Preprint submitted to IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION
Accepted article (Author's version) DOI: 10.1109/TEVC.2022.3153933

EDA++: Estimation of Distribution Algorithms with Feasibility Conserving Mechanisms for Constrained Continuous Optimization

Abolfazl Shirazi, Josu Ceberio, and Jose A. Lozano, Fellow, IEEE

***Abstract*—Handling non-linear constraints in continuous optimization is challenging, and finding a feasible solution is usually a difficult task. In the past few decades, various techniques have been developed to deal with linear and non-linear constraints. However, reaching feasible solutions has been a challenging task for most of these methods. In this paper, we adopt the framework of Estimation of Distribution Algorithms (EDAs) and propose a new algorithm (EDA++) equipped with some mechanisms to deal with non-linear constraints. These mechanisms are associated with different stages of the EDA, including seeding, learning and mapping. It is shown that, besides increasing the quality of the solutions in terms of objective values, the feasibility of the final solutions is guaranteed if an initial population of feasible solutions is seeded to the algorithm. The EDA with the proposed mechanisms is applied to two suites of benchmark problems for constrained continuous optimization and its performance is compared with some state-of-the-art algorithms and constraint handling methods. Conducted experiments confirm the speed, robustness and efficiency of the proposed algorithm in tackling various problems with linear and non-linear constraints.**

***Index Terms*—Estimation of Distribution Algorithms, Continuous Optimization, Non-linear Constraints, Seeding, Clustering, Mapping**

I. Introduction

MOST real world engineering optimization problems are in continuous domain with nonlinear constraints. In such problems, dealing with nonlinear constraints is challenging and usually it is difficult to find feasible solutions. Constrained optimization problems can be mathematically represented as:

$$\begin{aligned} &Minimize \quad F(x) \quad x = (x_1, x_2, ..., x_n) \\ &Subject\ to \quad G_i(x) \leq 0 \quad i = 1, ..., n_g \\ &\qquad\qquad H_i(x) = 0 \quad i = 1, ..., n_h \\ &\qquad\qquad x_{min} < x_i < x_{max} \end{aligned} \tag{1}$$

where the goal is to minimize the objective function $F(x)$ with respect to the n dimensional parameter vector $x \in R^n$, while the feasible region is restricted by x_{min} and x_{max} as the vectors for lower and upper boundaries, $G_i(x)$ as the function for n_g inequality constraints, and $H_i(x)$ as the function for n_h equality constraints.

In order to approach constrained continuous optimization problems, Evolutionary Algorithms (EAs) utilize a variety of techniques to hold the constraints and return the best possible solutions. In 2018, Coello [1] taxonomized these techniques in different categories. The recently developed algorithms utilize variety of these techniques to handle constraints. For example, in [2], a mutation operator incorporated with a repair projection method has been developed to satisfy the constraints. The presented algorithm is capable of solving optimization problems with linear constraints. In another research by the same authors [3], the parameter settings for the proposed algorithm has been deeply analyzed. A constraint handling method based on covariance matrix adaptation evolution strategy is presented by Sakamoto and Akimot [4]. In this work, the focus was mainly toward the invariance to the element-wise increasing transformation of the objective and constraint functions and the invariance to an affine transformation of the search space. Constraint handling techniques based on differential evolution (DE) algorithms are presented in [5] and [6]. In [7], the concept of multi-objective approach is utilized in designing a ranking strategy for handling constraints. Also, a hybrid surrogate-based-constrained optimization method, equipped with a new constraint-handling strategy is proposed in [8] to map the feasible region into the origin of the Euclidean subspace. Although lots of efforts are made for achieving a robust technique for handling constraints, finding feasible solutions is still a challenge in real-world optimization problems.

Considering these facts, developing novel methods for handling constraints, which are independent from the problem and are robust to any types of constraints (equality, inequality, linear, non-linear, etc.) is a need in the literature. In this research, several mechanisms for estimation of distribution algorithms (EDAs) [9] are proposed for handling constraints. EDAs are a class of EAs that work based on probabilistic models [9]. In an EDA, a probabilistic model is learned at each iteration and new solutions are sampled from that model. The obtained solutions have similar characteristics as those used for learning the model. One of the characteristics of EDAs is to have an explicit description of the promising solutions in terms of probabilistic models. Due to this feature, they have a great potential for enhancement toward further improvements.

Abolfazl Shirazi is with the Machine Learning Group, Basque Center for Applied Mathematics (BCAM), Bilbao, Spain, 48009 (Email: ashirazi@bcamath.org).

Josu Ceberio is with the Intelligent Systems Group and is a member of the Computer Science and Artificial Intelligence Department, University of the Basque Country (UPV/EHU), Donostia, Spain, 20018 (Email: josu.ceberio@ehu.eus).

Jose A. Lozano is with the Department of Computer Science, University of the Basque Country (UPV/EHU) and the Machine Learning Group, Basque Center for Applied Mathematics (BCAM), Bilbao, Spain, 48009 (Email: jlozano@bcamath.org).

This characteristic is the main motivation in this research and the effort here is to enhance the mechanisms of EDAs for handling constraints. In recent years, there have been some efforts in developing probabilistic models that only generate feasible solutions regarding specific types of constraints [10]. However, up to now, the developments were only applicable in combinatorial optimization [11]. In this paper, the main contribution is to present a new concept toward EDAs for handling constraints. In this regard, two of the mechanisms in EDAs, including **SEEDING** and **LEARNING**, have been enhanced and one additional mechanism, called **MAPPING**, have been developed in this research. Having these new mechanisms, the optimization process is modified towards satisfying the constraints, while minimizing the objective function.

This article is organized as follows. In Section II, an overview of the proposed algorithm is presented, where the mechanisms associated with the seeding, learning and mapping stages are briefly introduced. The next sections describe each of these mechanisms in detail. Section III is dedicated to the enhanced seeding mechanism. The learning mechanism based on feasibility conservation is elaborated in Section IV. In Section V, the mapping mechanism is explained. The results of the numerical experiments are provided in Section VI along with comparisons with state-of-the-art algorithms and methods. Conclusions are provided in Section VII.

II. Overview of the Proposed Algorithm

EDAs are a type of population-based evolutionary algorithms designed for solving numerical optimization problems. Based on machine learning techniques, at each iteration, EDAs learn a probabilistic model from a subset of the most promising solutions, trying to explicitly express the interrelations between the different variables of the problem. Then, by sampling the probabilistic model learned in the previous iteration, a new population of solutions is created. In the other words, EDAs work based on two major key methods: learning and sampling, where a probabilistic model that estimates the probability distribution of the selected solutions is learned and then utilized for sampling new individuals [9]. However, in constrained continuous optimization, there are no guarantees that the newly obtained solutions satisfy the constraints of the problem. As previously mentioned, the mechanisms proposed in this work are introduced in the framework of EDAs, named EDA++. The overall pseudo code of the proposed algorithm is in Algorithm 1.

Algorithm 1: Overall pseudo code of EDA++

Input: $F(x), G_i(x), H_i(x), x_{min}, x_{max}$
Parameters: N, i_m , $\epsilon, \gamma, S, \tau, \alpha, \lambda, N_\delta$, MapType
CONSTRUCT $C(x)$ FROM $[G_i(x), H_i(x), \epsilon]$
$[x,c,i] \leftarrow$ **SEEDING**($C(x)$,x_{min},x_{max},N,i_m,γ,S,τ)
$f \leftarrow$ **EVALUATION**(x,$F(x)$)
while $i < i_m$ **do**
  $[x_{sel}, f_{sel}] \leftarrow$ **SELECTION**(x,f,γ)
  $[\Phi, \phi] \leftarrow$ **LEARNING**(x_{sel},f_{sel},$C(x)$, α,λ)
  $x_{sam} \leftarrow$ **SAMPLING**(Φ,ϕ,N)
  $x_{rep} \leftarrow$ **REPAIRING**(x_{sam},x_{min},x_{max})
  $x_{map} \leftarrow$ **MAPPING**(x_{rep},Φ,ϕ, $C(x)$,N_δ,MapType)
  $f_{map} \leftarrow$ **EVALUATION**(x_{map},$F(x)$)
  $[x, f] \leftarrow$ **REPLACEMENT**(x_{map},f_{map},x,f)
  EXTRACT $[x_{best}, f_{best}]$ FROM $[x, f]$;
  UPDATE i
  if *stopping criteria are met* **then**
    BREAK;
EXTRACT $[x_{best}, f_{best}]$ FROM $[x, f]$;
Output: x_{best}, f_{best}

Following the pseudo code of Algorithm 1, the overall optimization process is as follows. In EDA++, the optimization process starts by forming the function $C(x)$ for measuring the infeasibility of solutions as the constraint violation:

$$C(x) = \frac{\sum_{i=1}^{n_g} \hat{G}_i(x) + \sum_{j=1}^{n_h} \hat{H}_j(x)}{n_g + n_h} \tag{2}$$

with $\hat{G}_i(x)$ and $\hat{H}_j(x)$ defined by:

$$\hat{G}_i(x) = \max(0, G_i(x)) \tag{3}$$

$$\hat{H}_j(x) = \begin{cases} |H_j(x)| & |H_j(x)| > \epsilon \\ 0 & |H_j(x)| \leq \epsilon \end{cases} \tag{4}$$

where ϵ is the error margin for equality constraints. Having N as the population size, i as the function evaluation counter, and i_m as the maximum number of function evaluation, the **SEEDING** mechanism is utilized to generate an initial feasible population. Having the initial feasible solutions, with corresponding objective values f obtained from **EVALUATION**, the main optimization loop starts. At each iteration, the algorithm begins by selecting the top promising individuals in the current population according to the **SELECTION** method. Truncation selection method [9] is used in this research, with γ as the truncation factor. Having the selected population x_{sel} and the corresponding objective values f_{sel}, a probability model is learned via the **LEARNING** mechanism. In the proposed learning mechanism, the selected population is divided into several clusters of solutions (Φ and ϕ) with respect to their constraint violation. Then, a mixture of models is learned, one component on top of each cluster, in such a way that the probability of sampling feasible solutions becomes high. Having the mixture of models, new solutions are sampled via the **SAMPLING** method as x_{sam}. The **REPAIRING** method simply refines the newly sampled solutions based on the vectors of boundaries x_{min} and x_{max}. This is done by replacing the out-of-bound solutions with the boundaries of decision variables as:

$$x = \max(\hat{x}, x_{min}) \tag{5}$$

$$x = \min(\hat{x}, x_{max}) \tag{6}$$

where $\hat{x}$ is the out-of-bound solution, and x_{min} and x_{max} are vectors of lower and upper boundaries of the decision variables respectively. It is noteworthy that this strategy might favor

the problems with optimal solutions located in the boundaries. Up to this point, the obtained solutions x_{rep} are likely to be inside the feasible region thanks to the seeding mechanism and the proposed learning mechanism, which is intended to generate feasible solutions. However, despite doing that, the algorithm sometimes generates infeasible solutions. As a result, a **MAPPING** mechanism guarantees the feasibility, and maps all possible infeasible solutions to the feasible region to form a completely feasible population x_{map}. After evaluating the objective value of the obtained solutions, f_{map}, via the **EVALUATION** process, the new individuals are combined with the individuals from the previous population, and the **REPLACEMENT** mechanism is invoked to form the new population and the corresponding objective values f in the current iteration. Population aggregation method is used for this mechanism in this research. As described, the overall concept of optimization process is toward building probabilistic model based on feasible solutions. Therefore, obtaining the initial feasible solution is a key step in the algorithm.

During the optimization process, the counter for function evaluation is updated in every iteration, according to the number of objective/constraint function calls within the proposed mechanisms. The process continues until at least one stopping criteria is met or the function evaluation counter reaches the maximum allowable limit. As explained, EDA++ benefits from three newly developed mechanisms, which are distinct from the typical EDAs. These mechanisms, including seeding, learning and mapping, are described in detail in the following sections.

Algorithm 2: Pseudo code of the seeding mechanism

Input: $C(x)$,x_{min},x_{max},N,i_m,γ,S,τ
$retryFlag \leftarrow false$; $i \leftarrow 0$
while $i < i_m$ **do**
  if $i = 0$ **then**
    $x \leftarrow$ UNIF. DIST. [x_{min},x_{max},N]
    $c \leftarrow$ **EVALUATION**(x,$C(x)$)
  else if $retryFlag = true$ **then**
    $retryFlag \leftarrow false$
    $x_1 \leftarrow$ **SELECTION**(x,c,τ)
    $x_2 \leftarrow$ UNIF. DIST. [x_{min},x_{max},$N(1-\tau)$]
    $x \leftarrow [x_1, x_2]$
    $c \leftarrow$ **EVALUATION**(x,$C(x)$)
  else
    $[x_{sel}, c_{sel}] \leftarrow$ **SELECTION**(x,c,γ)
    $\Phi \leftarrow$ **LEARNING**(x_{sel},c_{sel})
    $x_{sam} \leftarrow$ **SAMPLING**(Φ,N)
    $x_{rep} \leftarrow$ **REPAIRING**(x_{sam},x_{min},x_{max})
    $c_{rep} \leftarrow$ **EVALUATION**(x_{rep},$C(x)$)
    $[x, c] \leftarrow$ **REPLACEMENT**(x_{rep},c_{rep},x,c)
  UPDATE i
  if $max(c) = 0$ **then**
    BREAK;
  if *REMINDER*$(i, S) = 0$ **then**
    $retryFlag \leftarrow true$;
Output: x, c, i

III. Seeding

Providing initial feasible solutions is a priority in EDA++. The aim of the seeding mechanism is to ensure that the initial population is feasible regardless of the objective value of the solutions. The initial population containing only feasible solutions may be available and seeded to the algorithm initially. In this case, the seeding mechanism is skipped. However, if no initial feasible population is provided, the seeding mechanism is invoked. The pseudo code of this mechanism is shown in Algorithm 2.

As shown, the seeding mechanism includes an iterative optimization process based on a multivariate Gaussian EDA that considers the constraint violation function $C(x)$ in Eq. 2 as the temporary objective function. In this process, first an initial random population is created based on a uniform distribution of solutions within the boundaries of x_{min} and x_{max}. Then, the amount of constraint violation of the population is evaluated and if any infeasible solution exists within the population, the mechanism performs the multivariate Gaussian EDA to minimize the constraint violation. Other EDAs could be used as well, including the one with the advanced Gaussian model, which is described in the next section. This iterative process stops when all of the solutions in the population are feasible. Also, this process restarts every S number of iterations, while saving the high quality solutions in terms of constraint violation. In the case of a restart, the top τ fraction of the solutions with lowest constraint violation is saved and added to a new population of solutions with random uniform distribution. The search for the initial population of feasible solutions using this mechanism is a fundamental step. The success of this process relies on the complexity of the problem constraints. Finding feasible solutions has more priority over minimizing the objective function in this algorithm and if the seeding mechanism manages to find enough feasible solutions, the highest performance of the algorithm will be achieved. The mechanism continues searching for feasible solutions until enough solutions are obtained or the maximum function evaluation limit is reached.

IV. Learning

Having a feasible population, obtained from the seeding mechanism, the main loop of the optimization starts. A selection of high quality feasible solutions, x_{sel}, along with their corresponding objective values, f_{sel}, is chosen from the current population, and these are used to estimate the parameters of the probability model. The pseudo code of the learning mechanism is shown in Algorithm 3.

The main idea of the learning process is based on utilizing a mixture of Gaussian distributions as a probabilistic model whose density function is formalized as:

$$f(x) = \sum_{k=1}^{N} \pi_k f_k(x|\mu_k, \Sigma_k) \tag{7}$$

where each $f_k(x|\mu_k, \Sigma_k)$ component of the mixture is a multivariate Gaussian distribution, and μ_k and Σ_k are the mean value (the centroid) and the covariance matrix of the

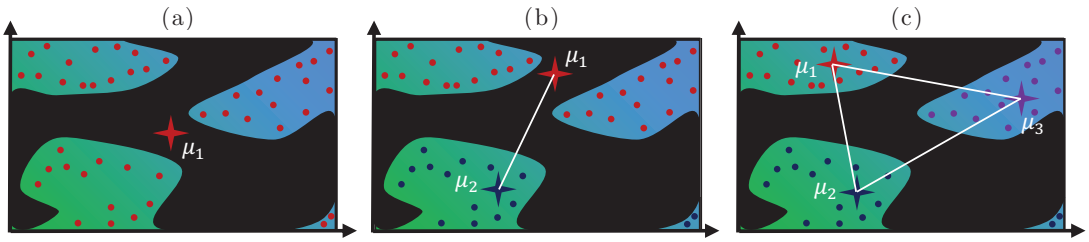

Fig. 1. Clustering the selected population within the learning mechanism

k model for $k = 1, \ldots, N$ respectively, with π_k as the mixing coefficient for the kth component.

In the proposed learning stage, the Gaussian mixture model is constructed in two steps. The first step consists of finding the minimum number of mixture components in which all the centroids (μ_k) are placed inside the feasible region. To this end, an iterative clustering process is developed. This step is represented in lines 2 to 7 of Algorithm 3 and the scheme of this process is illustrated in a schematic instance in Fig. 1.

In the plots, the infeasible region due to the constraints is depicted in black, while the feasible region is illustrated as the color-mapped area. The selected population is plotted and different numbers of clusters (k) are considered. In this research, k–$means$++ is chosen as the clustering method. However, other methods could also be considered. In plot (a), just one cluster is considered ($k = 1$) and thus there is one centroid, which is the mean value of the population. As can be seen, in this case the centroid resides in the infeasible solution, and this probabilistic model does not therefore meet the requirement of satisfying the constraints as the sampled solutions will be mostly in the infeasible region. In plot (b), the population is divided into two clusters ($k = 2$). The positions of the centroids indicate that one of them is inside the feasible region, while the other one is not, leading to the conclusion that this mixture model is also not suitable for constraints satisfaction. Considering three clusters ($k = 3$), yields plot (c) in Fig. 1. As can be appreciated, all centroids are inside the feasible region. Therefore, the mixture of Gaussian distributions model is learned by calculating the maximum likelihood estimators of the parameters of the components in this mixture, using the solutions in the respective clusters. Since all of the centroids are feasible in this case, any solution that is going to be sampled is likely to be inside the feasible domain.

Algorithm 3: Pseudo code of the learning mechanism

Input: x_{sel},f_{sel},$C(x)$, α,λ
1 $N_{sel} \leftarrow size(x_{sel})$
2 **for** $i \leftarrow 1$ **to** N_{sel} **do**
3 $[\iota, \mu] \leftarrow$ kmeans(x_{sel},i);
4 $c_\mu \leftarrow$ **EVALUATION**(μ,$C(x)$)
5 **if** $max(c_\mu) = 0$ **then**
6 BREAK;
7 CONSTRUCT Φ FROM $[\mu, x_{sel}(\iota)]$; $N_c \leftarrow size(\Phi)$
8 **for** $i \leftarrow 1$ **to** N_c **do**
9 EXTRACT $[\hat{x}, \hat{f}, \hat{\mu}, \hat{\sigma}]$ FROM $\Phi(i)$
10 $[\hat{x}_{sel}, \hat{f}_{sel}] \leftarrow$ **SELECTION**($\hat{x}$,$\hat{f}$,α)
11 $d \leftarrow ||\hat{x}_{sel} - \hat{\mu}||$; $j \leftarrow 0$
12 **if** $d > \lambda \times \hat{\sigma}$ **then**
13 $j \leftarrow j + 1$
14 CONSTRUCT $\hat{\phi}$ FROM $[\hat{\mu}, \hat{x}_{sel}]$; $\phi(j) \leftarrow \hat{\phi}$
Output: Φ, ϕ

This process is the first loop in Algorithm 3, representing the first step of the learning process. The obtained number of clusters, N_c, in this scenario is the minimum number of clusters with feasible centroids. Finalizing the process, the components Φ, referred to as the *parent clusters*, are extracted, which contain corresponding solutions $\hat{x}$, objective values $\hat{f}$, centroids $\hat{\mu}$ and covariances $\hat{\sigma}$. Although it is possible to continue increasing the number of components and obtain other mixtures of Gaussian distributions, the computation time will increase without any actual necessity as the objective is to find a minimum number of mixture components with feasible centroids. The main benefit of such a process is that having all of the centroids ($\hat{\mu}$) from the components inside the feasible region significantly reduces the chance of sampling infeasible solutions later on during the sampling process.

The described learning process satisfies the primary requirement for sampling feasible solutions. However, when the mapping mechanism (explained in the next section) is applied to the model that has been created based on this learning process, the covariance matrix tends to shrink, i.e., it loses diversity. This effect reduces the convergence rate of the optimization process. To overcome this drawback, in the next step of the learning process, more components are added to the model. This step is to compensate the covariance loss due to the mapping mechanism that is going to be used in the algorithm [12] after the sampling stage. In this step, for each component Φ_i, first, the top α percentage of the best solutions ($\hat{x}_{sel}$ and $\hat{f}_{sel}$) are selected. Then, the selected set of solutions is analyzed to see if they have outliers using the Z-score outlier detection method [13]. This method is represented as:

$$\frac{||\hat{x}_{sel} - \hat{\mu}||}{\hat{\sigma}} > \lambda \tag{8}$$

where λ is the distance threshold from the centroids $\hat{\mu}$. According to this mechanism, if an outlier solution is at the top α percentage of the best solutions, it will be considered as the centroid of a new component in the mixture $\hat{\phi}$, referred to as an *outlier-based cluster*. For the newly formed components, we assume an independent multivariate Gaussian distribution,

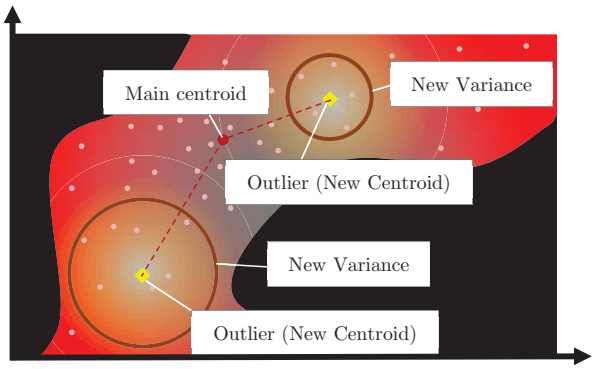

Fig. 2. Formation of outlier-based clusters within the learning mechanism

where the variance of each dimension is calculated as half of the distance from the initial centroid in each component. The illustration of this approach in a schematic instance is shown in Fig. 2 and in lines 8 to 14 of Algorithm 3.

As shown in this instance, the selected population inside the feasible region is depicted along with the corresponding centroid. According to these parameters, two outliers are detected that have the objective function values above a predefined threshold in this population, and they are therefore considered as the centroids for two new components in the mixture. The variance of the independent multivariate Gaussian distributions for each newly generated component is considered as half of the distance from the outlier to the centroid in the component. Overall, a mixture model of three Gaussian models is determined in this iteration: one initial component (parent cluster) and two additional components (outlier-based clusters) due to outliers. It is noteworthy that outlier-based clusters are formed regardless of the number of parent clusters. The aim of outlier based clusters it to increase the quality of the parent clusters and compensate unwanted covariance shrinking caused by the mapping mechanism, which will occur after sampling new individuals. Overall, Gaussian mixture distribution is formed based on the maximum likelihood estimation in each cluster, the sampled population is more likely to have solutions inside the feasible domain since the centroids are feasible.

V. Mapping

Although the proposed learning mechanism generates mixture components with mean values inside the feasible region, when sampling, it is still possible that some samples are generated outside the feasible region. In order to solve this problem, a mapping mechanism is introduced, which is utilized after the repairing process. The simple variation of this concept has been introduced in [14]. The application of this mapping mechanism in a schematic instance is illustrated in Fig. 3.

The presented plots illustrate the mapping mechanism in one iteration of the optimization process. As shown, the mapping method is based on the idea of shifting infeasible points toward their respective centroid in N_δ number of steps until they enter the feasible region. As shown in Fig. 3 (a), the probabilistic model in this iteration is a mixture of two Gaussian models. New solutions are sampled around the centroids. However, not all the samples are inside the feasible region. The feasible and infeasible solutions are marked separately, connected to their respective centroid. The points are mapped in a deterministic equally-spaced form toward the centroids, with N_δ as the maximum number of steps. The amount of displacement in each step is $(\mu_i - x_{j_0})/N_\delta$, where x_{j_0} is the infeasible solution to be mapped toward the centroid μ_i in the ith cluster. As a result, the distance between the infeasible point and the respective centroid is divided into N_δ steps. It is worth noting that, at the final step, the last displacement will be on the centroid. So, the feasibility is guaranteed no matter what the number of N_δ is. However, the higher the number of N_δ is, the more accurate the feasible and infeasible borders that will be discovered. Note that while moving toward the centroids, the mapping is stopped as the point enters the feasible region. The mapped solutions are shown in Fig. 3 (b).

Following the mapping mechanism, it can be highlighted that the number of solutions to be mapped depends on the shape and the boundary of the feasible and infeasible region in the solution domain. The percentage of mapped solutions is problem dependent and varies by the complexity of the constraints and other features such as number and type (equality/inequality) of constraints. Regarding the mapping mechanism, one can consider different approaches depending on their non-linear or stochastic nature. In the following, four alternatives are proposed, represented by *MapType* in

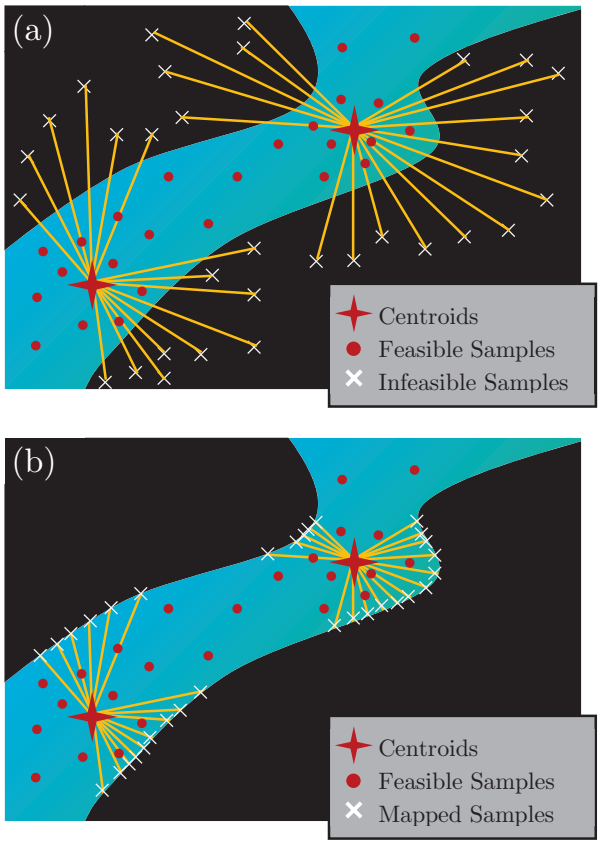

Fig. 3. The process of shifting infeasible individuals toward the centroids, (a) Before mapping (b) After mapping.

Algorithm 1. The major differences between the proposed mechanisms and the original basic concept in [14] are the inclusion of a stochastic behavior and a new heuristic method.

A. Linear Deterministic Mapping

The linear deterministic (LD) mapping is a straightforward method for shifting the infeasible point toward the respective centroid. In this method, the distance between the infeasible solution and the centroid is linearly divided into equal steps. The infeasible solution is moved from its initial position toward the centroid with respect to the steps. In each step the feasibility of the new solution is checked. The process stops when the shifted solution has entered the feasible region.

This mapping process can be represented as:

$$x_j^{new} = x_j + \delta \tag{9}$$

where x_j is the current point within the infeasible region, x_j^{new} is the new shifted solution towards the respective centroid μ_i in the component, and δ is the step calculated as:

$$\delta = \frac{|\mu_i - x_{j_0}|}{N_\delta} \tag{10}$$

where N_δ is the selective total number of steps for this mapping mechanism, and x_{j_0} is the initial position of the infeasible solution.

B. Linear Stochastic Mapping

The linear stochastic (LS) mapping is similar to LD. The only difference is that in each step, after obtaining the new solution, it will also be shifted in a random direction with a variable radius r as:

$$x_j^{new} = x_j + r \times \delta \tag{11}$$

where $0 < r < 1$. This forces a random movement of the point while mapping and may have some advantages depending on the solution domain as it produces diversity to the search.

C. Bisection Deterministic Mapping

Bisection deterministic (BD) mapping is based on repeatedly bisecting the interval defined by the centroid and the infeasible solution. At each step, the distance is divided in two by computing the midpoint of the interval as:

$$\delta = \frac{|\mu_i - x_j|}{2} \tag{12}$$

The feasibility of the midpoint solution is evaluated. If the new solution is feasible, the process stops. Otherwise, the process continues considering the interval between the new obtained solution and the centroid. This process is similar to the well-known bisection method in finding the root of a continuous function, and also the traditional process of binary search described in [14].

D. Bisection Stochastic Mapping

Likewise, the bisection stochastic (BS) mapping is similar to BD. The difference is that in each step, after obtaining the new solution, it will also be shifted in a random direction with a variable radius $0 < r < 1$.

VI. Experiments

The proposed algorithm is tested and compared with different constraint handling techniques and state-of-the-art constrained optimization algorithms [1]. In the first part of the experiment, the efficiency of the algorithm is verified against other constraint handling techniques on the well-known benchmark suite [15], which contains 13 constrained optimization problems, and also the proposed mapping mechanisms are analyzed. In the second experiment, the performance of the algorithm is compared with state-of-the-art algorithms on CEC 2020 test-suite benchmark [16], which contains 57 non-convex constrained optimization problems.

A. Common parameters setup

The experiments are conducted on HIPATIA cluster setup of BCAM, with 18 nodes including 672 cores (Processor Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz) and 3360 GB RAM for the aforementioned runs. In all of the experiments, the following predefined parameters are chosen for EDA++ since tuning the parameters was not the main goal of this research. The initial parameters of N, M, and ϵ are set according to each benchmark. In the seeding mechanism, parameters S and τ are considered as 100 and 0.2. The truncation factor γ for the selection method is chosen as 0.5. Within the learning process, the multivariate Gaussian model is utilized as the component mixture and k–$means$++ is used as the clustering method. The outlier detection parameters are chosen as $\lambda = 1$ and $\alpha = 0.01$. Within the sampling process, for a new size N population, several choices exist for the number of samples for each mixture component. The typical option, which is used in this research, is to dedicate an equal sample size to each component (i.e., N/k) for k components. However, this is an optional choice. Obviously dedicating more samples to the parent clusters or outlier-based clusters acts as a balancing parameter for exploration/exploitation behavior of the optimization algorithm. Also, the number of steps for the mapping mechanism is chosen as $N_\delta = 10$ for all proposed mapping methods. The type of the employed mapping mechanism will be specified in each experiment.

B. Part I - Analysis of EDA++ components

In the first experiment, the efficiency of the proposed algorithm is analyzed with respect to other constraint handling techniques, combined with well-known EAs. The benchmark suite consists of 13 constrained optimization problems in the well-known benchmark of [15]. Since the process of constraint handling method in EDA++ is based on probabilistic models, other algorithms with different techniques are considered for this experiment. In this regard, the performance of the proposed EDA++ is compared with three algorithms: Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [17]. As for the GA, the crossover rate, crossover range factor, mutation percentage and mutation rate of 0.7, 0.4,

[1] All materials for the experiments, including the codes and the results are available at https://github.com/abolfazlshirazi/EDAPP

0.3 and 0.2 are chosen, respectively. Also, the PSO is a vectorized Particle Swarm Optimization with personal learning coefficient a nd g lobal l earning c oefficients eq ual to 2, and inertia weight damping ratio of 0.99. For CMA-ES, default values are used for its parameters as described in [17]. Since the best parameter selection for the algorithms depends on each specific p roblem, t hese v alues a re c hosen a rbitrary for each algorithm. Therefore, it should be highlighted that the obtained results only hold for the given set of parameters. A deeper analysis of the robustness of the results based on the changes of the parameters is left to future works.

Handling the constraints in GA and PSO is based on a static penalty function with a constant coefficient. S everal penalty factors, such as $1, 100$ and 10000, are tested and the best performance of GA and PSO was found to be that with the highest value. The aim of this priliminary test for penalty factor selection was to make sure that the selected value is high enough to add sufficient p enalty t o t he o bjective f unction due to the constraint violation for all problems in the benchmark. In CMA-ES, the resampling technique, as used in [18], was considered as the constraint handling technique and the best obtained feasible solution is saved in each iteration.

In the first e xperiment, e ach a lgorithm i s r un 1 0 t imes for each of the 13 problems of the benchmark. All algorithms started with feasible initial populations. Also, the same initial population is considered in the run of each algorithm in order to have a fair comparison. Considering D as the number of dimensions in each problem, the population size and the number of iterations are considered as $20 \times D$ and $30 \times D$ respectively and no additional stopping criterion is assumed.

A summary of the obtained results is tabulated in Table I. In this table, the four proposed mapping mechanisms are tested in EDA++ along with GA, PSO and CMA-ES. The Relative Best Percentage (RBP) and Average Relative Percentage Deviation (ARPD) values [10] of the obtained feasible solutions across 10 repetitions by each algorithm are provided. RBP is calculated as

$$RBP = min(100 \times |\frac{\hat{f} - f_{best}}{f_{best}}|) \quad (13)$$

$$ARPD = mean(100 \times |\frac{\hat{f} - f_{best}}{f_{best}}|) \quad (14)$$

where $\hat{f}$ is the array of obtained solutions out of 10 runs for each algorithm on a specific problem, and f_{best} is the global best solution for the problem found out of all algorithms.

The results for PSO and GA are regarding the penalty factor of 10000 and as previously mentioned, CMA-ES benefits from the resampling technique for handling constraints. According to the obtained results, although none of the algorithms could find the global optimal solution in all problems, the performance of the proposed algorithm is competitive against GA and PSO incorporated with penalty function and CMA-ES with resampling technique. Comparing the best and mean values of the final solutions confirms the high efficiency of EDA++ relative to the others. It is worth noting that unlike GA, PSO and CMA-ES, EDA++ always returns feasible solutions. In some cases, GA or/and PSO failed to reach feasible solutions, even when the provided initial population is feasible. These cases include 1 run of GA for problem *g10*, 2 runs of PSO for problem *g06* and 4 runs of PSO for problem *g10*. On the other hand, the feasibility of the final solution is guaranteed in EDA++, when the initial feasible solution is provided.

CMA-ES does not return infeasible solutions as it is associated with a trigger that returns the best feasible solution found so far in every iteration after the resampling process. Evaluating the results for CMA-ES shows that, despite returning feasible solutions in all cases, solutions by CMA-ES have the lowest quality in comparison to GA, PSO and EDA++. Considering the quality of the obtained solutions in Table I, EDA++ is either quite superior or has the same performance as the other algorithms in finding good feasible solutions.

Analysis of the time burden of the proposed algorithm is depicted in Fig. 4. In this figure, the boxplot for the execution time of all algorithms is plotted for each problem. All runs are considered in this plot, regardless of whether they achieve feasible or infeasible solutions. Results indicate that the vectorized PSO has the fastest process, due to the parallel computing associated with the structure of the code. On the other hand, the execution time of the proposed algorithm is competitive with GA and CMA-ES. Also, the variance of execution time is higher in EDA++. This is due to the fact that various mapping mechanisms perform several iterations per each infeasible individual in every generation. Depending on how the infeasible solutions are distributed, the mapping mechanism takes variable times. Therefore, the number of iterations for mapping infeasible solutions toward the respective centroids makes the algorithm take longer to converge. However, in exchange for having all feasible solutions by the proposed EDA, the time burden is acceptable and competitive.

Detailed comparison between the proposed mapping mechanisms is depicted in Fig. 5. In this figure, two kinds of plots are illustrated for each of the problems. The top graphs are dedicated to the boxplots for the quality of the obtained solution

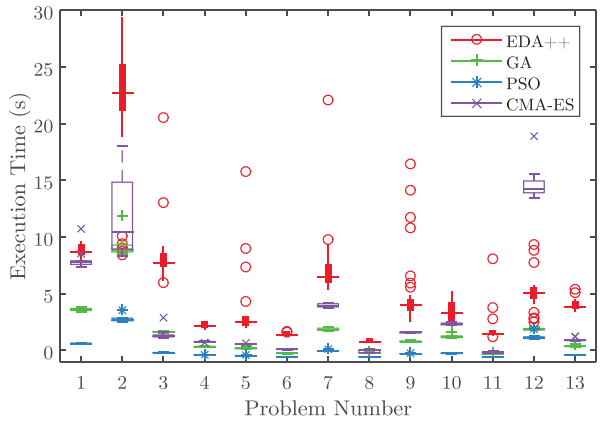

Fig. 4. Comparison of the execution times of the algorithms

TABLE I
RBP AND ARPD VALUES OF THE FEASIBLE SOLUTIONS OBTAINED AFTER 10 RUNS.
$*$ EQUALITY CONSTRAINTS ARE CONVERTED INTO INEQUALITY CONSTRAINTS WITH $\epsilon = 10^{-3}$.
† MAXIMIZATION PROBLEMS ARE CONVERTED INTO MINIMIZATION PROBLEMS.

RBP							
#	GA	PSO	CMA-ES	EDA++ (LD)	EDA++ (LS)	EDA++ (BD)	EDA++ (BS)
1	4.007e-09	**0.000e+00**	1.506e+00	2.728e-07	2.341e-06	3.640e-06	5.424e-06
2†	2.503e+00	4.250e+01	5.011e+01	5.595e+00	**0.000e+00**	2.157e+01	9.397e+00
3†*	2.744e+01	4.757e-01	9.193e+01	1.767e-03	1.866e-03	2.693e-03	**0.000e+00**
4	1.614e-01	4.306e-06	2.287e-01	1.479e-08	**0.000e+00**	6.186e-08	1.600e-08
5*	2.722e-04	2.722e-04	7.172e-04	**0.000e+00**	2.706e-07	5.071e-07	1.001e-06
6	1.042e+00	1.477e+00	1.270e+00	1.600e-04	1.333e-04	**0.000e+00**	5.703e-05
7	8.714e-01	**0.000e+00**	1.729e+01	1.467e+00	1.037e-01	2.377e+00	1.663e+00
8†	1.448e-14	3.483e-07	**0.000e+00**	1.448e-14	1.448e-14	1.448e-14	1.448e-14
9	7.073e-03	4.023e-03	7.612e-01	3.866e-03	**0.000e+00**	7.243e-03	2.722e-04
10*	2.779e+00	**0.000e+00**	8.424e+01	3.064e+00	2.800e+00	7.807e+00	4.204e+00
12†	2.373e-02	3.228e-03	1.511e-01	2.596e-06	2.257e-05	**0.000e+00**	4.241e-07
12	**0.000e+00**	2.909e-11	6.822e-07	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**
13*	2.015e-01	2.015e-01	2.278e+02	4.461e-02	4.603e-02	**0.000e+00**	4.629e-02
ARPD							
#	GA	PSO	CMA-ES	EDA++ (LD)	EDA++ (LS)	EDA++ (BD)	EDA++ (BS)
1	**5.817e-09**	7.490e+00	3.000e+00	3.485e-05	1.333e+00	1.334e+00	1.333e+00
2†	**5.436e+00**	5.243e+01	6.313e+01	1.789e+01	1.508e+01	4.101e+01	2.727e+01
3†*	5.147e+01	1.522e+00	9.828e+01	1.058e+00	2.745e-01	1.990e-02	**5.750e-03**
4	4.361e-01	3.207e-05	1.905e+00	3.231e-05	**5.874e-06**	1.903e-05	2.634e-05
5*	7.145e-04	7.145e-04	4.441e+00	4.659e-04	4.460e-04	4.394e-04	**4.250e-04**
6	9.397e+00	2.882e+00	6.265e+00	2.094e-03	1.880e-03	**1.074e-06**	2.088e-04
7	2.627e+00	**2.218e+00**	5.448e+02	5.497e+00	2.758e+00	6.160e+00	7.762e+00
8†	3.765e-14	3.907e-05	8.497e-02	2.028e-14	**1.738e-14**	2.028e-14	2.172e-14
9	4.303e-02	1.381e-02	1.887e+01	1.718e-02	8.018e-03	2.457e-02	**6.055e-03**
10*	8.236e+00	**2.694e+00**	1.926e+02	1.666e+01	1.369e+01	2.049e+01	1.650e+01
11	1.007e-01	6.427e-02	2.584e+00	1.134e-03	1.328e-03	**1.640e-04**	2.484e-04
12†	**0.000e+00**	4.536e-10	4.144e-03	**0.000e+00**	5.625e-02	5.625e-02	5.625e-02
13*	1.079e+01	1.079e+01	5.696e+03	1.057e+01	1.047e+01	1.040e+01	**1.020e+01**

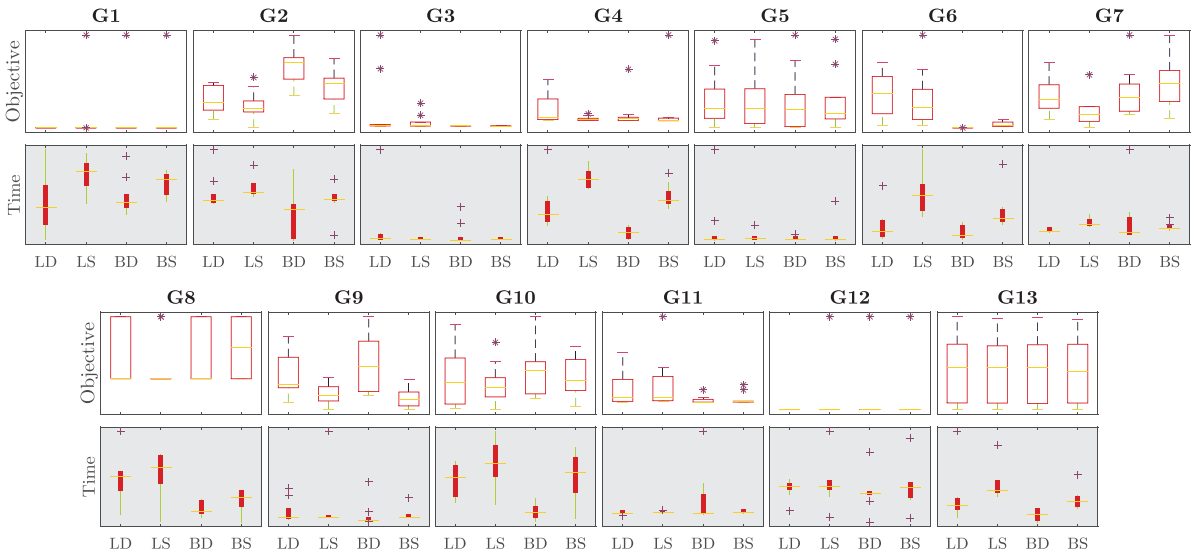

Fig. 5. Comparison of mapping mechanisms

and the second graphs include the boxplots for the execution times of the algorithms. Each boxplot separates the data for all four aforementioned mapping mechanisms. Results shows that none of the mapping mechanisms outperforms the rest in all problems. Therefore, the efficient mapping mechanism depends on the type of the constrained optimization problem.

C. Part II - Comparison with the state-of-art algorithms

In this experiment, we focused on the benchmark of CEC 2020 competition for real-world single objective constrained optimization[16]. This benchmark consists of 57 problems in six categories of problems including industrial chemical processes (1 to 7), process synthesis and design problems (8 to 14), mechanical engineering problems (15 to 33), power system problems (34 to 44), power electronic problems (45

to 50), and livestock feed ration optimization (51 to 57). An overview of the features of the problems in this benchmark is shown in Fig. 6. This figure shows the dimension, number of inequality and equality constraints and the feasibility ratio (F.R.) percentage [2] of the solution domain for all 57 problems. As shown, the majority of the problems have zero F.R. percentage due to the existence of equality constraints. Even some of the problems with only inequality constraints have zero F.R. percentage, since the defined inequality constraints have a similar effect to equality constraints. Also, two problems have an F.R. percentage of 100%, which makes them almost unconstrained problems.

As for the selection of the algorithms for comparison, several factors are considered as the main criteria in the experimental setup. Independency of the algorithms from external optimizers and toolboxes is considered as one of the criteria within the experimental setup. Moreover, the algorithms should not benefit from any prior computed data, such as gradient information of the constraints. Taking these considerations into account, COLSHADE [19], EnMODE [20], and BP-ϵMAg-ES [21] are chosen for this experiment. LSHADE44 [22] from CEC 2017 competition and CORCO [23] are also considered as complementary algorithms. Regarding this, the maximum allowable function evaluation for each problem is considered as:

$$MAX_{FEs} = \begin{cases} 1 \times 10^5 & D \leq 10 \\ 2 \times 10^5 & 10 < D \leq 30 \\ 4 \times 10^5 & 30 < D \leq 50 \\ 8 \times 10^5 & 50 < D \leq 150 \\ 10^6 & 150 < D \end{cases} \tag{15}$$

where D is the dimension of the problem. LD mapping method is chosen in this experiment and equality constraints are converted into inequality constraints with a threshold of $\epsilon = 10^{-4}$. The population size N for EDA++ is considered as

$$N = \min(200, \max(10 \times D, 50)) \tag{16}$$

Also, the default values of the parameters are considered for the rest of the algorithms. According to this setup, each algorithm is run 25 times for each problem and all of the solutions are saved. Results are provided in Table II, including the percentage of feasibility rate (FR) and the mean value of constraint violations (MV). As can be seen, EDA++ managed to find at least one feasible solution (non-zero FR) similar to COLSHADE, LSHADE44, EnMODE and CORCO. In this regard BP-ϵMAg-ES has the best performance. However, for the majority of the problems (1 to 33) EDA++ outperforms BP-ϵMAg-ES in terms of FR percentage. Moreover, for the rest of the problems, EDA++ has competitive performance with CORCO, COLSHADE, and EnMODE. The same results can be observed regarding the MV values. Superior performance has been observed for EDA++ in most of the problems (2-6, 8-10, 12-15, 17-24, 27-33, 44, 45). The performances of COLSHADE and EnMODE are competitive in this matter and they are close to EDA++. For the other problems, one observes a reasonable performance for EDA++ in lowering the constraint violations.

Besides the FR and MV values, the execution times of the algorithms are compared in Fig. 7. As the number of function evaluations is the same for each problem, the relative difference in the execution times can be analyzed. In this figure, the logarithmic scale of the execution time of the algorithms for all of the 57 problems in the 25 executions is compared. Each plot is dedicated to the problems in a unique category in this benchmark. As can be observed, CORCO and EDA++ superbly overpower the rest of the algorithms in almost every case since they require lower execution time. This difference is high in problems 34 to 44 and low in problems 15 to 33. It is worth noting that COLSHADE generally has the highest execution time in all of the categories except the last one, in which BP-ϵMAg-ES is the slowest algorithm.

Having the execution time and the quality of the obtained solutions, the efficiency of the algorithms can be analyzed. To this end, the following efficiency parameter is defined:

$$\Gamma(x) = \begin{cases} 1 + \Gamma_c(x) & \Gamma_c(x) > 0 \\ \Gamma_f(x) & \Gamma_c(x) = 0 \end{cases} \tag{17}$$

where Γ_c and Γ_f are the scaled values of objective function and constraint violation as:

$$\Gamma_f(x) = \frac{F(x) - F_{min}}{F_{max} - F_{min}} \tag{18}$$

$$\Gamma_c(x) = \frac{C(x) - C_{min}}{C_{max} - C_{min}} \tag{19}$$

where F_{min} and F_{max} are the minimum and maximum objective values found by any of the algorithms in the competition

[2]This feature is measured by evaluating 10000 random inputs, with uniform distribution between the upper and lower bounds for each problem.

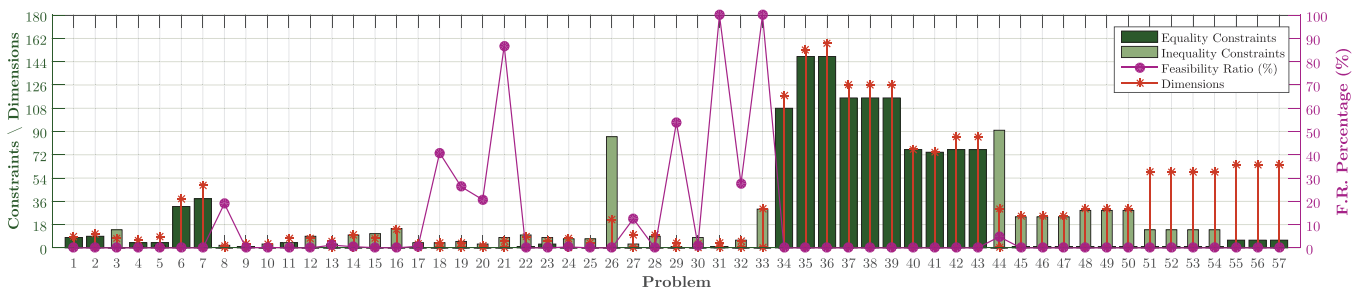

Fig. 6. Features of the benchmark problems in CEC 2020 competition on real-world constrained optimization

TABLE II
COMPARISON OF FEASIBILITY RATE (FR) AND MEAN VALUE OF CONSTRAINT VIOLATION (MV) OF THE ALGORITHMS IN CEC 2020 BENCHMARK

#	FR						MV					
	EDA++	BP-ϵMAg-ES	COLSHADE	LSHADE44	EnMODE	CORCO	EDA++	BP-ϵMAg-ES	COLSHADE	LSHADE44	EnMODE	CORCO
1	28	80	0	4	**100**	0	2.01e-03	1.00e+01	1.89e-01	3.10e-03	**0.00e+00**	1.02e+05
2	**100**	44	**100**	**100**	**100**	**100**	**0.00e+00**	2.28e+01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
3	**100**	64	**100**	**100**	**100**	4	**0.00e+00**	8.95e+02	**0.00e+00**	**0.00e+00**	**0.00e+00**	8.25e+01
4	**100**	**100**	**100**	**100**	**100**	84	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	2.09e-04
5	**100**	52	**100**	**100**	**100**	0	**0.00e+00**	1.77e+01	**0.00e+00**	**0.00e+00**	**0.00e+00**	1.93e+01
6	0	**20**	0	0	0	0	5.58e-01	1.58e+00	**6.90e-02**	1.66e-01	2.22e-01	2.36e+00
7	**0**	**0**	**0**	**0**	**0**	**0**	1.09e+00	6.67e-01	9.98e-02	**7.47e-02**	3.76e-01	1.38e+00
8	**100**	60	**100**	**100**	**100**	**100**	**0.00e+00**	8.75e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
9	**100**	88	**100**	**100**	**100**	**100**	**0.00e+00**	1.59e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
10	**100**	80	**100**	**100**	**100**	**100**	**0.00e+00**	7.17e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
11	20	80	4	96	**100**	4	1.25e-01	2.53e-01	9.39e-02	1.25e-01	**0.00e+00**	2.89e-03
12	**100**	64	**100**	**100**	**100**	**100**	**0.00e+00**	6.15e+01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
13	**100**	76	**100**	**100**	**100**	**100**	**0.00e+00**	2.28e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
14	**100**	**100**	**100**	**100**	**100**	84	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	4.06e-01
15	**100**	92	**100**	**100**	**100**	**100**	**0.00e+00**	1.02e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
16	88	96	**100**	**100**	**100**	**100**	6.46e-02	6.46e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
17	**100**	**100**	**100**	**100**	96	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	1.19e-01	**0.00e+00**
18	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
19	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
20	**100**	80	**100**	**100**	**100**	**100**	**0.00e+00**	5.22e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
21	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
22	**100**	68	96	**100**	**100**	80	**0.00e+00**	3.98e+00	9.09e+04	**0.00e+00**	**0.00e+00**	1.82e+04
23	**100**	88	**100**	**100**	**100**	**100**	**0.00e+00**	1.38e-03	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
24	**100**	92	**100**	**100**	**100**	**100**	**0.00e+00**	9.79e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
25	**100**	**100**	**100**	88	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	1.51e-04	**0.00e+00**	**0.00e+00**
26	68	20	**100**	**100**	**100**	92	2.30e-02	9.29e+00	**0.00e+00**	**0.00e+00**	**0.00e+00**	5.92e-03
27	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
28	**100**	88	**100**	**100**	**100**	**100**	**0.00e+00**	2.03e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
29	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
30	**100**	36	**100**	28	92	**100**	**0.00e+00**	8.14e+04	**0.00e+00**	2.33e+03	6.76e-02	**0.00e+00**
31	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
32	**100**	92	**100**	**100**	**100**	**100**	**0.00e+00**	1.22e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
33	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
34	0	**76**	0	0	0	0	4.33e-01	**1.33e-02**	1.66e-02	4.31e-02	1.46e-01	4.07e-01
35	0	**76**	0	0	0	0	4.85e+00	5.69e+00	**9.99e-02**	1.46e-01	1.98e+00	4.30e+00
36	0	**80**	0	0	0	0	4.54e+00	**2.51e-02**	1.18e-01	2.94e-01	3.23e+00	1.62e+01
37	0	**40**	0	0	0	0	2.93e-01	8.32e-02	**3.56e-02**	5.64e-02	2.44e-01	1.69e-01
38	0	**40**	0	0	0	0	2.85e-01	6.28e-02	**3.81e-02**	5.79e-02	1.30e-01	3.99e-01
39	0	**36**	0	0	0	0	2.79e-01	3.65e-01	**3.85e-02**	5.84e-02	1.29e-01	3.27e-01
40	0	**60**	0	0	0	0	8.71e+00	**4.69e-01**	9.53e-01	1.91e+00	1.56e+00	1.83e+00
41	0	**100**	0	0	0	0	9.11e+00	**0.00e+00**	6.71e-01	1.68e+00	7.38e+00	2.40e+01
42	0	**60**	0	0	0	0	1.55e+01	2.87e+00	**9.33e-01**	2.10e+00	2.31e+00	4.28e+00
43	0	**60**	0	0	0	0	1.42e+01	1.52e+00	**9.97e-01**	2.10e+00	2.80e+00	5.18e+00
44	**100**	**100**	**100**	**100**	**100**	**100**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
45	**100**	92	**100**	**100**	**100**	92	**0.00e+00**	1.22e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	8.59e-08
46	84	60	**100**	**100**	**100**	88	1.43e-02	3.01e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	4.94e-03
47	92	84	**100**	**100**	**100**	**100**	3.82e-03	2.31e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
48	28	84	**100**	88	88	88	4.82e-02	2.51e-03	**0.00e+00**	1.33e-02	1.69e-02	1.96e+00
49	32	60	**100**	**100**	88	76	2.13e-02	9.61e-04	**0.00e+00**	**0.00e+00**	1.55e-02	1.00e+00
50	4	**96**	88	0	8	48	3.33e-02	**6.71e-04**	2.66e-03	7.56e-03	1.53e-02	4.09e-01
51	**0**	**0**	**0**	**0**	**0**	**0**	3.82e-02	1.48e-01	**2.82e-06**	1.57e-05	6.82e-04	1.08e-02
52	88	68	**100**	**100**	**100**	84	1.58e-02	1.50e-02	**0.00e+00**	**0.00e+00**	**0.00e+00**	1.71e-03
53	16	4	**100**	**100**	4	0	3.47e-02	4.76e-02	**0.00e+00**	**0.00e+00**	5.20e-03	4.54e-03
54	0	0	**100**	**100**	0	0	5.76e-02	1.04e+00	**0.00e+00**	**0.00e+00**	1.35e-03	9.62e-03
55	0	0	**24**	0	0	0	1.21e-01	1.93e-02	**2.42e-04**	3.70e-03	4.94e-03	1.51e-02
56	**0**	**0**	**0**	**0**	**0**	**0**	6.63e-02	2.04e-02	**1.13e-03**	8.26e-03	1.01e-02	2.53e-02
57	0	4	**92**	0	0	0	1.93e-01	2.24e-02	**1.46e-04**	1.95e-03	1.45e-03	1.74e-02

regarding a specific problem. Similarly, C_{min} and C_{max} are the lowest and highest constraint violation achieved by any algorithms for each problem. The defined parameters scale the objective score Γ_f and constraint score Γ_c in the interval of 0 and 1 for each solution. Having these scores, the efficiency score Γ will be a score within the interval of $0 \leq \Gamma \leq 2$. Regarding this, all feasible and infeasible solutions will be inside the interval of $0 \leq \Gamma \leq 1$ and $1 \leq \Gamma \leq 2$ respectively. Obviously $\Gamma = 0$ means that the solution is feasible with the best objective value found. If $0 < \Gamma < 1$, it shows that the solution is feasible, but it is not the best solution found in terms of the objective value. If $\Gamma = 1$, it indicates that the solution is feasible (or almost feasible) with the worst objective value in comparison to other obtained feasible solutions. If $1 < \Gamma < 2$, it shows that the solution is infeasible with constraint violation less than the worst solution found. Finally, $\Gamma = 2$ indicates that the solution is infeasible and it has the highest amount of constraint violation. The scaled execution time $\Delta(x)$ is also defined as:

$$\Delta(x) = \frac{T(x) - T_{min}}{T_{max} - T_{min}} \tag{20}$$

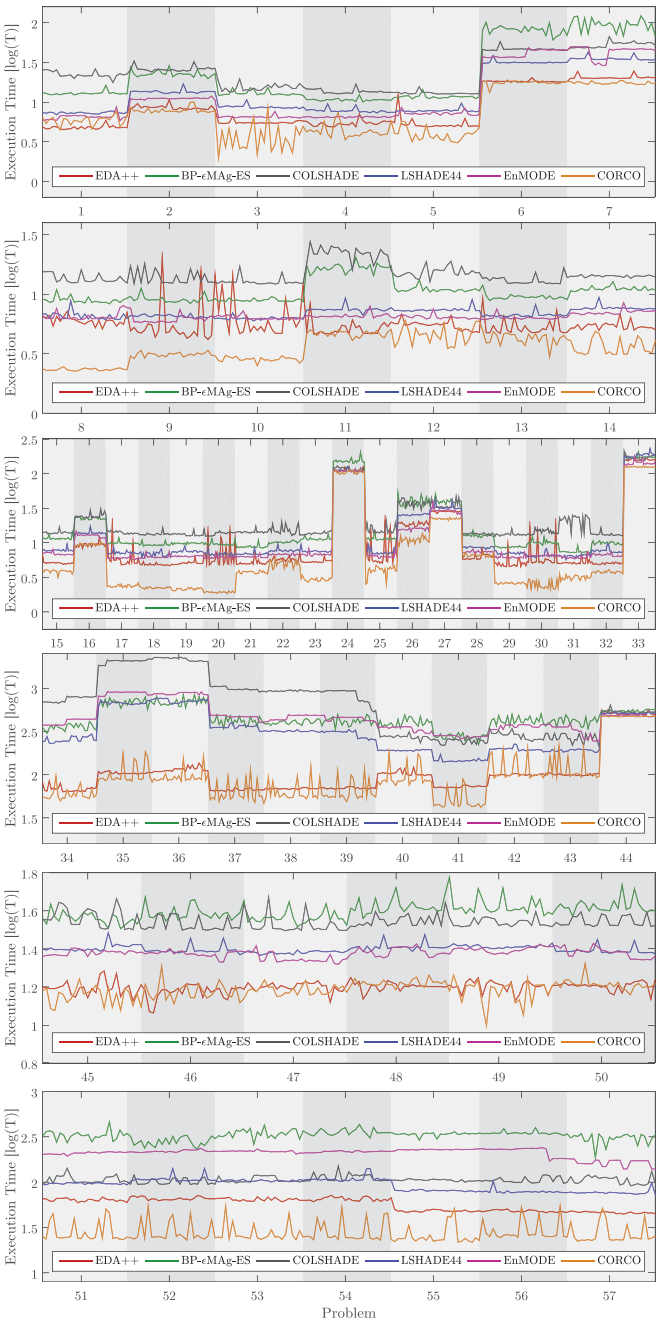

Fig. 7. Algorithms execution times in all repetitions

where $T(x)$ represents the execution time in obtaining the solution x, and T_{min} and T_{max} are the lowest and highest execution times between all algorithms for the corresponding problem.

The defined parameters are scaled scores, and are thus independent from the problem, and can be used to analyze the algorithms considering the entire benchmark. This analysis is shown in Fig. 8. In this figure, each graph is dedicated to an algorithm, plotting the efficiency score and the execution time score for all of the 1425 obtained solutions (25 runs for 57 problems). Obviously, the points closer to the origin represent better solutions in both terms of quality of the solution (objective value and feasibility) and the execution time. The border between feasible and infeasible regions ($\Gamma = 1$) is marked with a dashed line. Comparison of the distribution of points indicates that EDA++ has superior efficiency in comparison to the other algorithms. In this regard, CORCO has the efficiency closest to EDA++. Comparing the number of feasible solutions between CORCO and EDA++ shows that EDA++ has higher feasible solutions than CORCO. However, CORCO performs relatively faster. Also, BP-ϵMAg-ES has the highest number of points inside the feasible region. However, it generally has longer execution times in exchange for the satisfaction of constraints. To be more accurate in this analysis, the Pareto set of the solutions for each problem is extracted and plotted in Fig. 9. The reason for this plot is to find out the group of points that forms the Pareto sets from each algorithm. In other words, Fig. 9 indicates which algorithm has the highest number of dominant obtained solutions in terms of execution time and quality. As can be observed, the majority of the points correspond to EDA++, while LSHADE44 has the lowest number of points within the Pareto sets. The most competitive algorithms in comparison to EDA++ are BP-ϵMAg-ES in terms of feasibility and having a fair amount of dominant solutions with high qualities, and CORCO in terms of execution time.

VII. Conclusion

This paper presented EDA++, a new type of EDA for constrained continuous optimization. The mechanisms associated with the proposed algorithm make it efficient in finding high quality feasible solutions. The proposed mechanisms in this research interact with the seeding, learning and mapping methods within the optimization process. They include a mixture with feasible centroids, outlier detection and heuristic techniques for the mapping process. Taking advantage of these mechanisms, the algorithm has competitive performance in comparison to other state-of-the-art algorithms in this matter. Also, it is capable of generating only feasible solutions if the initial feasible population is provided, regardless of the type of the constraints or the problem. However, there are no guarantees that other techniques reach feasible solutions. The proposed method is generally faster than the state-of-the-art algorithms including COLSHADE, EnMODE, BP-ϵMAg-ES and LSHADE44. However, it consumes more time in comparison to traditional algorithms, such as GA and PSO, equipped with typical constraint handling techniques and also CORCO. Nevertheless, the feasibility of the solutions is a fair exchange when tackling continuous optimization problems with hard constraints.

The aim of the research in this paper was to initially fill the gap in handling constraints with some mechanisms associated with EDAs. This research can be extended to consider the vast majority of the characteristics of the proposed algorithm. One noteworthy point is that the high percentage of the time burden in the proposed method was due to the mapping process, which makes further enhancements in this mechanism in continuous domain a crucial need. Analyzing the percentage of mapped solutions considering the FR ratio of the problems is a good subject in this matter.

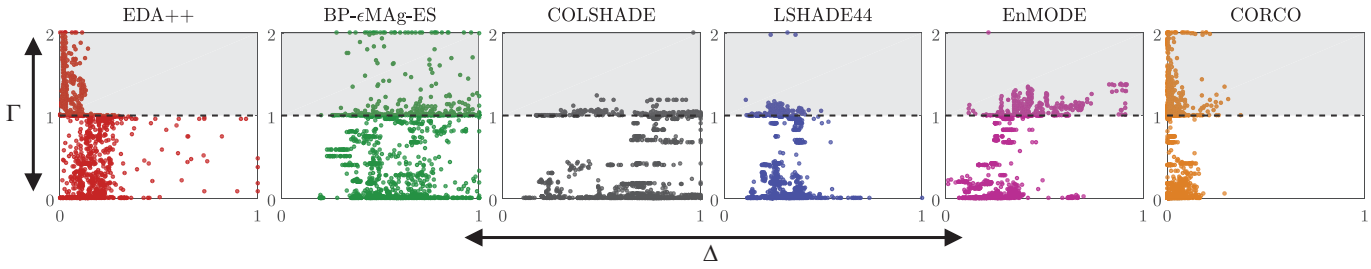

Fig. 8. Comparison of the efficiency of the algorithm Γ vs the scaled execution time Δ

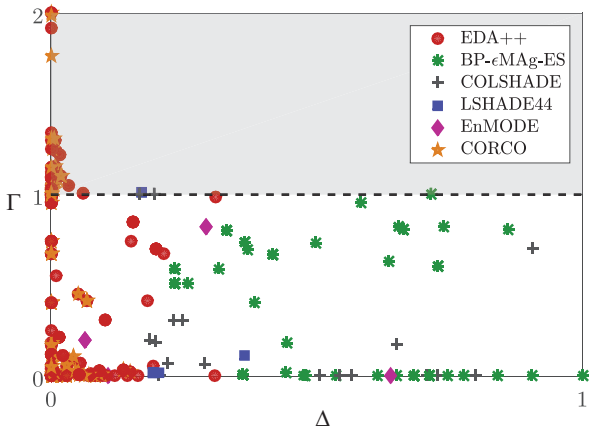

Fig. 9. Pareto sets of each of the 57 constrained problems from CEC2020 benchmark regarding 25 repetitions. Colored markers have been used to highlight the solutions obtained from each algorithm.

Acknowledgment

This research is supported by La Caixa Foundation Fellow-ship, the Basque Government through the BERC 2022-2025, Elkartek programs (project code KK-2020/00049), Spanish Ministry of Economy and Competitiveness MINECO: BCAM Severo Ochoa excellence accreditation SEV-2017-0718, TIN2016-78365R and TIN2017-82626R projects, Spanish Ministry of Science PID2019-106453GA-I00/AEI/10.13039/501100011033, Basque Government consolidated groups 2019-2021 IT1244-19.

References

[1] C. A. C. Coello, "Constraint-handling techniques used with evolutionary algorithms," in *Proc. Genetic and Evol. Comp. Conf. Comp.* ACM, 2018, pp. 773–799.

[2] P. Spettel, H.-G. Beyer, and M. Hellwig, "A covariance matrix self-adaptation evolution strategy for optimization under linear constraints," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 514–524, Jun. 2019.

[3] P. Spettel and H.-G. Beyer, "Analysis of the (μ/μi, λ)-σ-self-adaptation evolution strategy with repair by projection applied to a conically constrained problem," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2019.

[4] N. Sakamoto and Y. Akimoto, "Adaptive ranking based constraint handling for explicitly constrained black-box optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 700–708.

[5] C. Zhang, A. K. Qin, W. Shen, L. Gao, K. C. Tan, and X. Li, "ϵ -constrained differential evolution using an adaptive ϵ -level control method," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–17, 2020.

[6] B.-C. Wang, H.-X. Li, J.-P. Li, and Y. Wang, "Composite differential evolution for constrained evolutionary optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 7, pp. 1482–1495, Jul. 2019.

[7] A. Kumar, S. Das, and R. Mallipeddi, "A reference vector-based simplified covariance matrix adaptation evolution strategy for constrained global optimization," *IEEE Transactions on Cybernetics*, pp. 1–14, 2020.

[8] Y. Su, L. Xu, and E. D. Goodman, "Hybrid surrogate-based constrained optimization with a new constraint-handling method," *IEEE Transactions on Cybernetics*, pp. 1–14, 2020.

[9] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer, 2001.

[10] J. Ceberio, A. Mendiburu, and J. A. Lozano, "A square lattice probability model for optimising the graph partitioning problem," in *IEEE Cong. on Evol. Comp.* IEEE, Jun. 2017.

[11] ——, "Distance-based exponential probability models on constrained combinatorial optimization problems," in *Proc. Genetic and Evol. Comp. Conf. Comp.* ACM Press, 2018.

[12] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Comput. Method Appl. M.*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[13] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[14] G. Leguizamón and C. A. C. Coello, "Boundary search for constrained numerical optimization problems in ACO algorithms," in *Ant Colony Optimization and Swarm Intelligence*. Springer Berlin Heidelberg, 2006, pp. 108–119.

[15] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comp.*, vol. 4, no. 3, pp. 284–294, 2000.

[16] A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, P. N. Suganthan, and S. Das, "A test-suite of non-convex constrained optimization problems from the real-world and some baseline results," *Swarm and Evolutionary Computation*, p. 100693, 2020.

[17] M. Andersson, S. Bandaru, A. H. Ng, and A. Syberfeldt, "Parameter tuned cma-es on the cec 15 expensive problems," in *IEEE Cong. on Evol. Comp.* IEEE, May 2015.

[18] S. Rodrigues, P. Bauer, and P. A. Bosman, "A novel population-based multi-objective cma-es and the impact of different constraint handling techniques," in *Proc. Genetic and Evol. Comp. Conf. Comp.*, 2014, pp. 991–998.

[19] J. Gurrola-Ramos, A. Hernandez-Aguirre, and O. Dalmau-Cedeno, "COLSHADE for real-world single-objective constrained optimization problems," in *IEEE Cong. on Evol. Comp.* IEEE, Jul. 2020.

[20] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty, and M. J. Ryan, "Multi-operator differential evolution algorithm for solving real-world constrained optimization problems," in *IEEE Cong. on Evol. Comp.* IEEE, Jul. 2020.

[21] M. Hellwig and H.-G. Beyer, "A modified matrix adaptation evolution strategy with restarts for constrained real-world problems," in *IEEE Cong. on Evol. Comp.* IEEE, Jul. 2020.

[22] Z. Fan, Y. Fang, W. Li, Y. Yuan, Z. Wang, and X. Bian, "LSHADE44 with an improved constraint-handling method for solving constrained single-objective optimization problems," in *IEEE Cong. on Evol. Comp.* IEEE, Jul. 2018.

[23] Y. Wang, J.-P. Li, X. Xue, and B. chuan Wang, "Utilizing the correlation between constraints and objective function for constrained evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 29–43, Feb. 2020.