Turning high-dimensional optimization into computationally expensive optimization

Yang, Peng ; Tang, Ke; Yao, Xin

DOI:
10.1109/TEVC.2017.2672689

Turning High-dimensional Optimization into Computationally Expensive Optimization*

Peng Yang, Ke Tang, and Xin Yao†

February 16, 2017

Abstract

Divide-and-Conquer (DC) is conceptually well suited to deal with high-dimensional optimization problems by decomposing the original problem into multiple low-dimensional sub-problems, and tackling them separately. Nevertheless, the dimensionality mismatch between the original problem and sub-problems makes it non-trivial to precisely assess the quality of a candidate solution to a sub-problem, which has been a major hurdle for applying the idea of DC to non-separable high-dimensional optimization problems. In this paper, we suggest that searching a good solution to a sub-problem can be viewed as a computationally expensive problem and can be addressed with the aid of meta-models. As a result, a novel approach, namely Self-Evaluation Evolution (SEE) is proposed. Empirical studies have shown the advantages of SEE over 4 representative compared algorithms increase with the problem size on the CEC2010 large scale global optimization benchmark. The weakness of SEE is also analysed in the empirical studies.

1 Introduction

Evolutionary Algorithm (EAs) are powerful tools for complex optimization problems [1–4]. However, it has been frequently reported that the performance of existing EAs drops down dramatically as the dimensionality becomes high [5]. In the last decade, this issue has attracted lots of research interests [6]. Basic ideas include improving the search ability of existing EAs in high-dimensional solution space by re-balancing the exploration and exploitation [7–9] and simplifying high-dimensional problems for existing EAs via the Dimensionality Reduction techniques [10, 11] or the Divide-and-Conquer (DC) methodology [12, 13].

*This work has been accepted by IEEE Transactions on Evolutionary Computation on 03/02/2017.

†This work was supported in part by the National Natural Science Foundation of China under Grant 61329302 and Grant 61672478; in part by EPSRC under Grant EP/K001523/1 and EP/J017515/1, in part by the Royal Society Newton Advanced Fellowship under Grant NA150123, and SUSTech. Xin Yao was also supported by a Royal Society Wolfson Research Merit Award. Peng Yang and Ke Tang are with the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications, School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, Anhui 230027, China (e-mail: trevor@mail.ustc.edu.cn; ketang@ustc.edu.cn). Corresponding Author: Ke Tang. Xin Yao is with Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen 518055, China, and CERCIA, School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK (e-mail: x.yao@cs.bham.ac.uk)

In the context of evolutionary optimization, DC works by first dividing a high-dimensional problem into multiple exclusive low-dimensional sub-problems. Then, an existing EA is employed as the optimizer for each sub-problem. After that, the partial solutions, i.e., the solutions to the sub-problems, are merged to form the solution to the original problem. Since the partial solutions and the search space of the original problem are of different dimensionality, a partial solution cannot be naturally evaluated with respect to the original objective function. Such a mismatch in dimensionality induces one of the major challenges for applying DC to EAs.

One way to address the problem of dimensionality mismatch is to complement a partial solution to be a complete solution (to the original problem) before evaluating it with the original objective function. This type of works are ususally referred to as Cooperative Coevolution (CC) in the literature [14–16]. Notice that, the measured qualities of partial solutions largely depend on the selected complements, which may affect the rank of partial solutions and thereby the search biases. In case the sub-problems can be made independent of each other via some proper decomposition technique [13, 17–20], changes of complements will not perturb the rank of partial solutions [14, 20–23]. Unfortunately, no existing decomposition techniques are guaranteed to obtain the desired sub-problems. Besides, many problems are non-separable in essence. For this reason, the complements should be carefully chosen to correctly rank the partial solutions. However, to guarantee the correct rank, one has to exhaustively test all feasible samples in the whole solution space of the other sub-problems for each single partial solution [24], which is computationally prohibitive. In practice, existing CCs usually sample one or a few complements for each partial solution to alleviate the computational burden, while the performance can be significantly compromised when sub-problems are interdependent [13, 23, 25–27].

Since the dimensionality mismatch induces a computationally expensive problem, we propose in this paper an efficient meta-model technique to tackle it. Different from CC that complements partial solutions to match the dimensionality of the original objective function, we establish meta-models for each sub-problem. In this way, partial solutions to a sub-problem, without being complemented, can be directly compared via the corresponding meta-model. On this basis, a new approach, namely Self-Evaluation Evolution (SEE), is developed for high-dimensional optimization.

The remainder of the paper is organized as follows. In Section II, the existing works for high-dimensional optimization are briefly reviewed. Section III describes the challenge of the DC framework and proposes to view it as a computationally expensive problem. The proposed SEE is presented in details in Section IV and is empirically studied in Section V against 4 representatives of exiting works. All the 20 problems in the CEC'2010 large-scale optimization benchmark [28] are implemented with five different dimensionalities (from 1000D to 5000D) as the tested functions. Finally, conclusions will be drawn in Section VI.

2 Related Works

Generally, the major difficulty of EAs on high-dimensional optimization lies in that the solution space of a problem expands exponentially with the dimensionality. Such an expanded solution space quickly exceeds the search ability of existing EAs, and thus cannot be fully explored within a reasonable time budget.

The most straight-forward way might be to improve the search efficiency of existing EAs by either explicitly or implicitly re-balancing the exploration and exploita-

tion [7–9], which has been widely acknowledged as the key to a successful search [29]. However, such balance may vary from problem to problem, and may particularly be sensitive to the dimensionality, i.e., a refined computational resource allocation on a problem with a given dimensionality can be ineffective on the same problem with another dimensionality.

An alternative way is to simplify the original problem. Dimensionality Reduction is one type of such techniques that aims to project the original high-dimensional solution space onto lower dimensions where existing EAs are well-suited. Typical approaches in this category, e.g., Random Projection [10] and Random Embeddings [11], have obtained appealing performance in case the number of projected sub-spaces is sufficiently large to jointly represent the solution space or the solution space is embedded with effective low dimensions. However, to determine such lower dimensions appears to be non-trivial either in terms of the computational costs [10] or due to the unawareness of the problem structure [11].

Another way to simplify the problem is based on the DC methodology. DC deals with a high-dimensional optimization problem by first decomposing it into multiple exclusive low-dimensional sub-problems, and solves them separately. Since the low-dimensional sub-problems are expected to be solved more efficiently by existing EAs than the original problem, DC is conceptually well-suited to deal with the high-dimensional optimization problems. In the literature, the Cooperative Coevolution (CC) is a typical DC approach [14]. Given an appropriate sub-problem optimizer, CC works well on the so-called separable problems, for which the global optimal optimum can be found by optimizing one dimension at a time regardless of the values taken on the other dimensions [14, 20, 21]. If this condition does not hold, the performance of CC heavily relies on the decomposition method [13, 17–19, 23, 30], which aims to divide a high-dimensional problem in such a way that the global optimum can still be obtained by solving the sub-problems in a fully independent manner. In the past few years, a large variety of decomposition methods have been proposed [6, 13, 31]. Despite the performance enhancement brought by them, none of these methods are guaranteed to achieve the desired sub-problems. Meanwhile, a practical problem of interest may be fully non-separable such that the above-mentioned ideal decomposition does not even exist. Therefore, CC often faces a difficulty while dealing with interdependent sub-problems of correctly evaluating and ranking partial solutions in each sub-problem, which will be detailed in the next section. Unfortunately, it has been formally proven that performing such a task will consume prohibitive computational time [24]. Otherwise, CC would highly likely prematurely converge to a sub-optimum [15, 24, 32]. Some researchers tried to achieve a proper balance between the computational burden and the solution quality [25–27, 33], which, however, is another non-trivial task.

3 The Challenge of Divide-and-Conquer

In this section, we show that the challenge of DC mainly lies in the dimensionality mismatch between partial solutions and the objective function, where the partial solutions are difficult to be correctly ranked due to the overwhelming computational cost.

Before that, for clarity, the notations used in the rest of this paper will be introduced. Let $\mathbf{x}$ denote a given set of complete candidate solutions to the original objective function $\mathcal{F}$, $\mathbf{x}_i$ denote the i-th complete solution, and $\mathbf{x}_{1:N}$ denote the first N complete solutions in this set. Suppose a D-dimensional problem has been divided into M exclusive d_j-dimensional sub-problems ($\sum_{j=1}^{M} d_j = D$), then we denote the i-th

partial solution to the j-th sub-problem as $x_{i,j}$. Similarly, the first N partial solutions to the j-th sub-problem are denoted as $x_{1:N,j}$. In this case, a complete solution $\mathbf{x}_i$ can also be expressed as the stacked version of D partial solutions with the same i-index, i.e., $\mathbf{x}_{i,1:D}$. Accordingly, $x_{i,j}$ is also said to be the j-th component that belongs to the complete solution $\mathbf{x}_i$.

It is clear to see that the original D-dimensional objective function $\mathcal{F}$ cannot be used to directly evaluate the d_j-dimensional partial solutions in the j-th sub-problem, where $j = 1, 2, ..., M$. One straightforward way to address the dimensionality mismatch is to complement a partial solution to be a complete solution before evaluating it with the objective function. To complement a d_j-dimensional partial solution $x_{i,j}$, it is actually to fix the values for the decision variables involved in the other $M-1$ sub-problems. Hence, for each partial solution, there are $|\mathcal{X}|^{D-d_j}$ candidate complements available, where $\mathcal{X} \subseteq \mathbb{R}$ denotes a bounded range of each decision variable. It should be noted that, different complements will result in different objective function values to a partial solution. Thus, the rank of multiple partial solutions might also change over the complements. As the rank of partial solutions usually determines the search direction in the corresponding sub-problem, it is of importance to select a correct complement from $|\mathcal{X}|^{D-d_j}$ candidates for each partial solution so that they can be correctly ranked, which, however, is a non-trivial task.

Given a set of partial solutions, the correct ordering of them must be unique in terms of their real qualities in the solution space. Specifically, if a partial solution belongs to a good complete solution, it is said to be with good real quality, which is usually referred to as the building blocks hypothesis [34–38]. Hence, the correct rank of a set of partial solutions should be obtained by comparing the function values of the best complete solutions they belong to. More formally, in the j-th sub-problem, if the a-th partial solution $x_{a,j}$ is said to be better than the b-th partial solution $x_{b,j}$, the Eq.(1) must be satisfied (considering the minimization case).

$$\min_{\mathbf{x_c} \in S_\mathbf{c}} \mathcal{F}(x_{a,j}, \mathbf{x_c}) < \min_{\mathbf{x_c} \in S_\mathbf{c}} \mathcal{F}(x_{b,j}, \mathbf{x_c}), \tag{1}$$

where $\mathbf{x_c}$ denotes the candidate complements, $\mathbf{c} = [1, ..., j-1, j+1, ..., M]$ denotes all the sub-problems except the j-th one, $\mathcal{F}$ denotes the original objective function, and $S_\mathbf{c}$ denotes the corresponding complement space $\mathcal{X}^{D-d_j}$. Based on Eq.(1), each partial solution $x_{i,j}$ has its own correct complement, as defined in Eq.(2).

$$\mathbf{x}^*_\mathbf{c} = \arg\min_{\mathbf{x_c} \in S_\mathbf{c}} \mathcal{F}(x_{i,j}, \mathbf{x_c}). \tag{2}$$

Unfortunately, there is no way to guarantee the correct complement for a partial solution $x_{i,j}$, unless evaluating the combinations of $x_{i,j}$ with all possible complements in $\mathcal{X}^{D-d_j}$ by $\mathcal{F}$ and returning the best one, which is computationally overwhelming. As a result, the task of obtaining the correct rank of a set of partial solutions turns out to be computationally expensive. Existing DC based works, e.g., CC, aim to approximate the correct rank by heuristically sampling from the complement space.

Let us define a function $\mathcal{G}$ that can directly evaluate the real quality of a partial solution $x_{i,j}$ as Eq.(3).

$$\mathcal{G}(x_{i,j}) = \min_{\mathbf{x_c} \in S_\mathbf{c}} \mathcal{F}(x_{i,j}, \mathbf{x_c}). \tag{3}$$

In this case, the correct rank of a set of partial solution can be obtained by directly comparing their function values in terms of $\mathcal{G}$. Mathematically, given N partial solutions

Algorithm 1 SEE($\mathcal{F}$, T_{max}, λ, μ, ...)

Divide $\mathcal{F}$ into M exclusive sub-problems.
For $j = 1$ **to** M
  Initialize a meta-model $\mathcal{H}_j$.
  Initialize the search operators.
  Initialize μ parent partial solutions $x^p_{1:\mu,j}$ randomly.
EndFor
For $t = 1$ **to** T_{max}
  For $j = 1$ **to** M
    Generate λ offsprings $x^o_{1:\lambda,j}$ based on $x^p_{1:\mu,j}$.
    Rank $x^o_{1:\lambda,j}$ and $x^p_{1:\mu,j}$ in terms of $\mathcal{H}_j$.
    Update $x^p_{1:\mu,j}$.
    Collect the training data sampled from $\mathcal{F}$ for $\mathcal{H}_j$.
    Train $\mathcal{H}_j$.
    Adjust the j-th sub-problem optimizer.
  EndFor
EndFor
Output the best complete solution found so far.

$x_{1:N,j}$ to the j-th sub-problem, the correct rank of them is denoted as $\mathcal{R}$, and is defined as follows:

$$\mathcal{R}(x_{1:N,j}) = \underset{1\leq i\leq N}{\text{sort}}\, \mathcal{G}(x_{i,j}). \tag{4}$$

where sort() denotes a function that ranks the $x_{i:N,j}$ in an ascending order in terms of their qualities $\mathcal{G}(x_{i,j}), i = 1, 2, ..., N$.

To summarize, calculating $\mathcal{G}(x_{i,j})$ is computationally prohibitive, heuristically sampling the complement space in essence approximates $\mathcal{G}(x_{i,j})$. A natural alternative idea would be to directly build a rule or surrogate as the approximation of each $\mathcal{G}(x_{i,j})$ or even $\mathcal{R}(x_{1:N,j})$, which motivates the method proposed in Section IV.

4 Self-Evaluation Evolution

According to the discussions above, we regard the task of correctly ranking the partial solutions as a computationally expensive problem. In the literature, EAs handle the computationally expensive problems often with the aid of the meta-model (surrogate model) techniques [39, 40]. The underlying idea is to introduce a computationally less expensive function to approximate the original function. Inspired by that, we propose a novel DC based algorithm in this section, called the Self-Evaluation Evolution (SEE).

4.1 The framework of SEE

In SEE, the original objective function $\mathcal{F}$ is firstly divided into M sub-problems, each of which exclusively contains d_j decision variables. A meta-model is initialized for each sub-problem to approximate the correct rank $\mathcal{R}$, denoted as $\mathcal{H}_j$, $j = 1, 2, ..., M$. For each sub-problem, a sub-problem optimizer is constructed. The sub-problem optimizer can be any type of EAs. Without loss of generality, we consider a simple case

that the j-th sub-problem optimizer first uniformly randomly generates μ parent partial solutions $x^p_{1:\mu,j}$ at the initial stage, and then λ offspring partial solutions $x^o_{1:\lambda,j}$ are reproduced based on the parents at each iteration.[1] The offsprings $x^o_{1:\lambda,j}$ and parents $x^p_{1:\mu,j}$ are then ranked by the meta-model H_j, where the best μ partial solutions among them are kept as the new parents for the next iteration.

In addition to the evolution of sub-problems, there are two important steps in SEE. First, it is highly unlikely to obtain perfect meta-models that can always correctly rank the partial solutions at the initial stage. Hence, the meta-models should be adjusted during the optimization. Second, different stages in the search course may prefer different search operators. Therefore, the search operators also would better be adjusted during the search. Accordingly, the training data are required for the adjustments in both steps, which are usually the samples of the original solution space. For illustration, the framework of SEE is given in Algorithm 1.

4.2 An efficient meta-model for SEE

In the literature, different kinds of meta-models have been extensively studied [39,40]. Most of them do not explicitly rely on the problem structure. From this perspective, those meta-models can be expected to work well in SEE. Nevertheless, when considering the computational efficiency, those existing sophisticated meta-models may not be the best choices in SEE. To be specific, the effectiveness of traditional EAs are mostly observed on problems whose dimensionalities do not exceed 50. To solve each sub-problem well, the dimensionality of sub-problems should be manageable for the sub-problem optimizers. Comparatively, the dimensionality of a high-dimensional optimization problem usually exceeds 1000, and can even be billions [41]. As a result, there can be large numbers of meta-models to train at each iteration of the optimization, which will induce huge computational costs by using the commonly used meta-models like Support Vector Machine [42], Neural Network [43], and Gaussian Process [44].

Considering this, we specially design a very simple yet efficient rank-based meta-model for SEE so that the computational cost of the meta-model is kept very low. Different from existing meta-models that aim to approximate the global landscape of the solution space, the proposed meta-model only approximates the pairwise rank within the *local area* of each 1-dimensional sub-space. With such a simpler goal, building a good meta-model can be much easier and more efficient.

To describe the meta-model clearly, let us consider the rank between a parent partial solution $x^p_{1,j}$ and its offspring $x^o_{1,j}$, in the j-th sub-problem. In the 1-dimensional case, $x^o_{1,j}$ will be either smaller or larger than $x^p_{1,j}$ according to their scalar decision variable values. If the landscape changes either monotonously ascending or descending, the rank of $x^p_{1,j}$ and $x^o_{1,j}$ can be obtained directly without explicitly calculating the real qualities of them, i.e., $\mathcal{G}(x^p_{1,j})$ and $\mathcal{G}(x^o_{1,j})$. That is, for the minimization case, $\mathcal{G}(x^o_{1,j})$ is better than $\mathcal{G}(x^p_{1,j})$, as long as $x^o_{1,j}$ is smaller than $x^p_{1,j}$ while the landscape there goes ascending, or $x^o_{1,j}$ is larger than $x^p_{1,j}$ while the landscape there goes descending. Otherwise, $\mathcal{G}(x^o_{1,j})$ is worse than $\mathcal{G}(x^p_{1,j})$. Although the global landscape does not appear to be monotonous, especially for multi-modal problems, it is still optimistic to assume the local landscape changes monotonously within the small interval between $x^p_{1,j}$ and $x^o_{1,j}$, as long as they are generated via some local search operator. Then the meta-model needs to learn the likelihoods that the landscape goes ascending on the smaller side of

[1] The notations of parents and offsprings are distinguished with the superscript, where p is for parents and o is for offsprings.

$x^p_{1,j}$ and the landscape goes descending on the larger side of $x^p_{1,j}$. Formally, these two likelihoods are respectively denoted as $PS_{1,j}$ and $PL_{1,j}$ and defined as probabilities as below:

$$\begin{aligned} PS_{1,j} &= p(\mathcal{G}(x^o_{1,j}) < \mathcal{G}(x^p_{1,j}) | x^o_{1,j} < x^p_{1,j}) \\ PL_{1,j} &= p(\mathcal{G}(x^o_{1,j}) < \mathcal{G}(x^p_{1,j}) | x^o_{1,j} > x^p_{1,j}). \end{aligned} \tag{5}$$

In case $x^o_{1,j}$ equals to $x^p_{1,j}$, which is highly unlikely, $x^o_{1,j}$ is re-sampled. According to Eq.(5), given a pair of partial solutions $x^p_{1,j}$ and $x^o_{1,j}$, it is predictable that $\mathcal{G}(x^o_{1,j})$ is better than $\mathcal{G}(x^p_{1,j})$ with probability $PS_{1,j}$ (or $PL_{1,j}$) if $x^o_{1,j}$ is smaller (or larger) than $x^p_{1,j}$. Therefore, the meta-model $\mathcal{H}_j$ is defined as follows:

$$\begin{cases} \mathcal{G}(x^o_{1,j}) \leq \mathcal{G}(x^p_{1,j}) & \textbf{if} \quad (x^o_{1,j} < x^p_{1,j} \wedge PS_{1,j} \geq r) \\ & \qquad \vee (x^o_{1,j} > x^p_{1,j} \wedge PL_{1,j} \geq r) \\ \mathcal{G}(x^o_{1,j}) > \mathcal{G}(x^p_{1,j}) & \textbf{otherwise} \end{cases} \tag{6}$$

where r indicates a function that returns a random variable uniformly sampled in the range of $[0, 1]$.

In Eq.(6), $PS_{1,j}$ and $PL_{1,j}$ are simply initialized as 1.00, indicating that the initialized parent partial solution $x^p_{1,j}$ is assumed to be worse than any of its neighbours. $PS_{1,j}$ and $PL_{1,j}$ will vary during the search since the parent $x^p_{1,j}$ moves during the search course and the corresponding local landscape will be different. To train the meta-model, i.e., adjusting $PS_{1,j}$ and $PL_{1,j}$, we modify a well-known parameters tuning technique, i.e., the 1/5 successful rule [45], as follows:

$$\begin{aligned} PS_{1,j} &= PS_{1,j} \cdot \exp^{\frac{1}{\sqrt{2}}} [\mathbb{I}_{x^o_{1,j} < x^p_{1,j}} \cdot (\mathbb{I}_{\mathcal{F}(\mathbf{x}^p_1) \geq \mathcal{F}(\mathbf{x}^o_1)} - \frac{1}{5})] \\ PL_{1,j} &= PL_{1,j} \cdot \exp^{\frac{1}{\sqrt{2}}} [\mathbb{I}_{x^o_{1,j} > x^p_{1,j}} \cdot (\mathbb{I}_{\mathcal{F}(\mathbf{x}^p_1) \geq \mathcal{F}(\mathbf{x}^o_1)} - \frac{1}{5})] \end{aligned} \tag{7}$$

where $\mathbb{I}_a$ is an indicator function that returns 1 if a is true and 0 otherwise. The above equations state that, if this prediction of the meta-model is correct, the corresponding probability will be enlarged. Otherwise, the corresponding probability will be reduced. To check whether the prediction is correct, we should have compared $\mathcal{G}(x^o_{1,j})$ and $\mathcal{G}(x^p_{1,j})$ in Eq.(7), i.e., $\mathbb{I}_{\mathcal{G}(x^p_{1,j}) \geq \mathcal{G}(x^p_{1,j})}$, which is practically impossible. Instead, we compare $\mathcal{F}(\mathbf{x}^o_1)$ with $\mathcal{F}(\mathbf{x}^p_1)$ to check whether the prediction is correct, where $\mathcal{F}(\mathbf{x}^o_1)$ and $\mathcal{F}(\mathbf{x}^p_1)$ respectively denote the objective function values of the complete solutions $\mathbf{x}^o_1$ and $\mathbf{x}^p_1$.[2] Notice that, Eq.(7) adjusts $PS_{1,j}$ and $PL_{1,j}$ gradually with a constant factor. Eq.(7) works based on the assumption that the local landscape changes *smoothly*. More specifically, only when the landscapes in local areas change smoothly, the meta-model in previous iterations can be helpful for the current pairwise comparison. It again requires the sub-problem optimizer to be local search.

[2]As a reminder, $\mathbf{x}^o_1$ is the complete solution that $x^o_{1,j}$ belongs to, and can be expressed as $[x^o_{1,1}, x^o_{1,2}, ..., x^o_{1,D}]$. Similarly, $\mathbf{x}^p_1$ denotes the complete solution that $x^p_{1,j}$ belongs to, and can be expressed as $[x^p_{1,1}, x^p_{1,2}, ..., x^p_{1,D}]$.

4.3 Detailed steps of SEE

As highlighted above, to facilitate the proposed meta-model better, the sub-problems optimizers should be local search. By considering only one pair of parent and offspring in each sub-problem, as described above, one can easily establish a concrete sub-problem optimizer, which is a (1+1)-EA [45]. This idea is further generalized to the (1+λ)-EA based sub-problem optimizer for SEE. To be specific, SEE first randomly initializes a single parent partial solution $x^p_{1,j}$ in each j-th sub-problem. At each iteration, λ offsprings $x^o_{1:\lambda,j}$ are generated based on $x^p_{1,j}$ with different local search operators. And only the best one among the parent $x^p_{1,j}$ and λ offsprings $x^o_{1:\lambda,j}$ is preserved. As described above, there are two important issues in SEE that should be detailed: the local search operators and the update of the parent partial solution.

In this work, both the Gaussian mutation operator and the Cauchy mutation operator are employed as the local search operators. Specifically, in each sub-problem, n new partial solutions ($1 \leq n \leq \lambda$) are generated in terms of the Gaussian mutation, while the rest are generated by the Cauchy mutation. Given a parent solution $x^p_{1,j}$ in the j-th sub-problem, the Gaussian mutation operator generates an offspring partial solution $x^o_{i,j}$, $i = 1, 2, ..., n$, using Eq.(8):

$$x^o_{i,j} = x^p_{1,j} + \sigma_{i,j} \cdot \mathcal{N}(0, 1), \tag{8}$$

where $\mathcal{N}(0, 1)$ denotes a Gaussian random variable with zero mean and standard deviation 1. Analogously, the Cauchy mutation operator generates an offspring partial solution $x^o_{i,j}$, $i = n + 1, n + 2, ..., \lambda$, using Eq.(9):

$$x^o_{i,j} = x^p_{1,j} + \sigma_{i,j} \cdot \mathcal{C}(0, 1), \tag{9}$$

where $\mathcal{C}(0, 1)$ denotes a random variable subject to the standard Cauchy distribution. In general, the Gaussian mutation has "narrower" probability distribution than the Cauchy mutation. This means that the Gaussian mutation may be more compatible with the proposed meta-model while it searches slower than the Cauchy mutation [46, 47]. Therefore, the parameter n is set to balance the effectiveness of the proposed meta-model and the search efficiency. The $\sigma_{i,j}$ in Eqs.(8)-(9) denotes the corresponding search step-size. Generally, the value of $\sigma_{i,j}$ can be adapted during the search and may also vary over sub-problems optimizers. To keep it simple, all $\sigma_{i,j}$ are initialized with a same value, i.e., 1.00. Then, each $\sigma_{i,j}$ is adapted for every iteration again in terms of the 1/5 successful rule [45], as given in Eq.(10):

$$\sigma_{i,j} = \sigma_{i,j} \cdot \exp^{\frac{1}{\sqrt{2}}[\mathbb{I}_{x^o_{i,j} \neq x^p_{1,j}} \cdot (\mathbb{I}_{\mathcal{F}(\mathbf{x}^p_1) \geq \mathcal{F}(\mathbf{x}^o_i)} - \frac{1}{5})]} \tag{10}$$

Similar to Eq.(7), Eq.(10) states that, given $x^o_{i,j}$ is different from $x^p_{1,j}$, if the search process has successfully made a progress, the corresponding search step-size will be enlarged. Otherwise, the corresponding search step-size will be reduced.

For the update of the parent, given the accuracy of the proposed meta-model implicitly relies on the distance between the parent and each offspring, one meta-model may not work well for all offsprings, as the offsprings usually locate differently. Therefore, λ meta-models, denoted as $\mathcal{H}_{1:\lambda,j}$, are built for each sub-problem. In a sub-problem, each of the λ meta-models is used to compare a unique offspring and the parent partial solution. Although such a strategy is expected to more accurately tell whether an

offspring is better than the parent, it cannot identify the best offspring partial solution. Thus, after finishing the lambda pairwise comparisons, those offsprings that are predicted as better than the parent will be further compared in terms of the objective functions values of their complete solutions, so as to find the best one out of them.

Technically, in the j-th sub-problem, each i-th offspring $x^o_{i,j}$ is pairwise compared with $x^p_{1,j}$ via $\mathcal{H}_{i,j}$. If $x^o_{i,j}$ is predicted to be worse than $x^p_{1,j}$ according to Eq.(6), we discard it by letting let $x^o_{i,j} = x^p_{1,j}$, i.e., only the offsprings (to a sub-problem) predicted to be better than the parent $x^p_{1,j}$ will be kept. After executing this step for all sub-problems, we evaluate the resultant λ offspring complete solutions $\mathbf{x}^o_{1:\lambda}$ with the objective function $\mathcal{F}$, and each partial solution $x^o_{i,j}$ shares the same objective function value with that of its corresponding complete solution, i.e., $\mathcal{F}(\mathbf{x}^o_i)$. After that, in each j-th sub-problem, $x^p_{1,j}$ is updated with its best offspring in terms of the objective function values. Actually, for brevity, it is identical to directly replace the parent complete solution with the best offspring complete solution. That is, let $\mathbf{x}^p_1 = \text{argmin}_{\mathbf{x}^o_i}\mathcal{F}(\mathbf{x}^o_i)$, if $\mathcal{F}(\mathbf{x}^p_1) > \min\limits_{1\leq i\leq\lambda} \mathcal{F}(\mathbf{x}^o_i)$. Otherwise, keep $\mathbf{x}^p_1$ unchanged.

To summarize, the detailed pseudo-code of SEE is presented in Algorithm 2 for illustration[3]. As seen that, SEE generates new partial solutions at steps 11-16. After that, the partial solutions are ranked in terms of the meta-models at step 18 and step 20. Then according to the predictions of the meta-models, the offsprings worse than the parent are reset to the value of the parent, as seen in step 19 and step 21. The $\mathbf{x}^p_1$ is updated at steps 25-27. The training data are also collected while evaluating the complete solutions. Lastly, the search step-sizes and the parameters of the meta-models are updated at steps 28-34.

4.4 Different behaviors between SEE and CC

As both SEE and CC are motivated by the divide-and-conquer idea, the potential advantage of SEE over CC worth a few more discussion, from the following two aspects.

First, SEE consumes much less computational costs than CC based works. Suppose there are λ offsprings generated in each sub-problem at each iteration, the proposed meta-model costs $D \cdot \lambda$ times of Eqs.(6)-(7) and λ times of the objective function evaluations $\mathcal{F}$. Since Eqs.(6)-(7) are simple scalar calculations, it can be computationally much cheaper in comparison with the objective function evaluations. On the contrary, as the counterpart under the DC framework, traditional CC costs at least $\lambda \cdot M$ objective function evaluations at each iteration, where M is the number of sub-problems in a CC based work. Hence, although the proposed meta-model still cannot guarantee the correct rank of partial solutions, it gives an efficient way to produce an approximated rank, which is around M times faster than that of CC based works.

Besides, SEE may be less sensitive to the separability of problems than the CC based works. The reason is that the variables in different sub-problems of SEE are still possible to be evolved together as they are evolved in parallel, while in CC based works, the variables in different sub-problems will never be evolved together as they are evolved sequentially. In consequence, once the interdependent variables are grouped into different sub-problems, CC based work will not be able to optimize all of them in the same iteration.

[3]The Java source code can be seen at: `http://staff.ustc.edu.cn/~ketang/`

Table 1: 5 groups of problems with different configurations.

	Group 1	**Group 2**	**Group 3**	**Group 4**	**Group 5**
D	1000	2000	3000	4000	5000
m	50	100	150	200	250
T_{max}	6e5	1.2e6	1.8e6	2.4e6	3e6

5 Empirical Studies

In the experimental studies, SEE is first compared with 4 state-of-the-art algorithms on 20 problems with 5 different dimensionalities. Then how the proposed meta-model impacts the performance of SEE is also investigated.

5.1 Experiment protocol

The CEC'2010 large-scale optimization benchmark [28] is a widely used test suite to assess the performance high-dimensional optimization algorithms [7, 10, 13, 18, 19, 23, 48]. This test suite consists of four types of high-dimensional problems in terms of different degrees of separability. In each type of problems, both uni-modal problems and multi-modal problems are included. Although this benchmark originally consists of 20 continuous problems with 1000 variables, it is flexible to adjust the dimensionality to see how the tested algorithms scale on those 20 problems. Given the above features, the CEC'2010 large-scale optimization benchmark is adopted in our empirical studies. The test suite has two parameters that could be used to control the complexity of the problems. The first parameter is the dimensionality of each problem, i.e., D. The second parameter is the size of each group of interdependent variables in each problem, denoted as m. They were initially set to 1000 and 50, respectively. In order to see how SEE scales on those problems in comparison with the other 4 algorithms, we modified these two parameters to produce 5 groups of problems as shown in Table I.

The time budget, i.e., total number of function evaluations, is set to 6e5, 1.2e6, 1.8e6, 2.4e6, and 3e6, respectively for each group of problems. All the compared algorithms terminate when the time budget runs out. With this linearly increased time budget, it is easy to see how an algorithm scale on a problem. If an algorithm obtains the final solutions with the same quality on one problem with different dimensionalities, it is said to scale linearly with the dimensionality (see Figs. 1-2). The quality of the final solution is measured with function errors, i.e., the difference between the objective function value of the output solution and that of the optimal solution to the problem (which are known for these benchmark problems as 0.0). In other words, the smaller the function error is, the better the solution will be. If the function error is smaller than 1e-13, the output solution will be regarded as the global optimum. All the compared algorithms are repeated on each problem for 20 runs. The function errors of the corresponding solutions were recorded and averaged over 20 runs. The averaged function errors of all 5 algorithms on 20 problems with 5 different dimensionalities are respectively shown in Tables II-VI, together with the standard deviations. The best result on each problem is marked in gray. The two-sided Wilcoxon rank-sum test at a 0.05 significance level is also conducted to see whether the performances of two algorithms are statistically significantly different. The scalability curves of each algorithm on 5 groups of problems are shown in Figs.1-2, where the slower the curve of

Table 2: The average and standard deviation of function errors on the 1000-dimensional CEC'2010 large-scale global optimization benchmark.

Algo.		Chain	RPM	DECC-D	DECC-DG	SEE
F_1	Mean	3.09e+04	1.68e+07	3.33e+02	7.97e+06	6.99e-11
	Std	1.03e+04	1.77e+06	2.41e+01	4.93e+06	1.98e-11
F_2	Mean	2.74e+03	7.25e+02	3.13e+03	4.62e+03	8.77e+03
	Std	1.11e+02	1.94e+01	6.99e+01	1.67e+02	2.99e+02
F_3	Mean	3.82e+00	7.77e-05	6.54e-03	1.70e+01	1.99e+01
	Std	2.32e-02	3.67e-06	2.57e-04	4.14e-01	1.61e-02
F_4	Mean	6.08e+11	2.64e+12	3.02e+13	6.86e+13	2.58e+11
	Std	4.48e+10	5.90e+11	7.64e+12	1.21e+13	6.41e+10
F_5	Mean	8.50e+07	3.05e+08	2.68e+08	2.39e+08	5.85e+08
	Std	2.02e+07	1.17e+07	7.57e+07	2.42e+07	1.37e+08
F_6	Mean	2.13e+01	9.09e+01	1.00e+06	1.69e+01	1.99e+07
	Std	5.88e-02	2.91e+00	4.49e+06	5.23e-01	6.01e+04
F_7	Mean	5.68e+06	1.49e+09	1.76e+09	1.29e+09	3.14e-02
	Std	3.49e+05	4.38e+08	7.41e+08	4.50e+08	1.56e-02
F_8	Mean	1.33e+07	4.59e+07	1.30e+08	6.21e+07	1.82e+06
	Std	5.87e+06	3.71e+05	5.54e+07	2.72e+07	2.11e+06
F_9	Mean	7.14e+07	4.56e+07	4.52e+08	8.27e+08	2.67e+07
	Std	4.82e+06	1.74e+06	4.71e+07	6.95e+07	2.13e+06
F_{10}	Mean	3.00e+03	7.24e+02	1.33e+04	8.60e+03	1.27e+04
	Std	1.57e+02	1.08e+01	3.02e+02	1.13e+02	5.06e+02
F_{11}	Mean	4.97e+01	1.57e-04	3.74e+00	1.67e+01	2.19e+02
	Std	1.91e+01	7.21e-06	1.39e+01	3.78e-01	2.44e-01
F_{12}	Mean	4.75e+02	6.12e+02	4.82e+06	3.29e+05	2.60e+02
	Std	2.86e+01	2.37e+01	2.30e+05	1.57e+04	4.81e+01
F_{13}	Mean	9.01e+03	3.26e+04	5.75e+03	2.60e+10	7.12e+02
	Std	6.68e+03	8.50e+03	5.14e+03	4.32e+09	2.61e+02
F_{14}	Mean	1.44e+08	1.65e+08	1.52e+09	2.34e+09	9.88e+07
	Std	5.01e+06	2.72e+06	1.22e+08	1.10e+08	7.61e+06
F_{15}	Mean	5.43e+03	7.36e+02	1.63e+04	7.85e+03	1.50e+04
	Std	3.57e+03	1.64e+01	3.62e+02	8.90e+01	3.10e+02
F_{16}	Mean	1.13e+02	1.64e-04	3.37e+01	2.38e+01	3.97e+02
	Std	2.85e+01	8.40e-06	8.91e+01	1.05e+00	2.92e-01
F_{17}	Mean	2.73e+04	1.79e+04	8.39e+06	7.70e+05	7.40e+03
	Std	9.27e+02	4.35e+03	5.39e+05	2.85e+04	9.59e+02
F_{18}	Mean	1.32e+04	2.48e+04	3.02e+04	3.98e+11	3.14e+03
	Std	8.37e+03	3.63e+03	7.08e+03	2.79e+10	1.00e+03
F_{19}	Mean	1.38e+06	3.26e+06	2.22e+07	3.66e+06	7.13e+05
	Std	4.00e+04	6.80e+03	1.68e+06	2.16e+05	4.24e+04
F_{20}	Mean	1.12e+03	1.64e+03	5.75e+03	6.48e+10	1.43e+03
	Std	7.22e+01	1.42e+03	4.65e+02	8.50e+09	1.85e+02
w-d-l		11-1-8	11-0-9	14-0-6	12-0-8	-

Table 3: The average and standard deviation of function errors on the 2000-dimensional CEC'2010 large-scale global optimization benchmark.

Algo.		Chain	RPM	DECC-D	DECC-DG	SEE
F_1	Mean	7.39e+05	4.82e+09	1.34e+03	2.28e+08	2.19e-10
	Std	9.50e+04	2.04e+08	1.05e+02	1.07e+08	3.51e-11
F_2	Mean	6.66e+03	4.76e+03	6.29e+03	1.27e+04	1.81e+04
	Std	2.62e+02	2.09e+02	1.15e+02	2.50e+02	4.93e+02
F_3	Mean	7.57e+00	1.03e+01	9.74e-03	1.92e+01	1.99e+01
	Std	3.13e-01	3.49e-01	3.56e-04	1.23e-01	1.02e-02
F_4	Mean	8.07e+13	1.07e+15	7.08e+15	1.07e+16	4.89e+11
	Std	5.11e+13	2.28e+14	1.82e+15	1.90e+15	1.10e+11
F_5	Mean	4.34e+08	8.79e+07	1.20e+09	9.81e+08	1.23e+09
	Std	1.01e+08	1.67e+07	1.49e+08	9.79e+07	2.16e+08
F_6	Mean	2.01e+07	2.00e+07	2.09e+07	2.13e+07	1.99e+07
	Std	9.86e+04	2.58e+04	7.57e+04	2.06e+04	6.40e+04
F_7	Mean	1.17e+07	9.90e+08	1.94e+10	3.46e+10	7.33e+00
	Std	3.51e+05	2.67e+08	2.94e+09	3.78e+09	2.97e+00
F_8	Mean	7.47e+07	1.18e+08	1.77e+08	3.20e+08	9.99e+05
	Std	6.83e+07	5.73e+07	5.61e+07	5.54e+08	1.77e+06
F_9	Mean	1.10e+10	2.24e+11	1.25e+11	2.61e+11	6.95e+07
	Std	9.91e+08	7.90e+09	7.85e+09	1.51e+10	4.97e+06
F_{10}	Mean	2.54e+04	2.85e+04	2.74e+04	2.36e+04	2.49e+04
	Std	3.14e+03	9.19e+02	1.07e+03	3.54e+02	4.33e+02
F_{11}	Mean	2.23e+02	2.24e+02	2.37e+02	2.33e+02	2.19e+02
	Std	5.13e-01	2.04e-01	2.79e-01	3.09e-01	1.22e-01
F_{12}	Mean	2.26e+04	4.11e+06	1.24e+07	1.28e+06	1.68e+04
	Std	8.79e+02	9.62e+04	4.79e+05	4.39e+04	2.49e+03
F_{13}	Mean	1.50e+04	6.38e+06	1.08e+04	2.66e+10	1.43e+03
	Std	4.04e+03	8.30e+05	5.21e+03	3.98e+09	3.27e+02
F_{14}	Mean	1.95e+10	3.89e+11	2.38e+11	8.59e+11	2.61e+08
	Std	1.39e+09	1.85e+10	1.40e+10	5.40e+10	1.16e+07
F_{15}	Mean	3.70e+04	7.28e+04	3.46e+04	2.86e+04	3.08e+04
	Std	3.07e+03	3.14e+03	7.69e+02	5.71e+02	8.11e+02
F_{16}	Mean	4.04e+02	4.09e+02	4.32e+02	4.27e+02	3.98e+02
	Std	1.00e+00	4.28e-01	1.32e-01	1.70e-01	2.56e-01
F_{17}	Mean	3.15e+05	4.78e+06	2.55e+07	2.85e+06	1.59e+05
	Std	1.52e+04	1.17e+05	1.25e+06	6.44e+04	1.15e+04
F_{18}	Mean	3.97e+03	2.99e+09	2.38e+04	1.06e+12	4.65e+03
	Std	1.36e+03	6.69e+08	5.40e+03	4.64e+10	1.09e+03
F_{19}	Mean	4.55e+06	1.32e+07	6.15e+07	1.06e+07	2.61e+06
	Std	1.31e+05	6.99e+05	4.27e+06	5.01e+05	1.15e+05
F_{20}	Mean	2.33e+03	3.18e+09	7.20e+03	2.98e+12	3.19e+03
	Std	1.56e+02	6.65e+08	3.84e+02	1.74e+11	2.25e+02
w-d-l		14-1-5	17-0-3	17-1-2	15-0-5	-

Table 4: The average and standard deviation of function errors on the 3000-dimensional CEC'2010 large-scale global optimization benchmark.

Algo.		Chain	RPM	DECC-D	DECC-DG	SEE
F_1	Mean	3.17e+06	1.07e+10	1.92e+03	1.15e+09	6.37e-10
	Std	3.41e+05	3.73e+08	1.37e+02	4.92e+08	9.64e-11
F_2	Mean	1.10e+04	1.10e+04	9.39e+03	2.23e+04	2.74e+04
	Std	3.81e+02	3.39e+02	7.61e+01	5.18e+02	6.19e+02
F_3	Mean	1.05e+01	1.42e+01	9.65e-03	1.97e+01	1.99e+01
	Std	3.95e-01	2.18e-01	3.32e-04	8.48e-02	1.22e-02
F_4	Mean	1.36e+14	3.24e+15	1.58e+16	1.75e+16	6.25e+11
	Std	5.20e+13	3.70e+14	2.86e+15	3.23e+15	1.56e+11
F_5	Mean	9.73e+08	2.57e+08	2.48e+09	2.43e+09	2.03e+09
	Std	1.90e+08	2.98e+07	3.01e+08	2.08e+08	2.78e+08
F_6	Mean	2.01e+07	2.01e+07	2.10e+07	2.14e+07	1.99e+07
	Std	9.25e+04	3.16e+04	3.18e+04	1.72e+04	3.96e+04
F_7	Mean	1.74e+07	1.79e+10	4.01e+10	8.33e+10	4.21e+02
	Std	4.10e+05	2.80e+09	7.75e+09	7.64e+09	2.01e+02
F_8	Mean	1.17e+08	1.96e+09	1.85e+08	2.70e+08	8.63e+05
	Std	2.83e+07	1.80e+09	5.04e+07	6.69e+07	1.63e+06
F_9	Mean	2.38e+10	6.02e+11	2.71e+11	6.08e+11	1.33e+08
	Std	1.85e+09	1.99e+10	1.60e+10	5.09e+10	1.02e+07
F_{10}	Mean	5.02e+04	7.17e+04	4.65e+04	4.22e+04	3.78e+04
	Std	4.96e+03	2.81e+03	2.14e+03	8.07e+02	7.06e+02
F_{11}	Mean	2.23e+02	2.24e+02	2.37e+02	2.34e+02	2.19e+02
	Std	6.23e-01	2.01e-01	2.07e-01	1.46e-01	2.04e-01
F_{12}	Mean	1.18e+05	8.50e+06	2.30e+07	2.64e+06	9.34e+04
	Std	5.06e+03	1.79e+05	9.56e+05	6.71e+04	9.15e+03
F_{13}	Mean	4.62e+04	2.30e+08	5.91e+03	1.70e+11	1.97e+03
	Std	7.46e+03	7.29e+07	2.87e+03	1.66e+10	3.98e+02
F_{14}	Mean	5.17e+10	1.48e+12	5.99e+11	4.06e+12	4.91e+08
	Std	3.38e+09	5.27e+10	2.64e+10	3.28e+11	6.25e+07
F_{15}	Mean	1.14e+05	2.24e+05	6.01e+04	5.77e+04	4.64e+04
	Std	2.99e+04	5.80e+03	1.59e+03	1.07e+03	1.14e+03
F_{16}	Mean	4.03e+02	4.11e+02	4.33e+02	4.28e+02	3.98e+02
	Std	3.17e+00	2.15e-01	3.66e-01	1.49e-01	2.43e-01
F_{17}	Mean	9.65e+05	1.05e+07	4.74e+07	5.49e+06	5.74e+05
	Std	8.76e+04	3.63e+05	2.44e+06	1.13e+05	3.21e+04
F_{18}	Mean	3.63e+03	3.98e+10	2.64e+04	2.51e+12	6.57e+03
	Std	1.87e+02	6.49e+09	7.07e+03	1.07e+11	6.88e+02
F_{19}	Mean	8.91e+06	3.04e+07	1.15e+08	1.97e+07	5.16e+06
	Std	3.82e+05	1.61e+06	9.68e+06	5.06e+05	1.51e+05
F_{20}	Mean	3.62e+03	4.09e+10	1.11e+04	5.21e+12	5.10e+03
	Std	1.66e+02	2.82e+09	1.26e+03	5.94e+10	3.07e+02
w-d-l		15-0-5	17-0-3	18-0-2	18-0-2	-

Table 5: The average and standard deviation of function errors on the 4000-dimensional CEC'2010 large-scale global optimization benchmark.

Algo.		Chain	RPM	DECC-D	DECC-DG	SEE
F_1	Mean	8.14e+06	2.05e+10	2.45e+03	5.60e+10	1.55e-09
	Std	5.34e+05	8.68e+08	1.15e+02	3.70e+09	2.66e-10
F_2	Mean	1.53e+04	1.91e+04	1.25e+04	3.20e+04	3.72e+04
	Std	3.59e+02	4.10e+02	1.64e+02	3.82e+02	7.59e+02
F_3	Mean	1.32e+01	1.61e+01	9.33e-03	1.97e+01	1.99e+01
	Std	8.13e-01	1.98e-01	2.51e-04	4.03e-02	9.59e-03
F_4	Mean	2.25e+14	7.11e+15	2.73e+16	1.21e+16	1.16e+12
	Std	5.67e+13	9.59e+14	5.29e+15	1.97e+15	2.74e+11
F_5	Mean	1.61e+09	5.12e+08	4.15e+09	4.11e+09	2.85e+09
	Std	2.70e+08	4.14e+07	5.12e+08	4.00e+08	2.70e+08
F_6	Mean	2.01e+07	2.01e+07	2.10e+07	2.14e+07	1.99e+07
	Std	6.83e+04	2.35e+04	5.88e+04	1.42e+04	4.90e+04
F_7	Mean	2.34e+07	6.94e+10	8.38e+10	2.86e+11	1.02e+04
	Std	4.48e+05	8.66e+09	1.14e+10	2.69e+10	3.86e+03
F_8	Mean	1.56e+08	2.09e+08	2.31e+08	4.91e+08	1.20e+06
	Std	1.30e+07	5.34e+05	4.21e+07	7.19e+07	1.84e+06
F_9	Mean	3.86e+10	1.21e+12	4.74e+11	9.56e+11	2.01e+08
	Std	3.89e+09	4.24e+10	1.86e+10	7.16e+10	1.12e+07
F_{10}	Mean	8.04e+04	1.50e+05	6.69e+04	6.17e+04	5.03e+04
	Std	4.92e+03	5.41e+03	3.44e+03	1.65e+03	1.11e+03
F_{11}	Mean	2.23e+02	2.25e+02	2.38e+02	2.35e+02	2.19e+02
	Std	5.83e-01	1.73e-01	1.58e-01	9.15e-02	1.06e-01
F_{12}	Mean	3.28e+05	1.40e+07	3.47e+07	3.96e+06	2.49e+05
	Std	5.24e+03	1.86e+05	1.37e+06	9.62e+04	1.63e+04
F_{13}	Mean	6.06e+04	2.27e+09	8.36e+03	2.00e+11	2.90e+03
	Std	6.53e+03	2.89e+08	3.46e+03	1.44e+10	5.17e+02
F_{14}	Mean	9.40e+10	3.76e+12	1.03e+12	2.54e+12	7.67e+08
	Std	7.19e+09	8.49e+10	3.70e+10	1.30e+11	4.31e+07
F_{15}	Mean	2.98e+05	1.26e+06	8.40e+04	9.47e+04	6.14e+04
	Std	3.44e+04	1.59e+05	1.63e+03	1.51e+03	1.18e+03
F_{16}	Mean	4.03e+02	4.17e+02	4.33e+02	4.28e+02	3.98e+02
	Std	1.75e+00	5.83e+00	5.29e-01	1.29e-01	2.51e-01
F_{17}	Mean	1.85e+06	2.74e+07	7.50e+07	8.63e+06	1.22e+06
	Std	2.88e+05	3.64e+05	3.66e+06	1.84e+05	4.80e+04
F_{18}	Mean	6.05e+03	9.29e+11	2.54e+04	1.19e+12	9.03e+03
	Std	2.80e+02	8.46e+10	5.44e+03	6.71e+10	9.15e+02
F_{19}	Mean	1.40e+07	7.64e+07	1.58e+08	2.85e+07	8.28e+06
	Std	8.05e+05	3.81e+06	6.25e+06	5.33e+05	3.32e+05
F_{20}	Mean	4.53e+03	4.65e+11	1.65e+04	5.57e+12	7.17e+03
	Std	1.60e+02	9.91e+09	2.79e+03	2.48e+11	3.65e+02
w-d-l		15-0-5	17-0-3	18-0-2	18-0-2	-

Table 6: The average and standard deviation of function errors on the 5000-dimensional CEC'2010 large-scale global optimization benchmark.

Algo.		Chain	RPM	DECC-D	DECC-DG	SEE
F_1	Mean	1.60e+07	6.72e+10	3.09e+03	6.72e+10	3.44e-09
	Std	1.47e+06	2.77e+09	1.64e+02	2.68e+09	3.95e-10
F_2	Mean	1.99e+04	4.39e+04	1.57e+04	4.39e+04	4.64e+04
	Std	5.65e+02	6.26e+02	1.64e+02	6.26e+02	9.76e+02
F_3	Mean	1.42e+01	1.93e+01	9.58e-03	1.93e+01	1.99e+01
	Std	4.27e-02	7.05e-02	2.73e-04	7.05e-02	5.34e-03
F_4	Mean	2.16e+14	3.11e+16	4.29e+16	3.11e+16	1.29e+12
	Std	2.10e+13	2.56e+15	5.36e+15	2.56e+15	2.15e+11
F_5	Mean	2.73e+09	2.27e+09	5.76e+09	2.27e+09	3.27e+09
	Std	1.13e+09	2.16e+08	7.89e+08	2.16e+08	1.92e+08
F_6	Mean	2.02e+07	2.02e+07	2.11e+07	2.02e+07	1.99e+07
	Std	4.43e+04	3.42e+04	3.91e+04	3.42e+04	2.54e+04
F_7	Mean	2.83e+07	2.34e+11	1.33e+11	2.34e+11	1.14e+05
	Std	5.21e+05	1.67e+10	1.25e+10	6.75e+10	4.49e+04
F_8	Mean	1.93e+08	2.62e+08	3.74e+08	2.62e+08	6.26e+05
	Std	1.39e+07	7.35e+05	7.06e+07	7.35e+05	1.45e+06
F_9	Mean	5.36e+10	4.38e+12	7.86e+11	4.38e+12	3.07e+08
	Std	1.45e+10	1.84e+11	3.86e+10	1.84e+11	1.72e+07
F_{10}	Mean	1.18e+05	5.90e+05	9.17e+04	5.90e+05	6.34e+04
	Std	7.85e+03	1.83e+04	5.46e+03	1.83e+04	8.89e+02
F_{11}	Mean	2.23e+02	2.28e+02	2.38e+02	2.28e+02	2.19e+02
	Std	2.57e-01	1.73e-01	2.94e-01	1.73e-01	1.26e-01
F_{12}	Mean	6.72e+05	2.35e+07	4.70e+07	2.35e+07	4.78e+05
	Std	5.63e+03	4.51e+05	2.03e+06	4.51e+05	2.54e+04
F_{13}	Mean	1.34e+05	2.32e+11	1.55e+04	2.32e+11	3.47e+03
	Std	2.01e+04	9.56e+09	5.95e+03	9.56e+09	4.11e+02
F_{14}	Mean	1.37e+11	1.59e+13	1.59e+12	1.59e+13	1.08e+09
	Std	6.68e+09	4.01e+11	4.65e+10	4.01e+11	3.70e+07
F_{15}	Mean	5.62e+05	2.34e+06	1.15e+05	2.34e+06	7.67e+04
	Std	4.70e+04	1.32e+04	1.66e+03	1.32e+04	1.56e+03
F_{16}	Mean	4.04e+02	4.19e+02	4.32e+02	4.19e+02	3.98e+02
	Std	5.05e-01	3.49e-01	6.16e-01	3.49e-01	1.89e-01
F_{17}	Mean	3.03e+06	4.06e+07	1.07e+08	4.06e+07	2.11e+06
	Std	5.54e+04	1.10e+06	4.03e+06	1.10e+06	5.75e+04
F_{18}	Mean	6.82e+03	1.85e+12	4.13e+04	1.85e+12	1.15e+04
	Std	8.40e+02	5.48e+10	8.15e+03	5.48e+10	1.07e+03
F_{19}	Mean	1.94e+07	1.10e+08	2.58e+08	1.10e+08	1.15e+07
	Std	5.61e+05	3.24e+06	7.44e+06	3.24e+06	3.78e+05
F_{20}	Mean	5.60e+03	1.85e+12	1.72e+04	1.85e+12	9.40e+03
	Std	9.62e+01	1.51e+11	4.38e+02	1.51e+11	3.59e+02
w-d-l		15-1-4	17-0-3	18-0-2	18-0-2	-

an algorithm increases with the dimensionality, the better its scalability will be on that problem. Notice that, the curves describe the function errors at a log_{10} level. Hence, the Y-axis of the figures actually denotes the orders of magnitude of the function errors.

5.2 Algorithms settings

As introduced in Section II, there are three major ways to deal with the high-dimensional problems. That is, existing EAs with re-balanced exploitation and exploration, dimensionality reduction based EAs, and divide-and-conquer based EAs. Although the proposed SEE belongs to the last category, it would be more comprehensive to compare SEE with methods from all categories. Hence, we select 4 representatives as the compared algorithms, i.e., MA-SW-Chains [7], RP-Ens [10], DECC-D [48], and DECC-DG [13]. To be specific, MA-SW-Chains, which won the IEEE WCCI2010 large-scale global optimization competition, is a memetic algorithm that allocates a local search intensity for each candidate solution in terms of its historical search behaviour. The local search intensity is a parameter to balance the exploration and exploitation during search. RP-Ens is a random projection based Estimation of Distribution Algorithm (EDA) [49–51], where original problem is firstly projected onto a number of lower dimensional sub-spaces with randomly generated matrix. After that, Gaussian distribution based EDAs are employed to solve the lower-dimensional sub-problems. There is one parameter that has immediate influence on the performance of RP-Ens, i.e., the number of lower dimensional sub-spaces to be projected on. In general, large number of lower dimensional sub-spaces can enhance the performance of RP-Ens, while it will also induce more computational cost.

Both DECC-D and DECC-DG are cooperative coevolution based works that first divide the original problem into a number of sub-problems in order to minimize the interdependencies in-between. After that, SaNSDE [52], a self-adaptive variant of differential evolution algorithms is employed to solve each sub-problem cooperatively by sharing the partial solutions with best function values. Besides the similarity, DECC-D differs from DECC-DG on the decomposition strategy. DECC-D suggests that if two variables are interdependent, the performance improvement brought by optimizing one of them is limited. With this insight in mind, DECC-D decomposes the variables into multiple sub-problems on-line, i.e., the sub-problems vary from iteration to iteration. On the contrary, DECC-DG decomposes the original problem in an off-line manner. That is, DECC-DG solves a problem by two steps: first learning the interdependencies between variables and dividing them into multiple sub-problems; second optimizing those sub-problems by SaNSDE under the framework of CC. The decomposition strategy of DECC-DG is based on that if two variables are independent, changing the value of either variable will not perturb the rank of the corresponding complete solutions.

The source codes of the 4 compared algorithms are all available online and were used to carry out the experiments. All the compared algorithms are configured according to the suggestions in their original paper, except for RP-Ens. In [10], the number of lower dimensional sub-spaces of RP-Ens was set to 1000. Though it works well on 1000-dimensional problems [10], it quickly freezes the workstation when the dimensionality of problems becomes larger. According to the remarks in the source code of RP-Ens, such parameter can be set to 200 if the memory of computer is insufficient to store the matrices. Hence, we set the number of lower dimensional sub-spaces to 1000 for RP-Ens on the 1000-dimensional problems and 200 on the rest 4 groups of problems. DECC-DG will consume the function evaluations during the learning stage, which can be overwhelming on some problems and the total time budget (in terms of

Table 7: The features of six selected 1000-dimensional problems.

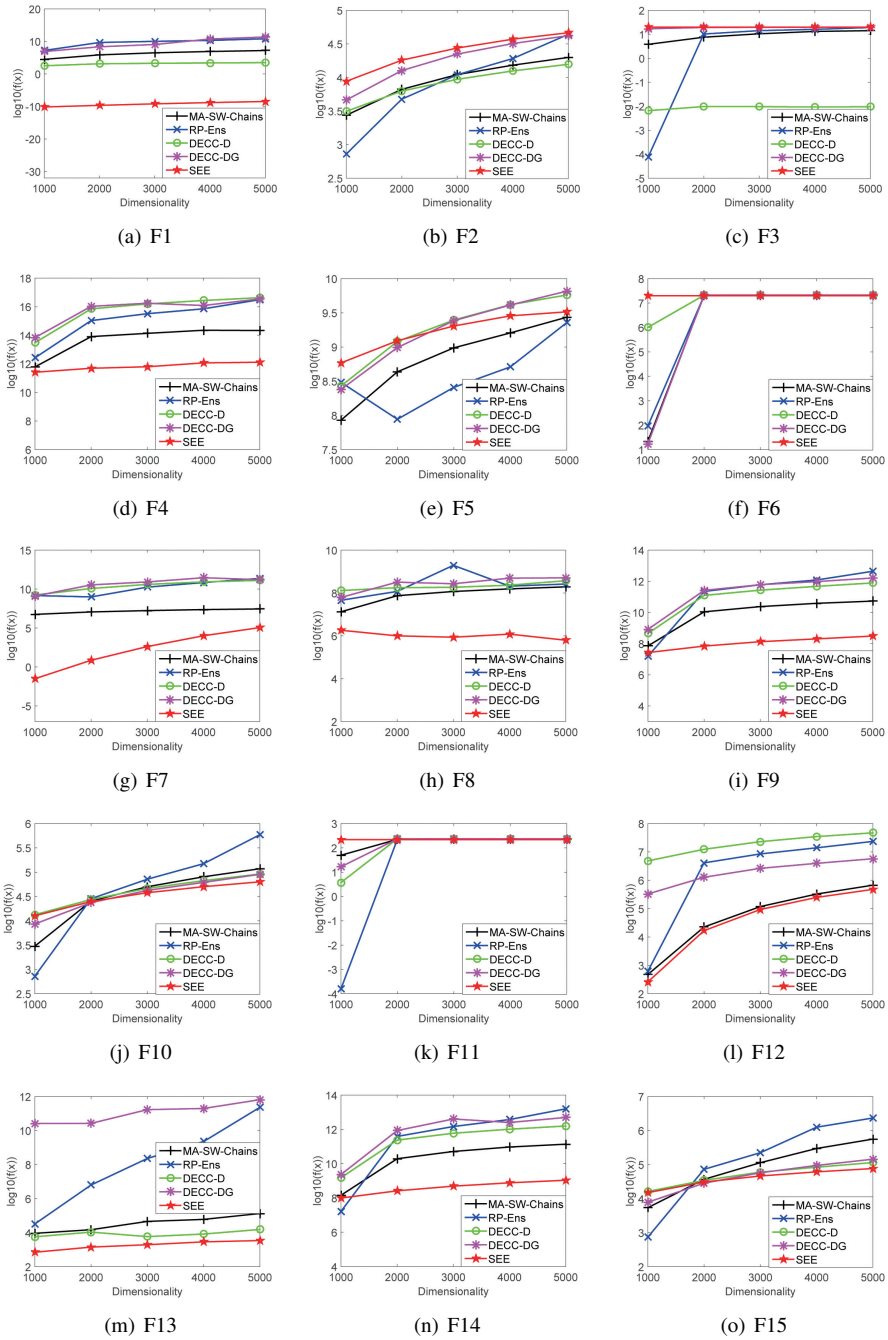
	F1	**F2**	**F7**
$Modality$	Uni-modal	Multi-modal	Uni-modal
$Separability$	Fully sepa.	Fully sepa.	Partially sepa.
$Smoothness$	Smooth	Rugged	Smooth
	F11	**F18**	**F20**
$Modality$	Multi-modal	Multi-modal	Multi-modal
$Separability$	Partially sepa.	Partially sepa.	Fully non-sepa.
$Smoothness$	Rugged	Smooth	Smooth

function evaluations) are not sufficient for completing the learning stage. In this comparison, the function evaluations consumed by DECC-DG during the learning stage is simply omitted. That is, after decomposition, the whole time budget will be used to optimize the sub-problems, which is the ideal case of DECC-DG. SEE has two parameters to set, which are all related to the sub-problem optimizer. The first parameter, i.e., the population size λ, is set to 10. The second parameter n is used to allocate the search resource for Gaussian mutation and Cauchy mutation. We simply set it to the half of λ, i.e., 5. This means neither Gaussian mutation nor Cauchy mutation is biased during the search of SEE.

5.3 Comparisons between SEE and four compared algorithms

A quick conclusion can be drawn from the results of Tables II-VI that SEE generally performs better than the 4 compared algorithms on the tested problems set. Besides, the advantages of SEE over the compared algorithms increase with dimensionality, which means that SEE has better scalability than the compared algorithms, at least on the 20 problems. This can be also verified in Figs.1-2 that, on most of the 20 problems, the function errors of SEE increase much slower than the curves of other 4 algorithms with the dimensionality.

Specially, F1 is the shifted Elliptic function. Intuitively, as F1 is both fully-separable and uni-modal, its global optimum can be easily obtained by separately optimizing one dimension at a time, regardless of the values taken on the other dimensions. It can be seen that the proposed SEE can always output good final solutions with function errors around 1e-10, which is very close to the global optimum. On the contrary, all the compared algorithms perform very poor on this problem that they fall at least 12 orders of magnitude behind SEE in terms of the function errors of final solutions. Such advantage of SEE does not disappear on partially separable or non-separable problems. For example, F4, F9, and F14 are variants of F1 that involve different degrees of rotated variables, which make the whole problems no longer fully separable. On those three problems, SEE is still significantly superior to the compared algorithms. Another example is the Schwefel problem 1.2 based problems. The Schwefel problem 1.2 is a fully non-separable problem. By combining it with the fully separable sphere function with different degrees, the F7, F12, F17, and F19 are constructed. To be specific, F7 has one single group of m interdependent variables. F12 has $\frac{D}{2m}$ groups of m interdependent variables. F17 has $\frac{D}{m}$ groups of m interdependent variables, and F19 is fully non-separable that involves one group of D interdependent variables. It can be



(a) F1 (b) F2 (c) F3

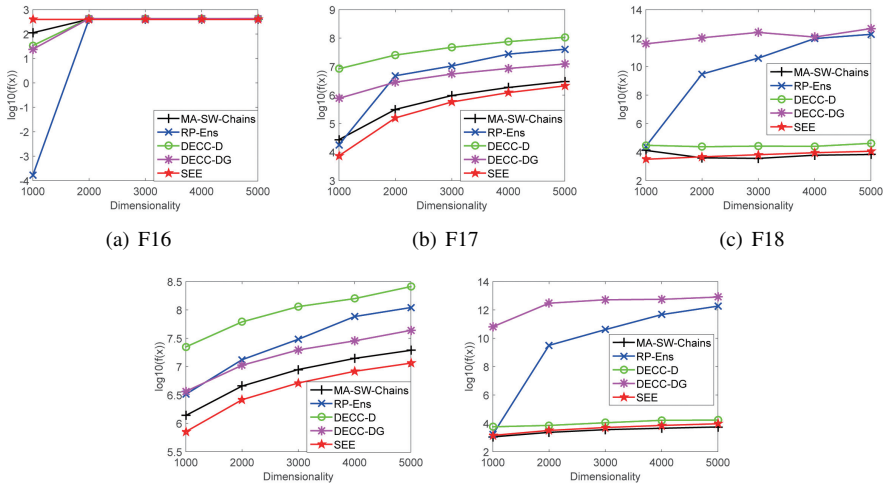
(d) F4 (e) F5 (f) F6

(g) F7 (h) F8 (i) F9

(j) F10 (k) F11 (l) F12

(m) F13 (n) F14 (o) F15

Figure 1: The scalability curves of 5 algorithms on F1-F15 problems. The Y-axis denotes the orders of magnitude of the function errors.



(a) F16 (b) F17 (c) F18

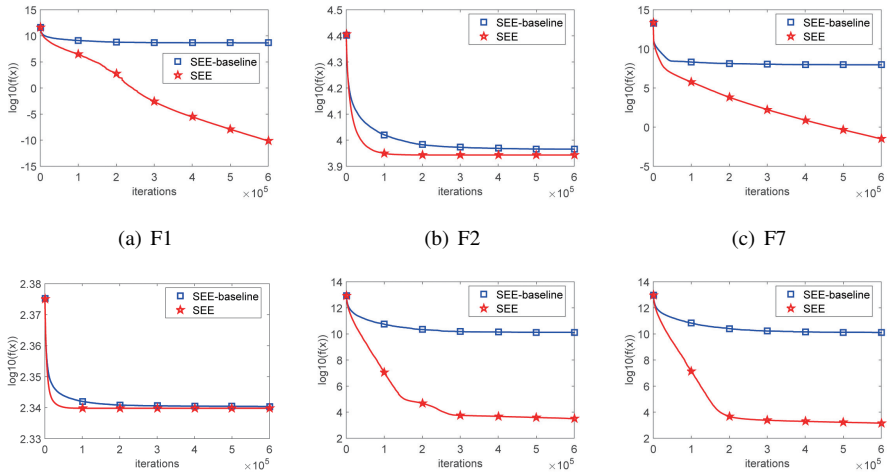
(d) F19 (e) F20

Figure 2: The scalability curves of 5 algorithms on F16-F20 problems. The Y-axis denotes the orders of magnitude of the function errors.

seen in Figs.1-2 that the scalability curves of SEE always keep lower than those of the compared algorithms, which means SEE performs the best on those 4 problems with different dimensionalities. Similar phenomena can be also observed on the well-known multi-modal Rosenbrock function based problems, i.e., F8, F13, F18, and F20. Notice that, although the results of SEE on F18 and F20 are slightly less accurate than that of the MA-WS-Chains, such differences can be insignificant when comparing with the huge function errors received by the other algorithms.

On the rest problems, i.e., F2, F3, F5, f6, F10, F11, F15, F16, SEE performs poorer than the compared algorithms in 1000-dimensional cases. The reason might lie in that those problems are either Rastrigin's function based or Ackley's function based, whose landscapes are rugged and the local areas change rapidly so that the proposed meta-model becomes less effective. Nevertheless, the 4 compared algorithms also drop their effectiveness down quickly when the dimensionality increases and even become inferior to SEE.

The two CC based algorithms, i.e., DECC-D and DECC-DG, perform much worse than SEE that their function errors on most of the 20 problems usually fall several orders of magnitude behind SEE, especially when the dimensionality increases. This phenomenon supports our suggestion that the DC methods can be facilitated better by self-evaluation with trained meta-models than cooperative coevolution. MA-SW-Chains and RP-Ens also obtain relatively good results on those 1000-dimensional problems. However, their performances deteriorate as the dimensionality increases. For MA-SW-Chains, the reason might be that the predefined local search intensity has well balanced the exploration and exploitation in 1000-dimensional solution space, while it is not suitable for higher dimensional problems. For RP-Ens, 1000 lower dimensions may cover the original solution space very well. However, when setting it to 200 to trade-off the computational cost, the performance of RP-Ens on the majority of the tested functions drops down dramatically. The reason might be that the sub-problems produced by such small number of random projections may not jointly cover the original



(a) F1 (b) F2 (c) F7

(d) F11 (e) F18 (f) F20

Figure 3: The convergence curves of SEE and SEE-baseline on 6 problems. The Y-axis denotes the orders of magnitude of the function errors.

solution space well.

5.4 Investigation of the meta-model

It is of importance to investigate how the meta-model impacts the performance of SEE. The most intuitive way might be to check the relation between the accuracy of the meta-model and the performance of SEE. However, as the ground truth of the rank of any pair of partial solutions is computationally prohibited to obtain, it is infeasible to calculate the accuracy of the meta-model. In this paper, we adopt an indirect method to investigate this issue. That is, a baseline meta-model is constructed by fixing the $PS_{i,j}$ and $PL_{i,j}$ as 0.5 along the whole optimization, which can be regarded as a random guess for ranking any pair of partial solutions. By replacing the original meta-model in SEE with this baseline meta-model, we obtain the resultant algorithm, denoted as SEE-baseline. Then the SEE-baseline is compared with SEE on the above benchmark problems. From their convergence curves on different problems, it will be easy to see the importance of the proposed meta-model to SEE. For brevity, we only list the comparisons on 6 selected 1000-dimensional problems in Fig.3, the comprehensive results will be available online[4]. The listed problems are selected in terms of their multimodality, separability, and landscape smoothness. The underlying idea is to show that the proposed meta-model is less sensitive to the multimodality and separability but heavily relies on the smoothness of the problem. These 6 problems are F1, F2, F7, F11, F18, F20, and their features are listed in Table VII.

As seen in Fig.3, SEE outperforms SEE-baseline on all the 6 problems, which means the proposed meta-model has a positive impact on the performance of SEE. Furthermore, Fig.3 also shows that the meta-model is not heavily influenced by the modality and separability of problem, which are both commonly concerned features of high-dimensional optimization problems. On the contrary, the effectiveness of the pro-

[4]The comprehensive results can be seen at: `http://staff.ustc.edu.cn/~ketang/`

posed meta-model deteriorates significantly when the landscape of a problem is rugged (Fig.3(b) and Fig.3(d)), where the assumption of the meta-model that the landscape changes smoothly does not hold.

6 Conclusion

This work investigated the general DC idea on high-dimensional optimization problems. We discussed that the major difficulty of DC in optimization lies in the dimensionality mismatch, where the partial solutions to each sub-problem can hardly be correctly evaluated. Hence, we suggest that dimensionality mismatch can be tackled as a computationally expensive problem. That is, we suggest evaluating partial solutions to each sub-problem separately by employing computationally cheap meta-models, and propose a novel SEE approach based on this idea. In SEE, the original objective function $\mathcal{F}$ is firstly divided into D 1-dimensional sub-problems. In each sub-problem, a $(1 + \lambda)$-EA is employed as the local search operator. That is, a parent partial solution (to a sub-problem) generates λ offsprings partial solutions with either Gaussian mutation or Cauchy mutation. A simple yet efficient meta-model is designed to predict the rank of a pair of partial solutions. Based on the rank and objective function values, the best one among the $(1 + \lambda)$ partial solutions is kept to the next iteration. To verify its effectiveness, empirical studies have been conducted to compare the SEE with 4 recently proposed representatives of high-dimensional optimization approaches. The results have shown that our work becomes more superior to the compared algorithms on the tested 20 functions with the increase of dimensionality. The impact of the meta-model on the performance of SEE is also investigated.

The proposed meta-model in SEE is implicitly based on the assumption that the local landscape of the solution space changes smoothly. Seen from the empirical results, on the problems that meet the assumption of local smoothness, the meta-model positively impacts the performance of SEE, while on the other problems, the meta-model loses its effectiveness. Besides, the proposed meta-model requires the sub-problem optimizer to be local search. In the future, more sophisticated meta-models may be incorporated into SEE to employ more powerful global optimization techniques as the sub-problem optimizers, such as differential evolution, estimation of distribution algorithms. By using more sophisticated meta-models, the assumption of local smoothness of the landscape may be alleviated.

References

[1] M. H. Tayarani-N., X. Yao, and H. Xu, "Meta-heuristic algorithms in car engine design: A literature survey," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 609–629, Oct 2015.

[2] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, June 2015.

[3] R. J. Preen and L. Bull, "Toward the coevolution of novel vertical-axis wind turbines," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 284–294, April 2015.

[4] W. B. Langdon and M. Harman, "Optimizing existing software with genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, Feb 2015.

[5] D. M. Cabrera, "Evolutionary algorithms for large-scale global optimisation: a snapshot, trends and challenges," *Progress in Artificial Intelligence*, vol. 5, no. 2, pp. 85–89, 2016.

[6] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Metaheuristics in large-scale global continues optimization: a survey," *Information Sciences*, vol. 295, pp. 407–428, 2015.

[7] D. Molina, M. Lozano, and F. Herrera, "MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *IEEE Congress on Evolutionary Computation*. Barcelona, Spain: IEEE, July 2010, pp. 1–8.

[8] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE transactions on cybernetics*, vol. 45, no. 2, pp. 191–204, 2015.

[9] H. Wang, Z. Wu, and S. Rahnamayan, "Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems," *Soft Computing*, vol. 15, no. 11, pp. 2127–2140, 2011.

[10] A. Kabán, J. Bootkrajang, and R. J. Durrant, "Toward large-scale continuous eda: A random matrix theory perspective," *Evolutionary computation*, 2015.

[11] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, N. Freitas *et al.*, "Bayesian optimization in high dimensions via random embeddings," in *International Joint Conferences on Artificial Intelligence (IJCAI)*, Beijing, China, Aug. 2013.

[12] W. Dong, T. Chen, P. Tiňo, and X. Yao, "Scaling up estimation of distribution algorithms for continuous optimization," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 6, pp. 797–822, 2013.

[13] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 378–393, 2014.

[14] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *International Conference on Parallel Problem Solving from Nature*. Jerusalem, Israel: Springer, Oct. 1994, pp. 249–257.

[15] R. P. Wiegand, "An analysis of cooperative coevolutionary algorithms," Ph.D. dissertation, Fairfax, VA, USA, 2004, aAI3108645.

[16] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary computation*, vol. 8, no. 1, pp. 1–29, 2000.

[17] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, 2012.

[18] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.

[19] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 2, p. 13, 2016.

[20] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *2001 IEEE Congress on Evolutionary Computation*, vol. 2. Seoul, South Korea: IEEE, 2001, pp. 1101–1108.

[21] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[22] T. Jansen and R. P. Wiegand, "The cooperative coevolutionary (1+1) ea," *Evolutionary Computation*, vol. 12, no. 4, pp. 405–434, 2004.

[23] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *International Conference on Parallel Problem Solving from Nature*. Krakow, Poland: Springer, Sept. 2010, pp. 300–309.

[24] L. Panait, "Theoretical convergence guarantees for cooperative coevolutionary algorithms," *Evolutionary computation*, vol. 18, no. 4, pp. 581–615, 2010.

[25] L. Panait, S. Luke, and J. F. Harrison, "Archive-based cooperative coevolutionary algorithms," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. Seattle, USA: ACM, July 2006, pp. 345–352.

[26] L. Panait and S. Luke, "Time-dependent collaboration schemes for cooperative coevolutionary algorithms," in *Proceedings of the 2005 AAAI Fall Symposium on Coevolutionary and Coadaptive Systems*, Virginia, USA, Nov. 2005.

[27] R.-L. Tang, Z. Wu, and Y.-J. Fang, "Adaptive multi-context cooperatively coevolving particle swarm optimization for large-scale problems," *Soft Computing*, pp. 1–20, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00500-016-2081-6

[28] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the cec'2010 special session and competition on large scale global optimization," Nature Inspired Computation and Applications Laboratory, University of Science and Technology of China, China, Tech. Rep., 2009.

[29] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, p. 35, 2013.

[30] Y. Mei, X. Li, and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 435–449, June 2014.

[31] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima, and M. Pelikan, "Dependency structure matrix, genetic algorithms, and effective recombination," *Evolutionary computation*, vol. 17, no. 4, pp. 595–626, 2009.

[32] R. P. Wiegand and J. Sarma, "Spatial embedding and loss of gradient in cooperative coevolutionary algorithms," in *International Conference on Parallel Problem Solving from Nature*. Birmingham, UK: Springer, Sept. 2004, pp. 912–921.

[33] R. P. Wiegand, W. C. Liles, and K. A. De Jong, "An empirical analysis of collaboration methods in cooperative coevolutionary algorithms," in *Proceedings of the genetic and evolutionary computation conference (GECCO)*, vol. 2611, San francisco, USA, July 2001, pp. 1235–1245.

[34] M. Iqbal, W. N. Browne, and M. Zhang, "Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 465–480, Aug 2014.

[35] R. A. Watson, "Analysis of recombinative algorithms on a non-separable building-block problem," *Foundations of genetic algorithms*, vol. 6, pp. 69–89, 2001.

[36] M. Munetomo and D. E. Goldberg, "A genetic algorithm using linkage identification by nonlinearity check," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1. Tokyo, Japan: IEEE, Oct. 1999, pp. 595–600.

[37] D. E. Goldberg, K. Deb, H. Kargupta, and G. R. Harik, "Rapid accurate optimization of difficult problems using fast messy genetic algorithms." in *Proceedings of the 5th International Conference on Genetic Algorithms*, vol. 5, Urbana-Champaign, USA, June 1993, pp. 56–64.

[38] M. Munetomo and D. E. Goldberg, "Linkage identification by non-monotonicity detection for overlapping functions," *Evolutionary computation*, vol. 7, no. 4, pp. 377–398, 1999.

[39] M. Tabatabaei, J. Hakanen, M. Hartikainen, K. Miettinen, and K. Sindhya, "A survey on handling computationally expensive multiobjective optimization problems using surrogates: non-nature inspired methods," *Structural and Multidisciplinary Optimization*, vol. 52, no. 1, pp. 1–25, 2015.

[40] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[41] K. Deb and C. Myburgh, "Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 653–660.

[42] A. Ciccazzo, G. D. Pillo, and V. Latorre, "Support vector machines for surrogate modeling of electronic circuits," *Neural Computing and Applications*, vol. 24, no. 1, pp. 69–76, 2014.

[43] Y. Tenne and S. W. Armfield, "A framework for memetic optimization using variable global and local surrogate models," *Soft Computing*, vol. 13, no. 8, p. 781, 2008.

[44] B. Liu, Q. Zhang, and G. G. E. Gielen, "A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 180–192, April 2014.

[45] N. Hansen, D. V. Arnold, and A. Auger, "Evolution strategies," in *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 871–898.

[46] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary computation*, vol. 3, no. 2, pp. 82–102, 1999.

[47] C.-Y. Lee and X. Yao, "Evolutionary programming using mutations based on the levy probability distribution," *IEEE Transactions on Evolutionary computation*, vol. 8, no. 1, pp. 1–13, Feb. 2004.

[48] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *2010 IEEE Congress on Evolutionary Computation*. Barcelona, Spain: IEEE, July 2010, pp. 1–8.

[49] P. Larranaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2002, vol. 2.

[50] J. Wang, K. Tang, J. A. Lozano, and X. Yao, "Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 96–109, Feb 2016.

[51] A. Zhou, J. Sun, and Q. Zhang, "An estimation of distribution algorithm with cheap and expensive local search methods," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 6, pp. 807–822, 2015.

[52] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *2008 IEEE Congress on Evolutionary Computation*. Hong Kong, China: IEEE, June 2008, pp. 1110–1116.

Algorithm 2 SEE($\mathcal{F}$, T_{max}, λ, $\mu = 1, n$)

Divide $\mathcal{F}$ into D exclusive sub-problems.
For $j = 1$ **to** D
  For $i = 1$ **to** λ
    Initialize $PS_{i,j}$ and $PL_{i,j}$ as 1.00.
    Initialize $\sigma_{i,j}$ as 1.00.
  EndFor
  Initialize $x^p_{1,j}$ randomly.
EndFor
For $t = 1$ **to** T_{max}
  For $j = 1$ **to** D
    For $i = 1$ **to** n
      $x^o_{i,j} = x^p_{1,j} + \sigma_{i,j} \cdot \mathcal{N}(0, 1)$.
    EndIf
    For $i = n + 1$ **to** λ
      $x^o_{i,j} = x^p_{1,j} + \sigma_{i,j} \cdot \mathcal{C}(0, 1)$.
    EndIf
    For $i = 1$ **to** λ
      If $x^o_{i,j} < x^p_{1,j} \wedge PS_{i,j} < r$
        $x^o_{i,j} = x^p_{1,j}$.
      Else $x^o_{i,j} > x^p_{1,j} \wedge PL_{i,j} < r$
        $x^o_{i,j} = x^p_{1,j}$.
      EndIf
    EndFor
  EndFor
  If $\mathcal{F}(\mathbf{x}^p_1) > \min_{1 \le i \le \lambda} \mathcal{F}(\mathbf{x}^o_i)$
    $\mathbf{x}^p_1 = \text{argmin}_{\mathbf{x}^o_i} \mathcal{F}(\mathbf{x}^o_i)$
  EndFor
  For $j = 1$ **to** D
    For $i = 1$ **to** λ
      $\sigma_{i,j} = \sigma_{i,j} \cdot \exp^{\frac{1}{\sqrt{2}}} [\mathbb{I}_{x^o_{i,j} \neq x^p_{1,j}} \cdot (\mathbb{I}_{\mathcal{F}(\mathbf{x}^p_1) \geq \mathcal{F}(\mathbf{x}^o_i)} - \frac{1}{5})]$.
      $PS_{i,j} = PS_{i,j} \cdot \exp^{\frac{1}{\sqrt{2}}} [\mathbb{I}_{x^o_{i,j} < x^p_{1,j}} \cdot (\mathbb{I}_{\mathcal{F}(\mathbf{x}^p_1) \geq \mathcal{F}(\mathbf{x}^o_i)} - \frac{1}{5})]$.
      $PL_{i,j} = PL_{i,j} \cdot \exp^{\frac{1}{\sqrt{2}}} [\mathbb{I}_{x^o_{i,j} > x^p_{1,j}} \cdot (\mathbb{I}_{\mathcal{F}(\mathbf{x}^p_1) \geq \mathcal{F}(\mathbf{x}^o_i)} - \frac{1}{5})]$.
    EndFor
  EndFor
EndFor
Output $\mathbf{x}^p_1$.