model performs better than the synchronous model. This is consistent with the results in previous studies (e.g., [7,8]). The asynchronous model is also competitive with the $(\mu + \lambda)$ model for $n = 40$. Compared to the STS model, the asynchronous model has the advantage that it does not require any parameter such as the subset size s.

Overall, we can say that the choice of a population model significantly influences the performance of DE for expensive optimization. The $(\mu + \lambda)$ and worst improvement models are the best when using the hand-tuned parameters.



(a) $n = 2$ (b) $n = 3$ (c) $n = 5$

(d) $n = 10$ (e) $n = 20$ (f) $n = 40$

Fig. 2: Results of the five population models with automatically-tuned parameters. The prefix "T-" means that the model uses the automatically-tuned parameters.

In contrast, the traditional synchronous model performs the worst in the five population models for all dimensions. Based on these observations, we do not recommend the use of the synchronous model for expensive optimization.

4.2 Effect of automatic algorithm configuration

This section investigates the effect of automatic algorithm configuration in the population models. Figure 2 shows results of the five population models with the automatically-tuned parameters. For details of the parameters, see Table 2.

Except for the STS model, the hand-tuned parameters are more suitable than the automatically-tuned parameters for $n \in \{2, 3, 5, 10\}$. In contrast, all the five models with the automatically-tuned parameters outperform those with the hand-tuned parameters for $n = 40$. The reason is discussed in Section 4.3. Although the $(\mu + \lambda)$ model with the automatically-tuned parameters performs the best exactly at $100 \times n$ evaluations for $n \in \{20, 40\}$, that with the hand-tuned parameters shows a good performance at the early stage.

In summary, we can obtain almost the same conclusion in Section 4.1 even when using the automatically-tuned parameters. For example, the synchronous model performs poorly even when using the automatically-tuned parameters.

4.3 Comparison to other optimizers

The results in Section 4.1 and 4.2 show that the worst improvement model with the hand-tuned parameters (WI) and the $(\mu+\lambda)$ model with the automatically-tuned parameters (T-Plus) perform the best in the five population models for $n \leq 10$ and $n \geq 20$, respectively. Here, we compare the two best population models with the following six optimizers. We downloaded the performance data of the six optimizers from the COCO data archive.

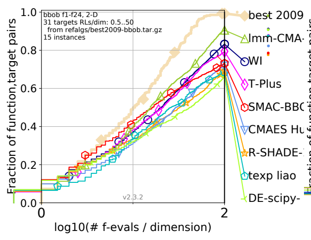
• `SMAC-BBOB` [23] is a Bayesian optimizer. Although the original version of SMAC [22] is an algorithm configurator as explained in Section 3, this version of SMAC (`SMAC-BBOB`) is a numerical optimizer. `SMAC-BBOB` is similar to EGO [25].
• `lmm-CMA` [2] is a surrogate-assisted CMA-ES with local meta-models [4].
• `CMAES_Hutter` [23] is a CMA-ES with the default parameters.
• `texp_liao` [29] is a CMA-ES with automatically-tuned parameters for expensive optimization as in Section 4.2. The irace tool [31] was used for tuning.
• `R-SHADE-10e2` [44] is a SHADE [43] with automatically-tuned parameters for expensive optimization as in Section 4.2. SHADE is one of the state-of-the-art adaptive DE algorithms. `R-SHADE-10e2` uses the synchronous model.
• `DE-scipy` [48] is DE from the Python SciPy library (`https://www.scipy.org/`). `DE-scipy` can be viewed as a DE used by a non-expert user. `DE-scipy` uses the asynchronous model.

Figure 3 shows results of the two population models and the six optimizers. For $n = 40$, only the data of `texp_liao`, `CMAES_Hutter`, and `DE-scipy` were available. Unsurprisingly, `SMAC-BBOB` and `lmm-CMA` perform the best at around $10 \times n$ and $100 \times n$ evaluations, respectively. We wanted to know how poorly the two population models perform compared to `SMAC-BBOB` and `lmm-CMA`.

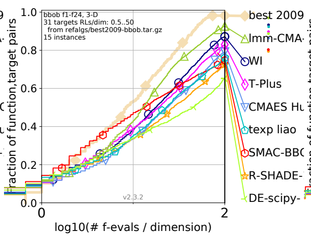
As seen from Figure 3, the $(\mu+\lambda)$ and worst improvement models perform significantly better than `R-SHADE-10e2` and `DE-scipy` for any n. This result provides important information to design an efficient DE, i.e., the basic DE with a suitable population model can outperform even SHADE. Our results indicate that the default version of `DE-scipy` is unsuitable for expensive optimization.

The $(\mu+\lambda)$ and worst improvement models perform significantly better than `CMAES_Hutter` and `texp_liao` for $n \in \{2, 3, 5\}$. The worst improvement model performs better than `CMAES_Hutter` and `texp_liao` until $100 \times n$ evaluations and performs similar to `CMAES_Hutter` and `texp_liao` exactly at $100 \times n$ evaluations. Although the two models perform significantly worse than `texp_liao` for $n \in \{20, 40\}$, they have a better performance than `CMAES_Hutter` at the early stage. The $(\mu+\lambda)$ model is also competitive with `CMAES_Hutter` at $100 \times n$ evaluations.

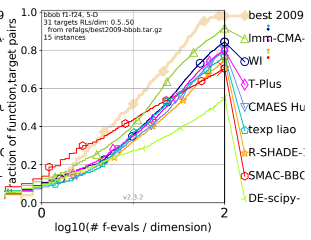
The $(\mu+\lambda)$ model in this section and `texp_liao` use the automatically-tuned parameters for expensive optimization. Interestingly, both optimizers perform well for high-dimensional functions ($n \geq 20$), but they perform poorly for low-dimensional functions ($n \leq 10$). This unintuitive observation may come from the difficulty in finding a parameter set with a good scalability to n. As reported in [40], automatically-tuned parameters are likely to fit for hard-to-solve training problems. Here, parameters in the $(\mu+\lambda)$ model were tuned on the 28 CEC2013 functions with $n \in \{2, 5, 10, 20, 40\}$, and parameters in `texp_liao` were tuned on



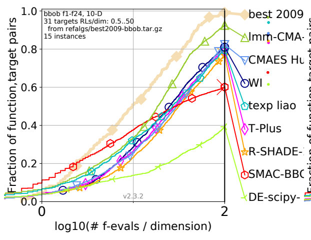
(a) $n = 2$



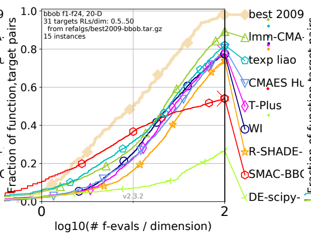
(b) $n = 3$



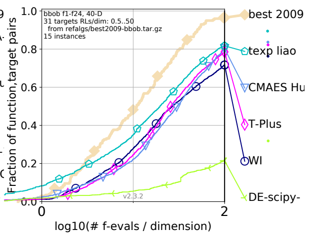
(c) $n = 5$



(d) $n = 10$



(e) $n = 20$



(f) $n = 40$

Fig. 3: Results of the two population models (WI and T-Plus) and the six optimizers.

the 19 SOCO functions [20] with $n \in \{5, 10, 20, 40\}$. It seems that parameters in the two optimizers fit only for high-dimensional functions. Although addressing this issue is beyond the scope of this paper, an in-depth analysis is needed.

5 Conclusion

We analyzed the performance of the five population models in DE for expensive optimization on the 24 BBOB functions. The traditional synchronous model performs the worst in most cases. In contrast, the worst improvement and $(\mu + \lambda)$ models are suitable for DE on a limited budget of evaluations. The worst improvement model with the hand-tuned parameters and the $(\mu + \lambda)$ model with the automatically-tuned parameters perform significantly better than CMA-ES for $n \in \{2, 3, 5\}$ and are competitive with CMA-ES for $n = 10$. In summary, our results demonstrated that DE with a suitable population model performs better than or similar to CMA-ES depending on the number of evaluations and dimensionality of a problem. This is the first study to report such a promising performance of DE for expensive optimization. DE with the two models should be a base-line for benchmarking new surrogate-assisted DE algorithms.

We believe that the poor performance of DE for $n \in \{20, 40\}$ compared to CMA-ES can be addressed by using an efficient parameter adaptation method for F and C [45]. It is promising to design a surrogate-assisted DE with the worst improvement and $(\mu + \lambda)$ models. It is also promising to design an algorithm portfolio [21,27,36] that consists of DE, CMA-ES, and Bayesian optimizers.

Acknowledgments. This work was supported by Leading Initiative for Excellent Young Researchers, MEXT, Japan.

References

1. M. M. Ali. Differential evolution with generalized differentials. *J. Comput. Appl. Math.*, 235(8):2205–2216, 2011.
2. A. Auger, D. Brockhoff, and N. Hansen. Benchmarking the local metamodel CMA-ES on the noiseless BBOB'2013 test bed. In *GECCO (Companion)*, pages 1225–1232, 2013.
3. T. Bartz-Beielstein and M. Zaefferer. Model-based methods for continuous and discrete global optimization. *Appl. Soft Comput.*, 55:154–167, 2017.
4. Z. Bouzarkouna, A. Auger, and D. Y. Ding. Local-meta-model CMA-ES for partially separable functions. In *GECCO*, pages 869–876, 2011.
5. J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE TEVC*, 10(6):646–657, 2006.
6. S. J. Daniels, A. A. M. Rahat, R. M. Everson, G. R. Tabor, and J. E. Fieldsend. A Suite of Computationally Expensive Shape Optimisation Problems Using Computational Fluid Dynamics. In *PPSN*, pages 296–307, 2018.
7. S. Das, S. S. Mullick, and P. N. Suganthan. Recent advances in differential evolution - an updated survey. *Swarm and Evol. Comput.*, 27:1–30, 2016.
8. B. Dorronsoro and P. Bouvry. Improving Classical and Decentralized Differential Evolution With New Mutation Operator and Population Topologies. *IEEE TEVC*, 15(1):67–98, 2011.
9. A. S. D. Dymond, A. P. Engelbrecht, and P. S. Heyns. The sensitivity of single objective optimization algorithm control parameter values under different computational constraints. In *IEEE CEC*, pages 1412–1419. IEEE, 2011.
10. M. Feurer and F. Hutter. Hyperparameter Optimization. In *Automated Machine Learning - Methods, Systems, Challenges*, pages 3–33. Springer, 2019.
11. R. Gämperle, S. D. Müller, and P. Koumoutsakos. A parameter study for differential evolution. In *Int. Conf. on Adv. in Intelligent Systems, Fuzzy Systems, Evol. Comput.*, pages 293–298, 2002.
12. J. Guo, Z. Li, and S. Yang. Accelerating differential evolution based on a subset-to-subset survivor selection operator. *Soft Comput.*, 23(12):4113–4130, 2019.
13. N. Hansen. The CMA Evolution Strategy: A Tutorial. *CoRR*, abs/1604.00772, 2016.
14. N. Hansen. A global surrogate assisted CMA-ES. In A. Auger and T. Stützle, editors, *GECCO*, pages 664–672, 2019.
15. N. Hansen, A. Auger, D. Brockhoff, D. Tušar, and T. Tušar. COCO: Performance Assessment. *CoRR*, abs/1605.03560, 2016.
16. N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *CoRR*, 2016.
17. N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posík. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *GECCO (Companion)*, pages 1689–1696, 2010.
18. N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical report, INRIA, 2009.

19. N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evol. Comput.*, 9(2):159–195, 2001.
20. F. Herrera, M. Lozano, and D. Molina. Test suite for the spec. iss. of Soft Computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. Technical report, Univ. of Granada, 2010.
21. M. D. Hoffman, E. Brochu, and N. de Freitas. Portfolio Allocation for Bayesian Optimization. In *UAI*, pages 327–336, 2011.
22. F. Hutter, F. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*, pages 507–523, 2011.
23. F. Hutter, H. H. Hoos, and K. Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *GECCO (Companion)*, pages 1209–1216, 2013.
24. Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evol. Comput.*, 1(2):61–70, 2011.
25. D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *J. Global Optimiz.*, 13(4):455–492, 1998.
26. A. Kayhani and D. V. Arnold. Design of a Surrogate Model Assisted (1 + 1)-ES. In *PPSN*, pages 16–28, 2018.
27. P. Kerschke and H. Trautmann. Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning. *Evol. Comput.*, 27(1):99–127, 2019.
28. J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Díaz. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. Technical report, Nanyang Technological Univ., 2013.
29. T. Liao and T. Stützle. Expensive optimization scenario: IPOP-CMA-ES with a population bound mechanism for noiseless function testbed. In *GECCO (Companion)*, pages 1185–1192, 2013.
30. B. Liu, Q. Zhang, and G. G. E. Gielen. A Gaussian Process Surrogate Model Assisted Evolutionary Algorithm for Medium Scale Expensive Optimization Problems. *IEEE TEVC*, 18(2):180–192, 2014.
31. M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.*, 3:43–58, 2016.
32. I. Loshchilov and F. Hutter. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *CoRR*, abs/1604.07269, 2016.
33. I. Loshchilov, M. Schoenauer, and M. Sebag. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *GECCO*, pages 321–328, 2012.
34. X. Lu, K. Tang, B. Sendhoff, and X. Yao. A new self-adaptation scheme for differential evolution. *Neurocomputing*, 146:2–16, 2014.
35. N. Lynn, M. Z. Ali, and P. N. Suganthan. Population topologies for particle swarm optimization and differential evolution. *Swarm and Evol. Comput.*, 39:24–35, 2018.
36. H. Mohammadi, R. L. Riche, and E. Touboul. Making EGO and CMA-ES Complementary for Global Optimization. In *LION*, 2015.
37. N. Noman and H. Iba. A new generation alternation model for differential evolution. In *GECCO*, pages 1265–1272, 2006.
38. A. P. Piotrowski. Review of Differential Evolution population size. *Swarm and Evol. Comput.*, 32:1–24, 2017.
39. S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition-Based Differential Evolution. *IEEE TEVC*, 12(1):64–79, 2008.
40. S. K. Smit and A. E. Eiben. Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. In *EvoApplicatons*, volume 6024, pages 542–551, 2010.

41. K. Socha. The Influence of Run-Time Limits on Choosing Ant System Parameters. In *GECCO*, pages 49–60, 2003.
42. R. Storn and K. Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Glo. Opt.*, 11(4):341–359, 1997.
43. R. Tanabe and A. Fukunaga. Success-History Based Parameter Adaptation for Differential Evolution. In *IEEE CEC*, pages 71–78, 2013.
44. R. Tanabe and A. Fukunaga. Tuning differential evolution for cheap, medium, and expensive computational budgets. In *IEEE CEC*, pages 2018–2025, 2015.
45. R. Tanabe and A. Fukunaga. Reviewing and Benchmarking Parameter Control Methods in Differential Evolution. *IEEE Trans. Cyber.*, 50(3):1170–1184, 2020.
46. R. Tanabe and A. S. Fukunaga. Improving the search performance of SHADE using linear population size reduction. In *IEEE CEC*, pages 1658–1665, 2014.
47. R. Thomsen. Multimodal optimization using crowding-based differential evolution. In *IEEE CEC*, pages 1382–1389, 2004.
48. K. Varelas and M. Dahito. Benchmarking multivariate solvers of SciPy on the noiseless testbed. In *GECCO (Companion)*, pages 1946–1954, 2019.
49. Y. Wang and Z. Cai. Constrained Evolutionary Optimization by Means of $(\mu + \lambda)$-Differential Evolution and Improved Adaptive Trade-Off Model. *Evol. Comput.*, 19(2):249–285, 2011.
50. Y. Wang, Z. Cai, and Q. Zhang. Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters. *IEEE TEVC*, 15(1):55–66, 2011.
51. Y. Wang, D. Yin, S. Yang, and G. Sun. Global and Local Surrogate-Assisted Differential Evolution for Expensive Constrained Optimization Problems With Inequality Constraints. *IEEE Trans. Cyber.*, 49(5):1642–1656, 2019.
52. M. Weber, F. Neri, and V. Tirronen. Distributed differential evolution with explorative–exploitative population families. *GPEM*, 10(4):343–371, 2009.
53. M. Wormington, C. Panaccione, K. M. Matney, and D. K. Bowen. Characterization of structures from X-ray scattering data using genetic algorithms. *Phil. Trans. R. Soc. Lond. A*, 357(1761):2827–2848, 1999.
54. M. Zhabitsky and E. Zhabitskaya. Asynchronous Differential Evolution with Adaptive Correlation Matrix. In *GECCO*, pages 455–462, 2013.
55. J. Zhang and A. Sanderson. *Adaptive differential evolution: a robust approach to multimodal problem optimization*, volume 1. Springer, 2009.
56. J. Zhang and A. C. Sanderson. DE-AEC: A differential evolution algorithm based on adaptive evolution control. In *IEEE CEC*, pages 3824–3830. IEEE, 2007.
57. J. Zhang and A. C. Sanderson. JADE: Adaptive Differential Evolution With Optional External Archive. *IEEE TEVC*, 13(5):945–958, 2009.
58. K. Zielinski, P. Weitkemper, R. Laur, and K. D. Kammeyer. Parameter study for differential evolution using a power allocation problem including interference cancellation. In *IEEE CEC*, pages 1857–1864, 2006.