Universiteit Leiden

Master Computer Science

Empirical Study of Dimensionality Reduction and Feature Selection Methods for Evolution Strategy in High Dimension

Name:	Yangjie Mei
Student ID:	S2368285
Date:	11/05/2021
Specialisation:	Computer Science and Advanced Data Analytics
1st supervisor:	Dr. H. Wang
2nd supervisor:	Prof.dr. T.H.W. Bäck

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Abstract

Over the past decades, evolution strategy is widely used as a heuristic method in some specific fields. However, when the dimensionality increases, the computational cost grows much faster than dimensionality. Therefore, how to tackle a good result with a small computation cost becomes a problem that needs to be address. In this thesis, we will use dimensionality reduction methods such as Principal Component Analysis (PCA), Random Forest (RF), Permutation, Boruta, and SHapley Additive exPlanations (SHAP) to reduce the covariance matrix dimension in different iterations of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) sampling process. In addition, this thesis evaluates the performance of the new CMA-ES, which combined dimensionality reduction methods, and regular CMA-ES in terms of convergence rates and optimal values on the problems of the Black-Box Optimization Benchmark (BBOB) problem set. Making a benchmark for all these new methods and analyzing the advantages and disadvantages of each algorithm.

Keywords— Evolutionary Algorithm, Principal component analysis, Random Forest, Permutation, Boruta, Shapley Additive Explanations, Covariance Matrix Adaptation Evolution Strategy, Black-Box Optimization Benchmarking

1 Introduction

Evolutionary algorithms are widely used as a good heuristic method to solve optimization problems in various laboratory scenarios as well as industrial scenarios. However, although evolutionary algorithms can find the best solution, sometimes its computational cost typically grows much faster than the dimensionality increases. Therefore, in this thesis, we aim to delve into potential methods to alleviate this problem. One method is to use dimensionality reduction[1] for continuous optimization problems. We plan to investigate the usefulness and effectiveness of various dimensionality reduction methods like PCA [2], Random Forest [3], Permutation[4], Boruta [5] and SHAP that based on Shapley value [6] and couple them with an evolutionary algorithm Covariance Matrix Adaptation Evolution Strategy(CMA-ES) [7], which is a widely-applied evolutionary algorithm for black-box optimization problems in continuous domains. Also, this integration of dimensionality reduction methods shall be implemented in an online manner, where the dimensionality reduction mapping and the inverse mapping thereof should be learned or adapted in each iteration of CMA-ES.

1.1 Research questions

The concept of CMA-ES has been published for several years, and its variants are considered the most advanced technique in evolutionary computation and are used in many different fields. At the same time, dimensionality reduction methods are widely used in various areas. Therefore, to explore the feasibility of combining the dimensionality reduction method with CMA-ES, we aim to answer the following questions.

1. What is the general impact of the dimensionality reduction method on the performance of CMA-ES?
2. Why are the variables reliable after dimensionality reduction?
3. Can the dimensionality reduction method really save time throughout the CMA-ES process?

1.2 Outline of the thesis

From the beginning, we focus on CMA-ES, a special evolutionary algorithm for black-box optimization problems in continuous domains. Its variances are used as a state-of-the-art method by different research groups and laboratories, and it is also applied in various industrial situations. Then, we investigate a class of dimensionality reduction methods such as PCA, RF, Permutation, Boruta, and SHAP. The methods can be divided into two types. The first one is feature projection, also called feature extraction [8] like PCA, which maps the data onto the first few components to obtain low-dimensional data while preserving as much variation in the data as possible. The second is the other four methods, which belong to the feature selection [9], where the most important features are selected, and the unimportant ones are discarded according to the importance calculated for the given data. In this thesis, we inserted feature projection and feature selection methods during the sampling operation of CMA-ES to accomplish the ability to transform variable dimensions in an iteration dynamically. Subsequently, we tested the new combined methods on the BBOB problem set, compared and analyzed their performance. We identified the advantages and disadvantages of the different techniques based on the results.

The rest of this thesis is structured as follows. Section 2 talks about the principle of CMA-ES and introduces the dimensionality reduction methods such as PCA, RF, Permutation, Boruta, and SHAP. This section also presents some related work that has been done to combine PCA with evolutionary algorithms, which can provide inspiration for the following research. Section 3 will talk about the combination of CMA-ES and dimensionality reduction methods. The experiment in this thesis is divided into two parts: Section 4 describes the experiment setup, and Section 5 discusses

the performance of the new methods based on the experimental results. Section 6 Summarize the experimental results and look forward to future research.

2 Background

CMA-ES is an evolutionary strategy for solving complex black-box optimization problems in continuous domains. CMA-ES and its variants are considered state-of-the-art evolutionary computational methods and have been adopted by many research laboratories and industrial settings around the world. The following section will first discuss the CMA-ES, followed by the dimensionality reduction method.

2.1 Covariance Matrix Adaptation Evolution Strategy

In contrast to GA, which is an evolutionary algorithm focusing on discrete problems, CMA-ES is excellent for continuous numerical optimization problems and is widely used in various laboratory and industrial scenarios. More importantly, compared to the most straightforward evolutionary strategy algorithm, CMA-ES algorithm gets the results of each iteration and adaptively increases or decreases the search space in the next generation of search.
CMA-ES uses the covariance matrix C to track the pairwise dependencies between samples obtained in the distribution. The new distribution parameter becomes:

$$\theta = (\mu, \sigma, C),\ p_\theta(x) \sim N(\mu, \sigma^2 C) \sim \mu + \sigma N(0, C) \tag{1}$$

where where $N(0, C)$ denotes a multivariate Gaussian distribution with mean 0 and covariance C, $p_\theta(x)$ indicates the populations, μ is the mean value of population, σ represents the step, C means the covariance matrix and C is symmetric.

Mutation The way to obtain the mutations in CMA-ES is to use the covariance matrix C to sample the results that satisfy a normal distribution where the expectation equals zero and stand deviation equals C. The following equation shows how CMA-ES samples the population:

$$x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)} \text{where } y_i \sim N(0, C^{(t)}), i = 1, ..., \lambda \tag{2}$$

Where λ is the size of offspring population, $x_i^{(t+1)}$ denotes the new candidates of the next generations, $\mu^{(t)}$ indicates the mean of elites of the last generation and $\sigma^{(t)}$ represents the step of the current generation and y_i obeys the normal distribution where the mean of distribution equals zero and the stand deviation equals $C^{(t)}$.

Recombination

$$\mu^{(t+1)} = \sum_{i=1}^{m} w_i x_{i:\lambda}^{(t+1)} \tag{3}$$

$$\sum_{i=1}^{m} w_i = 1, w_1 \geq w_2 \geq ...w_m > 0 \tag{4}$$

Where m is the number of samples which selected from $(x_1^{t+1}, ..., x_\lambda^{t+1})$, $\mu^{(t+1)}$ is the weighted means of m samples and $x_{i:\lambda}^{(t+1)}$ is the i_{th} best individual in $(x_1^{t+1}, ..., x_\lambda^{t+1})$

Step-size adaptation CMA-ES uses $\sigma^{(t)}$ to control the step of each iteration. To determine a good step-size, CMA-ES gets the sum of consecutive moving sequence $\frac{1}{\lambda}\sum_i^{\lambda} y_i^{(j)}, j = 1, 2, ..., t$ to get the evolution path p_σ and compare the evolution path with the path generated by random selection. If the evolution path is longer than the random selection path, CMA-ES will reduce the $\sigma^{(t)}$, and vice versa.

Covariance matrix The eigendecomposition of Covariance Matrix C obeys: $C = BD^2B^T$. The expression of Covariance Matrix C can re-estimate like this:

$$C_\lambda^{(t+1)} = \sum_{i=1}^{\lambda} w_i (x_{i:\lambda}^{(t+1)} - \mu^{(t)})(x_{i:\lambda}^{(t+1)} - \mu^{(t)})^T \tag{5}$$

In this formula, this estimate is reliable only if the population is large enough. However, in each iteration CMA-ES would like to have a fast and reliable iteration when the population size is lower. Therefore CMA-ES has a more reliable but more complex method to update C. It consists of two parts. The first method uses the history of C.

$$C^{(t+1)} = (1 - \alpha_{c_\lambda})C^{(t)} + \alpha_{c_\lambda} C_\lambda^{(t+1)} = (1 - \alpha_{c_\lambda})C^{(t)} + \alpha_{c_\lambda} \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^T} \tag{6}$$

where $\alpha_{c\lambda}$ is the learning rate of first way.
The second way is using an evolution path p_c, p_c also follows to the normal distribution $N(0, C)$:

$$p_c^{(t+1)} = (1 - \alpha_{c_p})p_c^{(t)} + \sqrt{\alpha_{c_p}(2 - \alpha_{c_p})\lambda} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \tag{7}$$

and CMA-ES uses p_c to update the covariance martix C:

$$C_\lambda^{(t+1)} = (1 - \alpha_{c_1})C^{(t)} + 1 - \alpha_{c_1} p_c^{(t+1)} p_c^{(t+1)^T} \tag{8}$$

Ultimately, this final update formula can be obtained by combining these two methods:

$$C^{(t+1)} = (1 - \alpha_{c_1} - \alpha_{c_\lambda})C^{(t)} + \alpha_{c_1} p_c^{(t+1)} p_c^{(t+1)^T} + \alpha_{c_\lambda} \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^T} \tag{9}$$

2.2 Dimensionality Reduction

Dimensionality reduction is a kind of method to convert high dimensional data to low dimensional data, while using this low dimensional data to retain some meaningful attributes of the original dimensional data to the maximum extent. The overall dimensionality reduction can be roughly divided into two categories: feature projection and feature selection. Feature projection converts data from high dimensionality to low dimensionality, and this transformation can be linear or nonlinear; in contrast to feature projection, feature selection is an attempt to find the best subset that can go on to represent a given set of input variables. In this experiment, we choose linear PCA as the feature projection method and select RF, Permutation, Boruta and SHAP as the feature selection methods.

2.2.1 Principal Component Analysis

Monitoring data with multiple variables and analyzing them is inevitable in many domains of research. Multivariate can provide a wealth of information, but it can also be computationally and analytically intensive. Also in global optimization problems, if the dimensionality increases, optimization algorithms usually require an exponentially increasing number of function evaluations [10] to

find the optimal value. Therefore, reducing variables while preserving their information becomes one of our research goals. And there are some kinds of known methods like single value decomposition (SVD) [11], PCA, linear discriminant analysis (LDA) [12] to reduce the dimensionality of the data. In our study, we choose linear-PCA as the feature projection method.

PCA is one of the most widely used algorithms for dimensionality reduction of data. the main idea of PCA is to map n-dimensional features to k-dimensions, where k is less than n. This k-dimension is a brand new orthogonal feature also called principal component, which is a k-dimensional feature reconstructed on the basis of the original n-dimensional features. PCA calculates the covariance matrix of the data matrix, then gets the eigenvalue eigenvectors of the covariance matrix, and selects the matrix composed of the eigenvectors corresponding to the k features with the largest eigenvalues (i.e., the largest variance), so that the data matrix can be transformed to the new space when the dimensionality reduction of the data features is achieved.

Also during the last few years, PCA has proven to be useful when combined with optimization algorithms. In 2016, Dimitrios Kapsoulis [13] introduced the combination method of Kernel-PCA [14] with evolutionary algorithm. In the iterative process of traditional evolutionary algorithms, there are four basic steps. Initialization, selection, crossover and mutation. Dimitrios Kapsoulis applied the Kernel-PCA method before the crossover operation to reduce the dimensionality of the variables and after the mutation operation to invert and map the variables at lower latitudes to the previous dimensions. Dimitrios Kapsoulis confirmed that in subsequent experiments, this new algorithm in the iterative process has a fast convergence rate as well as a relatively low computational cost.

In 2020, Elena Raponi1, Hao Wang [15] apply the PCA method on Bayesian Optimization (BO) [16] and run the benchmark on COCO[17] and assess the performance of PCA-BO in terms of the empirical convergence rate and CPU time on multi-modal problems from BBOB problem sets and gain some progress through the PCA method.

2.2.2 Random Forest

Unlike PCA, which is a feature mapping method. The methods we discuss next are another class of methods called feature selection, each of which has its importance assessment and selects the important features by their importance for the purpose of dimensionality reduction.

Random forests are often used as one of the feature selection methods in the data preprocessing phase of many problems. Random forest is a model consisting of multiple trees that embodies the concept of ensemble learning. Its basic components are decision trees or regression trees. In a random forest, each tree is a classifier or regressor and RF collects the results of each tree and summarizes a result. There are three methods to build decision trees: id3, C4.5, and CART. Inside the subsequent experiments, this thesis only chooses CART to build the tree because only CART is supported for regression problems.

CART CART's [18] full name is Classification And Regression Tree. This implies that it is also possible to build a regression tree compared to ID3 and C4.5. The CART algorithm is a binary recursive segmentation technique. The current sample can only be divided into two subsamples, so each non-leaf node generated has only two branches. Therefore, the tree generated by the CART algorithm is a binary tree with a simple structure. Even if a feature has multiple categories, the data can still only be divided into two parts.

When CART deals with classification problems, one very important difference from ID3 and C4.5 is that while ID3 and C4.5 use entropy as a criterion for reducing the instability of a data set, CART chooses the reduction of the Gini index [19] as a criterion instead. The Gini calculation is easier to implement and compute than the entropy calculation, which uses logarithmic operations and causes time consumption. Here is how the GINI impurity works in classification, if the dataset

D has K classes, each with probability p_k, the Gini coefficient can be expressed as follows.

$$Gini(D) = 1 - \sum_{k=1}^{K} p_k^2 \tag{10}$$

After the D is divided into D_1 and D_2 by the attribute A, the Gain Gini can be like this:

$$GI(A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_1|}{|D|} Gini(D_2) \tag{11}$$

where,

- $|D_1|$ - the sample amount of Dataset D_1
- $Gini(D_1)$ - the Gini index of Dataset D_1

CART will select the attribute with the smallest Gain Gini as the split point.
On the other hand, if the problem handled by random forest is a regression problem, CART uses least squares in constructing a binary tree to minimize the sum of residuals between the observations and the mean in each node. In the input space of the training data, each region is recursively divided into two subregions and the output value of each subregion is derived. A binomial tree is then constructed to find the best attribute j and cut point s.

$$\min_{j,s} = \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right] \tag{12}$$

where, $c_1 = ave(y_i | x_i \in R_1(j,s))$, $c_2 = ave(y_i | x_i \in R_2(j,s))$, each attribute j and cut point s has it's value and CART will split the point with the lowest value. CART will build the tree recursively according to the formula.

Using Random Forest for selection In 2009, Bjoern H Menze [20] combined the Gini importance selection in Random Forest with partial least squares regression [21]. The combined approach is applied to specific datasets such as medical diagnostics, chemical taxonomy, biomedical analysis, food science, etc. Finally, they not only reduce the dimensionality, but also erase the noise.

So how does RF select the features? At the beginning of building RF, RF will use Bootstrap [22] to initialize the dataset. Bootstrap sampling method selects N samples from the original N samples, and each instance is put back after each sampling operation. When N goes to infinity, the set of samples taken is about two-thirds of the original sample set, and these are the training set. The remaining one-third is called the "out-of-bag" (OOB) set. After the RF is built, the RF will use the OOB to evaluate the model. The details of the random forest algorithm are shown in Alg. 1.

Algorithm 1 Random Forest

Input: Dataset: D_{raw}, Maximum number of decision trees: n
Output: Random Forest: RF

1: **while** Not reach n **do**
2: Generate Dataset with the Bootstrap method
3: Generate the Decision Tree with the CART algorithm
4: **end while**
5: Use Out-of-Bag(OOB) to evaluate the Random Forest Model
6: **return** RF

After establishing the random forest, for different problems, the random forest will have different importance assessment criteria. If the problem is a classification problem, then RF uses the GINI index for feature selection; if the problem is a regression problem, then RF uses the residual sum of squares (RSS) as an indicator for feature selection. Regardless of the type of problem, after establishing the RF, the variable importance measure can be expressed as VIM.

Inside the classification problem, the GINI index is defined as GI. If we have a set of C attributes. $X_1, X_2, X_3, ..., X_C$], VIM^i_{jm} denotes the average change in node split impurity of attribute j on m nodes in the i_{th} decision tree, the GINI index can be computed as follows.

$$GI(m) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_1|}{|D|} Gini(D_2) \tag{13}$$

where, node m split the tree into two parts D_1 and $D2$, $Gini(D_1)$ means the GINI Index of D_1 and $Gini(D_2)$ is the GINI Index of D_2. After the tree is split by the node m, the variation of the GINI Index is shown like this:

$$VIM_{jm} = GI_m - GI_l - GI_r \tag{14}$$

Where GI_l means the GINI Index of the left part split by the node m and GI_r indicates the GINI Index of the right part split by the node m. If the problem is a regression problem, then the criteria for importance assessment are somewhat different. When the tree is partitioned by node m, the sum of squares of the residuals of the nodes is computable:

$$RSS = \sum_{x_i \in R_1(m)} (y_i - c_1)^2 + \sum_{x_i \in R_2(m)} (y_i - c_2)^2 \tag{15}$$

where $c_1 = ave(y_i | x_i \in R_1(m))$, $c_2 = ave(y_i | x_i \in R_2(m))$ and the variation of the VIM in this node is shown like this:

$$VIM_{jm} = RSS_m - RSS_l - RSS_r \tag{16}$$

If set M means all the nodes of attribute X_j in tree i. We calculate the importance of X_j in i tree as follows:

$$VIM^i_j = \sum_{m \in M} VIM^i_{jm} \tag{17}$$

and if RF has n trees, the importance of attribute j is the sum of all trees:

$$VIM^{Imp}_j = \sum_{i=1}^{n} VIM^i_j \tag{18}$$

Therefore, RF can obtain the importance ranking of all features in the dataset S by calculating the GINI index or RSS for all features, with larger VIM meaning more important.

2.2.3 Permutation Importance

Permutation is another method of feature selection. In 2017, Baptiste Gregorutti [23] used the Permutation importance metric as a ranking criterion for eliminating variables and demonstrated that this method can acquire good predictions with lower variables. Finding important features in Permutation is also very simple, when the data has been trained by a model such as RF, then at the beginning of the algorithm, Permutation will randomly shuffle one feature at a time and calculate the score of the corrupted dataset (e.g., the accuracy of the classifier or the Euclidean distance of the regressor), and the Permutation method will repeat this step k times and rank the importance based on the average score. The details of Permutation in random forests are shown in Alg.2.

In this algorithm Permutation will repeat a feature shuffle several times and average it, then get the importance ranking of the feature based on the score of each feature and select the features that we want to keep. The higher the score means the more important the feature is.

Algorithm 2 Permutation in Random Forest

Input: Fitted Random Forest Model M and dataset D
Output: Set of importance S

Compute the score s of the model M on dataset D
for each features f in dataset **do**
  Initialize the $Sum_f = 0$
  while repetition time not reach k **do**
    Randomly shuffle the column f of Dataset D and get the corrupted dataset D_c
    calculate the score s_f with the given model M and corrupted dataset D_c, add s_f into Sum_f: $Sum_f = Sum_f + s_f$
  end while
end for
The importance I_f for each feature f is: $I_f = s - \frac{Sum_f}{k}$
Rank I_f and insert most important features into set S
return S

2.2.4 Boruta

In 2010, Miron B. Kursa cite kursa2010feature introduced Boruta. As a new feature selection method, Boruta is very similar to Permutation, it differs from permutation in that it does not shuffle one feature at a time, it will shuffle each feature in a single run and then construct shadow features, which will be talked about later. The key ideas of Boruta are shadow features, impurity, Z-score and binomial distribution, which will be introduced one by one.

Shadow Features First, Boruta creates shadow features with the given dataset. In Boruta, features do not compete with other features. They will compete with newly created features, which are called shadow features. Boruta will randomly shuffle each feature and get the shadow dataset. Boruta will then merge the shadow dataset with the original dataset to get a new dataset that ends up with twice the number of columns as the old data frame.

Impurity After getting the new dataset, Boruta will use a model such as RF to train it and understand the importance of each feature (including shaded features). Here the importance is calculated using the GINI index or RSS as a criterion, which is shown in CART 2.2.2.

Z-score Boruta uses Z-score as a criterion for all features, and Boruta calculates the Z-score several times in succession. The Z-score is calculated by the following formula:

$$Z_{ij} = \frac{\overline{G_{ij}}}{\sum_{j=1}^{m} \overline{G_{ij}}},\ \overline{G_{ij}} = \frac{\sum_{k=1}^{T} G_{ij}^{k}}{T} \tag{19}$$

where, G_{ij}^{k} means the importance of j feature in k_{th} tree of i_{th} iteration and we totally have T trees. And the Z-score is shown like this:

$$\begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1m} & Z_{11}^{s} & \cdots & Z_{1m}^{s} \\ Z_{21} & Z_{22} & \cdots & Z_{2m} & Z_{21}^{s} & \cdots & Z_{2m}^{s} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ Z_{n1} & Z_{n2} & \cdots & Z_{nm} & Z_{n1}^{s} & \cdots & Z_{nm}^{s} \end{bmatrix}$$

Where Z_{ij}^{s} means the feature importance of shadow feature, now we compare the importance of each original feature with a threshold value, which is defined as the highest feature importance of

the shadow feature and can be expressed as $Z_n^{max} = max\{Z_{n1}^s, Z_{n2}^s..., Z_{nm}^s\}$. When the importance of a feature is above a threshold, we can call it a "hit" and we can get the number of "hits" for each original feature in n iterations.

Binomial distribution Boruta uses a binomial test to determine if a feature gives rise to a importance that is significantly lower than its shadow counterpart. With this test Boruta classifies the features into three categories: unimportant, important and uncertain, and in subsequent experiments we remove only the unimportant features.
Therefore, this section describes how Boruta determines the importance of features and selects them. Boruta's pseudo-code is shown in Alg. 3.

Algorithm 3 Boruta

Input: Dataset S, Iteration I, Random Forest method RF, tail accounts p_t
Output: List of Importance L

```
Generate empty List L
while Not reach iterations do
    Randomly shuffle each feature of the original dataframe and get the shadow dataframe, contract the shadow dataframe with the original dataframe to obtain a new dataset S_new
    Train model RF with S_new
    Calculate Z-score of each feature
    Z_max is the biggest Z-score of shadow features
    for each feature i of original dataframe do
        if Z_i larger than Z_max then
            Hit_i = Hit_i + 1
        end if
    end for
end while
Calculate the binomial distribution of I iterations and the probability p = 0.5
Append the feature which is in area of irresolution and area of acceptance into List L
return List L
```

2.2.5 Shapley Additive Explanations

In 2017, Lundberg, Scott M and Lee, and Su-In [24] proposed a unified framework. shapley, to help understand a model and to be able to obtain the importance of each feature. They argue that their Shapley provides better understanding of features compared to traditional methods.
This section will introduce a new approach Shapley Additive Explanations (SHAP) based on Shapley Value [6], which is a concept of using cooperative game theory. Then in the case of computer problems, SHAP uses the idea based on Shapley Value to rank the importance of features and select the most important ones. And unlike the traditional methods discussed before, SHAP focuses more on the importance of local features, and it can also integrate the importance of local features to find the importance of global features.

The main idea behind SHAP is the Shapley value, which Shapley created in 1953. It is a method of allocating expenses to players based on their contribution to the total expenses. Players cooperate in coalitions and receive a certain amount of revenue from this cooperation. And this can be used in machine learning models as well. Features are seen as "contributions" and SHAP can tell

us the impact of each feature in each sample. It can also tell us the impact of all the data. Here's an example to help understand how SHAP works.

Imagine that we have a model that uses features A, B and C to obtain predicted values V. SHAP considers every possible combination of feature outcomes to determine the importance of individual features; in this case, we have eight feature combinations because $2^3 = 8$. These combinations are shown in the figure 1. Each node represents a feature combination and each row represents a feature that is not included in the previous node. Now SHAP will train a model for each combination. Then, we take a new data x_0 and get the predicted value for each combination, and we get the result as shown in the figure 2. So, if we want to calculate the Shapley value of feature A we can do so.

$$\begin{aligned} SHAP_A(x_0) =& w_1 \times MC_{A,\{A\}}(x_0) + w_2 \times MC_{A,\{A,B\}}(x_0) + \\ & w_3 \times MC_{A,\{A,C\}}(x_0) + w_4 \times MC_{A,\{A,B,C\}}(x_0) \end{aligned}$$

where $MC_{A,\{A\}}(x_0) = Prediction_A(x_0) - Prediction(x_0) = 40 - 50 = -10$ and the contribution marginal weight of an feature-model is the reciprocal of the number of possible contributions of all feature-models and it also be write like this: $f \times \binom{F}{f}$, where f means the number of feature in current combination and F means the amount of all features. Therefore, $w_1 = w_4 = \frac{1}{3}$, $w_2 = w_3 = \frac{1}{6}$. Thus,

$$\begin{aligned} SHAP_A(x_0) =& (1 \times \binom{3}{1})^{-1} \times MC_{A,\{A\}}(x_0) + (2 \times \binom{3}{2})^{-1} \times MC_{A,\{A,B\}}(x_0) + \\ & (2 \times \binom{3}{2})^{-1} \times MC_{A,\{A,C\}}(x_0) + (3 \times \binom{3}{3})^{-1} \times MC_{A,\{A,B,C\}}(x_0) \\ =& -11.33 \end{aligned}$$

and we can also get the Shapley value of feature B: $SHAP_B(x_0) = -2.33$ and C: $SHAP_C(x_0) = 46.66$ and the mathematics explanation of Shapley values is like this:

$$\phi_i = \sum_{S \subseteq M \ \{i\}} \frac{|S|!(M - |S| - 1)!}{|M|!} [f(S \cup i) - f(S)] \tag{20}$$

where, S refers to a subset of features that doesn't include the feature i and $S \cup i$ is the subset that include features in S and also plus feature i and M means total set of all features.

Samely SHAP evaluates the importance of each feature in its unique way and is able to select the important features.

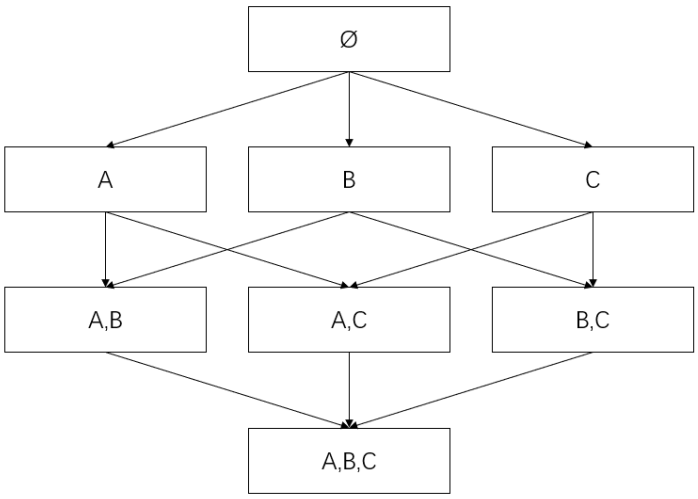

Figure 1: Each node in the figure represents one combination of features, each line means a feature which is not contained in former node

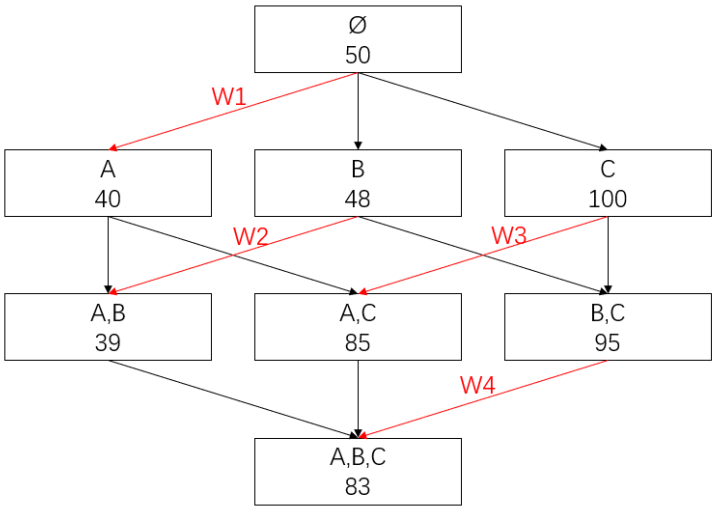

Figure 2: This figure also shows the predicted value of each model with the new data x_0

3 Methods

3.1 CMA-ES with Dimensionality Reduction Methods

This section will specifically talk about how to combine CMA-ES with the dimensionality reduction method. It is well known that there are two important steps in CMA-ES (2.1) **sample** and **Update**. In this thesis, we try to dynamically insert our dimensionality reduction method into the **Sample** part, which means that the dimensionality of the variables obtained from the **Sample** part of CMA-ES is different during each iteration.At the beginning of each iteration of the Sample operation, we will apply a dimensionality reduction method to the population, which can also be called elite, based on the population generated in the previous iteration. For the feature mapping approach, we will get a conversion matrix that can converts the covariance matrix C from high to low dimensions during sampling in CMA-ES; for the feature selection approach, we will get a ranking based on importance assessment, representing which features are important, and applying this ranking will also convert the covariance matrix from high to low dimensions. After obtaining the low-dimensional populations, our new method also needs to invert the low-dimensional variables back to the original dimensions for the subsequent **Update** section. Note that we will not use our downscaling method in the first iteration of CMA-ES, because in the first iteration CMA-ES does not have the populations from the previous iteration. Figure 3 shows the flow of this step, and all the dimensionality reduction used in our thesis adhere to this plan.

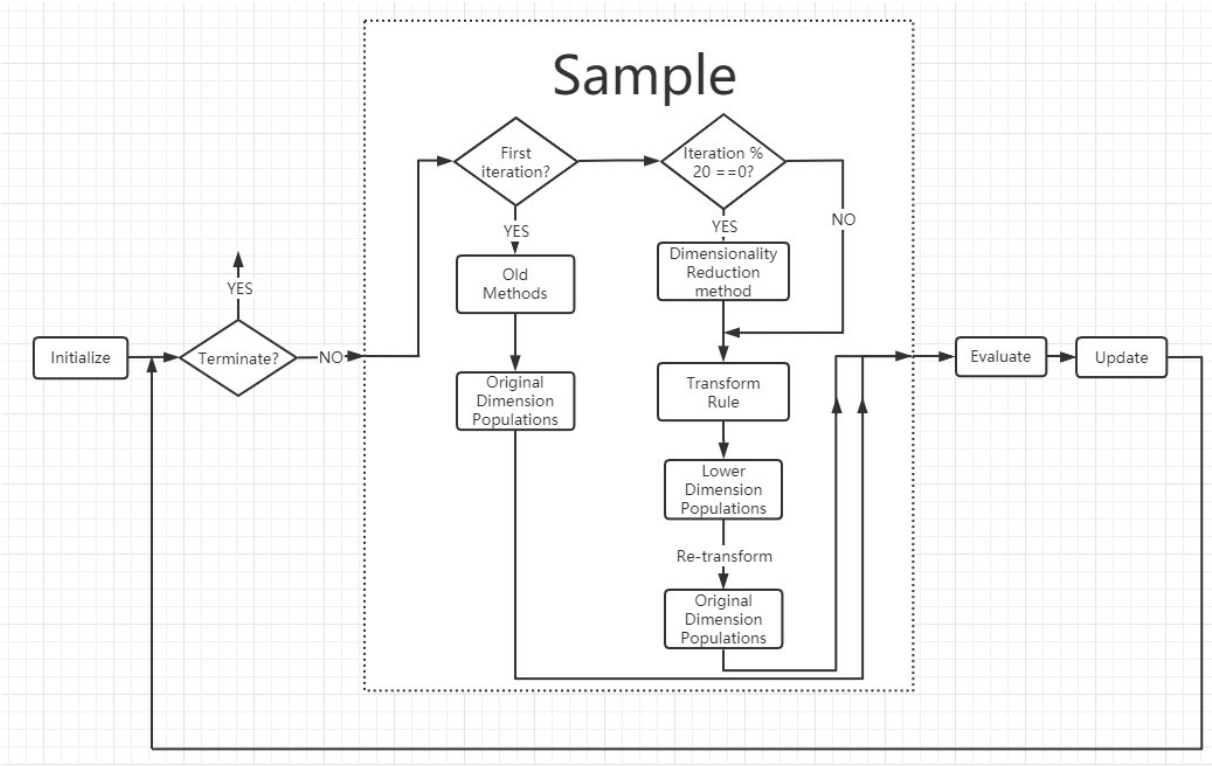

Figure 3: This figure shows how dimensionality reduction methods do in **Sample** Operation of CMA-ES

3.1.1 CMA-ES with PCA

The first algorithm to combine with CMA-ES is PCA, and the previous section shows that CMA-ES has two important parts, sampling and updating. Our proposed merging scheme is to combine PCA with the sampling method of CMA-ES by training PCA with the population data of the previous iteration before sampling starts, and obtaining the PCA transformation matrix P. The conversion matrix is then used to convert the covariance matrix of CMA-ES from high-dimensional to low-dimensional back to generate low-dimensional samples, and then the pseudo-inverse matrix P^T of the conversion matrix P is used to convert the low-dimensional samples to the original dimension. This concludes the merging of CMA-ES with PCA. Next, an example is used to help understand how this process works.

Before discussing the example we need to know that the calculation of the covariance satisfies the following equation:

$Cov(x_1 + x_2, y) = Cov(x_1, y) + Cov(x_2, y)$ and $Cov(a * x_1, b * x_2) = a * b * Cov(x_1, x_2)$

If the original dimension of the problem is three-dimensional, then the covariance matrix of three dimensions can be obtained $C^{old}\begin{bmatrix} c_{11}^{old} & c_{12}^{old} & c_{13}^{old} \\ c_{21}^{old} & c_{22}^{old} & c_{23}^{old} \\ c_{31}^{old} & c_{32}^{old} & c_{33}^{old} \end{bmatrix}$. Also using the population data from the previous iteration, PCA is applied to obtain the transformation matrix $P\begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$ This matrix converts 3D (x_1, x_2, x_3) into 2D (y_1, y_2), and the expression for the conversion can be written as follows:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P\begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

So the new variables are represented as follows: $y_1 = x_1 - x_3$ and $y_2 = -x_1 + x_2$. Using the newly obtained low-dimensional variables, we calculate the covariance $Cov(y_1, y_2)$ between the new variables y_1 and y_2:

$$\begin{aligned} Cov(y_1, y_2) &= Cov(x_1 - x_3, -x_1 + x_2) \\ Cov(x_1 - x_3, -x_1 + x_2) &= -Cov(x_1, x_1) + Cov(x_1, x_3) + Cov(x_1, x_2)Cov(x_1, x_2) \\ Cov(y_1, y_2) &= -c_{11}^{old} + c_{13}^{old} + c_{12}^{old} - c_{12}^{old} = c_{13}^{old} - c_{11}^{old} \\ Cov(y_1, y_2) &= c_{13}^{old} - c_{11}^{old} \end{aligned}$$

Similarly the covariance of the new variable with itself can be calculated to obtain the new low-dimensional covariance matrix. In general, with the given Covariance Matrix C^{old} and Transform Matrix P generated by PCA. The rules of gettting new $C_{i,j}^{new}$ can be like this:

$$C_{i,j}^{new} = \sum_{a=1}^{n}\sum_{b=1}^{n} P_{i,a}P_{j,b}Cov_{a,b}^{old},\ i, j <= m \tag{21}$$

After getting the new Covariance Matrix C^{new}, the sample formula is:

$$x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)}y_i^{(t+1)}\ where\ y_i \sim N(0, C_{new}^{(t)}) \tag{22}$$

and with the sampled solutions Y_{new}, CMA-ES-PCA will use the pseudo-inverse matrix P^T to get the population X: $X = P^T Y_{new}$

The complete algorithm is shown in Alg. 4. The bolded lines 6-11 explain how the new method combines CMA-ES and PCA. PCA obtains the transformation matrix(PCA retains 90% of the variance in experiment) from the population of the previous generation, and then transforms the covariance matrix C from the original dimensional matrix to a low-dimensional matrix. The low-dimensional

sample is obtained by sampling the low-dimensional CMA-ES, and the original dimensional population is obtained by multiplying it with the pseudo-inverse matrix of the transformation matrix. One thing to note is that the method does not use PCA to obtain the transformation matrix at every iteration, but updates the transformation matrix after every 20 generations.

Algorithm 4 Covariance Matrix Adaptation Evolution Strategy with PCA

Input: Learning rates: α_μ, α_σ, α_{cp}, α_{c1}, $\alpha_{c\lambda}$
Input: Generation Count: t = 0
Input: Attenuation Factor: d_σ
Input: Evolutionary Paths: $p_\sigma^{(0)} = 0$, $p_c^{(0)} = 0$
Input: Default Covariance Matrix: $C^{(0)} = I$
Output: $\mu^{(t)}, \sigma^{(t)}, C^{(t)}$

Sample $x_i^{(1)} = \mu^{(0)} + \sigma^{(0)} y_i^{(1)}$ where $y_i \sim N(0, C^{(0)})$, $i = 1, ..., \Lambda$
Evaluate $x_i^{(t+1)}$'s fitness, $i = 1, ..., \lambda$ of each candidate
Select top λ samples based on the fitness
Update the parameters
while Not hit stopping criteria **do**
  if The number of iterations is a multiple of twenty **then**
    Get the PCA Matrix P with the last generation's population
  end if
  Transform the Covariance Matrix from $C_\Lambda^{(t)}$ to $C_\Theta^{(t)}$ with the PCA Matrix P
  Sample $x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)}$ where $y_i \sim N(0, C^{(t)})$, $i = 1, ..., \Theta$
  Re-map $x_i^{(t+1)} = P^T x_i^{(t+1)}$
  Evaluate $x_i^{(t+1)}$'s fitness, $i = 1, ..., \lambda$ of each candidate
  Select top λ samples based on the best fitness
  $\mu^{(t+1)} \leftarrow \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})$
  $p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma (2 - \alpha_\sigma) \lambda} C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
  $\sigma^{(t+1)} \leftarrow \sigma^{(t)} exp(\frac{\alpha_\sigma}{d_\sigma} (\frac{||p_\sigma^{(t+1)}||}{E||N(0,I)||} - 1))$
  $p_c^{(t+1)} \leftarrow (1 - \alpha_{cp}) p_c^{(t)} + \sqrt{\alpha_{cp} (2 - \alpha_{cp}) \lambda} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
  $C^{(t+1)} \leftarrow (1 - \alpha_{c1}) C^{(t)} + \alpha_{c1} p_c^{(t+1)} p_c^{(t+1)^T}$
  $C^{(t+1)} \leftarrow (1 - \alpha_{c1} - \alpha_{c\lambda}) C^{(t)} + \alpha_{c1} p_c^{(t+1)} p_c^{(t+1)^T} + \alpha_{c\lambda} \sum_{i=1}^{\lambda} w_i y_{i:\lambda}^{(t+1)} y_{i:\lambda}^{(t+1)^T}$
  $t + 1 \leftarrow t$
end while
return $\mu^{(t)}, \sigma^{(t)}, C^{(t)}$

3.1.2 CMA-ES with Feature Selection

In this thesis, we chose four methods: RF, Permutation, Boruta and SHAP as feature selection to combine with CMA-ES. Similar to CMA-ES-PCA, we mainly applied the feature selection methods to the sampling process of CMA-ES. Using the last generation population, different feature selection methods can obtain the importance of each feature (variable), and then select the important features and discard the unimportant ones in order of importance. In our experiments, we take the importance values obtained by the feature selection methods as a percentage (the sum of the percentages of all features is 100%) and keep the features with 90% importance.

Subsequently, the new method obtains a low-dimensional covariance matrix by saving the rows inside the covariance matrix that belong to the important features and eliminating the rows that do not belong to the important features. The same sampling process is then performed to obtain the low-dimensional samples. For the unimportant features of the current generation, the value of these features from the previous iteration is used to gain the original dimensional population. The feature selection method is also used to find the importance of features every 20 generations and the feature selection method is not used in the first iteration because the feature selection methods need to find the importance of features by population.

The pseudo-code of CMA-ES with feature selection is shown in Alg. 5. This algorithm is similar to the above algorithm combining CMA-ES and PCA, the difference is that the Update parameter is simplified and the bolded lines 6-11 show how the new method works. line 7 uses feature selection to know which features are important, and then the covariance matrix is dimensioned down according to the important features to obtain a low Then, the covariance matrix is downscaled based on the important features, and then the missing values are inserted to obtain the final population.

Here is an example to help understand how CMA-ES-Feature-Selection works. If the original dimension of the problem is three, then the covariance matrix must also be three-dimensional and can be expressed as follows:

$$C^{old} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

In the current iteration, if a list of important features $L[1, 2]$ is obtained, which contains the important features selected by the feature selection method based on the population of the previous iteration. In this example, the feature selection method concludes that features one and two are more important than feature three. The new low-dimensional covariance matrix can be obtained by keeping the first two rows and the first two columns of the three-dimensional covariance matrix:

$$C^{new} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

For the general case. By the obtained list L containing the important features. The rule for obtaining the new covariance matrix C^{new} is to keep the columns and rows in the list L

$$C^{new}_{i,j} = C^{old}_{i,j},\ i \in L, j \in L \tag{23}$$

And sample the new populations by using the new Covariance Matrix C^{new}:

$$x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)}\ where\ y_i \sim N(0, C^{(t)}_{new}) \tag{24}$$

By this method, we are able to obtain the new low-dimensional population p_{new}. As for the unimportant features of the current generation, the value of these features from the previous iteration is used to fill to gain the original dimensional population.

All feature selection methods in this section are similar. The only difference is that each feature selection method has its unique way to select features.

3.2 Summary

In this section, we will describe how to combine CMA-ES with two types of dimensionality reduction methods: feature projection and feature selection. Different dimensionality reduction methods have their set of means to accomplish the downscaling effect during CMA-ES sampling, and each of these dimensionality reduction methods has been proven to be effective in their domain. Therefore, theoretically, our chosen dimensionality reduction method can make our variables maintain the original features of low dimensionality during the iterative process of CMA-ES populations. However, the specific performance depends on the subsequent experimental results and analysis.

Algorithm 5 Covariance Matrix Adaptation Evolution Strategy with Feature Selection Method

Input: Parameters: α_μ, α_σ, α_{cp}, α_{c1}, $\alpha_{c\lambda}$, d_σ, $p_\sigma^{(0)} = 0$, $p_c^{(0)} = 0$
Input: Generation Count: t = 0
Input: Feature Selection Method: FS
Output: $\mu^{(t)}, \sigma^{(t)}, C^{(t)}$

Sample $x_i^{(1)} = \mu^{(0)} + \sigma^{(0)} y_i^{(1)}$ where $y_i \sim N(0, C^{(0)})$, $i = 1, ..., \Lambda$
Evaluate $x_i^{(t+1)}$'s fitness, $i = 1, ..., \lambda$ of each candidate
Select top λ samples based on the fitness
Update the parameters
while Not hit stopping criteria **do**
  if The number of iterations is a multiple of twenty **then**
    Get the Important features IF with the last generation's population
  end if
  Transform the Covariance Matrix from $C_\Lambda^{(t)}$ to $C_\Theta^{(t)}$ with the IF
  Sample $x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)}$ where $y_i \sim N(0, C^{(t)})$, $i = 1, ..., \Theta$
  Complement the values of unimportant features
  Evaluate $x_i^{(t+1)}$'s fitness, $i = 1, ..., \lambda$ of each candidate
  Select top λ samples based on the best fitness
  Update all parameters
  $t + 1 \leftarrow t$
end while
return $\mu^{(t)}, \sigma^{(t)}, C^{(t)}$

4 Experiments

The purpose of the experiment we designed is to solve the three research questions we mentioned above: how the dimensionality reduction method affects CMA-ES, whether the variables reliable after dimensionality reduction, and whether the dimensionality reduction method can shorten the time consumption of CMA-ES. Therefore, the subsequent experiment can be divided into two parts, the first part is used to try to answer the first two questions, and the second part is to try to answer the last question.

4.1 Impact of dimensionality reduction and the reliable of variables generated by dimensionality reduction method

For the effect of the Dimensionality Reduction methods on the general of CMA-ES, we want to record the new dimensions obtained in the Dimensionality Reduction methods in each iteration and the average of the evaluated values of the target equations in each iteration. The best-so-far method is used to visualize the evaluated values and to analyze the similarities and differences of the different Dimensionality Reduction methods. The x- and y-axes are plotted logarithmically with e as the base to facilitate the reading and analysis. To determine the validity of the variables obtained by the Dimensionality Reduction method, we mainly want to figure out if the new method is as effective as the original CMA-ES in obtaining the optimal solutions.

4.1.1 CPU Time

To answer whether dimensionality reduction methods combined with CMA-ES can reduce the computation cost, we recorded the CPU time consumed by each method for each function in each dimension. In addition, we also record the total number of iterations for each method in each dimension for each problem going from the analysis.

4.2 Experimental Setup

All new methods are tested on ten multi-modal functions taken from the BBOB problem set [25] and compare their experimental result with CMA-ES. The methods were tested on functions F15-F24. F15-F19 are multi-modal functions with an adequate global structure, while F20-F24 are multi-modal functions with a weak global structure. All methods are tested on dimensions $D \in \{10, 40\}$ with $D \times 10^3$ function evaluations. The experiment is terminated by reaching a specified number of iterations, or the difference of evaluation value of two consecutive iterations is less than $1e^{-11}$. To make a statistically significant comparison, we conducted 30 independent runs for each pair of the algorithm, problem, and dimension (the problem instance is fixed to zero). Moreover, In ten dimensions, we added supplementary experiments to the new five methods, and supplementary experiments represent a certain probability that CMA-ES does not use the downsampling method in the sampling process but uses the original sampling method(this probability is 20% in our experiment). The experiment is worked on an Intel Intel® Core™ i7-9700K Processor @ 3.6GHz machine with single-thread mode.

5 Discussion

This section and the next section analyze the experimental results and attempt to draw a summative conclusion for the research question. This section is also divided into two parts based on our research question.

5.1 Dimensionality Reduction's general impact on CMA-ES and the reliable of lower variables

The most intuitive way to investigate the widespread impact of the dimensionality reduction method on CMA-ES is to compare the results of the new method and the old method in the same graph. Figure 4, Figure 5, Figure 6, record the evaluation value in each iteration, the average value of the final iteration, and the dimensionality change of the dimensionality reduction method during the iteration. In Figure 4, the length of different lines is different. The reason for this is that CMA-ES has its unique termination criteria. If CMA-ES reaches a specified number of iterations or the difference of evaluation value of two consecutive iterations is less than $1e^{-11}$, it will be eliminated.

5.1.1 Dimension 10

Overall, CMA-ES performs well for almost all functions, as can be seen in Figure 4, where CMA-ES converges quickly, and in Figure 5, where CMA-ES obtains the best optimal values in most functions. For the dimensionality reduction with CMA-ES, feature projection seems to be better than the feature selection. In all functions, PCA performs roughly similar to CMA-ES, the only difference is that PCA requires more iterations to converge, but it finds better results than CMA-ES for some specific functions. At the same time, we can also discover from the dimension diagram that PCA usually reduces the dimension to less than half of the initial dimension, which means that the Low-dimensional variables obtained from PCA can be relied on.

For the feature selection methods, Boruta stands out from the rest, while the other three algorithms: RF, Permutation, and SHAP performed relatively poorly in our experiments. As to why Boruta is much better than other feature selection methods, even getting better results than CMA-ES in some functions such as F16, F17, and F18. The dimensional plots in Figure 6 give some evidence. Moving average [26] is used to facilitate the analysis, the figure without the moving average method is shown in Appendix B. This graph shows the dimensionality change of the different methods at different iterations with different functions. What is more, we list the statistics of dimensional change in Appendix B. Here we give a part of this table, which is shown in Table.1. This table contains the dimensional changes of all methods only in functions 15 and 16 when the problem's dimension equals 10. By comparing the picture and the table, we found that Boruta does not have a strong dimensionality reduction effect than other feature selection methods. Unlike other algorithms that move to reduce the dimensionality to 1 or 2 dimensions, Boruta often retains the high dimensionality. For example, for function 16, Boruta considers all variables essential and no downscaling 37.66% of the total time. This means that when Boruta is combined with CMA-ES, the covariance matrix of CMA-ES retains original dimension at a specific period. Then naturally, the results obtained are relatively better than other feature selection methods.

	PCA		Boruta		RF		Permutation		SHAP	
D10	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)
F15	10	0.06	10	3.45	7	4.33			7	4.28
	5	4.54	6	26.27	6	12.99	10	2.38	5	4.50
	4	29.90	2	29.56	5	26.41	3	45.24	4	36.94
	3	58.60	1	40.72	4	38.69	2	52.38	3	4.50
	2	6.90			3	12.20			2	49.55
F16	5	16.14	10	37.66	6	17.18			6	4.33
	4	39.88	6	38.17	5	6.04	10	1.22	5	4.33
	3	18.69	2	8.06	4	13.75	2	47.56	4	26.41
	2	19.94	1	16.11	3	10.31	1	51.22	3	12.99
	1	5.30			2	45.02			2	47.62

Table 1: This table shows the dimension changes in dimensionality reduction for the **10D** problem, where only F15 and F16 are listed; the whole table is in the Appendix 3. In this table, D represents the unique value of reduced dimensions across all repetitions on this function. The percentage represents the percentage of times that dimension appears in all iterations. For each cell of data, only the top 5 dimensions in terms of the percentage of occurrences are kept.

5.1.2 Dimension 40

For the experiments results performed in high dimension. With the given Figure 4 5 6, we can conclude that PCA performs roughly the same as at low dimensions. It can obtain the optimal solution similar to the original CMA-ES and converges faster than other dimensionality reduction methods. For the feature selection method, Boruta does not show its absolute lead in the ten-dimensional experiments to other methods, and the lead is small. According to Table 4, we discovered that Boruta rarely considers all variables to be important in the 40-dimensional condition. Here a part of the table is given, which only contains the dimension changes in function 15 and 16, shown as Table 2. Boruta generally considers half or less of the variables to be important. Therefore the covariance matrix of CMA-ES will often reduce the dimensionality during the iterative process which is different from what it does in ten dimensions. So Boruta does not have a large lead in higher dimensions. As for other feature selection methods, although they are not as good as CMA-ES or

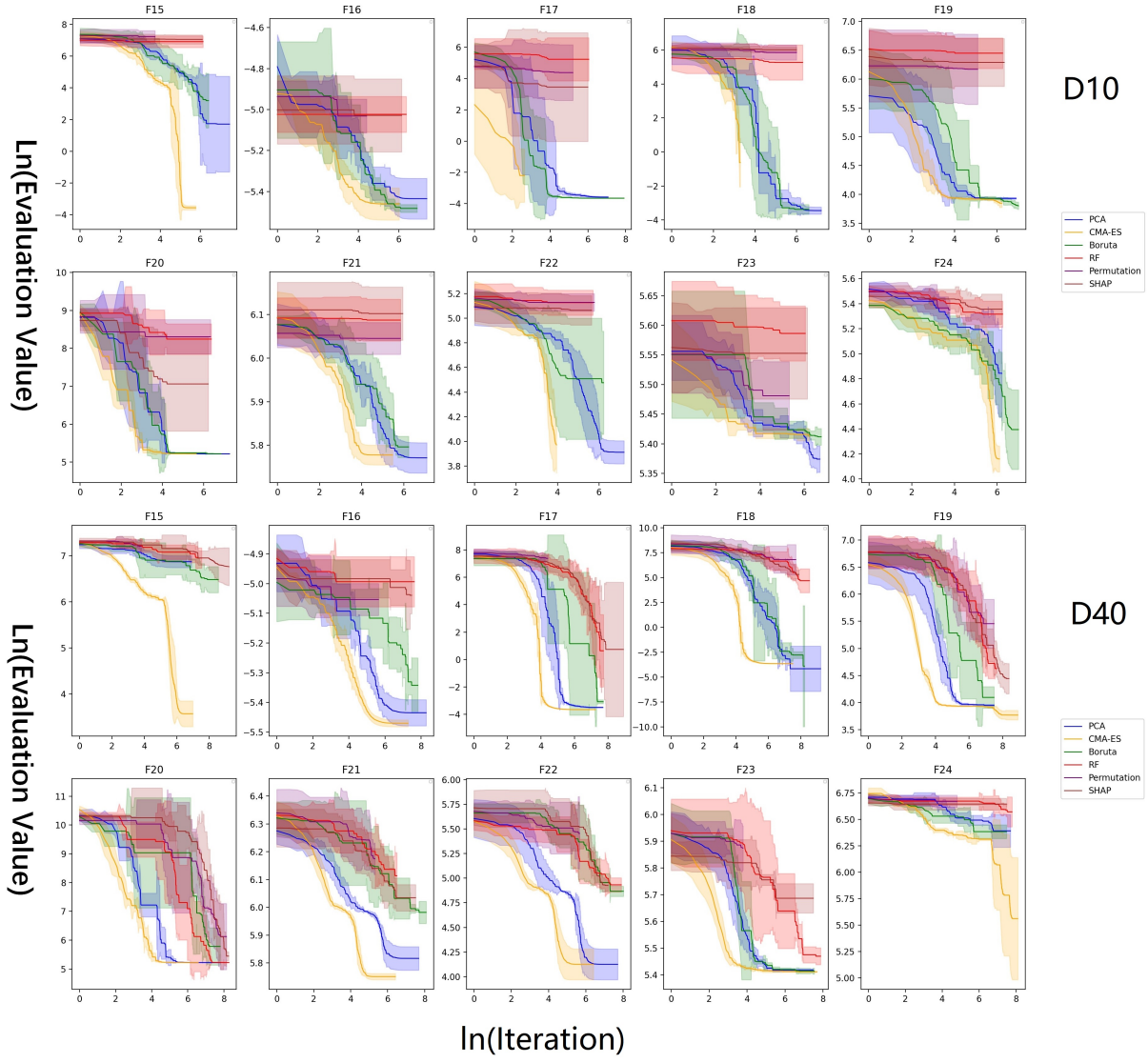

Figure 4: This figure records the performance of the dimensionality reduction method in combination with CMA-ES and the initial CMA-ES in 10 and 40 dimensions at different functions. The top one represents 10 dimensions, and the bottom one represents 40 dimensions. The horizontal coordinates represent the number of iterations, and the vertical coordinates represent the evaluation value at the current number of iterations. Both X and Y axes are logarithmically processed with e as the base. The shaded area indicates the 95% confidence interval of the mean target precision. The linear-scaled version can be found in Appendix A

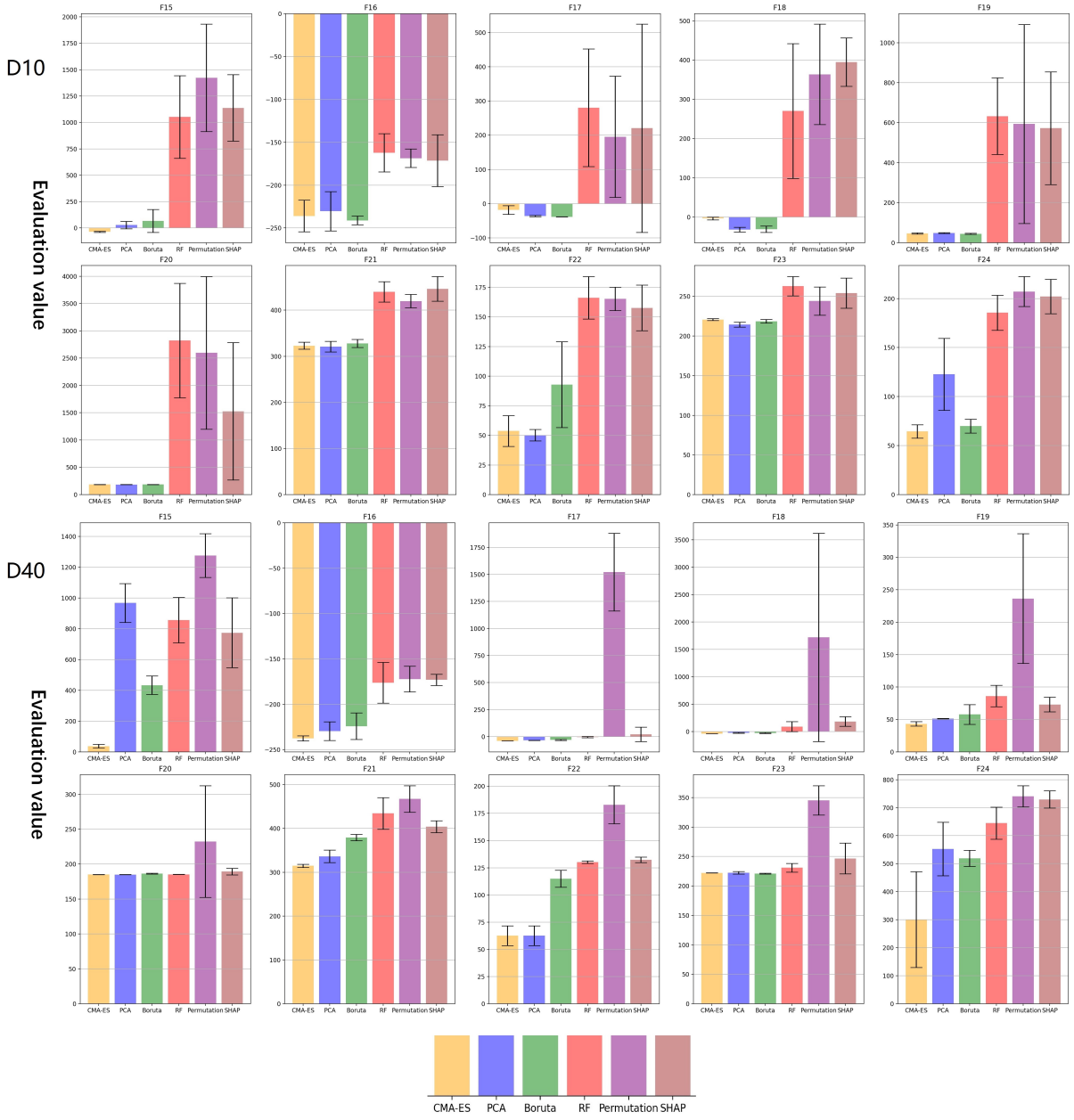

Figure 5: This figure explains the final results of the dimensionality reduction method combined with CMA-ES and general CMA-ES in 10 and 40 dimensions at different functions. The top one represents 10 dimensions, and the bottom one represents 40 dimensions. The different bars represent different methods, and the Y-axis represents the optimal value of each method under the current functions. The error bars used here donate the standard deviation

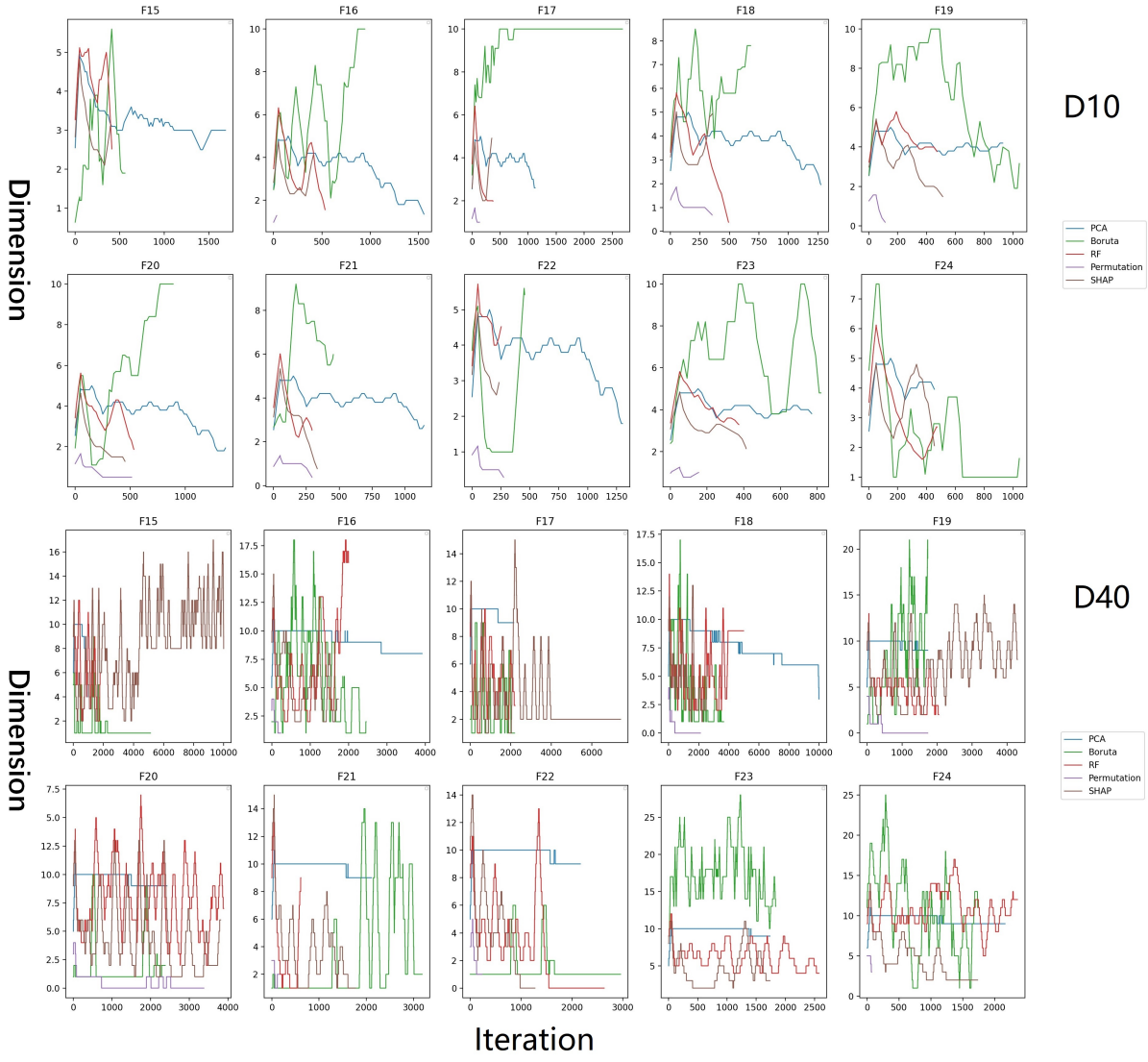

Figure 6: This figure records the dimensionality of the samples generated by the dimensionality reduction method combined with CMA-ES for each iteration with different functions in 10 and 40 dimensions. The moving average method is applied here to facilitate the observation of dimensional changes, and the size of the sliding window is 100. the top part of this plot represents 10 dimensions, and the bottom part represents 40 dimensions. The x-axis represents the number of iterations, and the y-axis represents the dimensionality. The graph without using the sliding window can be found in Figure.16

PCA, they still demonstrate their optimization ability and obtain optimal values similar to CMA-ES for some equations. And they all always turn 40-dimensional variables into about 10 in the iteration, which is similar to PCA in terms of the degree of dimensionality reduction.

	PCA		Boruta		RF		Permutation		SHAP	
D40	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)
F15	11	5.95	20	1.93	10	7.53	40	2.22	10	7.40
	10	56.51	4	60.39	8	8.61	10	42.22	9	5.00
	9	12.11	3	1.14	6	12.91	4	44.44	8	36.60
	8	10.09	2	11.20	4	18.24	2	11.11	4	7.80
	7	4.04	1	25.33	2	23.67			2	12.20
F16	40	0.03	20	12.75	12	6.90	40	0.35	8	8.97
	11	0.50	4	4.78	6	9.86	13	6.74	6	13.68
	10	42.84	3	3.95	4	16.76	4	7.09	4	14.57
	9	26.22	2	31.89	3	9.86	2	14.18	3	7.85
	8	30.41	1	38.62	2	20.70	1	71.63	2	29.15

Table 2: This table shows the dimension changes in dimensionality reduction for the **40D** problem, where only F15 and F16 are listed; the full table is in the Appendix B. In this table, D represents the unique value of reduced dimensions across all repetitions on this function. The percentage represents the percentage of times that dimension appears in all iterations. For each cell of data, only the top 5 dimensions in terms of the percentage of occurrences are kept.

5.1.3 Supplementary Experiment

Since in lower-dimensional problems, Boruta chooses to keep all variables during the iterations to obtain good experimental results. We decided to add one more set of experiments: under the ten-dimensional problem, for all the methods combining dimensionality reduction and CMA-ES. Suppose we deliberately choose to keep all variables at some points during the iterations (i.e., using the regular CMA-ES sampling method). Would it improve RF, Permutaion, and SHAP performance, which was mediocre in the original experiments? In the subsequent experiments, we set the probability p of this particular retention of all variables to 20%, leaving everything else unchanged to explore its impact. The results are shown in Figure 7, 8.

From the results, the main change is that RF, Permutation, and SHAP have improved compared to the previous ones, both in terms of iterative drop rate and error bar of the final results. For the error bar of the final results, we made a table 5, the average improvement rate for PCA came to 0.72%, for Boruta to 17.55%, for RF to 84.82%, for Permutation to 80.21% and SHAP to the average improvement came to 78.19%. All three algorithms: RF, Permutation, and SHAP, which were performed normally in the previous experiments, got about 80% improvement. Although Boruta's improvement was not significant, it should be noted that it obtained an optimal value similar to that of PCA after using this method, which means that its optimization ability is already identical to that of PCA.

The lack of improvement in PCA using the new method is understandable since PCA maps the low-dimensional variables back to higher dimensions with their inverse matrix without losing a lot of information during the inverse dimensionality reduction phase. The experimental results also confirm that the combination of PCA and CMA-ES does not differ significantly from the original CMA-ES in terms of optimization capability. PCA can even find better results than general CMA-ES on some specific functions. The only difference between them is the time, which will be discussed in the next section. Therefore, if the dimensionality reduction method is PCA, we can conclude that the

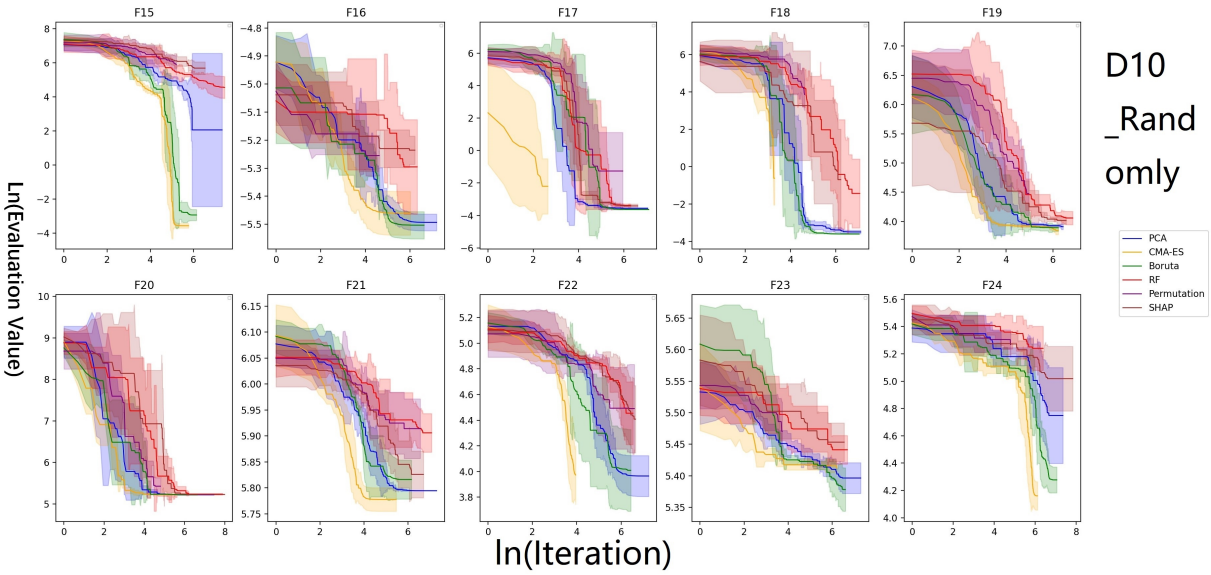

Figure 7: This graph records the evaluation value of each iteration of the dimensionality reduction method in 10 and 40 dimensions and different functions when "Guide" is added to the method (in the experiment, the probability is 20%). The shaded area indicates the 95% confidence interval of the mean target precision. Both X and Y axes are logarithmic with e as the base.

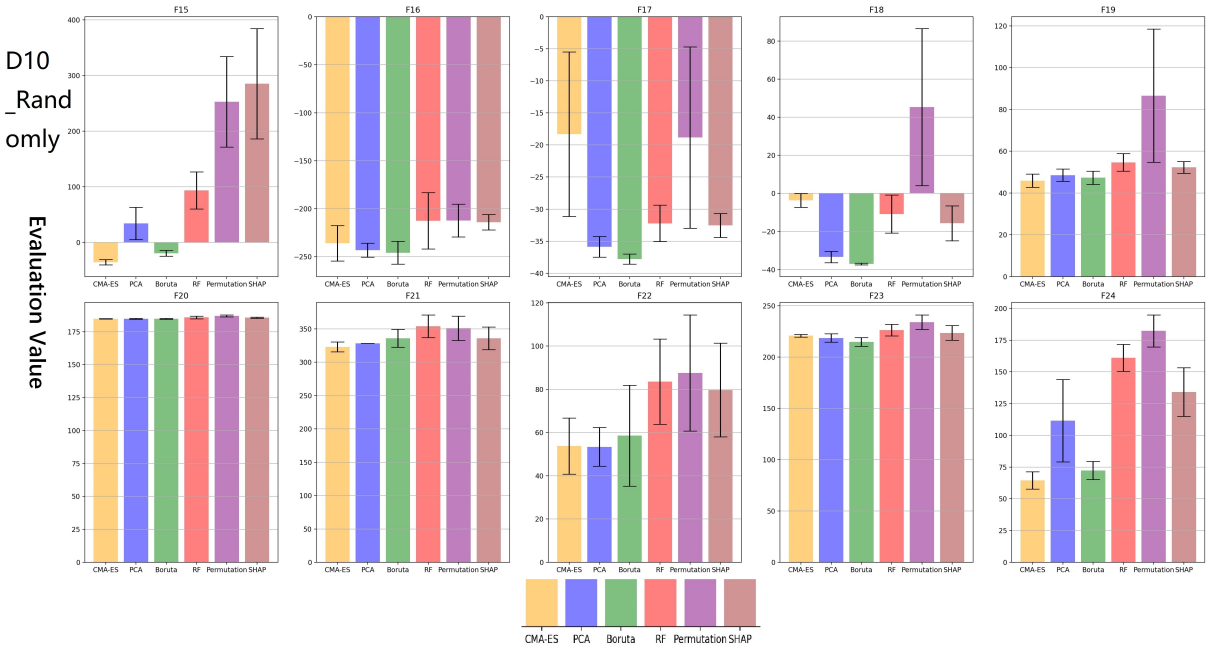

Figure 8: This figure explains the optimal values of different dimensionality reduction methods in different dimensions with different functions after using the "Guide" method. The different bars represent the different methods, and the Y-axis represents the optimal value of each method under the current function. The error bars used here are donated to the standard deviation

variables after dimensionality reduction are reliable. If the downscaling method is feature selection, the downscaled variables are less reliable than the feature projection method.

5.2 CPU Time

In this part, the main question we want to answer is whether the dimensionality reduction algorithm can reduce time consumption. Therefore, when doing the experiments, we recorded the CPU time of different methods for different functions in various dimensions and also the number of iterations at the end of the iteration. Since different methods have different iterations ending, we compare the CPU time consumed per iteration for each method as a basis for judging the computational consumption. We explored the time consumption of different methods by visualization. The results are shown in Fig 9, and the CPU time consumed by each iteration of each method is also recorded in Appendix D.

As can be seen from the figure, CMA-ES has the shortest computational consumption inside basically all algorithms for both high and low dimensional problems even though it consumes the most CPU time for some specific functions on low dimensions. Similarly, PCA and CMA-ES show similar results, with PCA outperforming all the feature selection methods in terms of computational consumption. Moreover, for low-dimensional problems, among the feature selection methods, Boruta tends to require more CPU time in a single iteration, Permutation follows, and then SHAP is the least time required method inside the feature selection algorithms.

In the graph of the results in 40 dimensions, we discover that CMA-ES does not perform as poorly as it does in functions 17, 18, and 22 in 10 dimensions. It is able to go inside most of the functions with a better score. In contrast, the dimensionality reduction methods do not appear to be as good. Our speculation is that each training time of each dimensionality reduction method's model increases after the dimensionality becomes higher, and the negative effect of this increase in training time outweighs the benefit to CMA-ES.

What is more, combined with Figure 6 and Table 6, although the single iteration of the dimensionality reduction method sometimes consumes less CPU time than CMA-ES in some specific functions. They also require more iterations to terminate the iterations, which will instead take more time than CMA-ES. Therefore, the answer to the research question of whether the dimensionality reduction algorithm can reduce the time consumption of CMA-ES is **No**. As for why the dimensionality reduction algorithm needs more iterations than CMA-ES, our deduction is that the dimensionality reduction algorithm can bring more search space to CMA-ES, which means that sometimes CMA-ES can jump out when it is stuck in a local optimum, and this can also explain why the combination of the dimensionality reduction algorithm with CMA-ES can sometimes obtain better results than usual. However, this also has the negative impact that the new CMA-ES cannot eliminate iterations quickly.

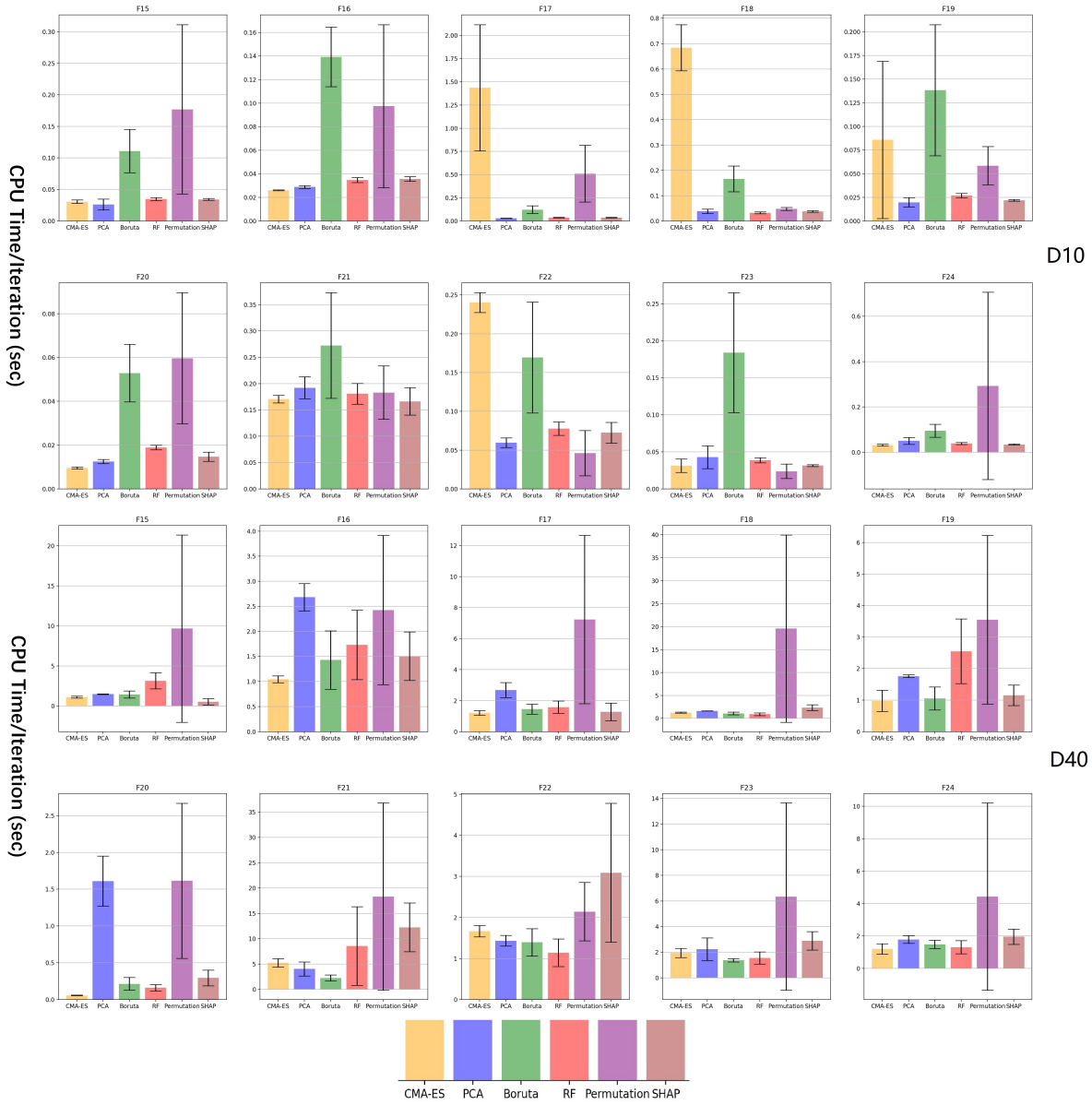

Figure 9: The graph explains the CPU time and the number of iterations required to complete the iterations for different methods with different functions in different dimensions. Different bars represent different method. The Y-axis is the CPU time divided by iterations. The error bars used here are donated by the standard deviation

6 Conclusions and Further Research

In general, the normal CMA-ES is better in terms of optimal values and number of iterations per function per dimension, followed closely by PCA. In terms of the optimal value results per function, PCA does not perform much worse than CMA-ES, especially in some specific functions, such as F16, F17. PCA shows better results than normal CMA-ES. We consider that adding PCA to CMA-ES can make the search space of CMA-ES larger, thus avoiding the dilemma of falling into the local optimum. However, the combination of PCA and CMA-ES requires more iterations and computation time.

For Boruta algorithm, the reason why it performs so well than other feature selection algorithms for low-dimensional problems is that it sometimes considers all variables to be important. So it keeps all variables, during which CMA-ES does not reduce the dimensionality, i.e., it performs particularly well as the regular CMA-ES. In high-dimensional problems, it does not keep all the variables compared to low-dimensional ones, so it does not outperform as much as before. For the RF, Permutation, and SHAP methods, they are not as good as Boruta for low-dimensional problems, but they do not perform much worse than Boruta for high-dimensional problems. For all feature selection algorithms, they are effective in combination with CMA-ES, but just not as effective as PCA and CMA-ES, because all feature selection algorithms need to eliminate some variables in the iterations, which is essential and useful for problems like sparse data filtering, but it is not particularly helpful for our experiments.

For the supplementary experiments, we draw inspiration from the Boruta experiments on whether we would have good results if we do not allow the dimensionality reduction method to reduce the dimensionality of CMA-ES at some specific times, i.e., using the traditional CMA-ES sampling method in some period. From my perspective, this method of keeping CMA-ES from being downsampled at certain times is a "Guide" that can lead the feature selection method down the right path if it goes off course. The RF, Permutation, and SHAP methods can achieve about 80% increase in the 10-dimensional problem if the "Guide" is used. Also, in some specific functions, F16, F17, F18, these new methods go beyond CMA-ES, which means the combination of the dimensionality reduction method and CMA-ES has the potential to do better than the normal CMA-ES. This can also be helpful for subsequent studies.

Last, the answer to our first research question: "What is the general impact of Dimensionality reduction method on the performance of CMA-ES?" is that all dimensionality reduction algorithms can make CMA-ES do its task and find the optimal value. However, feature selection is not as effective as feature projection algorithms. The combination of PCA and CMA-ES can find better results than traditional CMA-ES, although it requires more iterations to converge. However, according to the results of the supplementary experiment, we discovered that if we give a "Guide" to the dimensionality reduction methods during the iterative process, they can obtain a much better result and even better than the traditional CMA-ES in some specific functions. Then, the answer to the second research question about: "reliable of dimensionality reduction" is **Reliable**, which is described in detail in the supplementary experiment. For the last research question: "Does dimensionality reduction methods save time in the whole process of CMA-ES?" My answer is **No**. Although the dimensionality reduction algorithm gets shorter time consumption than CMA-ES under certain functions in some dimensions, its total time consumption increases due to the rise in the number of iterations. We consider this is because the dimensionality reduction method gives CMA-ES more space to search for the optimum, which prevents CMA-ES from falling into local optima, but with it comes an increase in the number of iterations and instability. So the dimensionality reduction method usually takes more time to find the optimum, and then it does not save time.

In further research, we want to improve our methods in four points:

1. Apply dimensionality reduction methods not only in the "Sample" part of CMA-ES but also in the "Update" part.

2. Use a longer history to train the dimensionality reduction methods.

3. Develop an algorithm that can automatically select the dimensionality reduction methods.

4. Add more Dimensionality reduction methods like LDA, T-SNE, and Autoencoders.

References

[1] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[2] Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, 2008.

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[5] Miron B Kursa, Witold R Rudnicki, et al. Feature selection with the boruta package. *J Stat Softw*, 36(11):1–13, 2010.

[6] Eyal Winter. The shapley value. *Handbook of game theory with economic applications*, 3:2025–2054, 2002.

[7] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[8] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[9] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[10] Ali Wagdy Mohamed. Solving large-scale global optimization problems using enhanced adaptive differential evolution algorithm. *Complex & Intelligent Systems*, 3(4):205–231, 2017.

[11] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

[12] Suresh Balakrishnama and Aravind Ganapathiraju. Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing*, 18(1998):1–8, 1998.

[13] D. Kapsoulis, K. Tsiakas, V. Asouti, and K. Giannakoglou. The use of kernel pca in evolutionary optimization for computationally demanding engineering applications. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

[14] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[15] Elena Raponi, Hao Wang, Mariusz Bujny, Simonetta Boria, and Carola Doerr. High dimensional bayesian optimization assisted by principal component analysis. In *International Conference on Parallel Problem Solving from Nature*, pages 169–183. Springer, 2020.

[16] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. Boa: The bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, volume 1, pages 525–532. Citeseer, 1999.

[17] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[18] Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14, 2000.

[19] Robert I Lerman and Shlomo Yitzhaki. A note on the calculation and interpretation of the gini index. *Economics Letters*, 15(3-4):363–368, 1984.

[20] Bjoern H Menze, B Michael Kelm, Ralf Masuch, Uwe Himmelreich, Peter Bachert, Wolfgang Petrich, and Fred A Hamprecht. A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC bioinformatics*, 10(1):1–16, 2009.

[21] Paul Geladi and Bruce R Kowalski. Partial least-squares regression: a tutorial. *Analytica chimica acta*, 185:1–17, 1986.

[22] Steven Abney. Bootstrapping. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 360–367, 2002.

[23] Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. Correlation and variable importance in random forests. *Statistics and Computing*, 27(3):659–678, 2017.

[24] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[25] Dimo Brockhoff, Tea Tusar, Anne Auger, and Nikolaus Hansen. Using well-understood single-objective functions in multiobjective black-box optimization test suites. *arXiv preprint arXiv:1604.00359*, 2016.

[26] Rob J. Hyndman. *Moving Averages*, pages 866–869. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

A The original result of experiment

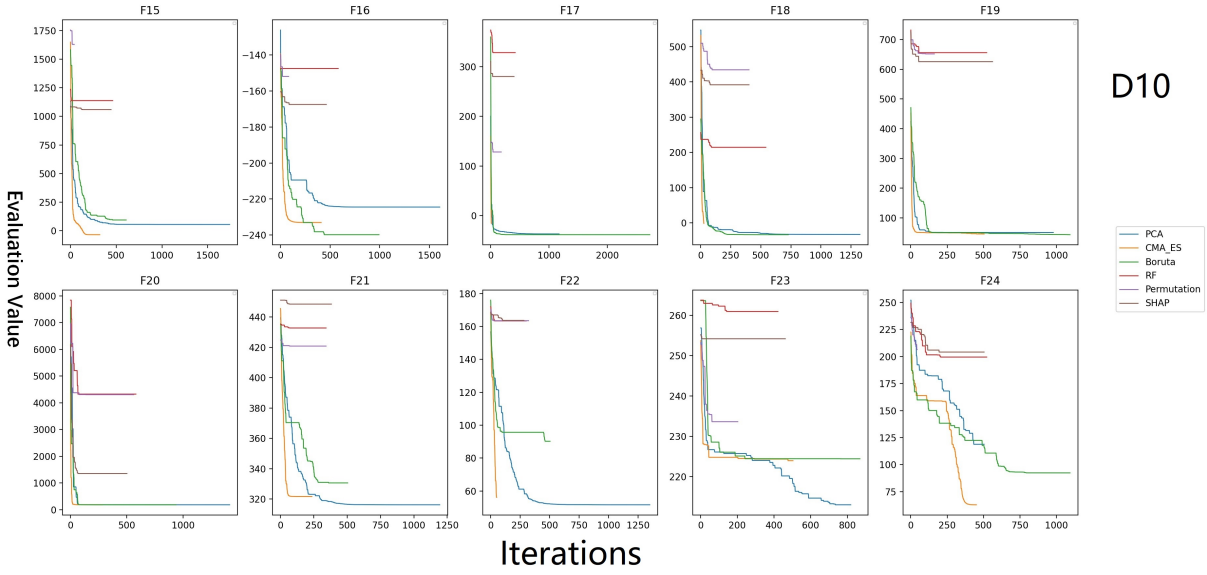

Figure 10: The figure documents the combination of the dimensionality reduction method with CMA-ES and the performance of CMA-ES in general under different functions in **10** dimensions. The horizontal coordinates represent the number of iterations, and the vertical coordinates represent the evaluated values at the current number of iterations.

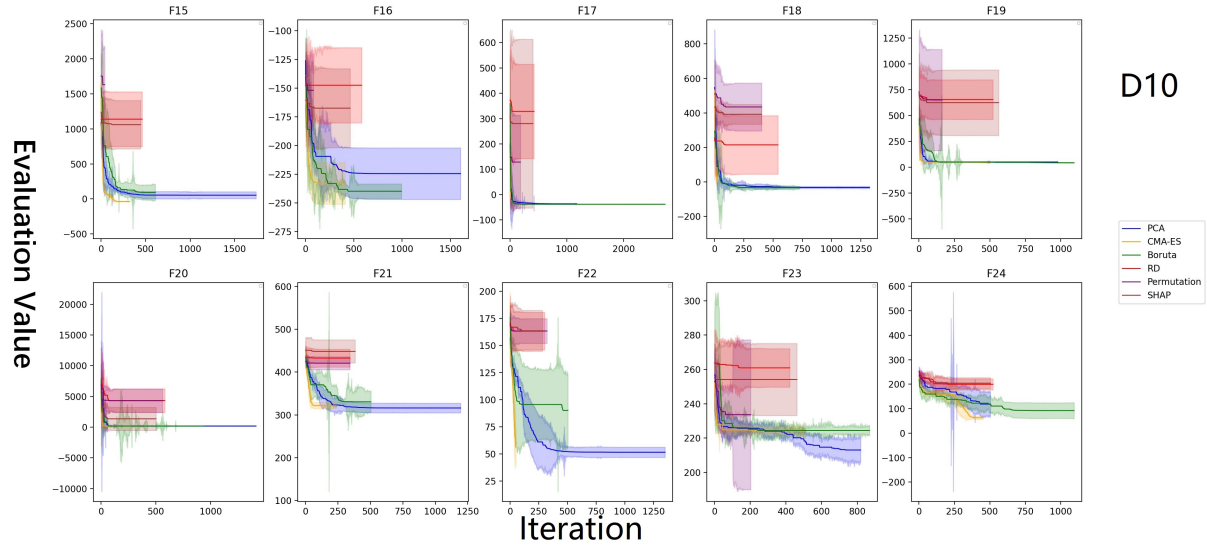

Figure 11: The figure documents the combination of the dimensionality reduction method with CMA-ES and the performance of CMA-ES in general under different functions in **10** dimensions. The shaded area indicates the 95% confidence interval of the mean target precision. The horizontal coordinates represent the number of iterations, and the vertical coordinates represent the evaluated values at the current number of iterations.

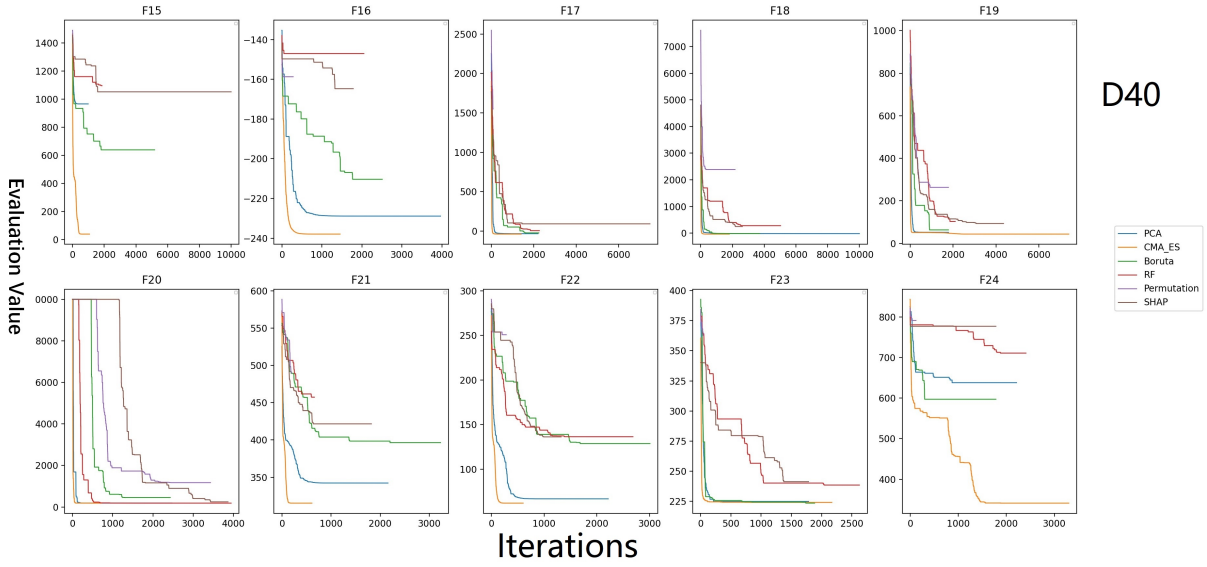

Figure 12: The figure documents the combination of the dimensionality reduction method with CMA-ES and the performance of CMA-ES in general under different functions in **40** dimensions. The horizontal coordinates represent the number of iterations, and the vertical coordinates represent the evaluated values at the current number of iterations.

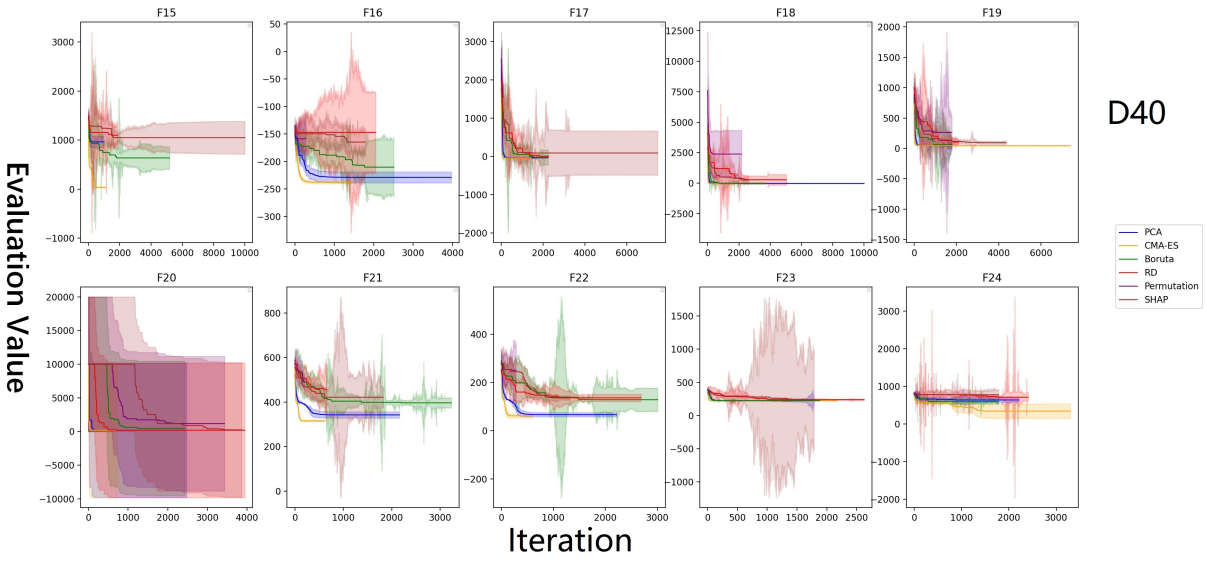

Figure 13: The figure documents the combination of the dimensionality reduction method with CMA-ES and the performance of CMA-ES in general under different functions in **40** dimensions. The shaded area indicates the 95% confidence interval of the mean target precision. The horizontal coordinates represent the number of iterations, and the vertical coordinates represent the evaluated values at the current number of iterations.

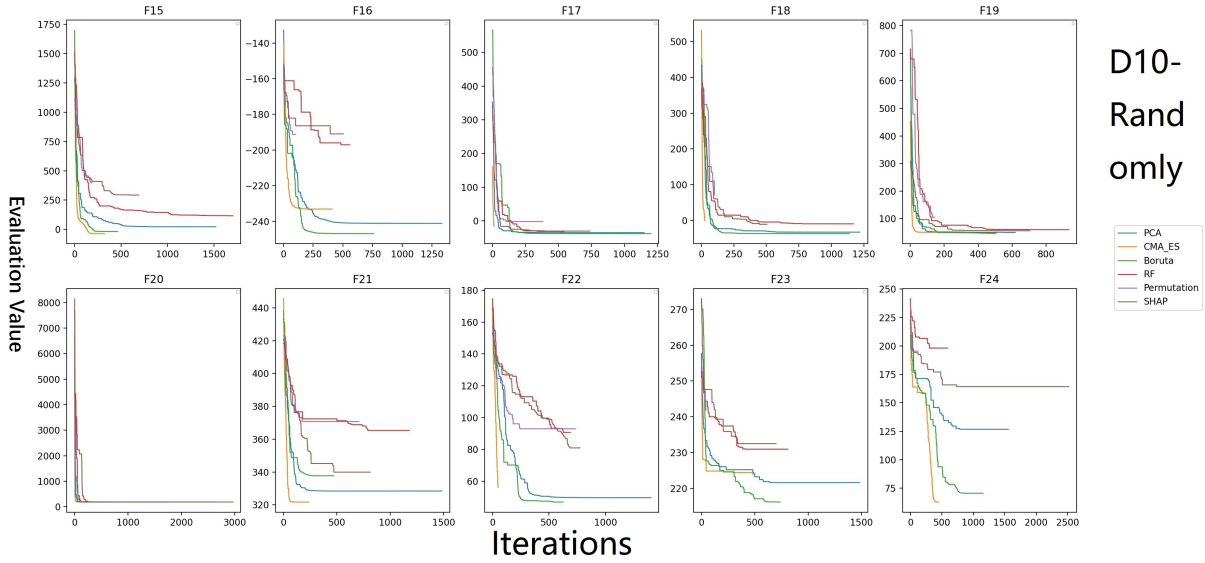

Figure 14: The graph records the overall performance of different dimensionality reduction methods and CMA-ES with different functions in **10** dimensions when the "Guide" method is used. The horizontal coordinates represent the number of iterations and the vertical coordinates represent the evaluated values at the current number of iterations.

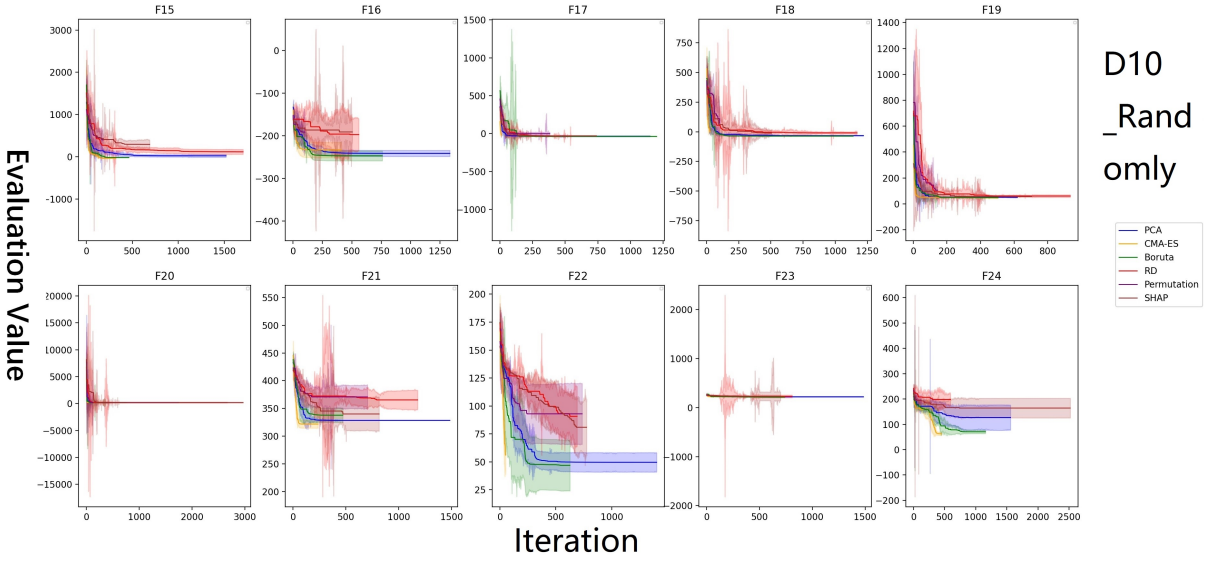

Figure 15: The graph records the overall performance of different dimensionality reduction methods and CMA-ES with different functions in **10** dimensions when the "Guide" method is used. The shaded area indicates the 95% confidence interval of the mean target precision. The horizontal coordinates represent the number of iterations and the vertical coordinates represent the evaluated values at the current number of iterations.

B Dimensional change of experiment

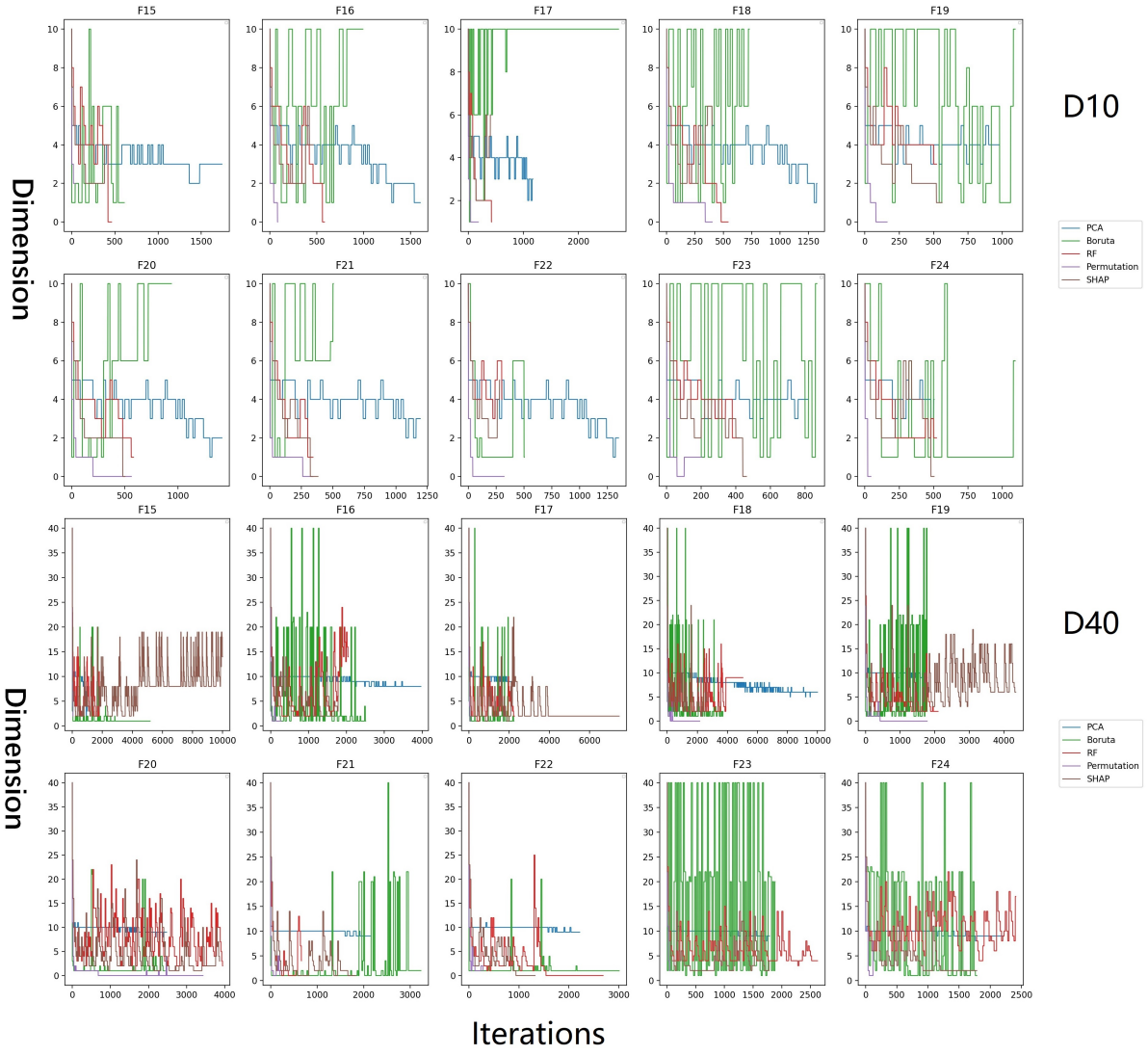

Figure 16: This figure records the dimensionality of the samples generated by the dimensionality reduction method combined with CMA-ES in 10 and 40 dimensions at different functions for each iteration. The top one represents 10 dimensions, and the bottom one represents 40 dimensions. The X-axis represents the number of iterations, and the Y-axis represents the dimensionality.

	PCA		Boruta		RF		Permutation		SHAP	
D10	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)
F15	10	0.06	10	3.45	7	4.33	10	2.38	7	4.28
	5	4.54	6	26.27	6	12.99	3	45.24	5	4.50
	4	29.90	2	29.56	5	26.41	2	52.38	4	36.94
	3	58.60	1	40.72	4	38.96			3	4.50
	2	6.90			3	12.20			2	49.55
F16	5	16.14	10	37.66	6	17.18	10	1.22	6	4.33
	4	39.88	6	38.17	5	6.04	2	47.56	5	4.33
	3	18.69	2	8.06	4	13.75	1	51.22	4	26.41
	2	19.94	1	16.11	3	10.31			3	12.99
	1	5.30			2	45.02			2	47.62
F17	10	0.09	10	91.94	7	4.74	10	0.55	6	10.20
	5	22.06	8	0.73	6	9.48	3	10.44	5	14.93
	4	54.51	6	5.86	4	9.48	2	10.99	4	24.88
	3	19.93	2	0.73	3	4.74	1	78.02	3	9.95
	2	3.41	1	0.73	2	66.35			2	39.80
F18	10	0.08	10	37.04	6	7.38	10	0.25	6	14.93
	5	19.56	7	2.74	5	18.45	4	4.73	5	10.45
	4	48.34	6	32.92	4	22.14	2	9.95	4	29.85
	3	22.66	2	13.58	2	22.14	1	85.07	3	9.95
	2	7.85	1	13.72	1	18.82			2	29.85
	1	1.51								
F19	10	0.10	10	43.42	8	7.47	10	0.66	6	7.12
	5	26.46	8	1.83	6	15.33	4	12.50	4	17.79
	4	61.18	6	29.25	5	7.66	2	13.16	3	21.35
	3	12.26	2	10.88	4	65.13	1	73.68	2	35.59
			1	1.46	3	4.21			1	7.47
F20	10	0.07			6	6.87	10	0.18	7	3.78
	5	18.32	10	36.21	4	44.67	4	3.38	6	3.98
	4	45.26	6	38.34	3	17.18	2	3.56	4	7.97
	3	21.22	2	8.52	2	20.62	1	92.88	3	7.97
	2	13.72	1	16.93	1	3.78			2	76.10
	1	1.41								
F21	10	0.07			6	6.87	10	0.18	7	3.78
	5	18.32	10	36.21	4	44.67	4	3.38	6	3.98
	4	45.26	6	38.34	3	17.18	2	3.56	4	7.97
	3	21.22	2	8.52	2	20.62	1	92.88	3	7.97
	2	13.72	1	16.93	1	3.78			2	76.10
	1	1.41								
F22	10	0.07			8	6.29			8	6.74
	5	19.20	10	3.97	6	26.49	10	0.31	6	7.09
	4	47.44	6	27.78	5	19.87	3	5.90	4	28.37
	3	22.24	2	7.94	4	40.40	1	93.79	3	28.37
	2	9.56	1	60.32	3	6.62			2	28.37
	1	1.48								
F23	10	0.12	10	54.09	6	14.22	10	0.49	7	4.11
	5	29.18	6	18.41	5	14.22	3	9.36	6	4.33
	4	58.49	2	9.21	4	47.39	1	90.15	4	25.97
	3	12.21	1	18.30	3	9.48			3	38.96
					2	9.95			2	26.41
F24	10	0.20	10	7.31	6	11.49	10	2.27	7	3.77
	5	43.45	6	15.90	5	3.83	3	97.73	6	11.90
	4	44.44	2	7.31	4	19.16			4	35.71
	3	11.90	1	69.47	3	11.49			3	19.84
					2	50.19			2	28.57

Table 3: This table shows the changes in dimensionality reduction for the **10D** problem. In this table, D represents the dimension that appears in this iteration, and the percentage represents the percentage of times that dimension appears in the iteration. For each cell of data, only the top 5 dimensions in terms of the percentage of occurrences are kept.

	PCA		Boruta		RF		Permutation		SHAP	
D40	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)	D	Percent(%)
F15	11	5.95	20	1.93	10	7.53	40	2.22	10	7.40
	10	56.51	4	60.39	8	8.61	10	42.22	9	5.00
	9	12.11	3	1.14	6	12.91	4	44.44	8	36.60
	8	10.09	2	11.20	4	18.24	2	11.11	4	7.80
	7	4.04	1	25.33	2	23.67			2	12.20
F16	40	0.03	20	12.75	12	6.90	40	0.35	8	8.97
	11	0.50	4	4.78	6	9.86	13	6.74	6	13.68
	10	42.84	3	3.95	4	16.76	4	7.09	4	14.57
	9	26.22	2	31.89	3	9.86	2	14.18	3	7.85
	8	30.41	1	38.62	2	20.70	1	71.63	2	29.15
F17	40	0.05	40	0.94	8	10.60	40	1.22	8	5.89
	11	1.82	20	5.36	6	11.48	12	23.17	6	2.94
	10	67.26	4	1.79	4	1.41	4	24.39	4	5.08
	9	30.88	2	25.90	3	10.60	2	24.39	2	72.71
			1	62.48	2	28.45	1	26.83	1	2.94
F18	10	15.59	21	3.77	9	24.91	40	0.05	6	9.15
	9	12.60	20	4.85	6	7.55	11	0.88	5	8.38
	8	23.20	3	4.85	4	13.51	4	1.85	4	23.78
	7	18.80	2	41.22	3	9.54	2	3.70	3	9.15
	6	29.20	1	39.36	2	24.63	1	93.52	2	28.96
	5	0.60								
F19	40	0.06	40	6.78	7	3.81	10	1.07	8	9.66
	11	2.24	22	6.73	6	9.53	5	1.12	6	17.68
	10	73.93	20	12.33	4	30.49	2	4.48	4	9.66
	9	22.65	2	29.32	3	10.48	4	1.12	3	7.82
	8	1.12	1	34.75	2	26.63	1	92.15	2	15.18
F20	40	0.04	22	0.82	10	10.65	40	0.03	6	6.21
	11	63.02	20	1.65	8	7.10	11	0.55	4	13.46
	10	1.62	4	3.29	6	15.22	4	1.17	3	5.28
	9	32.90	2	13.17	4	13.70	2	1.17	2	28.99
	8	2.43	1	79.42	2	9.18	1	97.08	1	19.67
F21	40	0.05	22	3.10	10	9.06	40	0.50	8	7.68
	11	2.73	21	1.86	6	6.04	9	9.41	4	9.88
	10	77.81	20	4.34	4	6.34	2	49.50	3	6.59
	9	19.41	2	25.67	2	27.19	1	30.69	2	31.83
			1	61.94	1	30.21	5	9.90	1	26.45
F22	40	0.05	20	1.33	6	5.22	10	6.69	8	7.56
	11	1.81	4	1.33	4	10.45	6	7.04	6	10.59
	10	76.74	3	1.33	3	5.97	3	7.04	4	13.62
	9	21.41	2	5.99	2	17.16	2	28.17	2	16.64
			1	89.98	1	47.76	1	50.70	1	33.43
F23	40	0.06	40	27.65	8	10.67	40	2.27	8	6.73
	11	3.31	22	7.43	7	4.57	11	43.18	6	8.97
	10	75.11	20	10.83	6	20.58	4	45.45	4	18.16
	9	24.53	2	27.55	5	9.15	2	9.09	3	14.57
	8	1.12	1	7.43	4	39.02			2	34.75
F24	40	0.05	40	6.78	14	11.62	40	2.27	8	7.85
	11	0.90	22	14.52	12	8.30	11	43.18	6	13.45
	10	59.58	20	13.45	10	16.60	4	45.45	4	13.45
	9	38.57	2	22.42	8	16.60	2	9.09	3	11.21
	8	0.90	1	28.25	6	10.79			2	41.70

Table 4: This table shows the changes in dimensionality reduction for the **40D** problem. In this table, D represents the dimension that appears in this iteration, and the percentage represents the percentage of times that dimension appears in the iteration. For each cell of data, only the top 5 dimensions in terms of the percentage of occurrences are kept.

C Improvement in Supplementary Experiment

	PCA			Boruta			RF			Permutation			SHAP		
	Prev	Current	Improve	Prev	Current	Improve	Prev	Current	Improve	Prev	Current	Improve	Prev	Current	Improve
F15	28.97	33.85	-16.84%	67.47	-19.70	129.19%	1052.03	93.41	91.12%	1423.39	252.79	82.24%	1138.29	285.28	74.94%
F16	-230.43	-243.13	-5.51%	-241.14	-245.79	-1.93%	-162.21	-212.53	-31.03%	-168.55	-212.20	-25.90%	-171.39	-214.00	-24.86%
F17	-36.11	-35.88	0.66%	-38.51	-37.80	1.85%	280.30	-32.22	111.49%	195.51	-18.83	109.63%	220.61	-32.54	114.75%
F18	-32.29	-33.35	-3.28%	-30.91	-36.99	-19.65%	270.22	-10.84	104.01%	363.70	45.31	87.54%	394.75	-15.64	103.96%
F19	48.56	48.54	0.05%	44.32	47.29	-6.70%	632.57	54.62	91.36%	593.60	86.56	85.42%	572.43	52.25	90.87%
F20	184.61	184.84	-0.12%	185.02	184.83	0.10%	2822.88	185.84	93.42%	2597.55	187.03	92.80%	1526.47	185.66	87.84%
F21	321.05	328.43	-2.30%	327.87	335.86	-2.44%	439.73	354.06	19.48%	419.76	350.92	16.40%	446.33	335.84	24.75%
F22	50.25	53.42	-6.30%	92.83	58.56	36.92%	166.24	83.53	49.75%	165.28	87.58	47.01%	157.64	79.70	49.44%
F23	214.36	218.58	-1.97%	218.55	214.64	1.79%	262.83	226.19	13.94%	244.07	233.95	4.15%	254.15	223.47	12.07%
F24	122.72	111.55	9.10%	69.86	72.39	-3.61%	185.54	161.06	13.20%	207.16	182.31	12.00%	202.05	134.10	33.63%
Avg	67.17	66.68	0.72%	69.54	57.33	17.55%	595.01	90.31	84.82%	604.15	119.54	80.21%	474.13	103.41	78.19%

Table 5: This table represents the difference before and after the "Guide" was used under the ten-dimensional problem, where the dimensionality reduction method has a 20% probability of not reducing the dimensionality. The values under Prev and Current represent the final evaluation value of the two methods respectively, and Improve represents the improvement value of using the latter method over the original method. Red data is the average values.

D Computation cost

	PCA		Boruta		RF		Permutation		SHAP		CMA-ES	
D10	CPU Time/Iter(sec)	Percent(%)	CPU Time/Iter(sec)	Percent(%)	CPU Time/Iter(sec)	Percent(%)	CPU Time/Iter(sec)	Percent(%)	CPU Time/Iter(sec)	Percent(%)	CPU Time/Iter(sec)	Percent(%)
F15	0.023778	79.30%	0.118464	395.09%	0.034564	115.27%	0.206194	687.68%	0.034461	114.93%	0.029984	100.00%
F16	0.028931	111.56%	0.132568	511.17%	0.034558	133.25%	0.108184	417.15%	0.036323	140.06%	0.025934	100.00%
F17	0.029021	1.87%	0.104379	6.73%	0.036758	2.37%	0.533913	34.45%	0.036818	2.38%	1.549944	100.00%
F18	0.039975	5.68%	0.189284	26.89%	0.03132	4.45%	0.047545	6.76%	0.038761	5.51%	0.703848	100.00%
F19	0.017302	16.77%	0.152201	147.54%	0.027211	26.38%	0.052734	51.12%	0.021607	20.95%	0.103159	100.00%
F20	0.012108	126.79%	0.05002	523.77%	0.018593	194.69%	0.053489	560.09%	0.01449	151.73%	0.00955	100.00%
F21	0.195358	114.97%	0.269877	158.82%	0.179014	105.35%	0.171798	101.10%	0.163316	96.11%	0.169922	100.00%
F22	0.059705	24.74%	0.186089	77.10%	0.081449	33.75%	0.039608	16.41%	0.076269	31.60%	0.241361	100.00%
F23	0.046708	144.66%	0.191484	593.03%	0.039544	122.47%	0.021808	67.54%	0.031394	97.23%	0.032289	100.00%
F24	0.05599	168.17%	0.089913	270.06%	0.038033	114.23%	0.338778	1017.53%	0.034201	102.72%	0.033294	100.00%
D40												
F15	1.492534	134.99%	1.279624	115.73%	3.510754	317.52%	11.776563	1065.09%	0.397532	35.95%	1.10569	100.00%
F16	2.703425	259.19%	1.496461	143.47%	1.869975	179.28%	2.737339	262.44%	1.602677	153.66%	1.043036	100.00%
F17	2.713127	216.40%	1.52315	121.48%	1.720872	137.25%	8.744808	697.47%	1.029842	82.14%	1.253784	100.00%
F18	1.63029	132.89%	1.100141	89.67%	0.782471	63.78%	9.830788	801.31%	2.527945	206.05%	1.226835	100.00%
F19	1.774037	186.93%	1.142688	120.40%	2.974737	313.44%	2.364934	249.19%	1.0576	111.44%	0.949062	100.00%
F20	1.621102	2841.15%	0.224417	393.31%	0.17094	299.59%	1.294787	2269.25%	0.326583	572.37%	0.057058	100.00%
F21	4.265215	83.74%	2.080908	40.86%	9.971553	195.78%	22.151512	434.91%	12.301203	241.52%	5.093315	100.00%
F22	1.447814	86.25%	1.285445	76.58%	0.986495	58.77%	2.399194	142.93%	3.10766	185.13%	1.678607	100.00%
F23	2.300985	120.22%	1.403513	73.33%	1.656345	86.54%	7.625	398.38%	3.181455	166.22%	1.914023	100.00%
F24	1.706661	149.67%	1.550094	135.94%	1.332336	116.84%	5.283235	463.33%	2.015325	176.74%	1.140267	100.00%

Table 6: This table shows the computational consumption of all methods for different functions in different dimensions. The computational consumption is obtained by dividing the CPU time by the number of iterations. The Percent under each method refers to the percentage form of the time consumption of that method divided by the time consumption of CMA-ES.